



SQL Anywhere® Server Programmierung

Version 16.0

Februar 2013

Version 16.0
Februar 2013

© 2013 SAP AG oder ein SAP-Konzernunternehmen. Alle Rechte vorbehalten.

Sie können diese Dokumentation (ganz oder teilweise) unter folgenden Bedingungen benutzen, reproduzieren und verteilen: 1) Sie müssen diese und alle anderen Urheberrechtsvermerke auf allen Kopien oder Auszügen der Dokumentation wiedergeben. 2) Sie dürfen die Dokumentation nicht verändern. 3) Sie dürfen nichts tun, aus dem abgeleitet werden könnte, dass Sie oder jemand anderer als SAP Verfasser oder Quelle der Dokumentation ist. Die hier enthaltenen Informationen können jederzeit ohne vorherigen Hinweis geändert werden.

Einige Softwareprodukte, die von der SAP AG oder einem ihrer Vertriebspartner vermarktet werden, enthalten Softwarekomponenten anderer Softwareanbieter. Die nationalen Produktspezifikationen können unterschiedlich sein.

Diese Dokumentationen werden von der SAP AG und ihren Tochtergesellschaften ("SAP Group") lediglich zu Informationszwecken bereitgestellt, ohne dass eine Gewährleistung oder eine Garantie irgendeiner Art gegeben wird. Die SAP Group übernimmt keine Verantwortung im Hinblick auf Fehler oder Auslassungen in den Dokumentationen. Die einzigen Garantien für Produkte und Dienstleistungen der SAP Group sind diejenigen, die in den mit den Produkten und Dienstleistungen eventuell gelieferten ausdrücklichen Garantieerklärungen enthalten sind. Keine der hier enthaltenen Informationen kann als Gewährung einer weitergehenden Garantie betrachtet werden.

SAP und weitere erwähnte SAP-Produkte und -Dienstleistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der SAP AG in Deutschland und anderen Ländern. Weitere Hinweise finden Sie unter <http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark>.

Inhalt

Über diese Dokumentation	ix
Anwendungsentwicklung mit SQL	1
Ausführung von SQL-Anweisungen in Anwendungen	1
Vorbereitete Anweisungen	2
Cursorverwendung	5
Cursor-Grundsätze	8
Cursortypen	15
SQL Anywhere-Cursor	16
Ergebnismenge-Deskriptoren	35
Transaktionen in Anwendungen	37
.NET-Anwendungsprogrammierung	41
SQL Anywhere .NET-Datenprovider	41
Praktische Einführungen zum .NET-Datenprovider	79
SQL Anywhere ASP.NET-Provider	97
SQL Anywhere-.NET-API-Referenz	105
OLE DB- und ADO-Entwicklung	337
OLE DB	337
ADO-Programmierung mit SQL Anywhere	338
OLE DB-Verbindungsparameter	345
OLE DB-Verbindungspooling	347
Microsoft-Verbindungsserver	348
Unterstützte OLE DB-Schnittstellen	352
OLE DB-Provider-Registrierung	357
ODBC-Unterstützung	359
Voraussetzungen für die Entwicklung von ODBC-Anwendungen	359
Entwicklung von ODBC-Anwendungen	361

ODBC-Beispiele	366
ODBC-Handles	368
ODBC-Verbindungsfunktionen	370
Von ODBC geänderte Serveroptionen	374
Erweiterte Verbindungsattribute für SQLSetConnectAttr	375
Hinweise zur Windows-Funktion DIIMain	377
Möglichkeiten zum Ausführen von SQL-Anweisungen	378
Hinweise zu 64-Bit-ODBC	382
Datenausrichtungsanforderungen	386
Ergebnismengen in ODBC-Anwendungen	388
Hinweise zu gespeicherten Prozeduren	393
ODBC Escape-Syntax	394
Fehlerbehandlung in ODBC	397
 Java in der Datenbank	 401
Informationen zu Java in der Datenbank	401
Häufig gestellte Fragen zu Java in der Datenbank	401
Fehlerbehandlung in Java	403
Praktische Einführung: Java in der Datenbank verwenden	404
Java-Klassen in Datenbanken installieren	410
Besondere Funktionen von Java-Klassen in der Datenbank	413
Java VM starten und stoppen	417
Shutdown-Hooks in der Java VM	417
 JDBC-Unterstützung	 419
JDBC-Anwendungen	419
JDBC-Treiber	420
JDBC-Programmstruktur	422
Unterschiede zwischen client- und serverseitigen JDBC-Verbindungen ..	422
SQL Anywhere JDBC-Treiber	423
jConnect-JDBC-Treiber	424
Verbindungen aus einer JDBC-Clientanwendung	427
Verbindung von einer serverseitigen JDBC-Klasse aus herstellen	431
Hinweise zu JDBC-Verbindungen	434

Datenzugriff mit JDBC	436
JDBC-Callbacks	445
JDBC-Escape-Syntax	449
JDBC 4.0-API-Unterstützung	453
OData-Unterstützung	455
OData Server-Architektur	455
Einschränkungen für den OData Server	456
Sicherheitshinweise zum OData Server	456
OData Server konfigurieren	457
HTTP-Server für OData-Vorgänge einrichten	461
OData Producer-Dienstmodelle erstellen	463
OData Server ausführen und Clients einrichten	464
OData Server-Dienstprogramm (<i>dbosrv16.exe</i>)	464
OData-Dienstprogramm Serverstopp (<i>dbostop.exe</i>)	465
Embedded SQL	467
Überblick über den Entwicklungsprozess	468
Der SQL-Präprozessor	469
Unterstützte Compiler	473
Header-Dateien für Embedded SQL	473
Importbibliotheken	474
Beispielprogram mit Embedded SQL	475
Struktur von Embedded SQL-Programmen	475
DBLIB unter Windows dynamisch laden	476
Beispielprogramme mit Embedded SQL	477
Datentypen in Embedded SQL	482
Hostvariablen in Embedded SQL	485
SQL-Kommunikationsbereich (SQLCA)	495
Static und Dynamic SQL	501
Der SQL-Deskriptor-Bereich (SQLDA)	504
Abruf von Daten mit Embedded SQL	514
Senden und Abrufen langer Werte mit Embedded SQL	522
Einfache gespeicherte Prozeduren in Embedded SQL	528

Anforderungsverwaltung mit Embedded SQL	531
Datenbanksicherung mit Embedded SQL	531
Referenz der Bibliotheksfunktionen	532
Zusammenfassung der Embedded SQL-Anweisungen	561
SQL Anywhere-Datenbank-API für C/C++	565
SQL Anywhere-C-API-Unterstützung	565
SQL Anywhere-C-API-Referenz	566
SQL Anywhere-Schnittstelle für externe Aufrufe	601
Prozeduren und Funktionen, die externe Aufrufe verwenden	601
Prototypen externer Funktionen	603
Schnittstellenmethoden für externen Funktionsaufruf	612
Behandlung von Datentypen	616
Externe Vorfilterbibliotheken entladen	618
Unterstützung für externe Umgebungen in SQL Anywhere	621
Die externe CLR-Umgebung	622
Die externen ESQL- und ODBC-Umgebungen	625
Die externe Java-Umgebung	635
Die externe PERL-Umgebung	640
Die externe PHP-Umgebung	644
Perl/DBI-Unterstützung	651
DBD::SQLAnywhere	651
DBD::SQLAnywhere unter Windows installieren	651
DBD::SQLAnywhere unter Unix und Mac OS X installieren	653
Perl-Skripten, die DBD::SQLAnywhere verwenden	654
Python-Unterstützung	661
sqlanydb	661
Installieren der Python-Unterstützung unter Windows	662

Installieren der Python-Unterstützung unter Unix und Mac OS X	662
Python-Skripten, die sqlanydb verwenden	663
PHP-Unterstützung	669
SQL Anywhere-PHP-Unterstützung	669
Überblick über die SQL Anywhere-PHP-API	684
Ruby-Unterstützung	717
SQL Anywhere-Ruby-API-Unterstützung	717
SQL Anywhere-Ruby-API-Referenz	725
Sybase Open Client-Unterstützung	747
Die Open Client-Architektur	748
Was Sie für die Entwicklung von Open Client-Anwendungen brauchen ...	749
Open Client-Datentypzuordnungen	749
SQL in Open Client-Anwendungen	751
Bekannte Open Client-Einschränkungen von SQL Anywhere	753
HTTP-Webdienste	755
SQL Anywhere als HTTP-Webserver	755
Zugriff auf Webdienste mithilfe von Webclients	794
Fehlercodereferenz für Webdienste	832
Beispiele für HTTP-Webdienste	834
Dreischichtige Datenverarbeitung und verteilte Transaktionen ...	865
Dreischichtige Datenverarbeitungsarchitektur	865
Verteilte Transaktionen	869
Datenbanktools-Schnittstelle (DBTools)	871
DBTools-Importbibliotheken	872
Initialisierung und Finalisierung der DBTools-Bibliothek	872

DBTools-Funktionsaufrufe	873
Callback-Funktionen	873
Versionsnummern und Kompatibilität	875
Bitfelder	876
Ein Beispiel für DBTools	876
SQL Anywhere-Datenbanktools - C-API-Referenz	878
Exit-Codes der Softwarekomponenten	945
 Deployment von Datenbanken und Anwendungen	 947
Deployment-Typen	947
Möglichkeiten zur Weitergabe von Dateien	948
Installationsverzeichnisse und Dateinamen	948
Deployment-Assistent	952
Dialogfreie Installationen mit dem SQL Anywhere- Installationsprogramm	956
Anforderungen für das Deployment von Clientanwendungen	960
Deployment von Administrationstools	992
Deployment der Dokumentation	1015
Deployment des Datenbankservers	1016
DLL-Registrierung unter Windows	1023
Deployment der Unterstützung für externe Umgebungen	1023
Verschlüsselungs-Deployment	1026
Deployment von Anwendungen mit eingebetteten Datenbanken	1027
 Index	 1033

Über diese Dokumentation

In diesem Handbuch wird beschrieben, wie Sie Datenbank Anwendungen mithilfe von C-, C++-, Java-, Perl-, PHP-, Python-, Ruby- und .NET-Programmiersprachen; wie etwa Visual Basic und Visual C#; schreiben. Es werden verschiedene Programmierschnittstellen beschrieben, wie ADO.NET, OLEDB und ODBC.

Anwendungsentwicklung mit SQL

In diesem Abschnitt finden Sie Informationen über die Verwendung von SQL in Anwendungen.

Ausführung von SQL-Anweisungen in Anwendungen

Wie Sie SQL-Anweisungen in Ihre Anwendung einbauen, hängt vom Entwicklungstool und der Programmierschnittstelle ab, die Sie verwenden.

- **ADO.NET** Sie können SQL-Anweisungen mit verschiedenen ADO.NET-Objekten ausführen. Das `SACCommand`-Objekt ist ein Beispiel:

```
SACCommand cmd = new SACCommand(  
    "DELETE FROM Employees WHERE EmployeeID = 105", conn );  
cmd.ExecuteNonQuery();
```

- **ODBC** Wenn Sie direkt in die ODBC-Programmierchnittstelle schreiben, erscheinen Ihre SQL-Anweisungen in Funktionsaufrufen. Beispiel: Der folgende C-Funktionsaufruf führt eine DELETE-Anweisung aus:

```
SQLExecDirect( stmt,  
    "DELETE FROM Employees  
    WHERE EmployeeID = 105",  
    SQL_NTS );
```

- **JDBC** Wenn Sie die JDBC-Programmierchnittstelle verwenden, können Sie SQL-Anweisungen ausführen, indem Sie Methoden des `statement`-Objekts aufrufen. Beispiel:

```
stmt.executeUpdate(  
    "DELETE FROM Employees  
    WHERE EmployeeID = 105" );
```

- **Embedded SQL** Wenn Sie Embedded SQL verwenden, erhalten die SQL-Anweisungen der C-Sprachen das Schlüsselwort `EXEC SQL` als Präfix. Der Code wird dann im Präprozessor verarbeitet, bevor er kompiliert wird. Beispiel:

```
EXEC SQL EXECUTE IMMEDIATE  
'DELETE FROM Employees  
WHERE EmployeeID = 105';
```

- **Sybase Open Client** Wenn Sie die Sybase Open Client- Schnittstelle verwenden, erscheinen Ihre SQL-Anweisungen in Funktionsaufrufen. Die folgenden beiden Aufrufe führen zum Beispiel eine DELETE-Anweisung aus:

```
ret = ct_command( cmd, CS_LANG_CMD,  
    "DELETE FROM Employees  
    WHERE EmployeeID=105"  
    CS_NULLTERM,  
    CS_UNUSED);  
ret = ct_send(cmd);
```

Weitere Hinweise über die Aufnahme von SQL in Ihre Anwendung finden Sie in Ihrer Entwicklungstool-Dokumentation. Wenn Sie ODBC oder JDBC verwenden, finden Sie weitere Hinweise im Software Development Kit für diese Schnittstellen.

Anwendungen im Datenbankserver

Auf vielerlei Weise funktionieren gespeicherte Prozeduren und Trigger wie Anwendungen oder Teile von Anwendungen, die im Server laufen. Sie können viele der hier beschriebenen Verfahren auch in gespeicherten Prozeduren verwenden.

Java-Klassen in der Datenbank können die JDBC-Schnittstelle genauso benutzen wie Java-Anwendungen dies außerhalb des Servers tun. In diesem Abschnitt werden einige Aspekte von JDBC beschrieben.

Siehe auch

- „SQL Anywhere .NET-Datenprovider“ auf Seite 41
- „ODBC-Unterstützung“ auf Seite 359
- „JDBC-Unterstützung“ auf Seite 419
- „Embedded SQL“ auf Seite 467
- „Sybase Open Client-Unterstützung“ auf Seite 747
- „Gespeicherte Prozeduren, Trigger, Batches und benutzerdefinierte Funktionen“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*]
- „JDBC-Unterstützung“ auf Seite 419

Vorbereitete Anweisungen

Sobald eine Anweisung an eine Datenbank gesendet wird, muss der Datenbankserver folgende Schritte ausführen:

- Er muss die Anweisung syntaktisch analysieren und in eine interne Form umwandeln. Dieser Prozess wird auch als **Vorbereiten** der Anweisung bezeichnet.
- Er muss alle Referenzen auf Datenbankobjekte auf ihre Richtigkeit prüfen, indem er beispielsweise überprüft, ob in der Abfrage genannte Spalten vorhanden sind.
- Wenn die Anweisung Joins oder Unterabfragen umfasst, generiert der Abfrageoptimierer einen Zugriffsplan.
- Er führt die Anweisung aus, nachdem alle diese Schritte durchgeführt wurden.

Die Wiederverwendung von vorbereiteten Anweisungen kann die Performance verbessern

Wenn Sie dieselbe Anweisung immer wieder verwenden, etwa das Einfügen von mehreren Zeilen in eine Tabelle, kommt es zu bedeutenden und unnötigen Overheads durch die wiederholte Vorbereitung der Anweisung. Um dies zu vermeiden, bieten einige Datenbank-Programmierschnittstellen eine Möglichkeit zur Verwendung von vorbereiteten Anweisungen. Eine **vorbereitete Anweisung** ist eine Anweisung, die eine Reihe von Platzhaltern enthält. Wenn Sie die Anweisung ausführen möchten, ordnen Sie den Platzhaltern Werte zu, statt die gesamte Anweisung erneut vorzubereiten.

Die Verwendung von vorbereiteten Anweisungen ist nützlich, wenn viele ähnliche Aktionen ausgeführt werden, z.B. das Einfügen zahlreicher Zeilen.

Im Allgemeinen erfordern vorbereitete Anweisungen die folgenden Schritte:

1. **Anweisung vorbereiten** In diesem Schritt versehen Sie die Anweisung im Allgemeinen mit Platzhalterzeichen anstelle von Werten.
2. **Vorbereitete Anweisung wiederholt ausführen** In diesem Schritt geben Sie Werte an, die jedes Mal verwendet werden sollen, wenn die Anweisung ausgeführt wird. Die Anweisung braucht nicht jedes Mal vorbereitet zu werden.
3. **Anweisung löschen** In diesem Schritt geben Sie die Ressourcen frei, die der vorbereiteten Anweisung zugeordnet waren. Einige Programmierschnittstellen verarbeiten diesen Schritt automatisch.

Keine Anweisungen vorbereiten, die nur einmal verwendet werden

Im Allgemeinen sollten Sie Anweisungen nicht vorbereiten, wenn sie nur einmal ausgeführt werden sollen. Bei getrennter Vorbereitung und Durchführung kommt es zu leichten Einbußen bei der Performance, und außerdem werden dadurch unnötig komplizierte Schritte in die Anwendung eingefügt.

Bei manchen Schnittstellen müssen Sie allerdings eine Anweisung vorbereiten, um sie mit einem Cursor zu verbinden.

Die Aufrufe für die Vorbereitung und Ausführung von Anweisungen sind nicht Teil der SQL und unterscheiden sich daher je nach verwendeter Schnittstelle. Jede Programmierschnittstelle von SQL Anywhere bietet eine Methode für die Verwendung von vorbereiteten Anweisungen.

Siehe auch

- [„Cursorverwendung“ auf Seite 5](#)

Vorbereitete Anweisungen – Überblick

In diesem Abschnitt wird ein kurzer Überblick über die Verwendung von vorbereiteten Anweisungen gegeben. Die allgemeine Vorgehensweise ist identisch, aber die Details unterscheiden sich je nach Schnittstelle. Ein Vergleich der Verwendung von vorbereiteten Anweisungen in den einzelnen Schnittstellen kann diesen Punkt vielleicht klarer darstellen.

In der Regel führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung zu verwenden:

1. Bereiten Sie die Anweisung vor.
2. Binden Sie die Parameter, die Werte in der Anweisung enthalten.
3. Weisen Sie den Bindungsparametern in der Anweisung Werte zu.
4. Führen Sie die Anweisung aus.
5. Wiederholen Sie, falls erforderlich, Schritte 3 und 4.
6. Löschen Sie die Anweisung, wenn Sie fertig sind. In JDBC löscht der Java-Mechanismus der Speicherbereinigung die Anweisung.

Vorbereitete Anweisungen in ADO.NET verwenden

In der Regel führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung in ADO.NET zu verwenden:

1. Erstellen Sie ein `SACommand`-Objekt, das die Anweisung enthält:

```
SACommand cmd = new SACommand(  
    "SELECT * FROM Employees WHERE Surname = ?", conn );
```

2. Deklarieren Sie Datentypen für beliebige Parameter in der Anweisung.

Verwenden Sie die Methode `SACommand.CreateParameter`.

```
SAParameter param = cmd.CreateParameter();  
param.SADbType = SADbType.Char;  
param.Direction = ParameterDirection.Input;  
param.Value = "Smith";  
cmd.Parameters.Add(param);
```

3. Bereiten Sie die Anweisung mit der `Prepare`-Methode vor.
4. Führen Sie die Anweisung aus:

```
SADataReader reader = cmd.ExecuteReader();
```

Ein Beispiel für das Vorbereiten von Anweisungen mit ADO.NET finden Sie im Quellcode unter `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\SimpleWin32`.

Vorbereitete Anweisungen in ODBC verwenden

In der Regel führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung in ODBC zu verwenden:

1. Bereiten Sie die Anweisung mit `SQLPrepare` vor.
2. Binden Sie die Anweisungsparameter mit `SQLBindParameter`.
3. Führen Sie die Anweisung mit `SQLExecute` aus.
4. Löschen Sie die Anweisung mit `SQLFreeStmt`.

Ein Beispiel für die Vorbereitung von Anweisungen mithilfe von ODBC finden Sie im Quellcode in `%SQLANYSAMPI6%\SQLAnywhere\ODBCPrepare`.

Weitere Hinweise zu vorbereiteten ODBC-Anweisungen finden Sie in der ODBC SDK-Dokumentation und unter „[Vorbereitete Anweisungen ausführen](#)“ auf Seite 380.

Vorbereitete Anweisungen in JDBC verwenden

In der Regel führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung in JDBC zu verwenden:

1. Bereiten Sie die Anweisung mit der Methode `prepareStatement` des Verbindungsobjekts vor. Daraus wird ein `PreparedStatement`-Objekt erzeugt.

2. Setzen Sie die Anweisungsparameter mit den entsprechenden **setType**-Methoden des PreparedStatement-Objekts. Hier ist *Type* der Datentyp, der zugewiesen wird.
3. Führen Sie die Anweisung mit der geeigneten Methode für das PreparedStatement-Objekt aus. Für Einfügungen, Aktualisierungen und Löschungen ist dies die Methode `executeUpdate`.

Ein Beispiel für die Vorbereitung von Anweisungen mithilfe von JDBC finden Sie in der Quellcode-Datei `%SQLANYSAMPI6%\SQLAnywhere\JDBC\JDBCExample.java`.

Weitere Hinweise zur Verwendung von vorbereiteten Anweisungen in JDBC finden Sie unter [„Vorbereitete Anweisungen für effizienteren Zugriff verwenden“ auf Seite 440](#).

Vorbereitete Anweisungen in Embedded SQL verwenden

In der Regel führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung in Embedded SQL zu verwenden:

1. Bereiten Sie die Anweisung mit der Anweisung `EXEC SQL PREPARE` vor.
2. Weisen Sie den Parametern in der Anweisung Werte zu.
3. Führen Sie die Anweisung mit der Anweisung `EXEC SQL EXECUTE` aus.
4. Geben Sie mithilfe der Anweisung `EXEC SQL DROP` die mit der Anweisung verbundenen Ressourcen frei.

Weitere Hinweise über vorbereitete Anweisungen in Embedded SQL finden Sie unter [„PREPARE-Anweisung \[ESQL\]“ \[SQL Anywhere Server - SQL-Referenzhandbuch\]](#).

Vorbereitete Anweisungen in Open Client verwenden

In der Regel führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung in Open Client zu verwenden:

1. Bereiten Sie die Anweisung mit der Funktion `ct_dynamic` mit `CS_PREPARE` als Typ-Parameter vor.
2. Legen Sie die Anweisungsparameter mit `ct_param` fest.
3. Führen Sie die Anweisung mit `ct_dynamic` mit `CS_EXECUTE` als Typ-Parameter aus.
4. Geben Sie die mit der Anweisung verbundenen Ressourcen frei, indem Sie `ct_dynamic` mit `CS_DEALLOC` als Typ-Parameter verwenden.

Weitere Hinweise zur Verwendung von vorbereiteten Anweisungen in Open Client finden Sie unter [„SQL in Open Client-Anwendungen“ auf Seite 751](#).

Cursorverwendung

Wenn Sie eine Abfrage in einer Anwendung ausführen, besteht die Ergebnismenge aus mehreren Zeilen. Im Allgemeinen wissen Sie nicht, wie viele Zeilen die Anwendung empfangen wird, bevor die Abfrage

ausgeführt worden ist. Mit einem Cursor können Sie Ergebnismengen von Abfragen in Anwendungen verarbeiten.

Wie Sie Cursor einsetzen, und welche Cursorarten verfügbar sind, hängt von der Programmierschnittstelle ab, die Sie benutzen. Eine Liste der Cursortypen, die in den einzelnen Schnittstellen verfügbar sind, finden Sie unter „[Cursorverfügbarkeit](#)“ auf Seite 15.

SQL Anywhere verfügt über mehrere Systemprozeduren, die ermitteln, welche Cursor für eine Verbindung verwendet werden, und was sie enthalten:

- „sa_list_cursors-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_describe_cursor-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_copy_cursor_to_temp_table-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Sie können mit einem Cursor folgende Aufgaben innerhalb einer Programmierschnittstelle durchführen:

- Die Ergebnisse einer Abfrage mittels Schleifen bearbeiten.
- Einfügungen, Aktualisierungen und Löschungen auf den darunterliegenden Daten an einem beliebigen Punkt in einer Ergebnismenge durchführen.

Zusätzlich ermöglichen Ihnen einige Programmierschnittstellen die Verwendung von Sonderfunktionen, um die Art zu optimieren, wie Ergebnismengen an Ihre Anwendung zurückgegeben werden, was beträchtliche Performance-Vorteile für Ihre Anwendung bietet.

Weitere Hinweise zu den verfügbaren Cursorarten in den einzelnen Programmierschnittstellen finden Sie unter „[Cursorverfügbarkeit](#)“ auf Seite 15.

Cursor

Ein **Cursor** ist ein Name, der einer Ergebnismenge zugeordnet ist. Die Ergebnismenge erhalten Sie durch eine SELECT-Anweisung oder den Aufruf einer gespeicherten Prozedur.

Ein Cursor ist ein Handle auf der Ergebnismenge. Zu jedem Zeitpunkt hat der Cursor eine genau definierte Position innerhalb der Ergebnismenge. Mit einem Cursor können Sie die Daten zeilenweise untersuchen und gegebenenfalls bearbeiten. In SQL Anywhere unterstützen Cursor Vorwärts- und Rückwärtsbewegungen durch die Abfrageergebnisse.

Cursorpositionen

Cursor können an den folgenden Stellen positioniert werden:

- Vor der ersten Zeile der Ergebnismenge.
- Auf einer Zeile in der Ergebnismenge.
- Nach der letzten Zeile der Ergebnismenge.

Absolute Zeile ab Start		Absolute Zeile ab Ende
0	Vor der ersten Zeile	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		-3
$n - 1$		-2
n		-1
$n + 1$	Nach der letzten Zeile	0

Die Cursorposition und die Ergebnismenge werden im Datenbankserver aufrechterhalten. Zeilen werden vom Client zur Anzeige oder Verarbeitung einzeln oder in Gruppen **abgerufen**. Dem Client muss nicht die gesamte Ergebnismenge übermittelt werden.

Vorteile der Cursorbenutzung

Obwohl serverseitige Cursor in Datenbank Anwendungen nicht erforderlich sind, bieten sie eine Reihe von Vorteilen. Ein serverseitiger Cursor ist einem clientseitigen Cursor aus folgenden Gründen vorzuziehen:

- **Antwortzeit** Serverseitige Cursor erfordern nicht, dass die gesamte Ergebnismenge zusammengestellt werden muss, bevor die erste Zeile vom Client abgerufen wird. Ein clientseitiger Cursor erfordert, dass die gesamte Ergebnismenge abgerufen und an den Client übertragen wird, bevor die erste Zeile vom Client abgerufen werden kann.
- **Clientseitiger Speicher** Bei großen Ergebnismengen kann das Sammeln der gesamten Ergebnismenge auf der Clientseite zu einer starken Speicherauslastung führen.
- **Parallelitätskontrolle** Wenn Sie Ihre Daten aktualisieren und keine serverseitigen Cursor in Ihrer Anwendung verwenden, müssen Sie separate SQL-Anweisungen wie UPDATE, INSERT oder DELETE an den Datenbankserver senden, um die Änderungen zu übernehmen. Dadurch können Parallelitätsprobleme entstehen, wenn entsprechende Zeilen in der Datenbank geändert wurden, seit die Ergebnismenge an den Client übertragen wurde. Daher können Aktualisierungen durch andere Clients verloren gehen.

Serverseitige Cursor können als Zeiger auf die zugrunde liegenden Daten fungieren und ermöglichen es Ihnen, geeignete Parallelitätsbeschränkungen für durch den Client vorgenommene Änderungen mithilfe einer entsprechenden Isolationsstufe festzulegen.

Cursor-Grundsätze

Um einen Cursor in ADO.NET, ODBC, JDBC oder Open Client zu verwenden, halten Sie die folgenden allgemeinen Schritte ein:

1. Bereiten Sie eine Anweisung vor und führen Sie sie aus:

Führen Sie eine Anweisung mit der normalen Methode für die Schnittstelle aus. Sie können die Anweisung vorbereiten und dann ausführen, oder die Anweisung direkt ausführen.

Bei ADO.NET gibt nur die Methode `SACommand.ExecuteReader` einen Cursor zurück. Er liefert einen nur lesenden Vorwärtscursor.

2. Überprüfen Sie, ob die Anweisung eine Ergebnismenge zurückgibt.

Ein Cursor wird implizit geöffnet, wenn eine Anweisung ausgeführt wird, die eine Ergebnismenge erstellt. Beim Öffnen eines Cursors wird er vor die erste Zeile der Ergebnismenge gesetzt.

3. Rufen Sie die Ergebnisse ab.

Obwohl ein einfacher Abrufvorgang den Cursor in die nächste Zeile der Ergebnismenge bewegt, ermöglicht SQL Anywhere auch kompliziertere Bewegungen in der Ergebnismenge.

4. Schließen Sie den Cursor.

Wenn Sie die Arbeit mit dem Cursor abgeschlossen haben, schließen Sie ihn, damit die ihm zugewiesenen Ressourcen freigegeben werden.

5. Geben Sie die Anweisung frei.

Wenn Sie eine vorbereitete Anweisung verwendet haben, geben Sie sie frei, um den benutzten Speicher wieder verfügbar zu machen.

Der Ansatz für die Verwendung eines Cursors in Embedded SQL unterscheidet sich von demjenigen in anderen Schnittstellen. Befolgen Sie diese allgemeinen Schritte, um einen Cursor in Embedded SQL zu verwenden:

1. Bereiten Sie eine Anweisung vor.

Cursor verwenden üblicherweise einen Anweisungs-Handle statt einer Zeichenfolge. Sie müssen eine Anweisung vorbereiten, damit ein Handle verfügbar ist.

2. Deklarieren Sie den Cursor.

Jeder Cursor bezieht sich auf eine einzelne SELECT- oder CALL-Anweisung. Wenn Sie einen Cursor deklarieren, geben Sie den Namen des Cursors und die Anweisung an, auf die er sich bezieht.

3. Öffnen Sie den Cursor.

Bei einer CALL-Anweisung wird durch das Öffnen des Cursors die Prozedur bis zu dem Punkt ausgeführt, an dem die erste Zeile bezogen werden kann.

4. Rufen Sie die Ergebnisse ab.

Obwohl ein einfacher Abrufvorgang den Cursor in die nächste Zeile der Ergebnismenge bewegt, ermöglicht SQL Anywhere auch kompliziertere Bewegungen in der Ergebnismenge. Die verfügbaren Abrufvorgänge hängen davon ab, wie Sie den Cursor deklarieren.

5. Schließen Sie den Cursor.

Wenn Sie die Arbeit mit dem Cursor abgeschlossen haben, schließen Sie ihn. Das setzt alle Ressourcen frei, die dem Cursor zugeordnet sind.

6. Löschen Sie die Anweisung.

Um den Speicher freizugeben, der der Anweisung zugeordnet war, müssen Sie die Anweisung löschen.

Siehe auch

- „Vorbereitete Anweisungen“ auf Seite 2
- „DECLARE CURSOR-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „OPEN-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „FETCH-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Abruf von Daten mit Embedded SQL“ auf Seite 514
- „CLOSE-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „DROP STATEMENT-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Cursor positionieren

Wenn ein Cursor geöffnet wird, wird er vor der ersten Zeile positioniert. Sie können den Cursor an eine absolute Position in Verhältnis zum Anfang oder zum Ende der Abfrageergebnisse positionieren oder ihn relativ zur aktuellen Cursorposition verschieben. Wie Sie im Einzelnen den Cursor verschieben und welche Vorgänge möglich sind, hängt von der Programmierschnittstelle ab.

Die Anzahl der Zeilenpositionen, die Sie mit einem Fetch-Vorgang abrufen können, wird durch die Größe einer Ganzzahl bestimmt. Mit einem Fetch-Vorgang können Sie Zeilen bis zu Nummer 2147483646 abrufen, wobei es sich um die größtmögliche Ganzzahl minus 1 handelt. Wenn Sie negative Werte verwenden (Zeilen in Bezug auf das Ende), können Sie Fetch-Vorgänge nach unten bis zum kleinsten negativen Wert, der in einer Ganzzahl möglich ist, plus 1 ausführen.

Sie können spezielle positionsbasierte Aktualisierungs- oder Löschvorgänge verwenden, um die Zeile an der aktuellen Cursorposition zu aktualisieren oder zu löschen. Wenn der Cursor vor der ersten Zeile oder nach der letzten Zeile positioniert ist, wird ein Fehler zurückgegeben, der darüber informiert, dass keine entsprechende Cursorzeile vorhanden ist.

Hinweis

Einfügungen und manche Aktualisierungsvorgänge mit asensitiven Cursors können Probleme mit der Cursorpositionierung verursachen. SQL Anywhere platziert eingefügte Zeilen an unvorhersehbaren Positionen innerhalb eines Cursors, falls die SELECT-Anweisung keine ORDER BY-Klausel hat. In einigen Fällen erscheint die eingefügte Zeile überhaupt erst, wenn der Cursor geschlossen und wieder geöffnet wurde. Bei SQL Anywhere tritt dies auf, wenn eine Arbeitstabelle erstellt werden musste, um den Cursor zu öffnen.

Die UPDATE-Anweisung kann bewirken, dass sich eine Zeile im Cursor verschiebt. Das passiert, wenn der Cursor eine ORDER BY-Klausel hat, die einen vorhandenen Index benutzt (es wird keine Arbeitstabelle erstellt). Mit der Verwendung eines statisch abrollenden Cursors werden diese Probleme vermieden, allerdings ist mehr Speicher und Verarbeitungsaufwand erforderlich.

Siehe auch

- „[Tipp: Arbeitstabellen in der Abfrageverarbeitung verwenden \("Alle Zeilen" als Optimierungsziel verwenden\)](#)“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*]

Cursorverhalten beim Öffnen von Cursors

Sie können die folgenden Aspekte des Cursorverhaltens konfigurieren, wenn Sie einen Cursor öffnen:

- **Isolationsstufe** Sie können die Isolationsstufen der Vorgänge für einen Cursor explizit so setzen, dass sie von der aktuellen Isolationsstufe der Transaktion verschieden sind. Dazu stellen Sie die `isolation_level`-Option ein.
- **Offen halten** Standardmäßig werden Cursor in Embedded SQL am Ende einer Transaktion geschlossen. Wenn Sie einen Cursor mit der Option `WITH HOLD` öffnen, können Sie ihn offen halten, bis die Verbindung beendet wird oder bis Sie ihn explizit schließen. ADO.NET, ODBC, JDBC und Open Client lassen Cursor beim Ende von Transaktionen standardmäßig geöffnet.

Siehe auch

- „[isolation_level-Option](#)“ [*SQL Anywhere Server - Datenbankadministration*]

Zeilenabruf durch einen Cursor

Die einfachste Möglichkeit, eine Ergebnismenge aus einer Abfrage mit einem Cursor zu verarbeiten, ist eine Schleife zum Durchlaufen aller Zeilen in der Ergebnismenge, bis keine Zeilen mehr vorhanden sind. Sie können diese Aufgabe mithilfe der folgenden Schritte ausführen:

1. Deklarieren und öffnen Sie den Cursor (Embedded SQL), oder führen Sie eine Anweisung aus, die eine Ergebnismenge (ODBC, JDBC, Open Client) oder ein `SADaReader`-Objekt (ADO.NET) zurückgibt.
2. Setzen Sie die Fetch-Vorgänge für die nächste Zeile fort, bis der Fehler `Zeile nicht gefunden` (Row Not Found) erscheint.

3. Schließen Sie den Cursor.

Wie die nächste Zeile abgerufen wird, hängt von der verwendeten Schnittstelle ab. Beispiel:

- **ADO.NET** Verwenden Sie die `SADataReader.Read`-Methode. Siehe [SADataReader.Read-Methode \[SQL Anywhere .NET\] auf Seite 246](#).
- **ODBC** `SQLFetch`, `SQLExtendedFetch` oder `SQLFetchScroll` verschieben den Cursor in die nächste Zeile und geben die Daten zurück.

Weitere Hinweise zur Verwendung eines Cursors in ODBC finden Sie unter [„Ergebnismengen in ODBC-Anwendungen“ auf Seite 388](#).

- **JDBC** Die `next`-Methode des `ResultSet`-Objekts bewegt den Cursor weiter und gibt die Daten zurück.

Weitere Hinweise zur Verwendung des `ResultSet`-Objekts in JDBC finden Sie unter [„Ergebnismengen aus Java zurückgeben“ auf Seite 443](#).

- **Embedded SQL** Die `FETCH`-Anweisung führt denselben Vorgang aus.

Weitere Hinweise zur Cursor-Verwendung in Embedded SQL finden Sie unter [„Cursor in Embedded SQL“ auf Seite 516](#).

- **Open Client** Die `ct_fetch`-Funktion verschiebt den Cursor in die nächste Zeile und gibt die Daten zurück.

Weitere Hinweise zur Verwendung eines Cursors in Open Client-Anwendungen finden Sie unter [„Cursor-Verwaltung in Open Client“ auf Seite 752](#).

Mehrzeiliger Abruf

Mehrzeilen-Fetch-Vorgänge sind nicht dasselbe wie Prefetch-Zeilen. Mehrzeilen-Abrufe werden von der Anwendung ausgeführt, während ein Vorab-Abrufen für die Anwendung nicht erkennbar ist und eine ähnliche Performance-Steigerung bietet. Das gleichzeitige Abrufen von mehreren Zeilen kann die Performance steigern.

Mehrzeilen-Fetch-Vorgänge

Einige Schnittstellen bieten Methoden zum gleichzeitigen Abrufen mehrerer Zeilen in die nächsten Felder eines Arrays. Im Allgemeinen gilt: Je weniger getrennte Fetch-Vorgänge Sie ausführen, desto weniger einzelne Anforderungen muss der Server bewältigen, und desto besser wird die Performance. Eine modifizierte `FETCH`-Anweisung, die mehrere Zeilen abrufen, wird auch ein **weiter Abruf** genannt. Ein Cursor, der Mehrzeilen-Abrufe ausführt, wird manchmal auch als **Block-Cursor** oder **Fetter Cursor** (engl.: fat cursor) bezeichnet.

Mehrzeilen-Fetch-Vorgänge verwenden

- In ODBC können Sie die Anzahl der Zeilen einstellen, die bei jedem Aufruf von `SQLFetchScroll` oder `SQLExtendedFetch` zurückgegeben werden, indem Sie das Attribut `SQL_ATTR_ROW_ARRAY_SIZE` oder `SQL_ROWSET_SIZE` setzen.

- In Embedded SQL verwendet die FETCH-Anweisung eine ARRAY-Klausel, um die Anzahl der Zeilen zu steuern, die durch einen Fetch-Vorgang gleichzeitig abgerufen werden.
- Open Client und JDBC unterstützen mehrzeilige Fetch-Vorgänge nicht. Sie verwenden Prefetch-Vorgänge.

Scrollfähige Cursor

ODBC und Embedded SQL bieten Methoden für die Verwendung von scrollfähigen und dynamischen scrollfähigen Cursortypen. Diese Methoden ermöglichen das gleichzeitige Vorwärts- oder Rückwärtsbewegen über mehrere Zeilen in einer Ergebnismenge.

Die Schnittstellen für JDBC und Open Client unterstützen keine scrollfähigen Cursor.

Prefetch-Vorgänge sind auf Vorgänge mit einem scrollfähigen Cursor nicht anwendbar. Zum Beispiel werden beim Abrufen einer Zeile in entgegengesetzter Richtung nicht mehrere vorherige Zeilen abgerufen.

Für die Zeilenänderung verwendete Cursor

Cursor können mehr als nur Ergebnisse aus einer Abfrage lesen. Sie können auch Daten in der Datenbank verändern, während Sie einen Cursor verarbeiten. Diese Vorgänge werden im Allgemeinen als **positionsbasierte** oder positionierte Einfügings-, Aktualisierungs- und Löschvorgänge oder PUT-Vorgänge (wenn es sich um ein INSERT handelt) bezeichnet.

Nicht alle Abfrage-Ergebnismengen ermöglichen positionsbasiertes Aktualisieren oder Löschen. Wenn Sie eine Abfrage in einer nicht aktualisierbaren Ansicht ausführen, werden in den Basistabellen keine Änderungen durchgeführt. Auch wenn die Abfrage einen Join enthält, müssen Sie angeben, aus welcher Tabelle Sie löschen wollen oder welche Spalten Sie aktualisieren wollen, wenn Sie die Vorgänge ausführen.

Einfügungen durch einen Cursor können nur durchgeführt werden, wenn nicht eingefügte Spalten in der Tabelle NULL zulassen oder Standardwerte haben.

Wenn mehrere Zeilen in einen wertsensitiven (Keyset-gesteuerten) Cursor eingefügt werden, erscheinen sie am Ende der Cursor-Ergebnismenge. Die Zeilen werden am Ende angezeigt, auch wenn sie nicht mit der WHERE-Klausel der Abfrage übereinstimmen oder eine ORDER BY-Klausel sie normalerweise an eine andere Stelle in der Ergebnismenge gesetzt hätte. Dieses Verhalten ist von der Programmierschnittstelle unabhängig. Beispielsweise gilt dies, wenn Sie die PUT-Anweisung aus Embedded SQL oder die ODBC-Funktion SQLBulkOperations verwenden. Der Wert einer autokrementierenden Spalte für die zuletzt eingefügte Zeile kann gefunden werden, indem Sie die letzte Zeile im Cursor auswählen. Beispiel: In Embedded SQL kann der Wert mit `FETCH ABSOLUTE -1 Cursorname` ermittelt werden. Als Folge dieses Verhaltens kann die erste Einfügung mit mehreren Zeilen für einen wertsensitiven Cursor teuer sein.

ODBC, JCBC, Embedded SQL und Open Client ermöglichen die Datenmanipulation mit einem Cursor, ADO.NET hingegen nicht. Mit dem Open Client können Sie Zeilen löschen und aktualisieren, aber Zeilen nur in einer Ein-Tabellenabfrage einfügen.

Aus welcher Tabelle werden Zeilen gelöscht?

Wenn Sie ein positionsbasiertes Löschen durch einen Cursor versuchen, wird die Tabelle, aus der Zeilen gelöscht werden, wie folgt festgelegt:

1. Wenn keine FROM-Klausel in der DELETE-Anweisung eingeschlossen ist, muss der Cursor nur für eine Tabelle gesetzt sein.
2. Wenn der Cursor für eine Join-Abfrage (einschließlich zum Benutzen einer Ansicht mit enthaltenem Join) gesetzt ist, muss die FROM-Klausel verwendet werden. Nur die aktuelle Zeile der angegebenen Tabelle wird gelöscht. Die anderen Tabellen des Joins sind nicht betroffen.
3. Wenn eine FROM-Klausel enthalten ist und kein Tabelleneigentümer angegeben wurde, ist der Tabellenangabewert der erste, der zu den Korrelationsnamen passt.
4. Wenn ein Korrelationsname existiert, ist der Tabellenangabename mit dem Korrelationsnamen identifiziert.
5. Wenn ein Korrelationsname nicht vorhanden ist, muss der Tabellenangabewert eindeutig als Tabellenname im Cursor identifizierbar sein.
6. Wenn eine FROM-Klausel enthalten ist und ein Tabelleneigentümer angegeben wurde, muss der Tabellenangabewert als Tabellenname im Cursor eindeutig identifizierbar sein.
7. Die positionsbasierte DELETE-Anweisung kann für einen Cursor verwendet werden, der für eine Ansicht geöffnet ist, solange die Ansicht aktualisierbar ist.

Siehe auch

- „FROM-Klausel“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Aktualisierbare Anweisungen

Dieser Abschnitt beschreibt, wie Klauseln in der SELECT-Anweisung aktualisierbare Anweisungen und Cursor beeinflussen.

Aktualisierbarkeit von schreibgeschützten Anweisungen

Die Angabe von FOR READ ONLY in der Cursordeklaration oder die Aufnahme einer FOR READ ONLY-Klausel in die Anweisung führt dazu, dass die Anweisung schreibgeschützt ist. D.h., dass eine FOR READ ONLY-Klausel oder die entsprechende schreibgeschützte Cursordeklaration bei Verwendung einer Client-API jede andere Aktualisierbarkeitsangabe außer Kraft setzt.

Wenn der äußerste Block einer SELECT-Anweisung eine ORDER BY-Klausel enthält und in der Anweisung kein FOR UPDATE angegeben wird, ist der Cursor schreibgeschützt. Wenn die SQL SELECT-Anweisung FOR XML angibt, ist der Cursor schreibgeschützt. Andernfalls ist der Cursor aktualisierbar.

Aktualisierbare Anweisungen und Parallelitätssteuerung

Bei aktualisierbaren Anweisungen stellt SQL Anywhere optimistische und pessimistische Parallelitätssteuerungsmechanismen für Cursor zur Verfügung, um zu gewährleisten, dass eine

Ergebnismenge während Abrollvorgängen konsistent bleibt. Diese Mechanismen stellen eine Alternative zur Verwendung eines INSENSITIVE-Cursors oder einer Snapshot-Isolation dar, auch wenn sie eine andere Semantik verwenden und andere Nachteile haben.

Die Angabe FOR UPDATE kann sich darauf auswirken, ob ein Cursor aktualisierbar ist. In SQL Anywhere hat die FOR UPDATE-Syntax allerdings keine weiteren Auswirkungen auf die Parallelitätssteuerung. Wenn FOR UPDATE mit zusätzlichen Parametern angegeben ist, ändert SQL Anywhere die Verarbeitung der Anweisung, um eine von zwei Parallelitätssteuerungsoptionen wie folgt einzubeziehen:

- **Pessimistisch** Für alle in die Ergebnismenge des Cursors abgerufene Zeilen erwirbt der Datenbankserver Absichtszeilensperren, um zu verhindern, dass die Zeilen von einer anderen Transaktion aktualisiert werden.
- **Optimistisch** Der vom Datenbankserver verwendete Cursortyp wird zu einem durch Keyset gesteuerten Cursor (insensitive Mitgliedschaft, wertsensitiv), damit die Anwendung benachrichtigt werden kann, wenn eine Zeile im Ergebnis von dieser oder einer anderen Transaktion geändert oder gelöscht wurde.

Pessimistische oder optimistische Parallelität wird auf der Cursorebene angegeben, und zwar entweder mittels DECLARE CURSOR- bzw. FOR-Anweisungen, oder mittels der Parallelitätseinstellungs-API bei einer spezifischen Programmierschnittstelle. Wenn eine Anweisung aktualisierbar ist und der Cursor keinen Parallelitätssteuerungsmechanismus angibt, wird die Angabe der Anweisung verwendet. Die Syntax lautet wie folgt:

- **FOR UPDATE BY LOCK** Der Datenbankserver erwirbt Absichtszeilensperren für abgerufene Zeilen der Ergebnismenge. Dies sind langfristige Sperren, die bis zum COMMIT oder ROLLBACK der Transaktion gehalten werden..
- **FOR UPDATE BY { VALUES | TIMESTAMP }** Der Datenbankserver verwendet einen keyset-driven Cursor, damit die Anwendung benachrichtigt werden kann, wenn Zeilen geändert oder gelöscht werden, während die Ergebnismenge durchlaufen wird.

Aktualisierbare Anweisungen beschränken

FOR UPDATE (*Spaltenliste*) erzwingt die Beschränkung, dass nur benannte Ergebnismengenattribute in einer nachfolgenden UPDATE WHERE CURRENT OF-Anweisung geändert werden können.

Siehe auch

- „DECLARE-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „FOR-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Abgebrochene Cursorvorgänge

Sie können eine Anforderung durch eine Schnittstellenfunktion abbrechen. Wenn Sie eine Anforderung abbrechen, die einen Cursorvorgang durchführt, ist die Position des Cursors unbestimmt. Nach dem Abbrechen der Anforderung müssen Sie die absolute Position des Cursors ermitteln oder ihn schließen.

Cursortypen

Dieser Abschnitt beschreibt Zuordnungen zwischen SQL Anywhere-Cursor und den Optionen, die Ihnen die Programmierschnittstellen bieten, die von SQL Anywhere unterstützt werden.

Siehe auch

- [„SQL Anywhere-Cursor“ auf Seite 16](#)

Cursorverfügbarkeit

Nicht alle Schnittstellen bieten Unterstützung für alle Cursortypen.

- ADO.NET liefert einen Cursor nur zum Weitergeben und nur zum Lesen.
- ADO/OLE DB und ODBC unterstützen alle Cursortypen.
- Embedded SQL unterstützt alle Cursortypen.
- Für JDBC:
 - Der SQL Anywhere JDBC-Treiber unterstützt die JDBC 4.0-Spezifikation und gestattet die Deklaration von insensitiven und sensitiven Cursoren sowie von asensitiven Vorwärtscursoren.
 - jConnect unterstützt die Deklaration von insensitiven und sensitiven Cursoren sowie von asensitiven Cursoren für die Weitergabe auf dieselbe Weise wie der SQL Anywhere JDBC-Treiber. Die zu Grunde liegende Implementierung von jConnect unterstützt jedoch nur asensitive Cursor-Semantiken.
- Sybase Open Client unterstützt lediglich asensitive Cursor. Außerdem kommt es zu erheblichen Performance-Einbußen, wenn aktualisierbare, nicht eindeutige Cursor benutzt werden.

Siehe auch

- [„Ergebnismengen in ODBC-Anwendungen“ auf Seite 388](#)
- [„Anforderungen für SQL Anywhere-Cursor“ auf Seite 33](#)

Cursoreigenschaften

Sie fordern einen Cursortyp entweder explizit oder implizit von der Programmierschnittstelle an. Unterschiedliche Schnittstellenbibliotheken bieten eine unterschiedliche Auswahl von Cursortypen an. JDBC und ODBC schreiben zum Beispiel unterschiedliche Cursortypen vor.

Jeder Cursortyp wird durch mehrere Merkmale definiert:

- **Eindeutigkeit** Wenn ein Cursor als eindeutig deklariert wird, zwingt dies die Abfrage, alle Spalten zurückzugeben, die für die eindeutige Identifizierung der einzelnen Zeilen erforderlich sind. Oft bedeutet dies, dass alle Spalten im Primärschlüssel zurückgegeben werden. Alle erforderlichen, aber nicht angegebenen Spalten werden der Ergebnismenge hinzugefügt. Der Standardcursortyp ist "Nicht eindeutig".

- **Aktualisierbarkeit** Ein als schreibgeschützt deklarierter Cursor kann nicht für ein positionsbasiertes Update oder eine Löschung verwendet werden. Der Standardcursortyp ist "Aktualisierbar".
- **Abrollfähigkeit** Sie können Cursor so deklarieren, dass Sie sich unterschiedlich verhalten, wenn Sie sich durch die Ergebnismenge bewegen. Manche Cursor können nur die aktuelle oder die nächste Zeile abrufen. Andere können sich vorwärts und rückwärts in der Ergebnismenge bewegen.
- **Sensitivität** Änderungen an der Datenbank können durch einen Cursor sichtbar sein, müssen es aber nicht.

Diese Eigenschaften haben möglicherweise signifikante Auswirkungen auf die Performance und Speichernutzung des Datenbankservers.

SQL Anywhere stellt Ihnen Cursor mit unterschiedlichen Zusammensetzungen dieser Eigenschaften zur Verfügung. Wenn Sie einen Cursor eines bestimmten Typs anfordern, versucht SQL Anywhere, diesen Merkmalen zu entsprechen.

Unter Umständen ist es nicht möglich, allen Eigenschaften zu entsprechen. Insensitive Cursor in SQL Anywhere müssen z.B. schreibgeschützt sein. Wenn Ihre Anwendung einen aktualisierbaren insensitive Cursor anfordert, wird stattdessen ein anderer Cursortyp (wertsensitiv) geliefert.

Lesezeichen und Cursor

ODBC bietet **Lesezeichen** bzw. Werte, die verwendet werden, um Zeilen in einem Cursor zu identifizieren. SQL Anywhere unterstützt Lesezeichen für wertsensitive und insensitive Cursor. Die ODBC-Cursortypen `SQL_CURSOR_STATIC` und `SQL_CURSOR_KEYSET_DRIVEN` unterstützen beispielsweise Lesezeichen, während die Cursortypen `SQL_CURSOR_DYNAMIC` und `SQL_CURSOR_FORWARD_ONLY` keine Lesezeichen unterstützen.

Block-Cursor

ODBC stellt einen Cursortyp namens Block-Cursor zur Verfügung. Wenn Sie einen solchen BLOCK-Cursor verwenden, können Sie mit `SQLFetchScroll` oder `SQLExtendedFetch` einen Zeilenblock anstelle einer einzelnen Zeile abrufen. Block-Cursor verhalten sich genauso wie Embedded SQL ARRAY-Abrufe.

SQL Anywhere-Cursor

Sobald ein Cursor geöffnet ist, hat er eine zugeordnete Ergebnismenge. Der Cursor bleibt eine Zeit lang geöffnet. Während dieser Zeit kann die dem Cursor zugeordnete Ergebnismenge geändert werden, entweder durch den Cursor selbst oder, abhängig von den Anforderungen der Isolationsstufe, durch andere Transaktionen. Manche Cursor ermöglichen es, Änderungen an den darunter liegenden Daten sichtbar zu machen, während bei Anderen diese Änderungen nicht auszumachen sind. Eine Sensitivität gegenüber Änderungen an den Basisdaten bewirkt unterschiedliches Cursor-Verhalten, was als **Cursor-Sensitivität** bezeichnet wird.

SQL Anywhere stellt Cursor mit einer Vielzahl von Sensitivitätseigenschaften zur Verfügung. Dieser Abschnitt beschreibt, was Sensitivität ist sowie die Sensitivitätseigenschaften von Cursors.

Dabei wird vorausgesetzt, dass Sie mit dem Abschnitt „Cursor“ auf Seite 6 vertraut sind.

Änderungen bei Mitgliedschaft, Reihenfolge und Werten

Änderungen an den darunter liegenden Daten können sich folgendermaßen auf die Ergebnismenge eines Cursors auswirken:

- **Mitgliedschaft** Die Menge der Zeilen in der Ergebnismenge, die durch ihre Primärschlüsselwerte gekennzeichnet sind.
- **Reihenfolge** Die Reihenfolge der Zeilen in der Ergebnismenge.
- **Wert** Die Werte der Zeilen in der Ergebnismenge.

Nehmen Sie zum Beispiel die folgende einfache Tabelle mit Mitarbeiterdaten (EmployeeID ist die Primärschlüsselspalte):

EmployeeID	Surname
1	Whitney
2	Cobb
3	Chin

Ein Cursor auf der folgenden Abfrage gibt alle Ergebnisse aus der Tabelle in der Reihenfolge des Primärschlüssels zurück.

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

Die Mitgliedschaft der Ergebnismenge könnte durch das Hinzufügen einer neuen Zeile oder das Löschen einer Zeile geändert werden. Die Werte könnten durch eine Namensänderung in der Tabelle geändert werden. Die Reihenfolge könnte geändert werden, indem der Primärschlüssel eines Mitarbeiters geändert wird.

Sichtbare und unsichtbare Änderungen

Abhängig von den Anforderungen der Isolationsstufe können Mitgliedschaft, Reihenfolge und Werte der Ergebnismenge eines Cursors geändert werden, nachdem der Cursor geöffnet wurde. Es hängt vom Typ des verwendeten Cursors ab, ob sich die Ergebnismenge, wie sie von der Anwendung gesehen wird, ändert, um diese Änderungen darzustellen.

Änderungen an den darunter liegenden Daten können durch den Cursor **sichtbar** oder **unsichtbar** sein. Eine sichtbare Änderung ist eine, die sich in der Ergebnismenge des Cursors widerspiegelt. Änderungen an den darunter liegenden Daten, die nicht in der Ergebnismenge, wie sie vom Cursor gesehen wird, widerspiegelt werden, sind unsichtbar.

Cursor-Sensitivität

SQL Anywhere-Cursor werden anhand ihrer Sensitivität gegenüber Änderungen an den Basisdaten eingeteilt. Die Cursor-Sensitivität wird also durch die sichtbaren Änderungen definiert.

- **Insensitive Cursor** Die Ergebnismenge ist unveränderlich, wenn der Cursor geöffnet ist. Änderungen an den darunter liegenden Daten sind nicht sichtbar.
- **Sensitive Cursor** Die Ergebnismenge kann sich ändern, nachdem der Cursor geöffnet wurde. Alle Änderungen an den darunter liegenden Daten sind sichtbar.
- **Asensitive Cursor** Änderungen können in der Mitgliedschaft, der Reihenfolge oder den Werten der Ergebnismenge, wie sie durch den Cursor gesehen wird, wiedergespiegelt werden, müssen es aber nicht. Siehe „[Asensitive Cursor](#)“ auf Seite 25.
- **Wertsensitive Cursor** Änderungen in der Reihenfolge oder den Werten der darunter liegenden Daten sind sichtbar. Die Mitgliedschaft der Ergebnismenge ist unveränderlich, wenn der Cursor geöffnet ist. Siehe „[Wertsensitive Cursor](#)“ auf Seite 26.

Die unterschiedlichen Anforderungen an Cursor bewirken unterschiedliche Beschränkungen bei der Ausführung, was sich wiederum auf die Performance auswirkt.

Siehe auch

- „[Insensitive Cursor](#)“ auf Seite 22
- „[Sensitive Cursor](#)“ auf Seite 23
- „[Cursor-Sensitivität und Performance](#)“ auf Seite 28

Beispiel für Cursor-Sensitivität: Eine gelöschte Zeile

Dieses Beispiel verwendet eine einfache Abfrage, um zu illustrieren, wie verschiedene Cursor auf eine Zeile in der Ergebnismenge reagieren, die gelöscht wird.

Es gibt die folgende Abfolge von Ereignissen:

1. Eine Anwendung öffnet in der Beispieldatenbank in der folgenden Abfrage einen Cursor.

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. Die Anwendung ruft die erste Zeile durch den Cursor ab (102).
3. Die Anwendung ruft die nächste Zeile durch den Cursor ab (105).
4. Eine weitere Anwendung löscht Mitarbeiter 102 (Whitney) und schreibt die Änderung fest.

In dieser Situation hängen die Ergebnisse der Cursor-Aktionen von der Cursor-Sensitivität ab.

- **Insensitive Cursor** Das DELETE wird weder in der Mitgliedschaft noch in den Werten der Ergebnisse, wie sie durch den Cursor gesehen werden, widerspiegelt:

Maßnahme	Ergebnis
Vorherige Zeile abrufen	Gibt die ursprüngliche Kopie der Zeile zurück (102).
Erste Zeile abrufen (absoluter Abruf)	Gibt die ursprüngliche Kopie der Zeile zurück (102).
Zweite Zeile abrufen (absoluter Abruf)	Gibt die ungeänderte Zeile zurück (105).

- **Sensitive Cursor** Die Mitgliedschaft der Ergebnismenge hat sich insofern geändert, dass jetzt Zeile 105 die erste Zeile in der Ergebnismenge ist:

Maßnahme	Ergebnis
Vorherige Zeile abrufen	Gibt Zeile nicht gefunden zurück. Es gibt keine vorherige Zeile.
Erste Zeile abrufen (absoluter Abruf)	Gibt Zeile 105 zurück.
Zweite Zeile abrufen (absoluter Abruf)	Gibt Zeile 160 zurück.

- **Wertsensitive Cursor** Die Mitgliedschaft der Ergebnismenge ist unveränderlich, und daher ist Zeile 105 weiterhin die zweite Zeile in der Ergebnismenge. Das DELETE wird in den Werten des Cursors widerspiegelt und erzeugt ein tatsächliches Loch in der Ergebnismenge.

Maßnahme	Ergebnis
Vorherige Zeile abrufen	Gibt keine aktuelle Cursorzeile zurück. Wo vorher die erste Zeile war, steht jetzt im Cursor eine Lücke.
Erste Zeile abrufen (absoluter Abruf)	Gibt keine aktuelle Cursorzeile zurück. Wo vorher die erste Zeile war, steht jetzt im Cursor eine Lücke.

Maßnahme	Ergebnis
Zweite Zeile abrufen (absoluter Abruf)	Gibt Zeile 105 zurück.

- **Asensitive Cursor** In Bezug auf Änderungen sind Mitgliedschaft und Werte der Ergebnismenge unbestimmt. Die Antwort auf einen Abruf der vorherigen Zeile, der ersten Zeile oder der zweiten Zeile hängt von der entsprechenden Optimierungsmethode für die Abfrage ab, nämlich ob die Methode die Erstellung einer Arbeitstabelle erfordert und ob die abgerufene Zeile vom Client vorab abgerufen wurde.

Der Vorteil von asensitiven Cursor liegt darin, dass für viele Anwendungen die Sensitivität unwichtig ist. Besonders wenn Sie einen nur lesenden Vorwärtscursor verwenden, sind keine der darunter liegenden Änderungen sichtbar. Auch wenn Sie auf einer hohen Isolationsstufe ausführen, sind darunter liegende Änderungen nicht zulässig.

Beispiel für Cursor-Sensitivität: Eine aktualisierte Zeile

Dieses Beispiel verwendet eine einfache Abfrage, um zu illustrieren, wie verschiedene Cursortypen auf eine Zeile in der Ergebnismenge reagieren, die aktualisiert wird und dadurch die Reihenfolge in der Ergebnismenge verändert.

Es gibt die folgende Abfolge von Ereignissen:

1. Eine Anwendung öffnet in der Beispieldatenbank in der folgenden Abfrage einen Cursor.

```
SELECT EmployeeID, Surname  
FROM Employees;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. Die Anwendung ruft die erste Zeile durch den Cursor ab (102).
3. Die Anwendung ruft die nächste Zeile durch den Cursor ab (105).
4. Eine weitere Transaktion aktualisiert die Mitarbeiter-ID des Mitarbeiters 102 (Whitney) auf 165 und schreibt die Änderung fest.

In dieser Situation hängen die Ergebnisse der Cursor-Aktionen von der Cursor-Sensitivität ab.

- **Insensitive Cursor** Das UPDATE wird weder in der Mitgliedschaft noch in den Werten der Ergebnisse, wie sie durch den Cursor gesehen werden, widerspiegelt:

Maßnahme	Ergebnis
Vorherige Zeile abrufen	Gibt die ursprüngliche Kopie der Zeile zurück (102).
Erste Zeile abrufen (absoluter Abruf)	Gibt die ursprüngliche Kopie der Zeile zurück (102).
Zweite Zeile abrufen (absoluter Abruf)	Gibt die ungeänderte Zeile zurück (105).

- **Sensitive Cursor** Die Mitgliedschaft der Ergebnismenge hat sich insofern geändert, dass jetzt Zeile 105 die erste Zeile in der Ergebnismenge ist:

Maßnahme	Ergebnis
Vorherige Zeile abrufen	Gibt SQLCODE 100 zurück. Die Mitgliedschaft der Ergebnismenge hat sich geändert und 105 ist jetzt die erste Zeile. Der Cursor wird auf die Position vor der ersten Zeile verschoben.
Erste Zeile abrufen (absoluter Abruf)	Gibt Zeile 105 zurück.
Zweite Zeile abrufen (absoluter Abruf)	Gibt Zeile 160 zurück.

Außerdem wird beim Abrufen durch einen sensitiven Cursor die Warnung `SQL_ROW_UPDATED_WARNING` ausgegeben, wenn die Zeile seit dem letzten Lesen geändert wurde. Die Warnung wird nur einmal ausgegeben. Aufeinander folgende `FETCH`-Vorgänge für dieselbe Zeile lösen keine weitere Warnung aus.

Ähnlich gibt auch eine positionsbasierte `UPDATE`- oder `DELETE`-Anweisung durch den Cursor auf einer Zeile, die seit dem letzten Abruf geändert wurde, die Fehlermeldung `SQL_ROW_UPDATED_SINCE_READ` aus. Eine Anwendung muss das Abrufen einer Zeile nochmals durchführen, damit `UPDATE` oder `DELETE` bei einem sensitiven Cursor funktionieren.

Eine Aktualisierung einer Spalte bewirkt die Warnung oder den Fehler, auch wenn die Spalte vom Cursor nicht referenziert wird. Beispiel: Ein Cursor auf einer Abfrage, der Surname zurückgibt, würde die Aktualisierung melden, auch wenn nur die Spalte Salary geändert worden wäre.

- **Wertsensitive Cursor** Die Mitgliedschaft der Ergebnismenge ist unveränderlich, und daher ist Zeile 105 weiterhin die zweite Zeile in der Ergebnismenge. Das UPDATE wird in den Werten des Cursors widerspiegelt und erzeugt ein tatsächliches "Loch" in der Ergebnismenge.

Maßnahme	Ergebnis
Vorherige Zeile abrufen	Gibt SQLCODE 100 zurück. Die Mitgliedschaft der Ergebnismenge hat sich geändert und 105 ist jetzt die erste Zeile. Der Cursor wird über dem Loch positioniert: er befindet sich vor Zeile 105.
Erste Zeile abrufen (absoluter Abruf)	Gibt SQLCODE -197 zurück. Die Mitgliedschaft der Ergebnismenge hat sich geändert und 105 ist jetzt die erste Zeile. Der Cursor wird über dem Loch positioniert: er befindet sich vor Zeile 105.
Zweite Zeile abrufen (absoluter Abruf)	Gibt Zeile 105 zurück.

- **Asensitive Cursor** In Bezug auf Änderungen sind Mitgliedschaft und Werte der Ergebnismenge unbestimmt. Die Antwort auf einen Abruf der vorherigen Zeile, der ersten Zeile oder der zweiten Zeile hängt von der entsprechenden Optimierungsmethode für die Abfrage ab, nämlich ob die Methode die Erstellung einer Arbeitstabelle erfordert und ob die abgerufene Zeile vom Client vorab abgerufen wurde.

Hinweis

Warnungs- und Fehlerbedingungen für Aktualisierungen kommen in Massenvorgängen nicht vor (Option -b für Datenbankserver).

Siehe auch

- „Zeile nicht gefunden“ [[Fehlermeldungen](#)]
- „Zeile wurde seit dem letzten Lesen aktualisiert“ [[Fehlermeldungen](#)]
- „Zeile seit dem letzten Lesen geändert - Vorgang abgebrochen“ [[Fehlermeldungen](#)]
- „Keine aktuelle Cursorzeile“ [[Fehlermeldungen](#)]

Insensitive Cursor

Diese Cursor haben insensitive Mitgliedschaft, Reihenfolge und Werte. Keine Änderungen, die nach dem Öffnen des Cursors durchgeführt werden, sind sichtbar.

Insensitive Cursor werden nur für schreibgeschützte Cursortypen verwendet.

Standards

Insensitive Cursor entsprechen der ISO/ANSI-Standarddefinition von insensitiven Cursor beziehungsweise den statischen ODBC-Cursor.

Programmierschnittstellen

Schnittstelle	Cursortyp	Kommentar
ODBC, ADO/OLE DB	Statisch	Wenn ein aktualisierbarer statischer Cursor angefordert wird, wird stattdessen ein wertsensitiver Cursor verwendet.
Embedded SQL	INSENSITIVE	
JDBC	INSENSITIVE	Insensitive Semantik wird nur vom SQL Anywhere JDBC-Treiber unterstützt.
Open Client	Nicht unterstützt	

Beschreibung

Insensitive Cursor geben immer Zeilen zurück, die den Auswahlkriterien der Abfrage entsprechen, und zwar in der durch eine ORDER BY-Klausel festgelegten Reihenfolge.

Die Ergebnismenge eines insensitive Cursors wird vollständig als Arbeitstabelle materialisiert, wenn der Cursor geöffnet wird. Das hat folgende Konsequenzen:

- Wenn die Ergebnismenge sehr umfangreich ist, sind die Festplatten- und Speicheranforderungen zum Verwalten des Ergebnisses möglicherweise von Bedeutung.
- Keine Zeile wird an die Anwendung zurückgegeben, bevor nicht die gesamte Ergebnismenge als eine Arbeitstabelle zusammengestellt ist. Bei komplexen Abfragen kann das zu einer Verzögerung führen, bevor die erste Zeile an die Anwendung zurückgegeben wird.
- Nachfolgende Zeilen können direkt von der Arbeitstabelle abgerufen werden, und werden daher rasch ausgegeben. Die Clientbibliothek kann mehrere Zeilen gleichzeitig vorab abrufen, was die Performance weiter steigert.
- Insensitive Cursor sind von ROLLBACK oder ROLLBACK TO SAVEPOINT nicht betroffen.

Sensitive Cursor

Sensitive Cursor können für schreibgeschützte oder aktualisierbare Cursortypen verwendet werden.

Diese Cursor haben sensitive Mitgliedschaft, Reihenfolge und Werte.

Standards

Sensitive Cursor entsprechen der ISO/ANSI-Standarddefinition von sensitiven Cursor beziehungsweise den dynamischen ODBC-Cursor.

Programmierschnittstellen

Schnittstelle	Cursortyp	Kommentar
ODBC, ADO/OLE DB	Dynamic	
Embedded SQL	SENSITIVE	Wird auch als Antwort auf eine Anfrage nach einem DYNAMIC SCROLL-Cursor geliefert, wenn keine Arbeitstabelle erforderlich und die Prefetch-Option auf OFF eingestellt ist.
JDBC	SENSITIVE	Sensitive Cursor werden vom SQL Anywhere JDBC-Treiber voll unterstützt.

Beschreibung

Prefetch-Vorgänge sind für sensitive Cursor deaktiviert. Alle Änderungen sind durch den Cursor sichtbar, sowohl die Änderungen durch den Cursor als auch die durch andere Transaktionen. Höhere Isolationsstufen können aufgrund von Sperren manche Änderungen verbergen, die von anderen Transaktionen durchgeführt werden.

Alle Änderungen von Mitgliedschaft, Reihenfolge und Spaltenwerten des Cursors sind sichtbar. Beispiel: Wenn ein sensitiver Cursor einen Join enthält und einer der Werte von einer der darunter liegenden Tabellen geändert wird, dann zeigen alle Ergebniszeilen, die aus dieser Basiszeile zusammengesetzt sind, den neuen Wert. Die Mitgliedschaft und Reihenfolge der Ergebnismenge können sich bei jedem Abruf ändern.

Sensitive Cursor geben immer Zeilen zurück, die den Auswahlkriterien der Abfrage entsprechen, und zwar in der durch eine ORDER BY-Klausel festgelegten Reihenfolge. Aktualisierungen können sich auf die Mitgliedschaft, Reihenfolge und Werte der Ergebnismenge auswirken.

Die Anforderungen von sensitiven Cursor bewirken bei der Implementierung von sensitiven Cursor folgende Einschränkungen:

- Zeilen können nicht vorab abgerufen werden, weil die Änderungen an solchen Zeilen durch den Cursor nicht sichtbar wären. Das kann sich auf die Performance auswirken.
- Sensitive Cursor müssen so implementiert werden, dass keine Arbeitstabellen zusammengestellt werden, weil Änderungen an den in der Arbeitstabelle gespeicherten Zeilen durch den Cursor nicht sichtbar wären.
- Die Bedingung, keine Arbeitstabelle zu verwenden, schränkt die Auswahl der Join-Methode durch den Optimierer ein und kann sich daher auf die Performance auswirken.
- Bei einigen Abfragen ist es unvermeidlich, dass der Optimierer einen Plan erstellt, der eine Arbeitstabelle enthält und somit keinen sensitiven Cursor erlaubt.

Arbeitstabellen werden üblicherweise zum Sortieren und Gruppieren von Zwischenergebnissen verwendet. Eine Arbeitstabelle ist zum Sortieren nicht erforderlich, wenn auf die Zeilen durch einen Index zugegriffen werden kann. Es ist nicht immer möglich vorherzusagen, welche Abfragen Arbeitstabellen verwenden, aber sie werden von folgenden Abfragen verwendet:

- UNION-Abfragen, auch wenn UNION ALL-Abfragen nicht unbedingt Arbeitstabellen verwenden.
- Anweisungen mit einer ORDER BY-Klausel, wenn es keinen Index auf der ORDER BY-Spalte gibt.
- Jede Abfrage, die mit einem Hash-Join optimiert ist.
- Viele Abfragen, die DISTINCT- oder GROUP BY-Klauseln betreffen.

In diesen Fällen gibt SQL Anywhere entweder eine Fehlermeldung an die Anwendung zurück, oder der Cursortyp wird in einen asensitiven Cursor geändert und eine Warnmeldung ausgegeben.

Siehe auch

- „Tipp: Arbeitstabellen in der Abfrageverarbeitung verwenden ("Alle Zeilen" als Optimierungsziel verwenden)“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)]

Asensitive Cursor

Diese Cursor haben keine genau definierte Sensitivität in ihrer Mitgliedschaft, ihrer Reihenfolge oder ihren Werten. Die Flexibilität, die sie in Bezug auf Sensitivität haben, ermöglicht es, asensitive Cursor für die Performance zu optimieren.

Asensitive Cursor werden nur für schreibgeschützte Cursortypen verwendet.

Standards

Asensitive Cursor entsprechen der ISO/ANSI-Standarddefinition von asensitiven Cursor beziehungsweise den ODBC-Cursor mit unbestimmter Sensitivität.

Programmierschnittstellen

Schnittstelle	Cursortyp
ODBC, ADO/OLE DB	Unspecified sensitivity
Embedded SQL	DYNAMIC SCROLL

Beschreibung

Die Anforderung eines asensitiven Cursors schränkt die Auswahl der Methoden kaum ein, die SQL Anywhere verwenden kann, um die Abfrage zu optimieren und Zeilen an die Anwendung zurückzugeben. Aus diesen Gründen erhalten Sie mit asensitiven Cursor die beste Performance. Vor allem steht es dem Optimierer frei, jede Maßnahme zur Materialisierung von Zwischenergebnissen, wie z.B. Arbeitstabellen, anzuwenden, und Zeilen können vom Client vorab abgerufen werden.

SQL Anywhere kann die Sichtbarkeit von Änderungen in den darunter liegenden Basiszeilen nicht garantieren. Einige Änderungen können sichtbar sein, andere nicht. Die Mitgliedschaft und Reihenfolge können sich mit jedem Abruf ändern. Besonders Aktualisierungen in Basiszeilen können dazu führen, dass nur einige der aktualisierten Spalten im Ergebnis des Cursors widerspiegelt werden.

Asensitive Cursor sind keine Garantie dafür, dass Zeilen zurückgegeben werden, die der Auswahl und Reihenfolge der Abfrage entsprechen. Die Zeilenmitgliedschaft steht zum Zeitpunkt des Öffnens des

Cursors fest, aber nachfolgende Änderungen an den darunter liegenden Werten werden in den Ergebnissen widerspiegelt.

Asensitive Cursor geben immer Zeilen zurück, die den WHERE- und ORDER BY-Klauseln des Kunden zu dem Zeitpunkt entsprachen, an dem die Cursor-Mitgliedschaft etabliert wurde. Wenn sich Spaltenwerte nach dem Öffnen des Cursors ändern, werden möglicherweise Zeilen zurückgegeben, die nicht mehr den WHERE- und ORDER BY-Klauseln entsprechen.

Wertsensitive Cursor

Bei wertsensitiven Cursor ist die Mitgliedschaft unbestimmt und die Reihenfolge und der Wert der Ergebnismenge sensitiv.

Wertsensitive Cursor können für schreibgeschützte oder aktualisierbare Cursortypen verwendet werden.

Standards

Wertsensitive Cursor entsprechen keiner ISO/ANSI-Standarddefinition. Sie entsprechen den Keyset-gesteuerten ODBC-Cursor.

Programmierschnittstellen

Schnittstelle	Cursortyp	Kommentar
ODBC, ADO/OLE DB	Durch Keyset gesteuert	
Embedded SQL	SCROLL	
JDBC	INSENSITIVE und CONCUR_UPDATABLE	Beim SQL Anywhere JDBC-Treiber wird die Anforderung eines INSENSITIVE Cursors mit einem wertsensitiven Cursor beantwortet.
Open Client und jConnect	Nicht unterstützt	

Beschreibung

Wenn eine Anwendung eine Zeile abrufen, die aus einer darunter liegenden geänderten Basiszeile besteht, dann muss der Anwendung der aktualisierte Wert sowie die SQL_ROW_UPDATED-Statusmeldung übermittelt werden. Wenn die Anwendung versucht, eine Zeile abzurufen, die aus einer darunter liegenden Basiszeile zusammengesetzt war, welche gelöscht worden ist, muss eine SQL_ROW_DELETED-Statusmeldung an die Anwendung übermittelt werden.

Eine Änderung am Primärschlüsselwert entfernt die Zeile aus der Ergebnismenge (dies wird als Löschen, gefolgt von Einfügen, behandelt). Ein Sonderfall tritt auf, wenn eine Zeile in der Ergebnismenge gelöscht wird (durch den Cursor oder von auswärts), und eine neue Zeile mit demselben Schlüsselwert eingefügt wird. Das führt dazu, dass die neue Zeile die alte Zeile in der ursprünglichen Position ersetzt.

Es gibt keine Garantie, dass die Zeilen in der Ergebnismenge dem Reihenfolgen- oder Auswahlkriterium der Abfrage entsprechen. Da die Zeilenmitgliedschaft zum Zeitpunkt des Öffnens festgelegt wird, führen

nachfolgende Änderungen, durch die eine Zeile nicht mehr der WHERE- oder ORDER BY-Klausel entspricht, nicht dazu, dass sich die Mitgliedschaft oder Position einer Zeile ändert.

Alle Werte sind in Bezug auf Änderungen, die durch den Cursor durchgeführt werden, sensitiv. Die Sensitivität der Mitgliedschaft gegenüber Änderungen, die durch den Cursor ausgeführt werden, wird durch die ODBC-Option `SQL_STATIC_SENSITIVITY` gesteuert. Wenn diese Option auf ON eingestellt ist, fügen Einfügungen durch den Cursor die Zeile dem Cursor hinzu. Ansonsten ist sie nicht in der Ergebnismenge enthalten. Löschungen durch den Cursor entfernen die Zeile aus der Ergebnismenge, wodurch eine Lücke vermieden wird, und geben die `SQL_ROW_DELETED`-Statusmeldung zurück.

Wertsensitive Cursor verwenden eine **Keyset-Tabelle**. Wenn der Cursor geöffnet wird, füllt SQL Anywhere eine Arbeitstabelle mit Kenndaten für jede Zeile an, die zur Ergebnismenge beiträgt. Wenn Sie die Ergebnismenge durchblättern, wird die Keyset-Tabelle zur Identifizierung der Mitgliedschaft der Ergebnismenge verwendet, aber die Daten werden, falls erforderlich, von den darunter liegenden Tabellen bezogen.

Die Eigenschaft der festen Mitgliedschaft von wertsensitiven Cursor ermöglicht es Ihrer Anwendung, sich an die Zeilenpositionen innerhalb eines Cursors zu erinnern, und stellt sicher, dass diese Positionen nicht geändert werden. Siehe [„Beispiel für Cursor-Sensitivität: Eine gelöschte Zeile“ auf Seite 18](#).

- Wenn eine Zeile seit dem Öffnen des Cursors aktualisiert beziehungsweise möglicherweise aktualisiert wurde, gibt SQL Anywhere die Warnung `SQLE_ROW_UPDATED_WARNING` zurück, wenn die Zeile abgerufen wird. Die Warnung wird nur einmal ausgegeben: Ein weiteres Abrufen der Zeile generiert keine Warnmeldung.

Eine Aktualisierung einer beliebigen Spalte in der Zeile verursacht diese Warnung sogar, wenn die aktualisierte Spalte nicht durch den Cursor referenziert wird. Ein Cursor auf Surname und GivenName würde beispielsweise die Aktualisierung melden, selbst wenn nur die Birthdate-Spalte geändert worden wäre. Diese Warn- und Fehlerbedingungen bei Aktualisierung treten nicht im Massenvorgangsmodus (Option `-b` für Datenbankserver) auf, wenn die Zeilensperre deaktiviert ist. Siehe [„Performance-Aspekte von Massenvorgängen“ \[SQL Anywhere Server - SQL-Benutzerhandbuch\]](#) und [„Zeile wurde seit dem letzten Lesen aktualisiert“ \[Fehlermeldungen\]](#).

- Der Versuch, eine positionsbasierte UPDATE- oder DELETE-Anweisung auf einer Zeile auszuführen, die seit dem letzten Abruf geändert wurde, gibt den `SQLE_ROW_UPDATED_SINCE_READ`-Fehler aus und bricht die Anweisung ab. Eine Anwendung muss die Zeile noch einmal mit `FETCH ABRUFEN`, bevor das UPDATE oder DELETE zugelassen wird.

Eine Aktualisierung einer beliebigen Spalte in der Zeile verursacht diesen Fehler. Dies ist sogar dann der Fall, wenn die aktualisierte Spalte nicht durch den Cursor referenziert wird. Der Fehler tritt nicht im Massenvorgangsmodus auf. Siehe [„Zeile seit dem letzten Lesen geändert - Vorgang abgebrochen“ \[Fehlermeldungen\]](#).

- Wenn eine Zeile, entweder durch den Cursor oder durch eine andere Transaktion, nach dem Öffnen des Cursors gelöscht wurde, entsteht im Cursor eine **Lücke**. Die Mitgliedschaft des Cursors steht fest, daher ist die Position einer Zeile reserviert, aber der DELETE-Vorgang wird im geänderten Wert für die Zeile wiedergespiegelt. Wenn Sie die Zeile an dieser Lücke abrufen, wird mit der Fehlermeldung `-197 SQLCODE` darauf hingewiesen, dass es keine aktuelle Zeile gibt, und der Cursor wird auf der Lücke belassen. Sie können Lücken vermeiden, indem Sie sensitive Cursor verwenden, da sich deren Mitgliedschaft zusammen mit den Werten verändert. Siehe [„Keine aktuelle Cursorzeile“ \[Fehlermeldungen\]](#).

Sie können für wertsensitive Cursor keine Zeilen vorab abrufen. Diese Anforderung kann sich auf die Performance auswirken.

Mehrere Zeilen einfügen

Wenn Sie mehrere Zeilen über einen wertsensitiven Cursor einfügen, erscheinen die neuen Zeilen am Ende der Ergebnismenge. Siehe „Für die Zeilenänderung verwendete Cursor“ auf Seite 12.

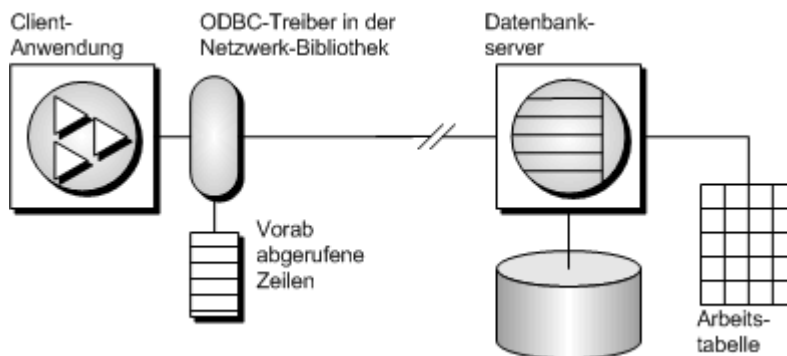
Cursor-Sensitivität und Performance

Es besteht eine Wechselwirkung zwischen Performance und anderen Cursor-Eigenschaften. Besonders wenn Sie einen Cursor aktualisierbar machen, führt das zu Beschränkungen der Abfrageverarbeitung und -zustellung, was die Performance vermindert. Auch können Anforderungen an die Cursor-Sensitivität die Performance einschränken.

Um zu verstehen, wie sich die Aktualisierbarkeit und Sensitivität von Cursor auf die Performance auswirkt, ist es hilfreich zu wissen, wie die Ergebnisse, die durch einen Cursor sichtbar sind, von der Datenbank an die Clientanwendung übermittelt werden.

Im Einzelnen können Ergebnisse aus Performance-Gründen an zwei dazwischengeschalteten Standorten gespeichert werden:

- **Arbeitstabellen** Sowohl Zwischen- als auch Endergebnisse können als Arbeitstabellen gespeichert werden. Wertsensitive Cursor verwenden eine Arbeitstabelle für Primärschlüsselwerte. Abfrageeigenschaften können ebenfalls dazu führen, dass der Optimierer in seinem gewählten Ausführungsplan Arbeitstabellen verwendet.
- **Prefetch-Vorgang** Der Client kann Zeilen vorab in einen clientseitigen Puffer abrufen, um separate Anfragen an den Datenbankserver für jede Zeile zu vermeiden.



Sensitivität und Aktualisierbarkeit beschränken die Verwendung von zwischengeschalteten Standorten.

Prefetch

Prefetch-Vorgänge und Mehrzeilen-Fetch-Vorgänge unterscheiden sich voneinander. Prefetch-Vorgänge können ohne explizite Anweisungen aus der Clientanwendung ausgeführt werden. Prefetch-Vorgänge

rufen Zeilen aus dem Server in einen Puffer auf dem Client ab, machen diese Zeilen aber für die Clientanwendung erst verfügbar, wenn die entsprechende Zeile von der Anwendung abgerufen wird.

Standardmäßig führt die Clientbibliothek von SQL Anywhere Prefetch-Vorgänge für mehrere Zeilen aus, wenn eine Anwendung eine einzelne Zeile abrufen. Die Clientbibliothek von SQL Anywhere speichert die zusätzlichen Zeilen in einem Puffer.

Prefetch-Vorgänge verbessern die Performance durch die Reduktion des Client/Server-Roundtrips und erhöhen den Durchsatz, indem ohne separate Anforderung für einzelne Zeilen oder Zeilenblöcke viele Zeilen verfügbar gemacht werden.

Weitere Hinweise zur Steuerung der Prefetch-Funktionen finden Sie unter „[prefetch-Option](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

Prefetch-Funktionen aus einer Anwendung steuern

- Die Prefetch-Option steuert, ob Prefetch-Vorgänge durchgeführt werden. Sie können die Prefetch-Option für eine einzelne Verbindung auf "Immer", "Bedingt" oder "Aus" setzen. Standardmäßig ist sie auf "Bedingt" gesetzt.

- In Embedded SQL können Sie die PREFETCH-Vorgänge auf Cursorbasis steuern, wenn Sie einen Cursor bei einem FETCH-Vorgang öffnen, indem Sie die BLOCK-Klausel verwenden.

Die Anwendung kann eine maximale Anzahl von Zeilen festlegen, die in einem einzelnen FETCH-Vorgang vom Server enthalten sein dürfen, indem die BLOCK-Klausel angegeben wird. Beispiel: Wenn Sie 5 Zeilen gleichzeitig abrufen und anzeigen, können Sie BLOCK 5 verwenden. Wenn Sie BLOCK 0 festlegen, wird jeweils 1 Datensatz abgerufen, und ein FETCH RELATIVE 0 ruft die Zeile nochmals vom Server ab.

Obwohl Sie die PREFETCH-Vorgänge auch durch einen Verbindungsparameter für die Anwendung ausschalten können, ist es effektiver, BLOCK 0 zu verwenden, als die Prefetch-Option auf Off zu setzen. Siehe „[prefetch-Option](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

- Prefetch-Vorgänge für wertsensitive Cursortypen sind standardmäßig deaktiviert.
- In Open Client können Sie das PREFETCH-Verhalten mit `ct_cursor` und `CS_CURSOR_ROWS` nach der Deklaration, aber vor dem Öffnen des Cursors steuern.

Prefetch erhöht die Anzahl der Prefetch-Zeilen dynamisch, wenn es möglich ist, Verbesserungen der Performance zu erzielen. Dies umfasst Cursor, die den folgenden Bedingungen entsprechen:

- Sie verwenden einen der unterstützten Cursortypen:
 - **ODBC und OLE DB** FORWARD-ONLY- und READ-ONLY-Cursor (Standard)
 - **Embedded SQL** DYNAMIC SCROLL- (Standard), NO SCROLL- und INSENSITIVE-Cursor
 - **ADO.NET** Alle Cursor
- Sie führen nur FETCH NEXT-Vorgänge aus (keine absoluten, relativen oder Rückwärts-Fetch-Vorgänge).

- Die Anwendung ändert den Typ der Hostvariablen zwischen Fetch-Vorgängen nicht und verwendet keine GET DATA-Anweisung, um Spaltendaten in Blöcken abzufragen (die Verwendung *einer* GET DATA-Anweisung zum Abrufen des Werts wird unterstützt).

Verlorene Aktualisierungen

Bei der Verwendung von aktualisierbaren Cursor muss darauf geachtet werden, dass keine Aktualisierungen verloren gehen. Eine verlorene Aktualisierung liegt vor, wenn zwei (oder mehr) Transaktionen dieselbe Zeile aktualisieren, aber keine der beiden Transaktionen von der Änderung durch die andere Transaktion weiß und daher die erste Änderung durch die zweite Änderung überschrieben wird. Dieses Problem wird im folgenden Beispiel erläutert:

1. Eine Anwendung öffnet in der Beispieldatenbank in der folgenden Abfrage einen Cursor.

```
SELECT ID, Quantity
FROM Products;
```

ID	Quantity
300	28
301	54
302	75
...	...

2. Die Anwendung ruft die Zeile mit ID = 300 durch den Cursor ab.
3. Eine weitere Transaktion aktualisiert die Zeile mit der folgenden Anweisung:

```
UPDATE Products
SET Quantity = Quantity - 10
WHERE ID = 300;
```

4. Die Anwendung aktualisiert dann die Zeile durch den Cursor auf einen Wert von (Quantity - 5).
5. Der korrekte Endwert für die Zeile sollte 13 sein. Wenn der Cursor die Zeile vorab abgerufen hätte, würde der neue Wert für die Zeile 23 lauten und die Aktualisierung der anderen Transaktion verloren gehen.

In einer Datenbankanwendung gibt es auf jeder Ebene das Potenzial für eine verlorene Aktualisierung, wenn eine Zeile geändert wird, ohne zuvor ihren Wert zu überprüfen. Auf höheren Isolationsstufen (2 und 3), können Sperren (Lese-, Absichts- und Schreibsperren) verwendet werden, um sicherzustellen, dass eine Zeile nicht geändert werden kann, wenn die Zeilen von der Anwendung gelesen wurde. Auf den Isolationsstufen 0 und 1 ist das Potenzial für einen Verlust von Aktualisierungen dagegen höher. Auf der Isolationsebene 0 werden keine Lesesperren erworben, um nachfolgende Änderungen der Daten zu verhindern, und die Isolationsebene 1 sperrt nur die aktuelle Zeile. Bei der Verwendung von Snapshot-Isolation kann es nicht zu verlorenen Aktualisierungen kommen, da jeder Versuch, den alten Wert zu

ändern, zu einem Aktualisierungskonflikt führt. Die Verwendung von Prefetch-Vorgängen auf Isolationsstufe 1 kann auch zu einem möglichen Verlust von Aktualisierungen führen, da die Ergebnismengenzeile, auf die die Anwendung positioniert ist und die sich im Prefetch-Puffer des Clients befindet, möglicherweise nicht dieselbe ist wie die aktuelle Zeile, auf die der Server im Cursor positioniert ist.

Um einen Aktualisierungsverlust auf Isolationsstufe 1 zu verhindern, unterstützt der Datenbankserver drei verschiedene Parallelitäts-Kontrollmechanismen, die von einer Anwendung festgelegt werden können:

1. Der Erwerb von Absichtszeilensperren für jede Zeile im Cursor, wenn er abgerufen wird. Absichtssperren verhindern, dass andere Transaktionen Absichts- oder Schreibsperren für dieselbe Zeile erwerben, und damit auch gleichzeitige Aktualisierungen. Absichtssperren blockieren jedoch keine Lesezeilensperren und haben damit keine Auswirkung auf die Parallelität reiner Leseanweisungen.
2. Die Verwendung eines wertsensitiven Cursors. Wertsensitive Cursor können für die Protokollierung verwendet werden, wenn eine zugrunde liegende Zeile geändert oder gelöscht wurde, sodass die Anwendung antworten kann.
3. Die Verwendung von FETCH FOR UPDATE, womit eine Absichtszeilensperre für die betreffende Zeile erworben wird.

Wie diese Alternativen angegeben werden, hängt von der Schnittstelle ab, die von der Anwendung verwendet wird. Für die ersten beiden Alternativen, die sich auf eine SELECT-Anweisung beziehen, gibt es folgende Möglichkeiten:

- In ODBC können keine Aktualisierungsverluste auftreten, da die Anwendung bei der Deklaration eines aktualisierbaren Cursors einen Cursor-Parallelitätsparameter für die Funktion `SQLSetStmtAttr` angeben muss. Bei diesem Parameter handelt es sich um `SQL_CONCUR_LOCK`, `SQL_CONCUR_VALUES`, `SQL_CONCUR_READ_ONLY` oder `SQL_CONCUR_TIMESTAMP`. Bei `SQL_CONCUR_LOCK` erwirbt der Datenbankserver Absichtszeilensperren. Bei `SQL_CONCUR_VALUES` und `SQL_CONCUR_TIMESTAMP` wird ein wertsensitiver Cursor verwendet. `SQL_CONCUR_READ_ONLY` wird für schreibgeschützte Cursor verwendet und ist die Standardeinstellung.
- In JDBC ist die Parallelitätseinstellung für eine Anweisung ähnlich wie bei ODBC. Der SQL Anywhere JDBC-Treiber unterstützt die JDBC-Parallelitätswerte `RESULTSET_CONCUR_READ_ONLY` und `RESULTSET_CONCUR_UPDATABLE`. Der erste Wert entspricht der ODBC-Parallelitätseinstellung `SQL_CONCUR_READ_ONLY` und legt eine reine Leseanweisung fest. Der zweite Wert entspricht der Einstellung ODBC `SQL_CONCUR_LOCK`, sodass Absichtszeilensperren verwendet werden, um einen Verlust von Aktualisierungen zu verhindern. Wertsensitive Cursor können in der JDBC 4.0-Spezifikation nicht direkt angegeben werden.
- In jConnect werden aktualisierbare Cursor auf der API-Ebene unterstützt, doch die zu Grunde liegende Implementierung (unter Verwendung von TDS) unterstützt keine Aktualisierungen durch einen Cursor. Stattdessen sendet jConnect eine eigene UPDATE-Anweisung an den Datenbankserver, um die angegebene Zeile zu aktualisieren. Um einen Aktualisierungsverlust zu vermeiden, muss die Anwendung auf Isolationsstufe 2 oder höher ausgeführt werden. Alternativ dazu kann die Anwendung separate UPDATE-Anweisungen vom Cursor ausführen. Es muss jedoch sichergestellt werden, dass

die UPDATE-Anweisung überprüft, ob die Zeilenwerte geändert wurden, seit die Zeile gelesen wurde, indem entsprechende Bedingungen in die WHERE-Klauseln der UPDATE-Anweisung positioniert werden.

- In Embedded SQL kann eine Parallelitätsangabe gesetzt werden, indem Syntax in die SELECT-Anweisung selbst oder in die Cursordeklaration aufgenommen wird. In der SELECT-Anweisung bewirkt die Syntax `SELECT...FOR UPDATE BY LOCK`, dass der Datenbankserver eine Absichtszeilensperre für die Ergebnismenge erwirbt.

`SELECT...FOR UPDATE BY [VALUES | TIMESTAMP]` verursacht dagegen, dass der Datenbankserver den Cursortyp in einen wertsensitiven Cursor ändert. Wenn eine bestimmte Zeile seit dem letzten Lesen durch den Cursor geändert wurde, erhält die Anwendung eine Warnung (`SQL_ROW_UPDATED_WARNING`) für eine FETCH-Anweisung bzw. einen Fehler (`SQL_ROW_UPDATED_SINCE_READ`) für eine UPDATE WHERE CURRENT OF-Anweisung. Wenn die Zeile gelöscht wurde, erhält die Anwendung ebenfalls einen Fehler (`SQL_NO_CURRENT_ROW`).

Die FETCH FOR UPDATE-Funktionalität wird ebenfalls von der Embedded SQL- und der ODBC-Schnittstelle unterstützt, wenngleich die Details abhängig von der verwendeten API-Schnittstelle unterschiedlich sind.

In Embedded SQL verwendet die Anwendung `FETCH FOR UPDATE` anstelle von `FETCH`, damit eine Absichtssperre für die Zeile erworben wird. In ODBC verwendet die Anwendung den API-Aufruf `SQLSetPos` mit dem Vorgangsargument `SQL_POSITION` oder `SQL_REFRESH` und dem Sperrentypargument `SQL_LOCK_EXCLUSIVE`, um eine Absichtssperre für eine Zeile zu erwerben. In SQL Anywhere sind dies langfristige Sperren, die gehalten werden, bis die Transaktion festgeschrieben oder zurückgesetzt wird.

Cursor-Sensitivität und Isolationsstufen

Sowohl die Cursor-Sensitivität als auch die Isolationsstufen beziehen sich auf die Kontrolle der Parallelität, jedoch auf unterschiedliche Weise und mit verschiedenen Vor- und Nachteilen.

Indem Sie eine Isolationsstufe für eine Transaktion auswählen (gewöhnlich auf der Verbindungsebene), legen Sie fest, welche Typen von Sperren wann auf Zeilen in der Datenbank plazierte werden. Sperren verhindern, dass andere Transaktionen auf die Zeilen in der Datenbank zugreifen oder sie verändern. Im Allgemeinen ist bei einer größeren Zahl von Sperren ein geringerer Umfang von Parallelität bei gleichzeitigen Transaktionen zu erwarten.

Sperren verhindern jedoch nicht, dass andere Teile einer Transaktion ausgeführt werden. Daher kann eine einzelne Transaktion, die mehrere aktualisierbare Cursor verwaltet, nicht mithilfe von Sperren verhindern, dass es zu Aktualisierungsverlusten kommt.

Mithilfe der Snapshot-Isolation soll erreicht werden, dass keine Lesesperren erforderlich sind, indem dafür gesorgt wird, dass jede Transaktion eine konsistente Ansicht der Datenbank erhält. Der Vorteil besteht darin, dass eine konsistente Ansicht der Datenbank abgefragt werden kann, ohne sich auf vollständige serialisierbare Transaktionen (Isolationsstufe 3) zu verlassen und ohne den Verlust der Parallelität, der durch die Verwendung der Isolationsstufe 3 entsteht. Die Snapshot-Isolation ist jedoch sehr teuer, da Kopien geänderter Zeilen verwaltet werden müssen, um die Anforderungen der

gleichzeitigen Snapshot-Transaktionen, die bereits ausgeführt werden, und die Snapshot-Transaktionen, die noch starten müssen, zu erfüllen. Aufgrund dieser Kopienverwaltung ist die Verwendung der Snapshot-Isolation für intensive Aktualisierungsvorgänge ungeeignet. Siehe [Snapshot-Isolationsstufe wählen \[SQL Anywhere Server - SQL-Benutzerhandbuch\]](#).

Die Cursor-Sensitivität legt allerdings fest, welche Änderungen für das Ergebnis des Cursors sichtbar sind. Da die Cursor-Sensitivität auf Cursor-Basis festgelegt wird, bezieht sich die Cursor-Sensitivität sowohl auf die Auswirkungen auf andere Transaktionen als auch auf die Aktualisierungsaktivität derselben Transaktion, wenngleich diese Auswirkungen vollständig vom Typ des festgelegten Cursors abhängen. Durch das Festlegen der Cursor-Sensitivität legen Sie nicht direkt fest, wann Sperren für Zeilen in der Datenbank erworben werden. Die Kombination der Cursor-Sensitivität und der Isolationsstufe steuert jedoch die verschiedenen Parallelitätsszenarien, die in einer bestimmten Anwendung möglich sind.

Anforderungen für SQL Anywhere-Cursor

Wenn Sie einen Cursortyp von Ihrer Clientanwendung aus anfordern, liefert SQL Anywhere einen Cursor. SQL Anywhere-Cursor werden nicht durch den in der Programmierschnittstelle festgelegten Typ definiert, sondern durch die Sensitivität der Ergebnismenge auf Änderungen in den darunter liegenden Daten. Abhängig vom verlangten Cursortyp liefert SQL Anywhere einen Cursor mit einem Verhalten, das dem Typ entspricht.

Die Sensitivität der SQL Anywhere-Cursor wird entsprechend der Cursortyp-Anfrage des Clients gesetzt.

ADO.NET

Schreibgeschützte Cursor nur für die Weitergabe sind bei der Verwendung von `SACommand.ExecuteReader` verfügbar. Das `SADDataAdapter`-Objekt verwendet eine clientseitige Ergebnismenge anstelle von Cursor. Siehe [SACommand-Klasse \[SQL Anywhere .NET\]](#) auf Seite 129.

ADO/OLE DB und ODBC

Die nachstehende Tabelle beschreibt die Cursor-Sensitivität, die bei verschiedenen scrollfähigen ODBC-Cursortypen eingestellt wird.

Scrollfähiger ODBC-Cursortyp	SQL Anywhere-Cursor
STATIC	Insensitiv
KEYSET-DRIVEN	Wertsensitiv
DYNAMIC	Sensitiv
MIXED	Wertsensitiv

Sie erhalten einen MIXED-Cursor, indem Sie den Cursortyp auf `SQL_CURSOR_KEYSET_DRIVEN` setzen und dann die Anzahl der Zeilen im Keyset für einen Keyset-gesteuerten Cursor mit

SQL_ATTR_KEYSET_SIZE festlegen. Wenn die Keyset-Größe 0 ist (Standardwert), ist der Cursor vollständig Keyset-gesteuert. Wenn die Keyset-Größe größer als 0 ist, ist der Cursor gemischt (Keyset-gesteuert innerhalb des Keysets und dynamisch außerhalb des Keysets). Die Keyset-Standardgröße ist 0. Es ist ein Fehler, wenn die Keyset-Größe größer als 0 und kleiner als die Zeilengruppe (SQL_ATTR_ROW_ARRAY_SIZE) ist.

Hinweise zu SQL Anywhere-Cursor und ihrem Verhalten finden Sie unter „[SQL Anywhere-Cursor](#)“ auf Seite 16.

Hinweise über das Anfordern eines Cursortyps in ODBC finden Sie unter „[ODBC-Cursormerkmale](#)“ auf Seite 389.

Ausnahmen

Wenn Sie einen STATIC-Cursor als aktualisierbar anfordern, wird stattdessen ein wertsensitiver Cursor geliefert und eine Warnung ausgegeben.

Wenn ein DYNAMIC- oder MIXED-Cursor angefordert wird und die Abfrage nicht ohne Arbeitstabellen ausgeführt werden kann, wird stattdessen ein asensitiver Cursor geliefert und eine Warnung ausgegeben.

JDBC

Die JDBC 4.0-Spezifikation unterstützt die folgenden drei Cursortypen: insensitiv, sensitiv und asensitiver Vorwärtscursor. Der SQL Anywhere JDBC-Treiber ist mit diesen JDBC-Spezifikationen kompatibel und unterstützt diese verschiedenen Cursortypen für ein JDBC ResultSet-Objekt. Es gibt jedoch Fälle, in denen der Datenbankserver keinen Zugriffsplan mit der erforderlichen Semantik für einen bestimmten Cursortyp erstellen kann. In diesen Fällen gibt der Datenbankserver entweder einen Fehler zurück oder er verwendet einen anderen Cursortyp. Siehe „[Sensitive Cursor](#)“ auf Seite 23.

Bei jConnect unterstützt das zu Grunde liegende Protokoll (TDS) Cursor auf dem Datenbankserver nur zum Weitergeben und nur zum Lesen, wenngleich jConnect die APIs zum Erstellen verschiedener Typen von Cursor gemäß der JDBC 2.0-Spezifikation unterstützt. Alle jConnect Cursor sind asensitiv, da das TDS-Protokoll die Ergebnisse der Anweisung in Blöcken puffert. Diese Blöcke von gepufferten Ergebnissen werden durchsucht, wenn die Anwendung einen insensitiven oder sensitiven Cursortyp suchen muss, der die Abrollfähigkeit unterstützt. Wenn die Anwendung vor den Anfang der zwischengespeicherten Ergebnismenge zurückblättert, wird die Anweisung erneut ausgeführt. Dies kann zu Inkonsistenzen der Daten führen, wenn die Daten zwischen den beiden Ausführungen der Anweisung geändert wurden.

Embedded SQL

Um einen Cursor von einer Embedded SQL-Anwendung anzufordern, geben Sie den Cursortyp in der DECLARE-Anweisung an. Die nachstehende Tabelle beschreibt die Cursor-Sensitivität, die als Antwort auf unterschiedliche Anforderungen eingestellt wird.

Cursortyp	SQL Anywhere-Cursor
NO SCROLL	Asensitiv
DYNAMIC SCROLL	Asensitiv
SCROLL	Wertsensitiv
INSENSITIVE	Insensitiv
SENSITIVE	Sensitiv

Ausnahmen

Wenn Sie einen DYNAMIC SCROLL- oder NO SCROLL-Cursor als UPDATABLE anfordern, wird ein sensativer oder wertsensitiver Cursor geliefert. Es ist nicht ausgemacht, welcher der Beiden geliefert wird. Diese Ungewissheit entspricht der Definition von asensitivem Verhalten.

Wenn Sie einen INSENSITIVE-Cursor als UPDATABLE anfordern, wird ein wertsensitiver Cursor geliefert.

Wenn Sie einen DYNAMIC SCROLL-Cursor anfordern, die Prefetch-Datenbankoption auf Off eingestellt ist und der Ausführungsplan der Abfrage keine Arbeitstabellen verlangt, wird möglicherweise ein sensativer Cursor geliefert. Wieder entspricht diese Ungewissheit der Definition von asensitivem Verhalten.

Open Client

Wie bei jConnect unterstützt das zu Grunde liegende Protokoll (TDS) für Open Client nur schreibgeschützte, asensitive Cursor, die nur zum Weitergeben bestimmt sind.

Ergebnismenge-Deskriptoren

Einige Anwendungen bauen SQL-Anweisungen auf, die in der Anwendung nicht ausgeführt werden können. Anweisungen hängen manchmal von einer Antwort des Benutzers ab, sodass die Anwendung erst dann erfährt, welche Daten abzurufen sind, wenn z.B. eine Berichtsanwendung dem Benutzer die Möglichkeit gibt, die anzuzeigenden Spalten auszuwählen.

In einem solchen Fall benötigt die Anwendung eine Methode, um Informationen über die Art der **Ergebnismenge** selbst sowie den Inhalt der Ergebnismenge zu erhalten. Die Informationen über die Art der Ergebnismenge werden als **Deskriptor** bezeichnet. Sie identifizieren die Datenstruktur einschließlich Anzahl und Typ der erwarteten Spalten. Wenn die Anwendung die Art der Ergebnismenge ermittelt hat, ist der Abruf des Inhalts ein einfacher Vorgang.

Diese **Ergebnismengen-Metadaten** (Informationen über Art und Inhalt der Daten) werden mithilfe von Deskriptoren bearbeitet. Das Beziehen und Verwalten von Ergebnismengen-Metadaten wird als **Beschreiben** bezeichnet.

Da Cursor im Allgemeinen Ergebnismengen produzieren, sind Deskriptoren und Cursor eng verknüpft, obwohl manche Schnittstellen die Verwendung von Deskriptoren vor dem Benutzer verbergen. Normalerweise gilt: Anweisungen, die Deskriptoren benötigen, sind entweder SELECT-Anweisungen oder gespeicherte Prozeduren, die Ergebnismengen zurückgeben.

Ein Deskriptor wird bei einem cursorbasierten Vorgang folgendermaßen eingesetzt:

1. Weisen Sie den Deskriptor zu. Dies kann implizit erfolgen, die explizite Zuweisung ist aber in manchen Schnittstellen zulässig.
2. Bereiten Sie die Anweisung vor.
3. Anweisung beschreiben. Wenn es sich bei der Anweisung um eine gespeicherte Prozedur oder um einen Batch handelt und die Ergebnismenge nicht durch eine Ergebnisklausel in der Prozedurdefinition definiert wird, sollte die Beschreibung nach dem Öffnen des Cursors erscheinen.
4. Deklarieren Sie einen Cursor für die Anweisung und öffnen Sie ihn (Embedded SQL), oder führen Sie die Anweisung aus.
5. Beziehen Sie den Deskriptor und ändern Sie erforderlichenfalls den zugewiesenen Bereich. Dies erfolgt oft implizit.
6. Rufen Sie die Anweisungsergebnisse ab und verarbeiten Sie sie.
7. Heben Sie die Zuweisung des Deskriptors auf.
8. Schließen Sie den Cursor.
9. Löschen Sie die Anweisung. Einige Schnittstellen führen dies automatisch durch.

Hinweise zur Implementierung

- In Embedded SQL enthält eine SQLDA-(SQL Descriptor Area) Struktur die Deskriptor-Informationen. Siehe „[Der SQL-Deskriptor-Bereich \(SQLDA\)](#)“ auf Seite 504.
- In ODBC ermöglicht ein Deskriptor-Handle, durch SQLAllocHandle zugewiesen, den Zugriff auf die Felder eines Deskriptors. Sie können diese Felder mit SQLSetDescRec, SQLSetDescField, SQLGetDescRec und SQLGetDescField verarbeiten.
Alternativ können Sie SQLDescribeCol and SQLColAttributes verwenden, um Spalteninformationen zu beziehen.
- In Open Client können Sie ct_dynamic für die Vorbereitung einer Anweisung und ct_describe zur Beschreibung der Ergebnismenge der Anweisung verwenden. Sie können aber auch ct_command verwenden, um eine SQL-Anweisung ohne vorherige Vorbereitung zu senden, und dann mit ct_results die zurückgegebenen Zeilen nacheinander verarbeiten. Dies ist die normalerweise benutzte Vorgehensweise bei der Open Client-Anwendungsentwicklung.
- In JDBC stellt die Klasse java.sql.ResultSetMetaData Informationen zu Ergebnismengen bereit.
- Sie können auch Deskriptoren verwenden, um Daten an den Datenbankserver zu senden (z.B. mit der INSERT-Anweisung), aber dies ist eine andere Art von Deskriptor als für die Ergebnismenge.

Weitere Hinweise zu Eingabe- und Ausgabeparametern für die DESCRIBE-Anweisung finden Sie unter „[DESCRIBE-Anweisung \[ESQL\]“ \[SQL Anywhere Server - SQL-Referenzhandbuch\]](#).

Transaktionen in Anwendungen

Transaktionen sind Zusammenstellungen einzelner SQL-Anweisungen. Entweder werden alle Anweisungen in der Transaktion ausgeführt oder keine. In diesem Abschnitt werden einige Aspekte der Transaktionen in Anwendungen behandelt.

Weitere Hinweise zu Transaktionen finden Sie unter „[Transaktionen und Isolationsstufen“ \[SQL Anywhere Server - SQL-Benutzerhandbuch\]](#).

Autocommit-Modus und manueller Commit-Modus

Datenbank-Programmierschnittstellen können entweder im **manuellen Commit-Modus (manueller Festschreibemodus)** oder im **Autocommit-Modus (automatischen Festschreibemodus)** operieren.

- **Manueller Commit-Modus** Vorgänge werden nur festgeschrieben, wenn Ihre Anwendung eine explizite Festschreibungsoperation durchführt oder der Datenbankserver eine automatische Festschreibung vollzieht, wie zum Beispiel beim Ausführen einer ALTER TABLE-Anweisung oder anderer Datendefinitionsanweisungen. Der manuelle Commit-Modus wird manchmal auch **chained-Modus** genannt.

Um in Ihren Anwendungen Transaktionen wie verschachtelte Transaktionen und Savepoints verwenden zu können, müssen Sie im manuellen Festschreibemodus operieren.

- **Autocommit-Modus** Jede Anweisung wird wie eine separate Transaktion behandelt. Die Wirkung des Autocommit-Modus ist dieselbe, wie wenn Sie an das Ende jeder SQL-Anweisung eine COMMIT-Anweisung anhängen. Der automatische Festschreibemodus wird manchmal auch **unchained-Modus** genannt.

Dieser Modus kann sich auf die Performance und das Verhalten Ihrer Anwendung auswirken. Verwenden Sie ihn nicht, wenn Ihre Anwendung Transaktionsintegrität erfordert.

Hinweise darüber, wie sich der Autocommit-Modus auf die Performance auswirkt, finden Sie unter „[Tipp: Autocommit-Modus ausschalten“ \[SQL Anywhere Server - SQL-Benutzerhandbuch\]](#).

Autocommit-Verhalten steuern

Wie Sie das Festschreibungsverhalten Ihrer Anwendung steuern hängt von der verwendeten Programmierschnittstelle ab. Die Implementierung von Autocommit kann, abhängig von der Schnittstelle, clientseitig oder serverseitig stattfinden. Siehe „[Autocommit-Implementierung“ auf Seite 39](#).

Steuern des Autocommit-Modus (ADO.NET)

In der Standardeinstellung arbeitet der ADO.NET-Provider im Autocommit-Modus. Mithilfe der Methode `SAConnection.BeginTransaction` können Sie explizite Transaktionen verwenden. Siehe „[Transaktionsverarbeitung“ auf Seite 64](#).

Steuern des Autocommit-Modus (OLE DB)

In der Standardeinstellung arbeitet OLE DB im Autocommit-Modus. Mithilfe der Methoden `ITransactionLocal::StartTransaction`, `ITransaction::Commit` und `ITransaction::Abort` können Sie explizite Transaktionen verwenden.

Steuern des Autocommit-Modus (ODBC)

In der Standardeinstellung arbeitet ODBC im Autocommit-Modus. Die Art, wie Sie Autocommit ausschalten, hängt davon ab, ob Sie ODBC direkt verwenden, oder ein Anwendungsentwicklungstool einsetzen. Wenn Sie direkt über die ODBC-Schnittstelle programmieren, stellen Sie das `SQL_ATTR_AUTOCOMMIT`-Verbindungsattribut ein.

Steuern des Autocommit-Modus (JDBC)

Standardmäßig arbeitet JDBC im Autocommit-Modus. Verwenden Sie zum Ausschalten von Autocommit die `setAutoCommit`-Methode des Verbindungsobjekts:

```
conn.setAutoCommit( false );
```

Steuern des Autocommit-Modus (Embedded SQL)

Standardmäßig operieren Embedded SQL-Anwendungen im manuellen Festschreibemodus. Um Autocommit einzuschalten, setzen Sie die Datenbankoption "chained" (serverseitige Option) auf Off, indem Sie eine Anweisung wie die folgende verwenden:

```
SET OPTION chained='Off';
```

Steuern des Autocommit-Modus (Open Client)

Standardmäßig operiert eine Verbindung, die durch Open Client hergestellt wird, im Autocommit-Modus. Sie können dieses Verhalten ändern, indem Sie die Datenbankoption chained (serverseitige Option) in Ihrer Anwendung auf On setzen, indem Sie eine Anweisung wie die folgende verwenden:

```
SET OPTION chained='On';
```

Steuern des Autocommit-Modus (PHP)

In der Standardeinstellung arbeitet PHP im Autocommit-Modus. Wenn Sie Autocommit deaktivieren möchten, verwenden Sie die `sasql_set_option`-Funktion:

```
$result = sasql_set_option( $conn, "auto_commit", "Off" );
```

Steuern des Autocommit-Modus (auf dem Server)

Der Datenbankserver arbeitet im manuellen Festschreibemodus. Um automatische Festschreibungen einzuschalten, setzen Sie die Datenbankoption chained (serverseitige Option) auf Off, indem Sie eine Anweisung wie die folgende verwenden:

```
SET OPTION chained='Off';
```

Wenn Sie eine Schnittstelle verwenden, die Festschreibungen auf der Clientseite steuert, kann das Festlegen der Datenbankoption chained (serverseitige Option) Auswirkungen auf die Performance bzw. das Verhalten der Anwendung haben. Es wird nicht empfohlen, den chained-Modus des Servers zu verwenden.

Siehe auch

- „Autocommit-Modus und manueller Commit-Modus“ auf Seite 37

Autocommit-Implementierung

Abhängig von der verwendeten Schnittstelle und dem Provider sowie davon, wie Sie das Autocommit-Verhalten steuern, verhält sich der Autocommit-Modus leicht unterschiedlich.

Der Autocommit-Modus kann auf zwei Arten implementiert werden:

- **Clientseitiges Autocommit** Wenn eine Anwendung Autocommit verwendet, sendet die Clientbibliothek eine COMMIT-Anweisung nach jeder ausgeführten SQL-Anweisung.

ADO.NET-, ADO/OLE DB-, ODBC-, PHP- und SQL Anywhere JDBC-Anwendungen steuern das Festschreibeverhalten von der Clientseite.
- **Serverseitiges Autocommit** Wenn eine Anwendung den chained-Modus deaktiviert, schreibt der Datenbankserver die Ergebnisse der einzelnen SQL-Anweisungen fest. Beim Sybase jConnect JDBC-Treiber wird dieses Verhalten durch die chained-Datenbankoption gesteuert.

Embedded SQL-, der jConnect-Treiber und Open Client-Anwendungen verändern das serverseitige Festschreibeverhalten (sie legen z.B. die chained-Option fest).

Bei zusammengesetzten Anweisungen wie gespeicherten Prozeduren oder Triggern gibt es einen Unterschied zwischen clientseitigem und serverseitigem Autocommit. Für den Client ist eine gespeicherte Prozedur eine einzelne Anweisung, und daher sendet Autocommit eine einzelne COMMIT-Anweisung, nachdem die gesamte Prozedur ausgeführt wurde. Aus der Perspektive des Datenbankservers kann die gespeicherte Prozedur aus vielen SQL-Anweisungen bestehen. Daher schreibt ein serverseitiges Autocommit die Ergebnisse der einzelnen SQL-Anweisungen innerhalb der Prozedur fest.

Hinweis

Clientseitige und serverseitige Implementierungen nicht vermischen. Kombinieren Sie die Einstellung der chained-Option nicht mit der Autocommit-Option in Ihrer ADO.NET-, OLE DB-, ODBC-, PHP- oder JDBC-Anwendung von SQL Anywhere.

Einstellungen der Isolationsstufe

Sie können die Isolationsstufe einer aktuellen Verbindung mit der Datenbankoption `isolation_level` einstellen.

Einige Schnittstellen, wie ODBC, ermöglichen das Setzen der Isolationsstufe für eine Verbindung beim Aufbau. Diese Isolationsstufe kann später mit der Datenbankoption `isolation_level` zurückgesetzt werden. Siehe „`isolation_level`-Option“ [*SQL Anywhere Server - Datenbankadministration*].

Sie können temporäre oder öffentliche Einstellungen für die `isolation_level`-Datenbankoption in einzelnen INSERT-, UPDATE-, DELETE-, SELECT-, UNION-, EXCEPT- und INTERSECT-Anweisungen aufheben, indem Sie eine OPTION-Klausel in die Anweisung aufnehmen.

Cursor und Transaktionen

Im Allgemeinen wird ein Cursor geschlossen, wenn ein COMMIT ausgeführt wird. Es gibt zwei Ausnahmen für dieses Verhalten.

- Die Datenbankoption `close_on_endtrans` ist auf Off gesetzt.
- Ein Cursor wird mit `WITH HOLD` geöffnet, was der Standard bei Open Client und JDBC ist.

Trifft einer dieser beiden Fälle zu, bleibt der Cursor bei COMMIT geöffnet.

ROLLBACK und Cursor

Beim Zurücksetzen einer Transaktion wird der Cursor geschlossen, außer wenn er mit `WITH HOLD` geöffnet wurde. Dem Inhalt eines Cursors können Sie nach einem Zurücksetzen nicht vertrauen.

Der projektierte ISO SQL3-Standard legt fest, dass bei einem Zurücksetzen alle Cursor geschlossen werden sollen. Sie können dieses Verhalten erzwingen, indem Sie die Option `ansi_close_cursors_on_rollback` auf On setzen.

Savepoints

Wenn eine Transaktion bis zu einem Savepoint zurückgesetzt wird und `ansi_close_cursors_on_rollback` auf On gesetzt ist, wird jeder nach dem SAVEPOINT geöffnete Cursor geschlossen (sogar die Cursor, die mit `WITH HOLD` geöffnet wurden).

Cursor und Isolationsstufe

Sie können die Isolationsstufe für eine Verbindung während einer Transaktion setzen, indem Sie die Anweisung `SET OPTION` verwenden, um die Option `isolation_level` zu ändern. Diese Änderung wirkt sich jedoch nicht auf geöffnete Cursor aus.

Ein Snapshot aller Zeilen, die zur Snapshot-Startzeit festgeschrieben sind, ist sichtbar, wenn die `WITH HOLD`-Klausel mit den Isolationsstufen `snapshot`, `statement-snapshot` und `readonly-statement-snapshot` verwendet wird. Ebenfalls sichtbar sind alle Änderungen, die von der aktuellen Verbindung seit dem Start der Transaktion, innerhalb der der Cursor geöffnet war, abgeschlossen wurden. Weitere Hinweise zu unterstützten Isolationsstufen finden Sie unter „Isolationsstufen und Konsistenz“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)] und „`isolation_level`-Option“ [[SQL Anywhere Server - Datenbankadministration](#)].

.NET-Anwendungsprogrammierung

Dieser Abschnitt beschäftigt sich mit der Verwendung von SQL Anywhere mit .NET, einschließlich der API für den SQL Anywhere-.NET-Datenprovider.

SQL Anywhere .NET-Datenprovider

In diesem Abschnitt wird folgendes beschrieben: die .NET-Unterstützung, einschließlich Tipps zur Verwendung des SQL Anywhere-.NET-Datenproviders in Visual Studio-Projekten, das Herstellen von Verbindungen mit Datenbanken sowie das Einfügen, Aktualisieren und Löschen von Zeilen in/aus Datenbanktabellen, das Aufrufen von gespeicherten Prozeduren, das Arbeiten mit Transaktionen und Grundlagen der Fehlerbehandlung.

SQL Anywhere .NET-Unterstützung

ADO.NET ist die neueste Datenzugriffs-API von Microsoft nach ODBC, OLEDB und ADO. Es ist die bevorzugte Datenzugriffskomponente für das Microsoft .NET Framework und ermöglicht Ihnen den Zugriff auf relationale Datenbanken.

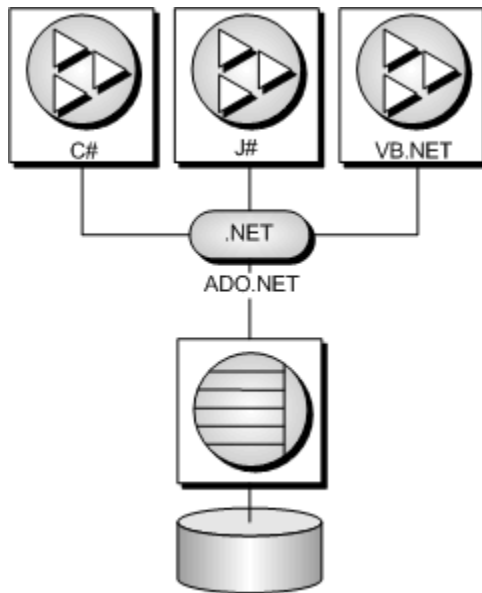
Der SQL Anywhere .NET-Datenprovider implementiert den `iAnywhere.Data.SQLAnywhere-Namespace` und ermöglicht Ihnen die Programmierung von Software in einer der von .NET unterstützten Programmiersprachen wie C# und Visual Basic .NET, um auf Daten in SQL Anywhere zuzugreifen.

Allgemeine Hinweise zum .NET-Datenzugriff finden Sie in der Microsoft-Dokumentation ".NET Data Access Architecture Guide" unter <http://msdn.microsoft.com/de-de/library/Ee817654%28pandp.10%29.aspx>.

ADO.NET-Anwendungen

Sie können Internet- und Intranet-Anwendungen mithilfe von objektorientierten Programmiersprachen entwickeln und diese Anwendungen dann mit dem ADO.Net-Datenprovider über SQL Anywhere verbinden.

Zur Entwicklung von Handheld-Datenbankanwendungen können Sie diesen Provider mit integrierten XML- und Webdienstfunktionen, der .NET-Skripterstellungsfunktionalität für die MobiLink-Synchronisation und einer UltraLite .NET-Komponente kombinieren und SQL Anywhere kann in das .NET Framework integriert werden.



Siehe auch

- [„Funktionen des SQL Anywhere .NET-Datenproviders“ auf Seite 42](#)
- [Namespace auf Seite 105](#)
- [„Praktische Einführungen zum .NET-Datenprovider“ auf Seite 79](#)

Funktionen des SQL Anywhere .NET-Datenproviders

SQL Anywhere unterstützt über drei unterschiedliche Namespaces die Microsoft .NET Framework-Versionen 2.0, 3.0, 3.5, 4.0 und 4.5.

- **iAnywhere.Data.SQLAnywhere** Das ADO.NET-Objektmodell ist ein allgemeines Datenzugriffsmodell. ADO.NET-Komponenten wurden entwickelt, um Datenmanipulation beim Datenzugriff zu verhindern. Dies wird durch zwei zentrale Komponenten von ADO.NET erreicht, nämlich das DataSet und den .NET Framework-Datenprovider, wobei es sich um eine Gruppe von Komponenten mit Connection-, Command-, DataReader- und DataAdapter-Objekten handelt. SQL Anywhere umfasst einen .NET Entity Framework-Datenprovider, der direkt mit einem SQL Anywhere-Datenbankserver kommuniziert, und zwar ohne den Overhead von OLE DB oder ODBC. Der SQL Anywhere-.NET-Datenprovider wird im .NET-Namespace als `iAnywhere.Data.SQLAnywhere` dargestellt.

Microsoft .NET Compact Framework ist das Entwicklungsframework für intelligente Geräte für Microsoft .NET. Der SQL Anywhere .NET Compact Framework-Datenprovider unterstützt Windows Mobile-Geräte. Compact Framework 2.0 und 3.5 werden unterstützt.

In diesem Dokument wird der Namespace für den SQL Anywhere-.NET-Datenprovider beschrieben.

Weitere Hinweise dazu, wie Sie mithilfe des ADO.NET-Objektmodells auf innerhalb einer SQL Anywhere-Datenbank gespeicherte Daten zugreifen können, insbesondere über die LINQ-Methodik

(Language Integrated Query to Entities), finden Sie im Whitepaper *SQL Anywhere and the ADO.NET Entity Framework* unter www.sybase.com/detail?id=1060541.

- **System.Data.OleDb** Dieser Namespace unterstützt OLE DB-Datenquellen. Er ist ein integrierter Bestandteil von Microsoft .NET Framework. Sie können System.Data.OleDb zusammen mit dem SQL Anywhere OLE DB-Provider, SAOLEDB, verwenden, um auf SQL Anywhere-Datenbanken zuzugreifen.
- **System.Data.Odbc** Dieser Namespace unterstützt ODBC-Datenquellen. Er ist ein integrierter Bestandteil von Microsoft .NET Framework. Sie können System.Data.Odbc zusammen mit dem SQL Anywhere ODBC-Treiber verwenden, um auf SQL Anywhere-Datenbanken zuzugreifen.

Für den Einsatz des SQL Anywhere .NET-Datenproviders sprechen einige wesentliche Vorteile:

- In der .NET-Umgebung stellt der SQL Anywhere .NET-Datenprovider nativen Zugriff auf eine SQL Anywhere-Datenbank bereit. Anders als andere unterstützte Provider kommuniziert er direkt mit einem SQL Anywhere Server und benötigt keine Brückentechnologie.
- Dies führt dazu, dass der SQL Anywhere .NET-Datenprovider schneller als der OLE DB- und ODBC-Datenprovider ist. Es wird empfohlen, Datenprovider für den Zugriff auf SQL Anywhere-Datenbanken zu verwenden.

.NET-Beispielprojekte

Mit dem SQL Anywhere .NET-Datenprovider werden mehrere Beispielprojekte bereitgestellt:

- **DeployUtility** Dieses Codebeispiel hilft beim Deployment der nicht verwalteten Codeteile des SQL Anywhere-.NET-Datenproviders in einem ClickOnce-Deployment. Weitere Hinweise finden Sie unter „[ClickOnce- und .NET-Datenprovider-DLLs für nicht verwalteten Code](#)“ auf Seite 963.
- **LinqSample** Ein Beispielprojekt für .NET Framework für Windows, das in die Sprache integrierte (LINQ) Abfrage-, Einstellungs- und Transformationsvorgänge zeigt, bei denen der SQL Anywhere-.NET-Datenprovider und C# verwendet werden.
- **SimpleCE** Ein Beispielprojekt für .NET Compact Framework für Windows Mobile mit einem einfachen Listefeld, in das Namen aus der Tabelle "Employee" eingelesen werden, wenn Sie auf die Schaltfläche **Connect** klicken.
- **SimpleWin32** Ein Beispielprojekt für .NET Framework für Windows, das ein einfaches Listefeld demonstriert, in das Namen aus der Tabelle "Employees" eingelesen werden, wenn Sie auf die Schaltfläche **Connect** klicken. Weitere Hinweise finden Sie unter „[Praktische Einführung: Verwendung des Codebeispiels "Simple"](#)“ auf Seite 79.
- **SimpleXML** Ein Beispielprojekt für .NET Framework für Windows, das veranschaulicht, wie Sie XML-Daten aus SQL Anywhere über ADO.NET erhalten. Beispiele für C#, Visual Basic, und Visual C++ werden bereitgestellt.

- **SimpleViewer** Ein Beispielprojekt für .NET Framework für Windows. Weitere Hinweise finden Sie unter „[Praktische Einführung: Eine einfache .NET-Datenbankanwendung mit Visual Studio entwickeln](#)“ auf Seite 87.
- **TableViewer** Ein Beispielprojekt für .NET Framework für Windows, das es Ihnen gestattet, SQL-Anweisungen einzugeben und auszuführen. Weitere Hinweise finden Sie unter „[Praktische Einführung: Verwendung des Codebeispiels "Table Viewer"](#)“ auf Seite 83.

.NET-Datenprovider in einem Visual Studio-Projekt einsetzen

Verwenden Sie den SQL Anywhere-.NET-Datenprovider, um .NET-Anwendungen mit Visual Studio zu erstellen, indem Sie sowohl eine Referenz auf den SQL Anywhere-.NET-Datenprovider als auch eine Zeile im Quellcode einfügen, die die SQL Anywhere-.NET-Datenprovider-Klassen referenziert.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Starten Sie Visual Studio und öffnen Sie Ihr Projekt.
2. Klicken Sie im **Projektmappen-Explorer**-Fenster mit der rechten Maustaste auf den Ordner **Verweise** und klicken Sie auf **Verweis hinzufügen**.

Die Referenz gibt an, welcher Provider einzubeziehen ist, und zeigt die Position des Codes für den SQL Anywhere-.NET-Datenprovider an.

3. Durchsuchen Sie auf der Registerkarte **.NET** die Liste nach einem der folgenden Einträge:
 - iAnywhere.Data.SQLAnywhere für .NET 2
 - iAnywhere.Data.SQLAnywhere für .NET 3.5
 - iAnywhere.Data.SQLAnywhere für .NET 4

Hinweis

Es gibt eine separate Version des Providers für Windows Mobile-Plattformen. Wenn Sie den SQL Anywhere-.NET-Datenprovider für Windows Mobile benötigen, klicken Sie auf die Registerkarte **Durchsuchen** und suchen Sie nach der Windows Mobile-Version des Providers. Der Standardspeicherort ist `%SQLANYI6%\CE\Assembly\V2`.

4. Klicken Sie auf den gewünschten Provider und anschließend auf **OK**.

Die DLL wird in den Ordner **Verweise** im Fenster **Projektmappen-Explorer** Ihres Projekts eingefügt.

5. Geben Sie eine Direktive zu Ihrem Quellcode an, um die Verwendung des Namespace für den SQL Anywhere-.NET-Datenprovider und der definierten Typen zu unterstützen.

Fügen Sie die folgende Zeile zu Ihrem Projekt hinzu:

- Wenn Sie mit C# arbeiten, fügen Sie die folgende Zeile zur Liste der **using**-Direktiven am Beginn Ihres Quellcodes hinzu:

```
using iAnywhere.Data.SQLAnywhere;
```

- Wenn Sie mit Visual Basic arbeiten, fügen Sie die folgende Zeile am Anfang des Quellcodes hinzu:

```
Imports iAnywhere.Data.SQLAnywhere
```

Ergebnisse

Der SQL Anywhere-.NET-Datenprovider wird für die Verwendung mit Ihrer .NET Anwendung eingerichtet.

Beispiel

Das folgende C#-Beispiel zeigt, wie Sie ein Verbindungsobjekt erstellen, wenn eine **using**-Direktive angegeben wurde:

```
SACConnection conn = new SACConnection();
```

Das folgende C#-Beispiel zeigt, wie Sie ein Verbindungsobjekt erstellen, wenn keine **using**-Direktive angegeben wurde:

```
iAnywhere.Data.SQLAnywhere.SACConnection conn =  
    new iAnywhere.Data.SQLAnywhere.SACConnection();
```

Das folgende Visual Basic-Beispiel zeigt, wie Sie ein Verbindungsobjekt erstellen, wenn eine **Imports**-Direktive angegeben wurde:

```
Dim conn As New SACConnection()
```

Das folgende Visual Basic-Beispiel zeigt, wie Sie ein Verbindungsobjekt erstellen, wenn keine **Imports**-Direktive angegeben wurde:

```
Dim conn As New iAnywhere.Data.SQLAnywhere.SACConnection()
```

Siehe auch

- [„Deployment von SQL Anywhere .NET-Datenprovider“ auf Seite 72](#)

Beispiele für eine .NET-Datenbankverbindung

Um eine Verbindung mit einer Datenbank herzustellen, muss ein SACConnection-Objekt erstellt werden. Sie können die Verbindungszeichenfolge beim Erstellen des Objekts angeben oder später einrichten, indem Sie dieConnectionString-Eigenschaft festlegen.

Eine sauber geschriebene Anwendung sollte Fehler behandeln können, die beim Herstellen der Verbindung mit einer Datenbank auftreten.

Eine Verbindung mit der Datenbank wird erstellt, wenn die Verbindung geöffnet wird, und freigegeben, wenn die Verbindung geschlossen wird.

C#-Beispiel mit SAConnection

Mit dem folgenden C#-Code wird ein Schaltflächen-Klickhandler erstellt, der eine Verbindung mit der SQL Anywhere-Beispieldatenbank öffnet und dann schließt. Eine Ausnahmeroutine ist enthalten.

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=SQL Anywhere 16 Demo");
    try
    {
        conn.Open();

        conn.Close();
    }
    catch (SAException ex)
    {
        MessageBox.Show(ex.Errors[0].Source + " : " +
            ex.Errors[0].Message + " (" +
            ex.Errors[0].NativeError.ToString() + ")",
            "Failed to connect");
    }
}
```

Visual Basic-Beispiel mit SAConnection

Mit dem folgenden Visual Basic-Code wird ein Schaltflächen-Klickhandler erstellt, der eine Verbindung mit der SQL Anywhere-Beispieldatenbank öffnet und dann schließt. Eine Ausnahmeroutine ist enthalten.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=SQL Anywhere 16 Demo")
    Try
        conn.Open()

        conn.Close()
    Catch ex As SAException
        MessageBox.Show(ex.Errors(0).Source & " : " & _
            ex.Errors(0).Message & " (" & _
            ex.Errors(0).NativeError.ToString() & ")", _
            "Failed to connect")
    End Try
End Sub
```

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)
- [SAConnection.ConnectionString-Eigenschaft \[SQL Anywhere .NET\] auf Seite 185](#)
- [SAConnectionStringBuilder-Klasse \[SQL Anywhere .NET\] auf Seite 189](#)
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]

Verbindungspooling

Der SQL Anywhere-.NET-Datenprovider unterstützt natives .NET-Verbindungspooling. Mit Verbindungspools kann Ihre Anwendung bestehende Verbindungen wiederverwenden, indem sie den Verbindungs-Handle in einem Pool speichert und nicht immer wieder eine neue Verbindung zur Datenbank erstellt. Das Verbindungspooling ist standardmäßig aktiviert.

Mit der **Pooling**-Option können Sie das Verbindungspooling aktivieren und deaktivieren. Die maximale Poolgröße wird in Ihrer Verbindungszeichenfolge mit der **Max Pool Size**-Option festgelegt. Die minimale oder anfängliche Poolgröße wird in Ihrer Verbindungszeichenfolge mit der **Min Pool Size**-Option festgelegt. Standardmäßig ist die maximale Poolgröße gleich 100 und die minimale Poolgröße gleich 0.

```
"Data Source=SQL Anywhere 16 Demo;Pooling=true;Max Pool Size=50;Min Pool Size=5"
```

Wenn Ihre Anwendung erstmals versucht, eine Verbindung zu einer Datenbank herzustellen, prüft sie den Pool auf eine bestehende Verbindung, die die von Ihnen angegebenen Verbindungsparameter verwendet. Wenn eine entsprechende Verbindung gefunden wurde, wird sie verwendet. Andernfalls wird eine neue Verbindung eingerichtet. Wenn Sie die Verbindung trennen, wird die Verbindung in den Pool zurückgegeben, damit sie wiederverwendet werden kann.

Der SQL Anywhere-Datenbankserver unterstützt ebenfalls Verbindungspooling. Diese Funktion wird mithilfe des Verbindungsparameters ConnectionPool (CPOOL) gesteuert. Der SQL Anywhere-.NET-Datenprovider verwendet diese Serverfunktion jedoch nicht und deaktiviert sie (CPOOL=NO). Das Verbindungspooling erfolgt stattdessen komplett in der .NET-Clientanwendung (clientseitiges Verbindungspooling).

Verbindungsstatus

Wenn Ihre Anwendung eine Verbindung zu der Datenbank hergestellt hat, können Sie den Verbindungsstatus prüfen, um sicherzugehen, dass die Verbindung offen ist, bevor eine Anforderung an den Datenbankserver abgesetzt wird. Wenn eine Verbindung geschlossen wird, können Sie eine entsprechende Meldung an den Benutzer zurückgeben bzw. versuchen, die Verbindung erneut zu öffnen.

Die SAConnection-Klasse verfügt über eine Eigenschaft namens State, anhand derer Sie den Zustand der Verbindung überprüfen können. Mögliche Zustandswerte sind ConnectionState.Open und ConnectionState.Closed.

Der folgende Code überprüft, ob das Verbindungsobjekt SAConnection initialisiert wurde. Wenn dies der Fall ist, wird geprüft, ob die Verbindung offen ist. Ist die Verbindung nicht offen, erhält der Benutzer eine entsprechende Meldung.

```
if ( conn == null || conn.State != ConnectionState.Open )
{
    MessageBox.Show( "Connect to a database first", "Not connected" );
    return;
}
```

Siehe auch

- [SAConnection.State-Eigenschaft \[SQL Anywhere .NET\] auf Seite 188](#)

Datenzugriff und Datenverarbeitung

Es gibt zwei Möglichkeiten, mit dem SQL Anywhere .NET-Datenprovider auf Daten zuzugreifen:

- **SACommand-Objekt** Das SACommand-Objekt ist die empfohlene Methode für den Zugriff auf Daten und ihre Verarbeitung in .NET.

Das `SACommand`-Objekt ermöglicht Ihnen, SQL-Anweisungen durchzuführen, mit denen Daten direkt aus der Datenbank abgerufen oder dort geändert werden. Mit dem `SACommand`-Objekt können Sie in der Datenbank direkt SQL-Anweisungen ausführen und gespeicherte Prozeduren aufrufen.

Innerhalb eines `SACommand`-Objekts wird das `SADataReader`-Objekt verwendet, um schreibgeschützte Ergebnismengen aus einer Abfrage oder gespeicherten Prozedur zurückzugeben. Das `SADataReader`-Objekt gibt jeweils nur eine Zeile zurück. Dies hat jedoch keine negative Auswirkung auf die Performance, da die clientseitigen SQL Anywhere-Bibliotheken Prefetch-Puffer verwenden, um gleichzeitig mehrere Zeilen im Voraus einzulesen.

Mithilfe des `SACommand`-Objekts können Sie Ihre Änderungen in Transaktionen zusammenfassen, anstatt den Autocommit-Modus zu verwenden. Wenn Sie das `SATransaction`-Objekt verwenden, werden Sperren auf die Zeilen gesetzt, damit andere Benutzer sie nicht ändern können.

Weitere Hinweise finden Sie unter [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#) und [SADataReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#).

- **SDataAdapter-Objekt** Das `SDataAdapter`-Objekt ruft die komplette Ergebnismenge in das `DataSet`-Objekt ab. Ein `DataSet`-Objekt ist ein Offline-Speicherort für aus der Datenbank abgerufene Daten. Sie können die Daten zuerst im `DataSet`-Objekt bearbeiten. Anschließend aktualisiert das `SDataAdapter`-Objekt die Datenbank mit den geänderten Daten aus dem `DataSet`-Objekt. Wenn Sie das `SDataAdapter`-Objekt verwenden, können auch andere Benutzer die Daten in Ihrem `DataSet`-Objekt ändern. Um daraus eventuell auftretende Konflikte zu lösen, müssen Sie eine entsprechende Logik in Ihre Anwendung einfügen.

Weitere Hinweise zu Konflikten finden Sie im Abschnitt [Konflikte beim Einsatz des SDataAdapter-Objekts lösen auf Seite 54](#).

Weitere Hinweise zum `SDataAdapter`-Objekt finden Sie im Abschnitt [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#).

Wenn Sie anstelle des `SDataAdapter`-Objekts das `SADataReader`-Objekt mit einem `SACommand`-Objekt verwenden, um Zeilen aus der Datenbank abzurufen, ergeben sich keinerlei negative Auswirkungen auf die Performance.

SACommand: Daten mit `ExecuteReader` und `ExecuteScalar` abrufen

Mit dem `SACommand`-Objekt können Sie in einer SQL Anywhere-Datenbank eine SQL-Anweisung ausführen oder eine gespeicherte Prozedur aufrufen. Sie können eine der folgenden Methoden verwenden, um Daten aus der Datenbank abzurufen:

- **ExecuteReader** Führt eine SQL-Abfrage aus, die eine Ergebnismenge zurückgibt. Diese Methode verwendet einen schreibgeschützten Cursor, der Daten nur für die Weitergabe ausliest. Sie können die Zeilen in der Ergebnismenge in einer Richtung durchsuchen.
- **ExecuteScalar** Führt eine SQL-Abfrage aus, die einen einzelnen Wert zurückgibt. Dabei kann es sich um die erste Spalte in der ersten Zeile der Ergebnismenge oder um eine SQL-Anweisung handeln, die einen Aggregatwert wie `COUNT` oder `AVG` zurückgibt. Diese Methode verwendet einen schreibgeschützten Cursor, der Daten nur für die Weitergabe ausliest.

Wenn Sie das `SACCommand`-Objekt verwenden, können Sie mithilfe des `SADataReader`-Objekts eine Ergebnismenge abrufen, die auf einem Join basiert. Änderungen (Einfügen, Aktualisieren oder Löschen) lassen sich jedoch nur an Daten vornehmen, die aus einer einzelnen Tabelle stammen. Sie können keine Ergebnismengen aktualisieren, die auf Joins basieren.

Wenn Sie das `SADataReader`-Objekt verwenden, sind mehrere Get-Methoden verfügbar, mit denen Sie die Ergebnisse im angegebenen Datentyp zurückgeben können.

C#-Beispiel mit `ExecuteReader`

Der folgende C#-Code öffnet eine Verbindung mit der SQL Anywhere-Beispieldatenbank und erstellt mithilfe der `ExecuteReader`-Methode eine Ergebnismenge, die die Nachnamen der Mitarbeiter in der Tabelle "Employees" enthält:

```
SACConnection conn = new SACConnection("Data Source=SQL Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand("SELECT Surname FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
while (reader.Read())
{
    listEmployees.Items.Add(reader.GetString(0));
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

Visual Basic-Beispiel mit `DataReader`

Der folgende Visual Basic-Code öffnet eine Verbindung mit der SQL Anywhere-Beispieldatenbank und erstellt mithilfe der `ExecuteReader`-Methode eine Ergebnismenge, die die Nachnamen der Mitarbeiter in der Tabelle "Employees" enthält:

```
Dim conn As New SACConnection("Data Source=SQL Anywhere 16 Demo")
Dim cmd As New SACCommand("SELECT Surname FROM Employees", conn)
Dim reader As SADataReader
conn.Open()
reader = cmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (reader.Read())
    ListEmployees.Items.Add(reader.GetString(0))
Loop
ListEmployees.EndUpdate()
conn.Close()
```

C#-Beispiel mit `ExecuteScalar`

Der folgende C#-Code öffnet eine Verbindung mit der SQL Anywhere-Beispieldatenbank und verwendet die `ExecuteScalar`-Methode, um die Anzahl der männlichen Mitarbeiter in der Tabelle "Employees" abzurufen:

```
SACConnection conn = new SACConnection("Data Source=SQL Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'", conn );
int count = (int) cmd.ExecuteScalar();
textBox1.Text = count.ToString();
conn.Close();
```

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)
- [SADataReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)
- [SACommand.ExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 145](#)
- [SACommand.ExecuteScalar-Methode \[SQL Anywhere .NET\] auf Seite 147](#)

SACommand: Ergebnismengenschema mit GetSchemaTable abrufen

Sie haben die Möglichkeit, Schemainformationen zu Spalten in einer Ergebnismenge abzurufen.

Die GetSchemaTable-Methode der SADataReader-Klasse ruft Informationen zur aktuellen Ergebnismenge ab. Die GetSchemaTable-Methode gibt das .NET-Standardobjekt DataTable zurück, das Informationen zu allen Spalten in der Ergebnismenge enthält, einschließlich Spalteneigenschaften.

C#-Beispiel mit Schemainformationen

Im folgenden Beispiel werden Informationen zu einer Ergebnismenge mit der GetSchemaTable-Methode abgerufen und das DataTable-Objekt wird an das Datenraster auf dem Bildschirm gebunden.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT * FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
DataTable schema = reader.GetSchemaTable();
reader.Close();
conn.Close();
dataGridView1.DataSource = schema;
```

Siehe auch

- [SADataReader.GetSchemaTable-Methode \[SQL Anywhere .NET\] auf Seite 239](#)

SACommand: Zeilen mit ExecuteNonQuery einfügen, löschen und aktualisieren

Um einen Einfüge-, Aktualisierungs- oder Löschvorgang mit dem SACommand-Objekt auszuführen, verwenden Sie die ExecuteNonQuery-Funktion. Die ExecuteNonQuery-Funktion setzt eine Abfrage (SQL-Anweisung oder gespeicherte Prozedur) ab, die keine Ergebnismenge zurückgibt.

Änderungen (Einfügen, Aktualisieren oder Löschen) sind jedoch nur an Daten möglich, die aus einer einzelnen Tabelle stammen. Sie können keine Ergebnismengen aktualisieren, die auf Joins basieren. Sie müssen mit einer Datenbank verbunden sein, um das SACommand-Objekt verwenden zu können.

Weitere Hinweise zum Abrufen von Primärschlüsselwerten für AUTOINCREMENT-Primärschlüssel finden Sie unter „[SACommand: Primärschlüsselwerte für neu eingefügte Zeilen abrufen](#)“ auf Seite 52.

Wenn Sie die Isolationsstufe für eine SQL-Anweisung festlegen möchten, müssen Sie das SACommand-Objekt als Teil eines SATransaction-Objekts verwenden. Wenn Sie Daten ohne SATransaction-Objekt

ändern, arbeitet der .NET-Datenprovider im Autocommit-Modus, sodass alle Änderungen sofort übernommen werden. Siehe „[Transaktionsverarbeitung](#)“ auf Seite 64.

C#-Beispiel mit ExecuteNonQuery DELETE und INSERT

Im folgenden Beispiel wird eine Verbindung mit der SQL Anywhere-Beispieldatenbank geöffnet und die ExecuteNonQuery-Methode verwendet, um alle Abteilungen zu entfernen, deren ID größer oder gleich 600 ist, und anschließend zwei neue Zeilen in die Tabelle "Departments" einzufügen. Die aktualisierte Tabelle wird in einem Datenraster angezeigt.

```
SAConnection conn = new SAConnection("Data Source=SQL Anywhere 16 Demo");
conn.Open();

SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600",
    conn);
deleteCmd.ExecuteNonQuery();

SACommand insertCmd = new SACommand(
    "INSERT INTO Departments(DepartmentID, DepartmentName) VALUES( ?, ? )",
    conn );
SAParameter parm = new SAParameter();
parm.SADBType = SADBType.Integer;
insertCmd.Parameters.Add( parm );
parm = new SAParameter();
parm.SADBType = SADBType.Char;
insertCmd.Parameters.Add( parm );

insertCmd.Parameters[0].Value = 600;
insertCmd.Parameters[1].Value = "Eastern Sales";
int recordsAffected = insertCmd.ExecuteNonQuery();

insertCmd.Parameters[0].Value = 700;
insertCmd.Parameters[1].Value = "Western Sales";
recordsAffected = insertCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

C#-Beispiel mit ExecuteNonQuery UPDATE

Im folgenden Beispiel wird eine Verbindung mit der SQL Anywhere-Beispieldatenbank geöffnet und die ExecuteNonQuery-Methode verwendet, um die DepartmentName-Spalte in allen Zeilen der Tabelle "Departments", deren DepartmentID-Wert 100 ist, auf "Engineering" zu aktualisieren. Die aktualisierte Tabelle wird in einem Datenraster angezeigt.

```
SAConnection conn = new SAConnection("Data Source=SQL Anywhere 16 Demo");
conn.Open();
```

```
SACommand updateCmd = new SACommand(
    "UPDATE Departments SET DepartmentName = 'Engineering' " +
    "WHERE DepartmentID = 100", conn );
int recordsAffected = updateCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)
- [SACommand.ExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 144](#)
- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)

SACommand: Primärschlüsselwerte für neu eingefügte Zeilen abrufen

Wenn die von Ihnen aktualisierte Tabelle einen autoinkrementierenden Primärschlüssel hat oder UUIDs verwendet bzw. der Primärschlüssel aus einem Primärschlüsselpool bezogen wird, können Sie mithilfe einer gespeicherten Prozedur die von der Datenquelle generierten Primärschlüsselwerte abrufen.

C#-Beispiel mit SACommand für Primärschlüssel

Das folgende Beispiel zeigt, wie der Primärschlüssel abgefragt werden kann, der für eine neu eingefügte Zeile erstellt wird. Das Beispiel verwendet ein SACommand-Objekt, um eine in SQL geschriebene gespeicherte Prozedur und ein SAParameter-Objekt aufzurufen, das den Primärschlüssel abruft, der von ihm zurückgegeben wird. Für Demonstrationszwecke erstellt das Beispiel eine Beispieltabelle (adodotnet_primarykey) und die gespeicherte Prozedur (sp_adodotnet_primarykey), die verwendet wird, um Zeilen einzufügen und Primärschlüsselwerte zurückzugeben.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ( " +
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE sp_adodotnet_primarykey(" +
```

```

        "out p_id int, in p_name char(40) )" +
        "BEGIN " +
        "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
        "SELECT @@IDENTITY INTO p_id;" +
        "END";
cmd.ExecuteNonQuery();

cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;

SAPParameter parmId = new SAPParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add(parmId);

SAPParameter parmName = new SAPParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add(parmName);

parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id1);

parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id2);

parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id3);

parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id4);

cmd.CommandText = "SELECT * FROM adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
SADataReader dr = cmd.ExecuteReader();
conn.Close();
dataGridView1.DataSource = dr;

```

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)
- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)

SADDataAdapter: Übersicht

Das SADDataAdapter-Objekt ruft eine Ergebnismenge in ein DataTable-Objekt ab. Ein DataSet-Objekt ist eine Sammlung von Tabellen (DataTable-Objekten) sowie den Beziehungen und Integritätsregeln zwischen diesen Tabellen. Das DataSet-Objekt wird in das .NET Framework integriert und ist unabhängig von dem Datenprovider, der für die Verbindung mit Ihrer Datenbank verwendet wird.

Wenn Sie das `SDataAdapter`-Objekt verwenden, müssen Sie mit der Datenbank verbunden sein, um ein `DataTable`-Objekt zu füllen und die Datenbank mit Änderungen zu aktualisieren, die am `DataTable`-Objekt vorgenommen wurden. Sobald das `DataTable`-Objekt mit Daten gefüllt ist, können Sie es jedoch auch ohne bestehende Datenbankverbindung ändern.

Wenn Sie Ihre Änderungen nicht sofort in die Datenbank übernehmen möchten, können Sie das `DataSet`-Objekt einschließlich Daten bzw. Schema mithilfe der `WriteXML`-Methode in eine XML-Datei schreiben. Dann können Sie die Änderungen zu einem späteren Zeitpunkt übernehmen, indem Sie das `DataSet`-Objekt mit der `ReadXML`-Methode laden. Im Folgenden finden Sie zwei Beispiele.

```
ds.WriteXml("Employees.xml");  
ds.WriteXml("EmployeesWithSchema.xml", XmlWriteMode.WriteSchema);
```

Weitere Hinweise finden Sie in der Dokumentation zum .NET Framework zu `WriteXml` und `ReadXml`.

Wenn Sie die `Update`-Methode aufrufen, um Änderungen aus dem `DataSet`-Objekt in die Datenbank zu übernehmen, analysiert das `SDataAdapter`-Objekt die durchgeführten Änderungen und ruft dann die geeigneten Anweisungen (`INSERT`, `UPDATE` oder `DELETE`) auf. Wenn Sie das `DataSet`-Objekt verwenden, können Sie Änderungen (Einfügen, Aktualisieren oder Löschen) nur an Daten vornehmen, die aus einer einzelnen Tabelle stammen. Sie können keine Ergebnismengen aktualisieren, die auf Joins basieren. Wenn ein anderer Benutzer die Zeile, die Sie aktualisieren möchten, gesperrt hat, wird eine Ausnahmebedingung generiert.

Vorsicht

Alle Änderungen, die Sie im `DataSet`-Objekt vornehmen, werden offline durchgeführt. Ihre Anwendung setzt keine Sperren für diese Zeilen in der Datenbank. Ihre Anwendung muss so konzipiert sein, dass sie Konflikte löst, die eventuell auftreten, wenn die am `DataSet`-Objekt vorgenommenen Änderungen in die Datenbank übertragen werden und ein anderer Benutzer diese Daten inzwischen in der Datenbank verändert hat.

Konflikte beim Einsatz des `SDataAdapter`-Objekts lösen

Wenn Sie das `SDataAdapter`-Objekt verwenden, werden die Zeilen in der Datenbank nicht mit Sperren versehen. Daher könnten Konflikte auftreten, wenn Sie Änderungen aus dem `DataSet`-Objekt in der Datenbank übernehmen. Ihre Anwendung muss daher Logik enthalten, die auftretende Konflikte löst oder protokolliert.

Konflikte, die von Ihrer Anwendungslogik verarbeitet werden müssen, sind u.a.:

- **Eindeutige Primärschlüssel** Wenn zwei Benutzer neue Zeilen in eine Tabelle einfügen, muss jede Zeile einen eindeutigen Primärschlüssel haben. Bei Tabellen mit automatisch inkrementierten Primärschlüsseln können die Werte im `DataSet`-Objekt die Synchronität mit den Werten in der Datenquelle verlieren.

Hinweise zum Abruf von Primärschlüsselwerten für automatisch inkrementierte Primärschlüssel finden Sie im Abschnitt „[SDataAdapter: Primärschlüsselwerte für neu eingefügte Zeilen abrufen](#)“ auf Seite 60.

- **Aktualisierungen, die an demselben Wert vorgenommen werden** Wenn zwei Benutzer denselben Wert ändern, muss die Logik Ihrer Anwendung ermitteln können, welcher Wert richtig ist.

- **Schemaänderungen** Wenn ein Benutzer das Schema einer Tabelle verändert, die Sie im DataSet-Objekt aktualisiert haben, schlägt die Aktualisierung bei der Übernahme der Änderungen in die Datenbank fehl.
- **Parallelität von Daten** Gleichzeitig laufende Anwendungen müssen einen einheitlichen Datenbestand anzeigen. Das SDataAdapter-Objekt setzt keine Sperren auf Zeilen, die es abrufen. Daher kann ein anderer Benutzer einen Wert in der Datenbank aktualisieren, nachdem Sie Daten aus dem DataSet-Objekt abgerufen haben und offline daran arbeiten.

Viele dieser potenziellen Probleme können vermieden werden, indem Sie Änderungen mithilfe der Objekte SACommand, SDataReader und SATransaction in die Datenbank übernehmen. Dabei wird die Verwendung des SATransaction-Objekts empfohlen, da Sie damit die Isolationsstufe für die Transaktion festlegen können und Sperren auf abgerufene Zeilen gesetzt werden, damit andere Benutzer diese nicht verändern können.

Weitere Hinweise zur Verwendung von Transaktionen für die Übernahme Ihrer Änderungen in die Datenbank finden Sie im Abschnitt „[SACommand: Zeilen mit ExecuteNonQuery einfügen, löschen und aktualisieren](#)“ auf Seite 50.

Um den Prozess der Konfliktlösung zu vereinfachen, können Sie Ihre INSERT-, UPDATE- oder DELETE-Anweisung so schreiben, dass sie als Aufruf einer gespeicherten Prozedur durchgeführt wird. Wenn Sie INSERT-, UPDATE- und DELETE-Anweisungen in gespeicherte Prozeduren einschließen, können Sie eventuell auftretende Fehler abfangen. Zusätzlich zur Anweisung können Sie Logik für die Fehlerbehandlung in die gespeicherte Prozedur einfügen, damit geeignete Maßnahmen getroffen werden, wenn der Vorgang fehlschlägt, z.B. indem die Fehlermeldung in ein Log eingetragen oder der Vorgang erneut versucht wird.

SDataAdapter: Daten mit Fill in ein DataTable-Objekt abrufen

Mit dem SDataAdapter-Objekt können Sie eine Ergebnismenge anzeigen. Verwenden Sie dabei die Fill-Methode, um ein DataTable-Objekt mit den Ergebnissen aus einer Abfrage zu füllen und anschließend an ein Anzeigeraster zu binden.

Beim Einrichten eines SDataAdapter-Objekts können Sie eine SQL-Anweisung angeben, die eine Ergebnismenge zurückgibt. Wenn Fill aufgerufen wird, um ein DataTable-Objekt zu füllen, werden alle Zeilen in einem Vorgang mit einem schreibgeschützten Vorwärtscursor abgerufen. Sobald alle Zeilen in der Ergebnismenge gelesen wurden, wird der Cursor geschlossen. Änderungen an den Zeilen in einem DataTable-Objekt können mit der Update-Methode in die Datenbank übernommen werden.

Mithilfe des SDataAdapter-Objekts können Sie eine Ergebnismenge abrufen, die auf einem Join basiert. Änderungen (Einfügen, Aktualisieren oder Löschen) lassen sich jedoch nur an Daten vornehmen, die aus einer einzelnen Tabelle stammen. Sie können keine Ergebnismengen aktualisieren, die auf Joins basieren.

Vorsicht

Änderungen, die Sie an einem DataTable-Objekt vornehmen, werden unabhängig von der ursprünglichen Datenbanktabelle durchgeführt. Ihre Anwendung setzt keine Sperren für diese Zeilen in der Datenbank. Ihre Anwendung muss so entwickelt werden, dass sie Konflikte löst, die möglicherweise auftreten, wenn die am DataTable-Objekt vorgenommenen Änderungen in die Datenbank übertragen werden und ein anderer Benutzer diese Daten inzwischen in der Datenbank geändert hat.

C#-Beispiel für SDataAdapter mit Fill-Methode und einem DataTable-Objekt

Das folgende Beispiel zeigt, wie ein DataTable-Objekt mithilfe des SDataAdapter-Objekts mit Daten gefüllt wird. Dabei werden ein neues DataTable-Objekt namens Results und ein neues SDataAdapter-Objekt erstellt. Die Fill-Methode des SDataAdapter-Objekts wird verwendet, um das DataTable-Objekt mit den Ergebnissen der Abfrage zu füllen. Anschließend wird das DataTable-Objekt an das Raster auf dem Bildschirm gebunden.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
DataTable dt = new DataTable("Results");
SDataAdapter da = new SDataAdapter("SELECT * FROM Employees", conn);
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

C#-Beispiel für SDataAdapter mit Fill-Methode und einem DataSet-Objekt

Das folgende Beispiel zeigt, wie ein DataSet-Objekt mithilfe des SDataAdapter-Objekts mit Daten gefüllt wird. Dabei werden ein neues DataSet-Objekt und ein neues SDataAdapter-Objekt erstellt. Die Fill-Methode des SDataAdapter-Objekts wird verwendet, um im DataSet-Objekt eine DataTable-Tabelle namens Results zu erstellen und anschließend mit den Ergebnissen der Abfrage zu füllen. Anschließend wird das DataSet-Objekt "Results" an das Raster auf dem Bildschirm gebunden.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
DataSet ds = new DataSet();
SDataAdapter da = new SDataAdapter("SELECT * FROM Employees", conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

Siehe auch

- [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)

SDataAdapter: DataTable-Objekte mit FillSchema formatieren

Das SDataAdapter-Objekt ermöglicht es Ihnen, mithilfe der FillSchema-Methode das Schema eines DataTable-Objekts so zu konfigurieren, dass es zu demjenigen einer bestimmten Abfrage passt. Die Attribute der Spalten im DataTable-Objekt stimmen mit denjenigen in der SelectCommand-Methode des SDataAdapter-Objekts überein. Anders als bei der Fill-Methode werden keine Zeilen im DataTable-Objekt gespeichert.

C#-Beispiel für SDataAdapter mit FillSchema-Methode und einem DataTable-Objekt

Das folgende Beispiel zeigt, wie Sie mithilfe der FillSchema-Methode ein neues DataTable-Objekt mit demselben Schema wie eine Ergebnismenge einrichten. Anschließend wird das DataTable-Objekt "Additions" an das Raster auf dem Bildschirm gebunden.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SDataAdapter da = new SDataAdapter("SELECT * FROM Employees", conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = dt;
```

C#-Beispiel für SDataAdapter mit FillSchema-Methode und einem DataSet-Objekt

Das folgende Beispiel zeigt, wie Sie mithilfe der FillSchema-Methode ein neues DataTable-Objekt mit demselben Schema wie eine Ergebnismenge einrichten. Das DataTable-Objekt wird mithilfe der Merge-Methode zum DataSet-Objekt hinzugefügt. Anschließend wird das DataTable-Objekt "Additions" an das Raster auf dem Bildschirm gebunden.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SDataAdapter da = new SDataAdapter("SELECT * FROM Employees", conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
DataSet ds = new DataSet();
ds.Merge(dt);
conn.Close();
dataGridView1.DataSource = ds.Tables["Additions"];
```

Siehe auch

- [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)

SDataAdapter: Zeilen mit Update einfügen**C#-Beispiel für SDataAdapter mit Insert**

Das folgende Beispiel zeigt, wie Sie die Update-Methode von SDataAdapter verwenden können, um Zeilen zu einer Tabelle hinzuzufügen. Im Beispiel wird die Tabelle "Departments" mithilfe der SelectCommand-Eigenschaft und der Fill-Methode von SDataAdapter in ein DataTable-Objekt abgerufen. Anschließend werden zwei neue Zeilen zum DataTable-Objekt hinzugefügt und die Tabelle "Departments" wird mithilfe der InsertCommand-Eigenschaft und der Update-Methode von SDataAdapter aus dem DataTable-Objekt aktualisiert.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.InsertCommand = new SACommand(
```

```
"INSERT INTO Departments( DepartmentID, DepartmentName ) " +
"VALUES( ?, ? )", conn );
da.InsertCommand.UpdatedRowSource =
    UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADBType = SADBType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADBType = SADBType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

DataRow row1 = dataTable.NewRow();
row1[0] = 600;
row1[1] = "Eastern Sales";
dataTable.Rows.Add( row1 );

DataRow row2 = dataTable.NewRow();
row2[0] = 700;
row2[1] = "Western Sales";
dataTable.Rows.Add( row2 );

rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

Siehe auch

- [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)

SDataAdapter: Zeilen mit Update löschen

C#-Beispiel für SDataAdapter mit Delete

Das folgende Beispiel zeigt, wie Sie die Update-Methode von SDataAdapter verwenden können, um Zeilen aus einer Tabelle zu löschen. Im Beispiel werden zwei neue Zeilen zur Tabelle Departments hinzugefügt und dann wird diese Tabelle mithilfe der SelectCommand-Eigenschaft und der Fill-Methode von SDataAdapter in ein DataTable-Objekt abgerufen. Anschließend werden einige Zeilen aus dem DataTable-Objekt gelöscht und die Tabelle "Departments" wird mithilfe der DeleteCommand-Eigenschaft und der Update-Methode von SDataAdapter aus dem DataTable-Objekt aktualisiert.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
```

```

prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn);
da.DeleteCommand = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID = ?",
    conn);
da.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADBType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add(parm);

DataTable dataTable = new DataTable("Departments");
int rowCount = da.Fill(dataTable);

foreach (DataRow row in dataTable.Rows)
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row.Delete();
    }
}
rowCount = da.Update(dataTable);

dataTable.Clear();
rowCount = da.Fill(dataTable);
conn.Close();
dataGridView1.DataSource = dataTable;

```

Siehe auch

- [SADataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)

SADataAdapter: Zeilen mit Update aktualisieren**C#-Beispiel für SADataAdapter mit Update**

Das folgende Beispiel zeigt, wie Sie die Update-Methode von SADataAdapter verwenden können, um Zeilen in einer Tabelle zu aktualisieren. Im Beispiel werden zwei neue Zeilen zur Tabelle Departments hinzugefügt und dann wird diese Tabelle mithilfe der SelectCommand-Eigenschaft und der Fill-Methode von SADataAdapter in ein DataTable-Objekt abgerufen. Anschließend werden einige Werte im DataTable-Objekt geändert und die Tabelle "Departments" wird mithilfe der UpdateCommand-Eigenschaft und der Update-Methode von SADataAdapter aus dem DataTable-Objekt aktualisiert.

```

SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =

```

```
"INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
"INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.UpdateCommand = new SACommand(
    "UPDATE Departments SET DepartmentName = ? " +
    "WHERE DepartmentID = ?",
    conn );
da.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADBType = SADBType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.UpdateCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADBType = SADBType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.UpdateCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

foreach ( DataRow row in dataTable.Rows )
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row[1] = (string)row[1] + "_Updated";
    }
}
rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

Siehe auch

- [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)

SDataAdapter: Primärschlüsselwerte für neu eingefügte Zeilen abrufen

Wenn die von Ihnen aktualisierte Tabelle einen autoinkrementierenden Primärschlüssel hat oder UUIDs verwendet bzw. der Primärschlüssel aus einem Primärschlüsselpool bezogen wird, können Sie mithilfe einer gespeicherten Prozedur die von der Datenquelle generierten Primärschlüsselwerte abrufen.

C#- Beispiel: SDataAdapter mit Primärschlüssel

Das folgende Beispiel zeigt, wie Sie den Primärschlüssel abrufen können, der für eine neu eingefügte Zeile erstellt wird. Im Beispiel wird ein SDataAdapter-Objekt verwendet, um eine gespeicherte SQL-Prozedur aufzurufen und ein SAParameter-Objekt, das den von dieser zurückzugebenden Primärschlüssel abrufen. Für Demonstrationszwecke wird im Beispiel eine Beispieltabelle (adodotnet_primarykey) erstellt sowie die gespeicherte Prozedur (sp_adodotnet_primarykey), mit der Zeilen eingefügt und Primärschlüsselwerte zurückgegeben werden sollen.

```

SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();

SACCommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ( " +
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE sp_adodotnet_primarykey( " +
    "out p_id int, in p_name char(40) ) " +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

da.SelectCommand = new SACCommand(
    "SELECT * FROM adodotnet_primarykey", conn);

da.InsertCommand = new SACCommand(
    "sp_adodotnet_primarykey", conn);
da.InsertCommand.CommandType = CommandType.StoredProcedure;
da.InsertCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;

SAParameter parmId = new SAParameter();
parmId.SADBType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "ID";
parmId.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADBType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "Name";
parmName.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmName);

DataTable dataTable = new DataTable("Departments");
da.FillSchema(dataTable, SchemaType.Source);

DataRow row = dataTable.NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";

```

```
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataTable.Rows.Add(row);

DataSet ds = new DataSet();
ds.Merge(dataTable);
da.Update(ds, "Departments");

conn.Close();
dataGridView1.DataSource = ds.Tables["Departments"];
```

Siehe auch

- [SADDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)
- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)

BLOBs

Bei einem Fetch langer Zeichenfolgenwerte oder von Binärdaten stehen Methoden bereit, mit den Sie Daten in Teilen abrufen können. Verwenden Sie bei Binärdaten die GetBytes-Methode, bei Zeichenfolgendaten die GetChars-Methode. Sonst werden BLOB-Daten in derselben Art verarbeitet wie andere Daten, die Sie aus der Datenbank abrufen.

C#-Beispiel: GetChars BLOB

Im folgenden Beispiel werden drei Spalten aus einer Ergebnismenge gelesen. Die ersten beiden Spalten enthalten Ganzzahlen, während die dritte vom Typ LONG VARCHAR ist. Die Länge der dritten Spalte wird berechnet, indem diese Spalte mit der GetChars-Methode in Abschnitten von 100 Zeichen gelesen wird.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SACCommand cmd = new SACCommand("SELECT * FROM MarketingInformation", conn);
SADataReader reader = cmd.ExecuteReader();

int idValue;
int productIdValue;
int length = 100;
char[] buf = new char[length];
while (reader.Read())
{
    idValue = reader.GetInt32(0);
    productIdValue = reader.GetInt32(1);
    long blobLength = 0;
    long charsRead;
    while ((charsRead = reader.GetChars(2, blobLength, buf, 0, length))
```

```
        == (long)length)
    {
        blobLength += charsRead;
    }
    blobLength += charsRead;
}
reader.Close();
conn.Close();
```

Siehe auch

- [SADataReader.GetBytes-Methode \[SQL Anywhere .NET\] auf Seite 228](#)
- [SADataReader.GetChars-Methode \[SQL Anywhere .NET\] auf Seite 230](#)

Zeitwerte

Das .NET-Framework hat keine "Time"-Struktur (Zeitstruktur). Wenn Sie Zeitwerte aus SQL Anywhere abrufen möchten, müssen Sie die GetTimeSpan-Methode verwenden. Diese Methode gibt die Daten als TimeSpan-Objekt für .NET Framework zurück.

C#-Beispiel: TimeSpan

Im folgenden Beispiel wird die GetTimeSpan-Methode verwendet, um den Zeitwert als TimeSpan-Objekt zurückzugeben.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
SACCommand cmd = new SACCommand("SELECT 123, CURRENT TIME", conn);
SADataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    int ID = reader.GetInt32(0);
    TimeSpan time = reader.GetTimeSpan(1);
}
reader.Close();
conn.Close();
```

Siehe auch

- [SADataReader.GetTimeSpan-Methode \[SQL Anywhere .NET\] auf Seite 241](#)

Gespeicherte Prozeduren

Sie können gespeicherte SQL-Prozeduren mit dem SQL Anywhere-.NET-Datenprovider verwenden.

Die ExecuteReader-Methode wird verwendet, um gespeicherte Prozeduren aufzurufen, die Ergebnismengen zurückgeben, während die ExecuteNonQuery-Methode verwendet wird, um gespeicherte Prozeduren aufzurufen, die keine Ergebnismengen zurückgeben. Die ExecuteScalar-Methode wird verwendet, um gespeicherte Prozeduren aufzurufen, die nur einen einzelnen Wert zurückgeben.

Sie können SAParameter-Objekte verwenden, um Parameter an eine gespeicherte Prozedur zu übergeben.

C#-Beispiel: Aufruf einer gespeicherten Prozedur mit Parametern

Das folgende Beispiel zeigt zwei Möglichkeiten, eine gespeicherte Prozedur aufzurufen und einen Parameter an sie zu übergeben. Im Beispiel wird ein `SADataReader`-Objekt verwendet, um die von der gespeicherten Prozedur zurückgegebene Ergebnismenge abzurufen.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
bool method1 = true;

SACCommand cmd = new SACCommand("", conn);
if (method1)
{
    cmd.CommandText = "ShowProductInfo";
    cmd.CommandType = CommandType.StoredProcedure;
}
else
{
    cmd.CommandText = "call ShowProductInfo(?)";
    cmd.CommandType = CommandType.Text;
}

SAParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Integer;
param.Direction = ParameterDirection.Input;
param.Value = 301;
cmd.Parameters.Add(param);

SADataReader reader = cmd.ExecuteReader();
reader.Read();
int ID = reader.GetInt32(0);
string name = reader.GetString(1);
string description = reader.GetString(2);
decimal price = reader.GetDecimal(6);
reader.Close();

listBox1.BeginUpdate();
listBox1.Items.Add("Name=" + name +
    " Description=" + description + " Price=" + price);
listBox1.EndUpdate();

conn.Close();
```

Siehe auch

- [SACCommand.ExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 145](#)
- [SACCommand.ExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 144](#)
- [SACCommand.ExecuteScalar-Methode \[SQL Anywhere .NET\] auf Seite 147](#)
- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [„SACCommand: Zeilen mit ExecuteNonQuery einfügen, löschen und aktualisieren“ auf Seite 50](#)
- [„SACCommand: Daten mit ExecuteReader und ExecuteScalar abrufen“ auf Seite 48](#)

Transaktionsverarbeitung

Mit dem SQL Anywhere .NET-Datenprovider können Sie das `SATransaction`-Objekt verwenden, um Anweisungen zu gruppieren. Jede Transaktion endet mit `COMMIT` oder `ROLLBACK`, sodass Ihre Änderungen in der Datenbank festgeschrieben oder alle Vorgänge in der Transaktion zurückgesetzt werden. Wenn die Transaktion abgeschlossen ist, müssen Sie ein neues `SATransaction`-Objekt erstellen,

um weitere Änderungen durchzuführen. Dieses Verhalten ist anders als bei ODBC und Embedded SQL, wo eine Transaktion, nachdem Sie eine COMMIT- oder ROLLBACK-Anweisung ausgeführt haben, bestehen bleibt, bis die Transaktion geschlossen wird.

Wenn Sie keine Transaktion erstellen, arbeitet der SQL Anywhere .NET-Datenprovider standardmäßig im Autocommit-Modus. Nach jeder Insert-, Update- oder Delete-Anweisung wird ein implizites COMMIT gesetzt, und wenn ein Vorgang abgeschlossen ist, wird die Änderung in der Datenbank durchgeführt. In diesem Fall können die Änderungen nicht zurückgesetzt werden.

Einstellungen der Isolationsstufe für Transaktionen

Die Datenbank-Isolationsstufe wird standardmäßig für Transaktionen verwendet. Sie können die Isolationsstufe für eine Transaktion mit der IsolationLevel-Eigenschaft definieren, wenn Sie die Transaktion beginnen. Die Isolationsstufe gilt für alle Anweisungen, die innerhalb der Transaktion ausgeführt werden. Der SQL Anywhere .NET-Datenprovider unterstützt die Snapshot-Isolation.

Welche Sperren SQL Anywhere verwendet, wenn Sie eine SQL-Anweisung ausführen, hängt von der Isolationsstufe der Transaktion ab.

Verteilte Transaktionsverarbeitung

In .NET Framework 2.0 wurde der neue Namespace System.Transactions eingeführt, der Klassen zur Erstellung transaktionaler Anwendungen enthält. Clientanwendungen können verteilte Transaktionen mit einem oder mehreren Teilnehmern erstellen und an diesen beteiligt sein. Clientanwendungen können mit der Klasse TransactionScope implizit Transaktionen erstellen. Das Verbindungsobjekt kann feststellen, ob eine von TransactionScope erstellte Ambient-Transaktion vorhanden ist und diese automatisch eintragen. Clientanwendungen können auch mit CommittableTransaction eine festschreibbare Transaktion erstellen und die Methode EnlistTransaction aufrufen, um sie aufzulisten. Diese Funktion wird vom SQL Anywhere-.NET -Datenprovider unterstützt. Verteilte Transaktionen haben einen erheblichen Performance-Overhead. Es wird empfohlen, für nicht verteilte Transaktionen Datenbanktransaktionen zu verwenden.

C#-Beispiel für SATransaction

Das folgenden Beispiel zeigt, wie Sie eine INSERT-Anweisung so in eine Transaktion einbinden können, dass sie festgeschrieben oder zurückgesetzt werden kann. Eine Transaktion wird mit einem SATransaction-Objekt erstellt und mithilfe eines SACommand-Objekts mit der Ausführung einer SQL-Anweisung verknüpft. Isolationsstufe 2 (RepeatableRead) wird angegeben, damit andere Datenbankbenutzer die Zeile nicht aktualisieren können. Die Sperre der Zeile wird aufgehoben, wenn die Transaktion festgeschrieben oder zurückgesetzt wird. Wenn Sie keine Transaktion verwenden, arbeitet der SQL Anywhere-.NET-Datenprovider im Autocommit-Modus und Sie können Änderungen, die Sie an der Datenbank vorgenommen haben, nicht zurücksetzen.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 16 Demo" );
conn.Open();
string stmt = "UPDATE Products SET UnitPrice = 2000.00 " +
    "WHERE Name = 'Tee shirt'";
bool goAhead = false;

SATransaction trans = conn.BeginTransaction(SAIsolationLevel.RepeatableRead);
SACommand cmd = new SACommand(stmt, conn, trans);
int rowsAffected = cmd.ExecuteNonQuery();
if (goAhead)
```

```
        trans.Commit();  
    else  
        trans.Rollback();  
    conn.Close();
```

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)
- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)
- [SAIsolationLevel-Enumeration \[SQL Anywhere .NET\] auf Seite 333](#)
- [„Isolationsstufen und Konsistenz“ \[SQL Anywhere Server - SQL-Benutzerhandbuch\]](#)
- [„Sperren bei Abfragen“ \[SQL Anywhere Server - SQL-Benutzerhandbuch\]](#)

Fehlerbehandlung

Ihre Anwendung muss so entwickelt werden, dass sie auftretende Fehler behandeln kann.

Der SQL Anywhere .NET-Datenprovider erstellt ein `SAException`-Objekt und gibt eine Ausnahme aus, wenn während der Ausführung Fehler auftreten. Jedes `SAException`-Objekt besteht aus einer Liste von `SAError`-Objekten, und diese Fehlerobjekte enthalten Fehlermeldungen und Fehlercodes.

Fehler sind nicht dasselbe wie Konflikte. Konflikte entstehen, wenn Änderungen in der Datenbank übernommen werden. Ihre Anwendung muss einen Prozess enthalten, um korrekte Werte zu berechnen oder eventuelle Konflikte zu protokollieren.

C#-Beispiel für die Fehlerbehandlung

Mit dem folgenden C#-Code wird ein Schaltflächen-Klickhandler erstellt, der eine Verbindung mit der SQL Anywhere-Beispieldatenbank öffnet. Wenn die Verbindung nicht hergestellt werden kann, zeigt die Ausnahmeroutine eine oder mehrere Meldungen an.

```
private void button1_Click(object sender, EventArgs e)  
{  
    SAConnection conn = new SAConnection("Data Source=SQL Anywhere 16 Demo");  
    try  
    {  
        conn.Open();  
    }  
    catch (SAException ex)  
    {  
        for (int i = 0; i < ex.Errors.Count; i++)  
        {  
            MessageBox.Show(ex.Errors[i].Source + " : " +  
                ex.Errors[i].Message + " (" +  
                ex.Errors[i].NativeError.ToString() + ")",  
                "Failed to connect");  
        }  
    }  
}
```

Visual Basic-Beispiel für die Fehlerbehandlung

Mit dem folgenden Visual Basic-Code wird ein Schaltflächen-Klickhandler erstellt, der eine Verbindung mit der SQL Anywhere-Beispieldatenbank öffnet. Wenn die Verbindung nicht hergestellt werden kann, zeigt die Ausnahmeroutine eine oder mehrere Meldungen an.

```

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=SQL Anywhere 16 Demo")
    Try
        conn.Open()
    Catch ex As SAException
        For i = 0 To ex.Errors.Count - 1
            MessageBox.Show(ex.Errors(i).Source & " : " & _
                ex.Errors(i).Message & " (" & _
                ex.Errors(i).NativeError.ToString() & ")", _
                "Failed to connect")
        Next i
    End Try
End Sub

```

Siehe auch

- [SAError-Klasse \[SQL Anywhere .NET\] auf Seite 252](#)
- [SAException-Klasse \[SQL Anywhere .NET\] auf Seite 256](#)
- [SAFactory-Klasse \[SQL Anywhere .NET\] auf Seite 260](#)
- [Konflikte beim Einsatz des SDataAdapter-Objekts lösen auf Seite 54](#)
- [„Erklärung des Beispielprojekts "Simple"“ auf Seite 81](#)
- [„Erklärung des Beispielprojekts "Table Viewer"“ auf Seite 85](#)

Unterstützung für Entity Framework

Der SQL Anywhere-.NET-Datenprovider unterstützt Entity Framework 5.0, ein separates Paket von Microsoft. Wenn Sie Entity Framework 5.0 verwenden möchten, müssen Sie es mit dem Microsoft NuGet Package Manager zu Visual Studio hinzufügen.

Eine der neuen Funktionen von Entity Framework ist Code First. Sie ermöglicht einen anderen Entwicklungs-Workflow: Definition von Datenmodellobjekten durch Schreiben von C#- oder VB.NET-Klassen, die Datenbankobjekten zugeordnet werden, ohne einen Designer öffnen oder eine XML-Zuordnungsdatei definieren zu müssen. Optional kann eine zusätzliche Konfiguration mit Daten-Annotations oder der Fluent API durchgeführt werden. Modelle können verwendet werden, um ein Datenbankschema zu generieren oder einer vorhandenen Datenbank zuzuordnen.

Dies ist ein Beispiel, bei dem neue Datenbankobjekte unter Zuhilfenahme des Modells erstellt werden:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table("EdmCategories", Schema = "DBA" )]
    public class Category
    {
        public string CategoryID { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
    }
}

```

```
        public virtual ICollection<Product> Products { get; set; }
    }

    [Table( "EdmProducts", Schema = "DBA" )]
    public class Product
    {
        public int ProductId { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
        public string CategoryID { get; set; }

        public virtual Category Category { get; set; }
    }

    [Table( "EdmSuppliers", Schema = "DBA" )]
    public class Supplier
    {
        [Key]
        public string SupplierCode { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
        {
            modelBuilder.Entity<Supplier>().Property( s =>
s.Name ).IsRequired();
        }
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new SAConnectionFactory();
            Database.SetInitializer<Context>( new
DropCreateDatabaseAlways<Context>() );

            using ( var db = new Context( "DSN=SQL Anywhere 16 Demo" ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}
```

Um dieses Beispiel zu erstellen und auszuführen, müssen die folgenden Assembly-Referenzen hinzugefügt werden:

```
EntityFramework
iAnywhere.Data.SQLAnywhere.v4.0
System.ComponentModel.DataAnnotations
System.Data.Entity
```

Dies ist ein anderes Beispiel mit Zuordnung zu einer vorhandenen Datenbank:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "Customers", Schema = "GROUPO" )]
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

        public virtual ICollection<Contact> Contacts { get; set; }
    }

    [Table( "Contacts", Schema = "GROUPO" )]
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
    }

    class Program
    {
        static void Main( string[] args )
        {
        }
    }
}
```

```
{
    Database.DefaultConnectionFactory = new SAConnectionFactory();
    Database.SetInitializer<Context>( null );

    using ( var db = new Context( "DSN=SQL Anywhere 16 Demo" ) )
    {
        foreach ( var customer in db.Customers.ToList() )
        {
            Console.WriteLine( "Customer - " + string.Format(
                "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}",
                customer.ID, customer.SurName, customer.GivenName,
                customer.Street, customer.City, customer.State,
                customer.Country, customer.PostalCode,
                customer.Phone, customer.CompanyName ) );

            foreach ( var contact in customer.Contacts )
            {
                Console.WriteLine( "    Contact - " + string.Format(
                    "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
                    {9}, {10}",
                    contact.ID, contact.SurName, contact.GivenName,
                    contact.Title,
                    contact.Street, contact.City, contact.State,
                    contact.Country, contact.PostalCode,
                    contact.Phone, contact.Fax ) );
            }
        }
    }
}
```

Zwischen dem Microsoft .NET Framework Data Provider für SQL Server (SqlClient) und dem SQL Anywhere-.NET-Datenprovider gibt es einige Detailunterschiede in der Implementierung, die Sie beachten müssen.

1. Eine neue Klasse SAConnectionFactory (implementiert IDbConnectionFactory) ist enthalten. Sie legen die Database.DefaultConnectionFactory mit einer Instanz von SAConnectionFactory fest, bevor Sie ein Datenmodell erstellen. Dies wird nachstehend gezeigt:

```
Database.DefaultConnectionFactory = new SAConnectionFactory();
```

2. Das wichtigsten Prinzip von Entity Framework Code First ist die Programmierung nach Konventionen. Das Entity Framework leitet das Datenmodell durch Programmierungskonventionen ab. Entity Framework führt vieles auch implizit durch. In einigen Fällen realisiert der Entwickler vielleicht nicht alle diese Entity Framework-Konventionen. Aber einige Programmkonventionen sind für SQL Anywhere nicht sinnvoll. Es gibt einen großen Unterschied zwischen SQL Server und SQL Anywhere. Jede SQL Server-Instanz verwaltet mehrere Datenbanken. Jede SQL Anywhere-Datenbank ist hingegen eine einzelne Datei.
 - Wenn der Benutzer ein benutzerdefiniertes DbContext-Objekt mit einem Konstruktor ohne Parameter erstellt, verbindet sich SqlClient mithilfe der integrierten Sicherheit mit SQL Server Express auf dem lokalen Computer. Der SQL Anywhere-Provider stellt eine Verbindung mit dem Standardserver mithilfe eines integrierten Logins her, wenn der Benutzer bereits eine Login-Zuordnung erstellt hat.
 - SqlClient löscht die vorhandene Datenbank und erstellt eine neue Datenbank, wenn das Entity Framework DbDeleteDatabase oder DbCreateDatabase aufruft (nur SQL Server Express). Der

SQL Anywhere-Provider wird nie eine Datenbank entfernen oder erstellen. Er erstellt oder entfernt die Datenbankobjekte (Tabellen, Beziehungen, Integritätsregeln beispielsweise). Der Benutzer muss die Datenbank zunächst erstellen.

- Die IDbConnectionFactory.CreateConnection Methode behandelt den Zeichenfolgenparameter "nameOrConnectionString" als Datenbanknamen (anfänglicher Katalog für SQL Server) oder als Verbindungszeichenfolge. Wenn der Benutzer die Verbindungszeichenfolge für DbContext nicht angibt, verbindet sich SqlConnection automatisch mit dem SQL Express-Server auf dem lokalen Computer und verwendet dabei den Namespace der benutzerdefinierten DbContext-Klasse als anfänglichen Katalog. Für SQL Anywhere kann dieser Parameter nur eine Verbindungszeichenfolge enthalten. Ein Datenbankname wird ignoriert und integriertes Login wird anstelle dessen verwendet.
3. Die SQL Server SqlConnection-API ordnet eine Spalte mit dem Daten-Annotationsattribut TimeStamp dem SQL Server-Datentyp timestamp/rowversion zu. Bei Entwicklern wird SQL Server timestamp/rowversion oft falsch verstanden. Der SQL Server timestamp/rowversion-Datentyp unterscheidet sich von SQL Anywhere und den meisten anderen Datenbankherstellern:
- SQL Server timestamp/rowversion ist binary(8). Er unterstützt keine kombinierten Datums- und Uhrzeitwerte. SQL Anywhere unterstützt einen Datentyp mit dem Namen timestamp, der dem SQL Server datetime-Datentyp entspricht.
 - SQL Server timestamp/rowversion-Werte sind garantiert eindeutig. SQL Anywhere-Zeitstempelwerte sind nicht eindeutig.
 - Ein SQL Server timestamp/rowversion-Wert ändert sich jedes Mal, wenn die Zeile aktualisiert wird.

Das TimeStamp Daten-Annotation-Attribut wird vom SQL Anywhere-Provider nicht unterstützt.

4. Standardmäßig setzt Entity Framework 4.1 den Schema- oder Eigentümernamen auf **dbo**, also das standardmäßige Schema von SQL Server. Allerdings ist **dbo** für SQL Anywhere nicht geeignet. Für SQL Anywhere müssen Sie den Schemanamen (**GROUPO** beispielsweise) mit dem Tabellennamen mit über Daten-Annotations oder die Fluent API angeben. Hier ist ein Beispiel:

```
namespace CodeFirstTest
{
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

        public virtual ICollection<Contact> Contacts { get; set; }
    }
}
```

```
public class Contact
{
    [Key()]
    public int ID { get; set; }
    public string SurName { get; set; }
    public string GivenName { get; set; }
    public string Title { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Country { get; set; }
    public string PostalCode { get; set; }
    public string Phone { get; set; }
    public string Fax { get; set; }

    [ForeignKey( "Customer" )]
    public int CustomerID { get; set; }
    public virtual Customer Customer { get; set; }
}

[Table( "Departments", Schema = "GROUPO" )]
public class Department
{
    [Key()]
    public int DepartmentID { get; set; }
    public string DepartmentName { get; set; }
    public int DepartmentHeadID { get; set; }
}

public class Context : DbContext
{
    public Context() : base() { }
    public Context( string connStr ) : base( connStr ) { }

    public DbSet<Contact> Contacts { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Department> Departments { get; set; }

    protected override void OnModelCreating( DbModelBuilder
modelBuilder )
    {
        modelBuilder.Entity<Contact>().ToTable( "Contacts",
"GROUPO" );
        modelBuilder.Entity<Customer>().ToTable( "Customers",
"GROUPO" );
    }
}
```

Deployment von SQL Anywhere .NET-Datenprovider

Um den SQL Anywhere .NET-Datenprovider zu verwenden, müssen folgende Komponenten auf Ihrem Computer oder Handheld-Gerät installiert sein:

- .NET Framework bzw. .NET Compact Framework Version 2.0 oder später.
- Visual Studio 2005 oder später, oder ein .NET-Compiler wie C# (nur für die Entwicklung erforderlich).

Der Code für den SQL Anywhere-.NET-Datenprovider befindet sich in einer DLL für jede Plattform.

Für Windows erforderliche Dateien

Für Windows (außer Windows Mobile) ist eine der folgenden DLLs erforderlich:

- *%SQLANY16%\V2\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll*
- *%SQLANY16%\V2\Assembly\V3.5\iAnywhere.Data.SQLAnywhere.v3.5.dll*
- *%SQLANY16%\V2\Assembly\V4\iAnywhere.Data.SQLAnywhere.v4.0.dll*

Die Wahl der DLL hängt davon ab, welche Version von .NET Sie verwenden möchten.

Die Windows-Version des Providers erfordert außerdem die folgenden DLLs.

- ***policy.16.0.iAnywhere.Data.SQLAnywhere.dll*** Die Richtliniendatei kann verwendet werden, um die Provider-Version außer Kraft zu setzen, mit der die Anwendung erstellt wurde. Die Richtliniendatei wird von Sybase jedes Mal aktualisiert, wenn eine Aktualisierung für den Provider freigegeben wird. Es gibt auch Richtliniendateien für den Provider der Version 3.5 (*policy.16.0.iAnywhere.Data.SQLAnywhere.v3.5.dll*) und den Provider der Version 4.0 (*policy.16.0.iAnywhere.Data.SQLAnywhere.v4.0.dll*).
- ***dblg16.dll*** Diese Sprachen-DLL enthält vom Provider ausgegebene Meldungen in Englisch (en). Sie ist in vielen anderen Sprachen erhältlich, einschließlich Chinesisch (zh), Französisch (fr), Deutsch (de) und Japanisch (jp).
- ***dbcon16.dll*** Der Unterstützungscod für das Fenster **Mit SQL Anywhere verbinden** ist in dieser DLL enthalten.

Für Windows Mobile erforderliche Dateien

Für Windows Mobile sind die folgenden DLLs erforderlich:

- ***%SQLANY16%\CE\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll*** Dies ist die Version des Providers, der Compact Framework Version 2 für Windows Mobile unterstützt.
- ***%SQLANY16%\CE\Assembly\V3.5\iAnywhere.Data.SQLAnywhere.dll*** Dies ist die Version des Providers, der Compact Framework Version 3.5 für Windows Mobile unterstützt.
- ***%SQLANY16%\CE\Arm.50\dblg16.dll*** Diese Sprachen-DLL enthält vom Provider ausgegebene Meldungen in Englisch (en). Sie ist in vielen anderen Sprachen erhältlich, einschließlich Chinesisch (zh), Französisch (fr), Deutsch (de) und Japanisch (jp).
- ***%SQLANY16%\CE\Arm.50\dbcon16.dll*** Der Unterstützungscod für das Fenster **Mit SQL Anywhere verbinden** ist in dieser DLL enthalten.

Visual Studio stellt die .NET-Datenprovider-DLL (*iAnywhere.Data.SQLAnywhere.dll* oder *iAnywhere.Data.SQLAnywhere.v3.5.dll*) zusammen mit Ihrem Programm auf Ihrem Gerät bereit. Wenn Sie nicht Visual Studio verwenden, müssen Sie die Datenprovider-DLL zusammen mit Ihrer Anwendung auf das Gerät kopieren. Sie kann entweder in dasselbe Verzeichnis wie Ihre Anwendung oder in das Verzeichnis *\Windows* gelegt werden.

dbdata.dll für den SQL Anywhere-.NET-Datenprovider

Wenn der SQL Anywhere .NET Datenprovider zum ersten Mal von einer .NET Anwendung geladen wird (in der Regel beim Herstellen einer Datenbankverbindung mithilfe von `SAConnection`), entpackt er eine DLL, die den nicht verwalteten Code des Providers enthält. Die Datei `dbdata 16.dll` wird vom Provider in einem Unterverzeichnis des Verzeichnisses abgelegt, das mit der folgenden Strategie identifiziert wird.

1. Das erste Verzeichnis, in dem das Speichern versucht wird, ist dasjenige, das durch die erste der folgenden Optionen zurückgegeben wird:
 - Der durch die Umgebungsvariable **TMP** identifizierte Pfad.
 - Der durch die Umgebungsvariable **TEMP** identifizierte Pfad.
 - Der durch die Umgebungsvariable **USERPROFILE** identifizierte Pfad.
 - Das Windows-Verzeichnis.
2. Wenn die angegebenen Verzeichnisse nicht zugänglich sind, versucht der Provider, das aktuelle Arbeitsverzeichnis zu verwenden.
3. Wenn das aktuelle Arbeitsverzeichnis nicht zugänglich ist, versucht der Provider, das Verzeichnis zu verwenden, aus dem die Anwendung selbst geladen wurde.

Das Unterverzeichnis hat die Form einer GUID mit einem Suffix, das die Versionsnummer, den Bitwert der DLL und eine Indexnummer für die Gewährleistung der Eindeutigkeit enthält. Das folgende Beispiel zeigt einen möglichen Unterverzeichnisnamen.

```
{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_160.x64_1
```

Registrierung der SQL Anywhere .NET-Datenprovider-DLL

Die Windows-Version der SQL Anywhere-.NET-Datenprovider-DLL (`%SQLANY16%\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll`) wird bei der Installation von SQL Anywhere im globalen Assembly-Cache registriert. Unter Windows Mobile brauchen Sie die DLL nicht zu registrieren.

Bei einem Deployment des SQL Anywhere-.NET-Datenproviders können Sie sie unter Verwendung des Dienstprogramms **gacutil** registrieren, das im Microsoft SDK enthalten ist.

Um den SQL Anywhere-.NET-Datenprovider beim Deployment des Providers als `DbProviderFactory`-Instanz zu registrieren, müssen Sie der .NET-Datei `machine.config` einen Eintrag hinzufügen. Ein Eintrag ähnlich wie im folgenden Beispiel muss in den Abschnitt `<DbProviderFactories>` eingefügt werden.

```
<add invariant="iAnywhere.Data.SQLAnywhere"
name="SQL Anywhere 16 Data Provider"
description=".Net Framework Data Provider for SQL Anywhere 16"
type="iAnywhere.Data.SQLAnywhere.SAFactory, iAnywhere.Data.SQLAnywhere.v3.5,
Version=16.0.0.14033, Culture=neutral, PublicKeyToken=f222fc4333e0d400"/>
```

Die Versionsnummer muss mit der Version des Providers übereinstimmen, den Sie installieren. Die Konfigurationsdatei befindet sich in `\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`. Für 64-Bit-Windows-Systeme gibt es eine zweite Konfigurationsdatei unter der `Framework64`-Struktur, die ebenfalls geändert werden muss.

Unterstützung der .NET-Protokollierung

Der SQL Anywhere-.NET-Datenprovider unterstützt die Protokollierung mithilfe der .NET-Protokollierungsfunktion. Die Protokollierung wird unter Windows Mobile nicht unterstützt.

Standardmäßig ist die Protokollierung deaktiviert. Wenn Sie die Protokollierung aktivieren möchten, geben Sie die zu verwendende Quelle in der Konfigurationsdatei Ihrer Anwendung an.

Im Folgenden finden Sie ein Beispiel einer Konfigurationsdatei:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
  <source name="iAnywhere.Data.SQLAnywhere"
    switchName="SASourceSwitch"
    switchType="System.Diagnostics.SourceSwitch">
    <listeners>
      <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
      <add name="EventListener"
        type="System.Diagnostics.EventLogTraceListener"
        initializeData="MyEventLog"/>
      <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
      <remove name="Default"/>
    </listeners>
    </source>
  </sources>
<switches>
  <add name="SASourceSwitch" value="All"/>
  <add name="SATraceAllSwitch" value="1" />
  <add name="SATraceExceptionSwitch" value="1" />
  <add name="SATraceFunctionSwitch" value="1" />
  <add name="SATracePoolingSwitch" value="1" />
  <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

Es gibt vier Arten von Trace-Listener, die in der oben gezeigten Konfigurationsdatei zu sehen sind.

- **ConsoleTraceListener** Protokollierung oder Fehlersuchausgabe wird an die Standardausgabe oder den STDERR-Datenstrom geleitet. Wenn Sie Microsoft Visual Studio verwenden, erscheint die Ausgabe im **Ausgabe**-Fenster.
- **DefaultTraceListener** Dieser Listener wird mit dem Namen "Default" automatisch den Debug.Listeners- und Trace.Listeners-Sammlungen hinzugefügt. Protokollierung oder Fehlersuchausgabe wird an die Standardausgabe oder den STDERR-Datenstrom geleitet. Wenn Sie Microsoft Visual Studio verwenden, erscheint die Ausgabe im **Ausgabe**-Fenster. Um ein Duplizierung des von **ConsoleTraceListener** produzierten Ausgabe zu vermeiden, wurde dieser Listener entfernt.
- **EventLogTraceListener** Protokollierung oder Fehlersuchausgabe wird an ein **EventLog** geleitet, das in der Option **initializeData** definiert wird. In dem Beispiel heißt das Ereignisprotokoll

MyEventLog. Schreiben in das Systemereignisprotokoll erfordert keine Administratorrechte und ist keine empfohlene Methode für die Fehlersuche in Anwendungen.

- **TextWriterTraceListener** Protokollierung oder Fehlersuchausgabe wird an einen **TextWriter** geleitet, der den Datenstrom in die Datei schreibt, die in der Option **initializeData** festgelegt ist.

Um die Protokollierung in einem der oben beschriebenen Trace-Listener zu deaktivieren, entfernen Sie den entsprechenden **add**-Eintrag aus **<listeners>**.

Die Trace-Konfigurationsinformationen werden in den Projektordner der Anwendung in der Datei *App.config* gespeichert. Wenn die Datei nicht existiert, kann sie mit Visual Studio erstellt werden:

Hinzufügen » Neues Element, Auswahl von **Anwendungskonfigurationsdatei**.

traceOutputOptions können für alle Listener festgelegt werden und enthalten:

- **Callstack** Schreibt den Aufruf-Stack, der vom Rückgabewert der Eigenschaft `Environment.StackTrace` repräsentiert wird.
- **DateTime** Schreibt das Datum und die Uhrzeit.
- **LogicalOperationStack** Schreibt den Stack für die logischen Vorgänge, der vom Rückgabewert der Eigenschaft `CorrelationManager.LogicalOperationStack` repräsentiert wird.
- **None** Schreibt keine Elemente.
- **ProcessId** Schreibt die Prozess-ID, die vom Rückgabewert der Eigenschaft `Process.Id` repräsentiert wird.
- **ThreadId** Schreibt die Thread-ID, die vom Rückgabewert der Eigenschaft `Thread.ManagedThreadId` für den aktuellen Thread repräsentiert wird.
- **Timestamp** Schreibt den Zeitstempel, der vom Rückgabewert der Methode `System.Diagnostics.Stopwatch.GetTimeStamp` repräsentiert wird.

Die Beispielkonfigurationsdatei (siehe oben) legt die Trace-Ausgabeoptionen nur für **TextWriterTraceListener** fest.

Mit spezifischen Protokolloptionen können Sie begrenzen, welche Elemente protokolliert werden. Standardmäßig sind alle Trace-Optionen mit numerischen Werten auf 0 gesetzt. Zu den Trace-Optionen, die Sie festlegen können, gehören folgende:

- **SASourceSwitch** `SASourceSwitch` kann jeden der folgenden Werte annehmen. Wenn der Wert **Off** ist, erfolgt keine Protokollierung.

Off Lässt keine Ereignisse durch.

Critical Lässt nur kritische Ereignisse durch.

Error Lässt kritische und Fehlerereignisse durch.

Warning Lässt kritische, Fehler- und Warnungereignisse durch.

Information Lässt kritische, Fehler-, Warnungs- und Informationsereignisse durch.

Verbose Lässt kritische, Fehler-, Warnungs-, Informations- und Verbose-Ereignisse durch.

ActivityTracing Lässt Stopp-, Start-, Aussetzen-, Übertragungs- und Wiederaufnahmeereignisse durch.

All Lässt alle Ereignisse durch.

Beispiel für eine Einstellung:

```
<add name="SASourceSwitch" value="Error"/>
```

- **SATraceAllSwitch** Alle Trace-Optionen sind aktiviert. Es ist in diesem Fall nicht erforderlich, weitere Optionen anzugeben, da damit alle ausgewählt werden. Wenn Sie diese Option auswählen, können Sie keine einzelnen Optionen deaktivieren. Es ist z.B. nicht möglich, die Protokollierung von Ausnahmebedingungen mit folgenden Anweisungen zu deaktivieren.

```
<add name="SATraceAllSwitch" value="1" />
<add name="SATraceExceptionSwitch" value="0" />
```

- **SATraceExceptionSwitch** Alle Ausnahmebedingungen werden protokolliert. Protokollmeldungen haben das folgende Format.

```
<Type|ERR> message='message_text'[ nativeError=error_number]
```

Der Text nativeError=error_number wird nur angezeigt, wenn ein SAException-Objekt vorhanden ist.

- **SATraceFunctionSwitch** Alle Eintritte/Austritte des Funktionsbereichs werden protokolliert. Protokollmeldungen können die folgenden Formate haben.

```
enter_nnn <sa.class_name.method_name|API> [object_id#][parameter_names]
leave_nnn
```

"nnn" steht für eine Ganzzahl, die die Verschachtelungsebene 1, 2, 3 usw. des Bereichs darstellt. Der optionale Parameter parameter_names ist eine Liste von durch Leerzeichen getrennten Parameternamen.

- **SATracePoolingSwitch** Alle Verbindungspools werden protokolliert. Protokollmeldungen können die folgenden Formate haben.

```
<sa.ConnectionPool.AllocateConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.RemoveConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.ReturnConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.ReuseConnection|CPOOL>
connectionString='connection_text'
```

- **SATracePropertySwitch** Alle Einstellungen und Abrufe von Eigenschaften werden protokolliert. Protokollmeldungen können die folgenden Formate haben.

```
<sa.class_name.get_property_name|API> object_id#
<sa.class_name.set_property_name|API> object_id#
```

Weitere Hinweise finden Sie unter "Tracing Data Access" unter <http://msdn.microsoft.com/de-de/library/ms971550.aspx>.

Konfigurieren einer Windows-Anwendung für die Protokollierung

Die Aktivierung der Protokollierung in der TableViewer-Beispielanwendung umfasst das Erstellen einer Konfigurationsdatei, die die Listener ConsoleTraceListener und TextWriterTraceListener referenziert, den Standard-Listener entfernt und alle Parameter aktiviert, die sonst auf 0 gesetzt würden.

Voraussetzungen

Visual Studio muss installiert sein.

Kontext und Bemerkungen

Die Protokollierung wird unter Windows Mobile nicht unterstützt.

Aufgabe

1. Öffnen Sie das TableViewer-Beispiel in Visual Studio.

Starten Sie Visual Studio und öffnen Sie `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\TableViewer\TableViewer.sln`.

2. Erstellen Sie eine Anwendungsdatei namens `App.config` und kopieren Sie die folgende Konfiguration:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="iAnywhere.Data.SQLAnywhere"
        switchName="SASourceSwitch"
        switchType="System.Diagnostics.SourceSwitch">
        <listeners>
          <add name="ConsoleListener"
            type="System.Diagnostics.ConsoleTraceListener"/>
          <add name="TraceLogListener"
            type="System.Diagnostics.TextWriterTraceListener"
            initializeData="myTrace.log"
            traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
          <remove name="Default"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="SASourceSwitch" value="All"/>
      <add name="SATraceAllSwitch" value="1" />
      <add name="SATraceExceptionSwitch" value="1" />
      <add name="SATraceFunctionSwitch" value="1" />
      <add name="SATracePoolingSwitch" value="1" />
      <add name="SATracePropertySwitch" value="1" />
    </switches>
  </system.diagnostics>
</configuration>
```

3. Erstellen Sie die Anwendung.
4. Klicken Sie auf **Debuggen » Debuggen starten**.

Ergebnisse

Wenn die Ausführung der Anwendung abgeschlossen ist, wird die Trace-Ausgabe in der Datei *bin\Debug\myTrace.log* erfasst.

Nächste Schritte

Zeigen Sie das Trace-Protokoll im Fenster **Ausgabe** von Visual Studio an.

Siehe auch

- <http://msdn.microsoft.com/de-de/library/ms971550.aspx>

Praktische Einführungen zum .NET-Datenprovider

In diesem Abschnitt wird erklärt, wie Sie die Beispielprojekte "Simple" und "Table Viewer" verwenden, die zum Lieferumfang des SQL Anywhere-.NET-Datenproviders gehören. Die Beispielprojekte können mit Visual Studio 2005 oder späteren Versionen benutzt werden. Die Beispielprojekte wurden mit Visual Studio 2005 entwickelt. Wenn Sie eine spätere Version verwenden, müssen Sie möglicherweise den Visual Studio **Upgrade-Assistenten** ausführen. Dieser Abschnitt enthält außerdem eine praktische Einführung, mit der Sie Schritt für Schritt die Simple Viewer .NET-Datenbankanwendung mit Visual Studio erstellen.

Praktische Einführung: Verwendung des Codebeispiels "Simple"

Das Projekt "Simple" verwendet den .NET-Datenprovider, um eine Ergebnismenge aus dem Datenbankserver abzurufen.

Voraussetzungen

Visual Studio und das .NET-Framework müssen auf Ihrem Computer installiert sein.

Sie müssen das SELECT ANY TABLE-Systemprivileg haben.

Kontext und Bemerkungen

Das Projekt "Simple" wird mit den SQL Anywhere-Beispieldateien mitgeliefert. Es zeigt ein einfaches Listenfeld, das mit Namen aus der Tabelle "Employees" gefüllt ist.

Aufgabe

1. Starten Sie Visual Studio.

2. Klicken Sie auf **Datei » Öffnen » Projekt**.
3. Wechseln Sie zu `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\SimpleWin32` und öffnen Sie das Projekt *Simple.sln*.
4. Wenn Sie den SQL Anywhere .NET-Datenprovider in einem Projekt verwenden, müssen Sie einen Verweis auf den Datenprovider einfügen. Dies wurde im Codebeispiel "Simple" bereits getan. Um den Verweis auf den Datenprovider (*iAnywhere.Data.SQLAnywhere*) einzufügen, öffnen Sie den Ordner **Verweise** im **Projektmappen-Explorer**-Fenster.
5. Sie müssen auch eine `using`-Direktive in Ihren Quellcode einfügen, die auf die Datenprovider-Klassen verweist. Dies wurde im Codebeispiel "Simple" bereits getan. So zeigen Sie die `using`-Direktive an:
 - Öffnen Sie den Quellcode für das Projekt. Rechtsklicken Sie im Fenster **Projektmappen-Explorer** auf *Form1.cs*, und klicken Sie auf **Code anzeigen**.

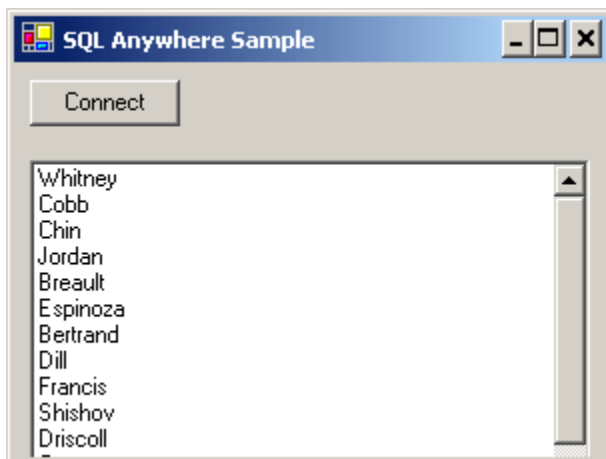
Unter den `using`-Direktiven im oberen Abschnitt müsste folgende Zeile zu sehen sein:

```
using iAnywhere.Data.SQLAnywhere;
```

Diese Zeile ist für C#-Projekte erforderlich. Wenn Sie Visual Basic .NET verwenden, müssen Sie die Zeile `Imports` in Ihren Quellcode einfügen.

6. Klicken Sie auf **Debuggen » Starten ohne Debuggen** oder drücken Sie Strg+F5, um den Beispielcode "Simple" auszuführen.
7. Klicken Sie im Fenster **SQL Anywhere-Beispiel** auf **Verbinden**.

Die Anwendung stellt eine Verbindung zur SQL Anywhere-Beispieldatenbank her und setzt den Nachnamen der einzelnen Mitarbeiter folgendermaßen in das Fenster:



8. Schließen Sie das **SQL Anywhere Sample**-Fenster, um die Anwendung zu schließen und die Verbindung mit der Beispieldatenbank zu trennen. Damit wird auch der Datenbankserver heruntergefahren.

Ergebnisse

Sie haben eine einfache .NET-Anwendung erstellt und ausgeführt, die den SQL Anywhere .NET-Datenprovider verwendet, um eine Ergebnismenge aus einem SQL Anywhere-Datenbankserver abzurufen.

Nächste Schritte

Keine.

Beispiel

Die vollständige Anwendung befindet sich im SQL Anywhere-Beispielverzeichnis unter `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\SimpleWin32`.

Siehe auch

- „[.NET-Datenprovider in einem Visual Studio-Projekt einsetzen](#)“ auf Seite 44
- „[Erklärung des Beispielprojekts "Simple"](#)“ auf Seite 81

Erklärung des Beispielprojekts "Simple"

In diesem Abschnitt werden einige Kernfunktionen des SQL Anywhere .NET-Datenproviders anhand der entsprechenden Abschnitte des Beispielcodes "Simple" erläutert. Der Beispielcode "Simple" verwendet die SQL Anywhere-Beispieldatenbank *demo.db*, die sich in Ihrem SQL Anywhere-Beispielverzeichnis befindet.

Weitere Hinweise zum Speicherort des SQL Anywhere-Beispielverzeichnisses finden Sie unter [Beispielverzeichnis \[SQL Anywhere Server - Datenbankadministration\]](#).

Hinweise zur Beispieldatenbank und zu den darin enthaltenen Tabellen sowie zu ihren Beziehungen untereinander finden Sie unter „[SQL Anywhere-Beispieldatenbank](#)“ [[SQL Anywhere 16 - Einführung](#)].

In diesem Abschnitt wird der Beispielcode ausschnittsweise beschrieben. Eine vollständige Beschreibung ist hier nicht enthalten. Den gesamten Code finden Sie im Beispielprojekt unter `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\SimpleWin32`.

Steuerelemente deklarieren

Das folgende Codefragment deklariert eine Schaltfläche mit der Bezeichnung `btnConnect` und ein Listenfeld mit der Bezeichnung `listEmployees`.

```
private System.Windows.Forms.Button btnConnect;  
private System.Windows.Forms.ListBox listEmployees;
```

Herstellen einer Verbindung zur Datenbank

Die `btnConnect_Click`-Methode deklariert und initialisiert ein `SACConnection`-Verbindungsobjekt.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 16  
Demo;UID=DBA;PWD=sql" );
```

Das `SACConnection`-Objekt stellt mithilfe der Verbindungszeichenfolge eine Verbindung zur SQL Anywhere-Beispieldatenbank her, wenn die `Open`-Methode aufgerufen wird.

```
conn.Open();
```

Weitere Hinweise zum `SACConnection`-Objekt finden Sie unter [SACConnection-Klasse \[SQL Anywhere .NET\]](#) auf Seite 168.

Abfrage definieren

Eine SQL-Anweisung wird mithilfe eines `SACCommand`-Objekts ausgeführt. Der folgende Code deklariert und erstellt mit dem `SACCommand`-Konstruktor ein Befehlsobjekt. Dieser Konstruktor akzeptiert die auszuführende Abfrage in Form einer Zeichenfolge zusammen mit dem `SACConnection`-Objekt, das die Verbindung repräsentiert, auf der die Abfrage ausgeführt wird.

```
SACCommand cmd = new SACCommand( "SELECT Surname FROM Employees", conn );
```

Weitere Hinweise zum `SACCommand`-Objekt finden Sie unter [SACCommand-Klasse \[SQL Anywhere .NET\]](#) auf Seite 129.

Ergebnisse anzeigen

Die Ergebnisse der Abfrage werden mit einem `SADataReader`-Objekt abgefragt. Der folgende Code deklariert und erstellt mit dem `ExecuteReader`-Konstruktor ein `SADataReader`-Objekt. Dieser Konstruktor ist ein Mitglied des `SACCommand`-Objekts "cmd", das zuvor deklariert wurde. `ExecuteReader` sendet den Befehltext zur Ausführung an die Verbindung und erstellt einen `SADataReader`.

```
SADataReader reader = cmd.ExecuteReader();
```

Der nachstehende Code führt einen Schleifendurchlauf durch die Zeilen im `SADataReader`-Objekt durch und fügt sie in das Listenfeld-Steuererelement ein. Sobald die `Read`-Methode aufgerufen wird, erhält das `DataReader`-Objekt eine weitere Zeile aus der Ergebnismenge. Für jede gelesene Zeile wird ein neues Element in das Listenfeld eingefügt. Der `DataReader` verwendet die `GetString`-Methode mit dem Argument 0, um die erste Spalte aus der Ergebnismengenzeile abzurufen.

```
listEmployees.BeginUpdate();  
while( reader.Read() ) {  
    listEmployees.Items.Add( reader.GetString( 0 ) );  
}  
listEmployees.EndUpdate();
```

Weitere Hinweise zum `SADataReader`-Objekt entnehmen Sie dem Abschnitt [SADataReader-Klasse \[SQL Anywhere .NET\]](#) auf Seite 223.

Abschluss

Das nachstehende Codefragment am Ende der Methode schließt die `DataReader`- und `Connection`-Objekte.

```
reader.Close();  
conn.Close();
```

Fehlerbehandlung

Alle Fehler, die während der Ausführung auftreten und aus SQL Anywhere .NET-Datenprovider-Objekten stammen, werden in einem Fenster angezeigt. Der nachstehende Beispielcode fängt den Fehler ab und zeigt die dazugehörige Meldung an:

```
catch( SAException ex ) {  
    MessageBox.Show( ex.Errors[0].Message );  
}
```

Weitere Hinweise zum SAException-Objekt finden Sie im Abschnitt [SAException-Klasse \[SQL Anywhere .NET\]](#) auf Seite 256.

Praktische Einführung: Verwendung des Codebeispiels "Table Viewer"

Das TableViewer-Projekt verwendet den .NET-Datenprovider, um eine Verbindung mit einer Datenbank herzustellen, SQL-Anweisungen auszuführen und die Ergebnisse mit einem DataGrid-Objekt anzuzeigen.

Voraussetzungen

Visual Studio und das .NET-Framework müssen auf Ihrem Computer installiert sein.

Sie müssen das SELECT ANY TABLE-Systemprivileg haben.

Kontext und Bemerkungen

Das Projekt "TableViewer" wird mit den SQL Anywhere-Beispieldateien mitgeliefert. Das "Table Viewer"-Projekt ist komplexer als das "Simple"-Projekt. Sie können es verwenden, um eine Verbindung mit einer Datenbank herzustellen, eine Tabelle auszuwählen und SQL-Anweisungen in der Datenbank auszuführen.

Aufgabe

1. Starten Sie Visual Studio.
2. Klicken Sie auf **Datei » Öffnen » Projekt**.
3. Wechseln Sie zu %SQLANYAMP16%\SQLAnywhere\ADO.NET\TableViewer und öffnen Sie das Projekt *TableViewer.sln*.
4. Um den SQL Anywhere-.NET-Datenprovider in einem Projekt verwenden zu können, müssen Sie eine Referenz auf die Datenprovider-DLL hinzufügen. Dies wurde im Codebeispiel "Table Viewer" bereits getan. Um den Verweis auf den Datenprovider (*iAnywhere.Data.SQLAnywhere*) einzufügen, öffnen Sie den Ordner **Verweise** im **Projektmappen-Explorer**-Fenster.
5. Sie müssen auch eine `using`-Direktive in Ihren Quellcode einfügen, die auf die Datenprovider-Klassen verweist. Dies wurde im Codebeispiel "Table Viewer" bereits getan. So zeigen Sie die `using`-Direktive an:

- Öffnen Sie den Quellcode für das Projekt. Rechtsklicken Sie im Fenster **Projektmappen-Explorer** auf *TableViewer.cs* und klicken Sie auf **Code anzeigen**.
- Unter den `using`-Direktiven im oberen Abschnitt müsste folgende Zeile zu sehen sein:

```
using iAnywhere.Data.SQLAnywhere;
```

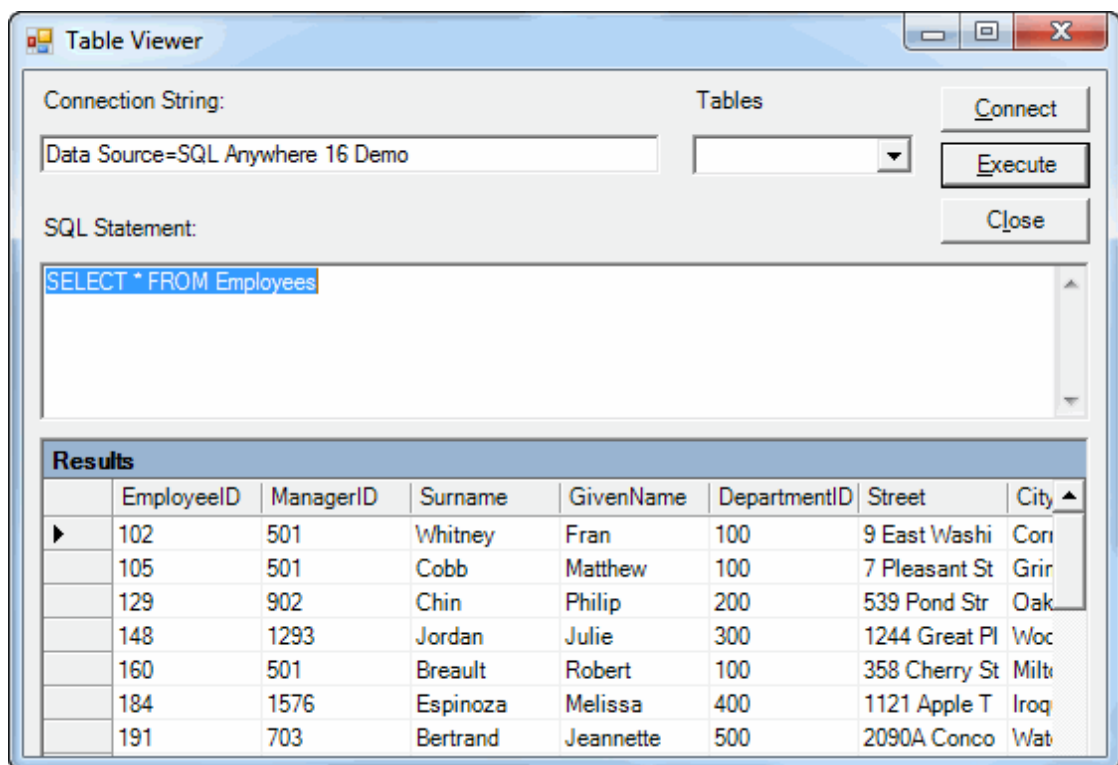
Diese Zeile ist für C#-Projekte erforderlich. Wenn Sie Visual Basic verwenden, müssen Sie die Zeile `Imports` in Ihren Quellcode einfügen.

6. Klicken Sie auf **Debuggen » Starten ohne Debuggen** oder drücken Sie Strg+F5, um den Beispielpcode "Table Viewer" auszuführen.

Die Anwendung verbindet sich mit der SQL Anywhere-Beispieldatenbank.

7. Klicken Sie im Fenster **Table Viewer** auf **Connect**.
8. Klicken Sie im Fenster **Table Viewer** auf **Execute**.

Die Anwendung ruft die Daten aus der Tabelle "Employees" in der Beispieldatenbank ab und setzt die Abfrageergebnisse in das Datenraster **Results**.



Sie können auch andere SQL-Anweisungen aus dieser Anwendung ausführen: Geben Sie eine SQL-Anweisung in den Fensterausschnitt **SQL Statement** ein und klicken Sie auf **Execute**.

9. Schließen Sie das **Table Viewer**-Fenster, um die Anwendung zu schließen und die Verbindung mit der Beispieldatenbank zu trennen. Damit wird auch der Datenbankserver heruntergefahren.

Ergebnisse

Sie haben eine .NET-Anwendung erstellt und ausgeführt, die den .NET-Datenprovider verwendet, um eine Verbindung mit einer Datenbank herzustellen, SQL-Anweisungen auszuführen und die Ergebnisse mit einem DataGrid-Objekt anzuzeigen.

Nächste Schritte

Keine.

Beispiel

Die vollständige Anwendung befindet sich im SQL Anywhere-Beispielverzeichnis unter `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\TableViewer`.

Siehe auch

- „[.NET-Datenprovider in einem Visual Studio-Projekt einsetzen](#)“ auf Seite 44
- „[Erklärung des Beispielprojekts "Table Viewer"](#)“ auf Seite 85

Erklärung des Beispielprojekts "Table Viewer"

In diesem Abschnitt werden einige Kernfunktionen des SQL Anywhere .NET-Datenproviders vorgestellt, indem entsprechende Abschnitte des Beispielcodes "Table Viewer" genauer betrachtet werden. Das Projekt "Table Viewer" benutzt die SQL Anywhere-Beispieldatenbank *demo.db*, die sich in Ihrem SQL Anywhere-Beispielverzeichnis befindet.

Weitere Hinweise zum Speicherort des SQL Anywhere-Beispielverzeichnisses finden Sie unter [Beispielverzeichnis \[SQL Anywhere Server - Datenbankadministration\]](#).

Hinweise zur Beispieldatenbank und zu den darin enthaltenen Tabellen sowie zu ihren Beziehungen untereinander finden Sie unter „[SQL Anywhere-Beispieldatenbank](#)“ [[SQL Anywhere 16 - Einführung](#)].

In diesem Abschnitt wird der Beispielcode ausschnittsweise beschrieben. Eine vollständige Beschreibung ist hier nicht enthalten. Den gesamten Code finden Sie im Beispielprojekt unter `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\TableViewer`.

Steuerelemente deklarieren

Das nachstehende Codefragment deklariert zwei Label namens `label1` und `label2`, ein Textfeld namens `txtConnectionString`, eine Schaltfläche namens `btnConnect`, ein Textfeld namens `txtSQLStatement`, eine Schaltfläche namens `btnExecute` und ein Datenraster namens `dgResults`.

```
private System.Windows.Forms.Label label1;  
private System.Windows.Forms.TextBox txtConnectionString;  
private System.Windows.Forms.Label label2;  
private System.Windows.Forms.Button btnConnect;  
private System.Windows.Forms.TextBox txtSQLStatement;
```

```
private System.Windows.Forms.Button btnExecute;  
private System.Windows.Forms.DataGrid dgResults;
```

Verbindungsobjekt deklarieren

Ein nicht initialisiertes SQL Anywhere-Verbindungsobjekt wird mit dem Typ `SACConnection` deklariert. Für eine eindeutige Verbindung mit einer SQL Anywhere-Datenquelle wird das `SACConnection`-Objekt verwendet.

```
private SACConnection _conn;
```

Weitere Hinweise zur `SACConnection`-Klasse finden Sie unter [SACConnection-Klasse \[SQL Anywhere .NET\]](#) auf Seite 168.

Herstellen einer Verbindung zur Datenbank

Die Text-Eigenschaft des `txtConnectionString`-Objekts hat den Standardwert "Data Source=SQL Anywhere 16 Demo". Dieser Wert kann vom Anwendungsbenuer überschrieben werden, indem er im `txtConnectionString`-Textfeld einen neuen Wert eingibt. Sie können sehen, wie dieser Standardwert festgelegt ist, indem Sie in *TableViewer.cs* den Bereich bzw. Abschnitt mit der Bezeichnung "Windows Form Designer Generated Code" öffnen. In diesem Abschnitt befindet sich die folgende Codezeile.

```
this.txtConnectionString.Text = "Data Source=SQL Anywhere 16 Demo";
```

Weiter unten verwendet das `SACConnection`-Objekt die Verbindungszeichenfolge, um eine Verbindung zu einer Datenbank herzustellen. Der folgende Code erstellt mithilfe des `SACConnection`-Konstruktors mit der Verbindungszeichenfolge ein neues Verbindungsobjekt. Anschließend wird die Verbindung mit der `Open`-Methode eingerichtet.

```
_conn = new SACConnection( txtConnectionString.Text );  
_conn.Open();
```

Weitere Hinweise zum Konstruktor `SACConnection` finden Sie unter [SACConnection.SACConnection-Konstruktor \[SQL Anywhere .NET\]](#) auf Seite 170.

Abfrage definieren

Die Text-Eigenschaft des `txtSQLStatement`-Objekts hat den Standardwert "SELECT * FROM Employees". Dieser Wert kann vom Anwendungsbenuer überschrieben werden, indem er im `txtSQLStatement`-Textfeld einen neuen Wert eingibt.

Die SQL-Anweisung wird mithilfe eines `SACCommand`-Objekts ausgeführt. Der folgende Code deklariert und erstellt mit dem `SACCommand`-Konstruktor ein Befehlsobjekt. Dieser Konstruktor akzeptiert die auszuführende Abfrage in Form einer Zeichenfolge zusammen mit dem `SACConnection`-Objekt, das die Verbindung repräsentiert, auf der die Abfrage ausgeführt wird.

```
SACCommand cmd = new SACCommand( txtSQLStatement.Text.Trim(), _conn );
```

Weitere Hinweise zum `SACCommand`-Objekt finden Sie unter [SACCommand-Klasse \[SQL Anywhere .NET\]](#) auf Seite 129.

Ergebnisse anzeigen

Die Ergebnisse der Abfrage werden mit einem `SADataReader`-Objekt abgefragt. Der folgende Code deklariert und erstellt mit dem `ExecuteReader`-Konstruktor ein `SADataReader`-Objekt. Dieser Konstruktor

ist ein Mitglied des SACommand-Objekts "cmd", das zuvor deklariert wurde. ExecuteReader sendet den Befehlstext zur Ausführung an die Verbindung und erstellt einen SADATAReader.

```
SADATAReader dr = cmd.ExecuteReader();
```

Das folgende Codefragment stellt eine Verbindung des SADATAReader-Objekts mit dem DataGridView-Objekt her, sodass die Ergebnisspalten im Fenster angezeigt werden. Anschließend wird das SADATAReader-Objekt geschlossen.

```
dgResults.DataSource = dr;  
dr.Close();
```

Weitere Hinweise zum SADATAReader-Objekt entnehmen Sie dem Abschnitt [SADATAReader-Klasse \[SQL Anywhere .NET\]](#) auf Seite 223.

Fehlerbehandlung

Wenn bei der Verbindungsherstellung oder beim Ausfüllen des Kombinationsfeld Tables durch die Anwendung ein Fehler auftritt, fängt der nachstehende Programmcode den Fehler ab und zeigt eine Fehlermeldung an:

```
try {  
    _conn = new SACConnection( txtConnectString.Text );  
    _conn.Open();  
  
    SACommand cmd = new SACommand( "SELECT table_name FROM sys.systable " +  
        "WHERE creator = 101 AND table_type != 'TEXT'", _conn );  
    SADATAReader dr = cmd.ExecuteReader();  
  
    comboBoxTables.Items.Clear();  
    while ( dr.Read() ) {  
        comboBoxTables.Items.Add( dr.GetString( 0 ) );  
    }  
    dr.Close();  
} catch( SAException ex ) {  
    MessageBox.Show( ex.Errors[0].Source + " : " + ex.Errors[0].Message + " (" +  
        +  
        ex.Errors[0].NativeError.ToString() + ")",  
        "Failed to connect" );  
}
```

Weitere Hinweise zum SAException-Objekt finden Sie im Abschnitt [SAException-Klasse \[SQL Anywhere .NET\]](#) auf Seite 256.

Praktische Einführung: Eine einfache .NET-Datenbankanwendung mit Visual Studio entwickeln

Dieser Abschnitt enthält eine praktische Einführung, mit der Sie Schritt für Schritt die Simple Viewer .NET-Datenbankanwendung mit Visual Studio erstellen.

Privilegien

Sie müssen das SELECT ANY TABLE-Systemprivileg haben.

Lektion 1: Erstellen von "Table Viewer"

In dieser Lektion verwenden Sie Microsoft Visual Studio, den Server-Explorer und den SQL Anywhere-.NET-Datenprovider, um eine Anwendung zu erstellen, die auf eine der Tabellen in der SQL Anywhere-Beispieldatenbank zugreift und es Ihnen ermöglicht, Zeilen zu untersuchen und Aktualisierungen auszuführen.

Voraussetzungen

Visual Studio und das .NET-Framework müssen auf Ihrem Computer installiert sein.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Eine einfache .NET-Datenbankanwendung mit Visual Studio entwickeln](#)“.

Kontext und Bemerkungen

Diese praktische Einführung basiert auf Visual Studio und dem .NET Framework. Die vollständige Anwendung finden Sie im ADO.NET-Projekt `%SQLANYSAMPI6%\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln`.

Aufgabe

1. Starten Sie Visual Studio.
2. Klicken Sie auf **Datei » Neu » Projekt**.
Es erscheint das Fenster **Neues Projekt**.
 - a. Im linken Fensterausschnitt des Fensters **Neues Projekt** wählen Sie **Visual Basic** oder **Visual C#** als Programmiersprache aus.
 - b. Klicken Sie in der Unterkategorie **Windows** auf **Windows-Anwendung** (VS 2005) oder **Windows Forms-Anwendung** (VS 2008/2010).
 - c. Im Feld **Projektname** geben Sie **MySimpleViewer** ein.
 - d. Klicken Sie auf **OK**, um ein neues Projekt zu erstellen.
3. Klicken Sie auf **Ansicht » Server-Explorer**.
4. Klicken Sie im Fenster **Server-Explorer** mit der rechten Maustaste auf **Datenverbindungen** und klicken Sie dann auf **Verbindung hinzufügen**.
5. Im Fenster **Verbindung hinzufügen** führen Sie Folgendes durch:
 - a. Wenn Sie das Fenster **Verbindung hinzufügen** noch nicht bei anderen Projekten verwendet haben, wird eine Liste von Datenquellen angezeigt. Klicken Sie auf **SQL Anywhere** in der angezeigten Liste der Datenquellen aus.
Wenn Sie das Fenster **Verbindung hinzufügen** bereits verwendet haben, klicken Sie auf **Ändern**, um die Datenquelle auf **SQL Anywhere (SQL Anywhere 11)** zu setzen.
 - b. Unter **Datenquelle**, klicken Sie auf **Name der ODBC-Datenquelle** und den Typ **SQL Anywhere 16 Demo**.

Hinweis

Wenn Sie den Visual Studio-Assistenten zum Hinzufügen einer Verbindung unter 64-Bit-Windows verwenden, werden nur die 64-Bit-System-Datenquellennamen (Data Source Names, DSN) in die Benutzerdatenbankquellennamen (Benutzer-DSN) einbezogen. 32-Bit-System-DSNs werden nicht angezeigt. In der 32-Bit-Entwicklungsumgebung von Visual Studio wird mit der Schaltfläche "Verbindung testen" versucht, eine Verbindung mit dem 32-Bit-Äquivalent des 64-Bit-System-DSN herzustellen. Wenn der 32-Bit-System-DSN nicht existiert, schlägt der Test fehl.

- c. Klicken Sie auf **Verbindung testen**, um zu überprüfen, ob Sie eine Verbindung zur Beispieldatenbank herstellen können.
- d. Klicken Sie auf **OK**.

Eine neue Verbindung namens **SQL Anywhere.demo16** erscheint im Fenster **Server-Explorer**.

- 6. Erweitern Sie die Verbindung **SQL Anywhere.demo16** im Fenster **Server-Explorer**, bis die Tabellennamen angezeigt werden.

(Nur Visual Studio 2005) Versuchen Sie Folgendes:

- a. Rechtsklicken Sie auf die Products-Tabelle und klicken Sie auf **Tabellendaten anzeigen**.
Dies zeigt die Zeilen und Spalten der Products-Tabelle in einem Fenster an.
- b. Schließen Sie das Tabellendaten-Fenster.

- 7. Klicken Sie auf **Daten » Neue Datenquelle hinzufügen**.

- 8. Im **Assistenten zum Konfigurieren von Datenquellen** führen Sie Folgendes durch:

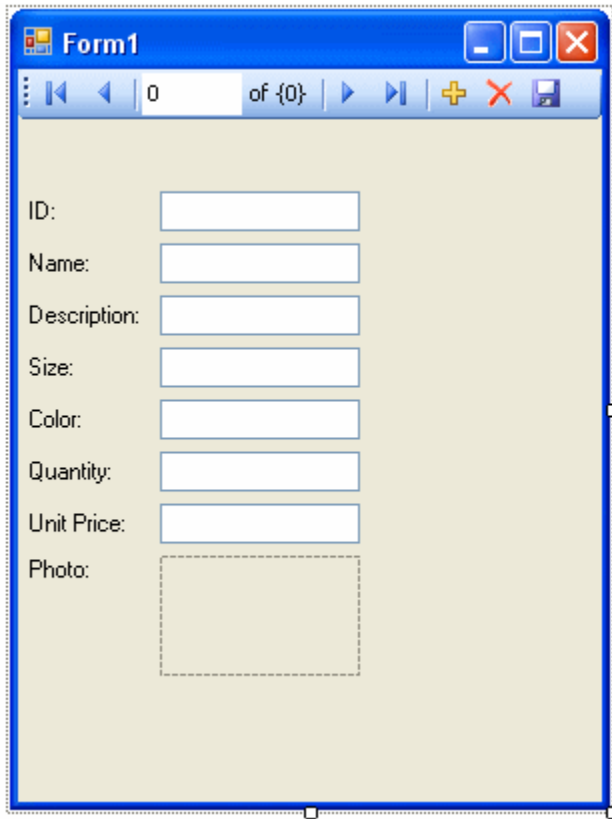
- a. Auf der Seite **Typ der Datenquelle** klicken Sie auf **Datenbank** und dann auf **Weiter**.
- b. (Nur Visual Studio 2010) Klicken Sie auf der Seite **Datenbankmodell** auf **DataSet** und anschließend auf **Weiter**.
- c. Auf der Seite **Datenverbindung** klicken Sie auf **SQL Anywhere.demo16** und dann auf **Weiter**.
- d. Auf der Seite **Verbindungszeichenfolge speichern** vergewissern Sie sich, dass **Ja, die Verbindung speichern unter** ausgewählt ist, und klicken Sie auf **Weiter**.
- e. Auf der Seite **Datenbankobjekte auswählen** klicken Sie auf **Tabellen** und dann auf **Fertig stellen**.

- 9. Klicken Sie auf **Daten » Datenquellen anzeigen**.

Das Fenster **Datenquellen** erscheint.

Erweitern Sie die Products-Tabelle im Fenster **Datenquellen**.

- a. Klicken Sie auf Products und wählen Sie **Details** aus der Dropdown-Liste.
- b. Klicken Sie auf "Photo" und klicken Sie auf **PictureBox** in der Dropdown-Liste.
- c. Klicken Sie auf die Products-Tabelle und ziehen Sie sie auf Ihr Formular (Form1).



Ein DataSet-Steuerelement und mehrere bezeichnete Textfelder erscheinen auf dem Formular.

10. Auf dem Formular klicken Sie auf die PictureBox neben "Photo".
 - a. Ändern Sie die Form der Box in ein Quadrat.
 - b. Klicken Sie auf den Pfeil-nach-rechts in der oberen rechten Ecke der PictureBox.
Das Fenster **PictureBox-Aufgaben** wird geöffnet.
 - c. In der Dropdown-Liste **Größenmodus** klicken Sie auf **Zoom**.
 - d. Um das Fenster **PictureBox-Aufgaben** zu schließen, klicken Sie auf eine Stelle außerhalb des Fensters.
11. Das Projekt kompilieren und ausführen
 - a. Klicken Sie auf **Erstellen » Projektmappe erstellen**.
 - b. Klicken Sie auf **Debuggen » Debuggen starten**.

Die Anwendung stellt eine Verbindung zur SQL Anywhere-Beispieldatenbank her und zeigt die erste Zeile der Products-Tabelle in den Textfeldern und der PictureBox an.

- c. Sie können die Schaltflächen des Steuerelements verwenden, um durch die Zeilen der Ergebnismenge zu blättern.
- d. Sie können direkt zu einer Zeile in der Ergebnismenge gehen, indem Sie die Zeilennummer in das Abroll-Steuerelement eingeben.
- e. Sie können Werte in der Ergebnismenge aktualisieren, indem Sie die Textfelder verwenden, und sie speichern, indem Sie auf die Schaltfläche **Daten speichern** klicken.

12. Beenden Sie die Anwendung und speichern Sie anschließend Ihr Projekt.

Ergebnisse

Sie haben nun eine einfache, aber leistungsstarke Anwendung mit Visual Studio, dem Server-Explorer und dem SQL Anywhere-.NET-Datenprovider erstellt.

Nächste Schritte

In der nächsten Lektion fügen Sie ein Datenraster-Steuerelement zu dem Formular hinzu, das in der vorherigen Lektion entwickelt wurde. Gehen Sie weiter zu [„Lektion 2: Synchronisierende Datensteuerelemente hinzufügen“ auf Seite 92.](#)

Lektion 2: Synchronisierende Datensteuerelemente hinzufügen

In dieser Lektion fügen Sie ein Datenraster-Steuerelement zu dem Formular hinzu, das in der vorherigen Lektion entwickelt wurde. Dieses Steuerelement wird automatisch aktualisiert, während Sie durch die Ergebnismenge navigieren.

Voraussetzungen

Diese Lektion ist eine Fortsetzung der unter „[Lektion 1: Erstellen von "Table Viewer"](#)“ auf Seite 88 beschriebenen Lektion.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Java in der Datenbank verwenden](#)“.

Kontext und Bemerkungen

Die vollständige Anwendung finden Sie im ADO.NET-Projekt `%SQLANYAMP16%\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln`.

Aufgabe

1. Starten Sie Visual Studio und öffnen Sie Ihr Projekt MySimpleViewer. Die Erstellung dieses Projekts erfolgte in „[Lektion 1: Erstellen von "Table Viewer"](#)“ auf Seite 88.
2. Rechtsklicken Sie auf DataSet1 im Fenster **Datenquellen** und klicken Sie auf **DataSet mit Designer bearbeiten**.
3. Rechtsklicken Sie auf einen leeren Bereich im Fenster **DataSet-Designer** und klicken Sie auf **Hinzufügen » TableAdapter**.
4. Im **TableAdapter-Konfigurations-Assistenten** führen Sie Folgendes durch:
 - a. Auf der Seite **Datenverbindung wählen** klicken Sie auf **Weiter**.
 - b. Klicken Sie auf der Seite **Befehlstyp wählen** auf **SQL-Anweisungen verwenden** und klicken Sie auf **Weiter**.
 - c. Auf der Seite **SQL-Anweisung eingeben** klicken Sie auf **Abfrage-Generator**.
 - d. Im Fenster **Tabelle hinzufügen** klicken Sie auf die Registerkarte **Ansichten**, klicken auf die Ansicht **ViewSalesOrders** und klicken auf **Hinzufügen**.
 - e. Klicken Sie auf **Schließen**, um das Fenster **Tabelle hinzufügen** zu schließen.
5. Erweitern Sie das Fenster **Abfrage-Generator**, bis alle Abschnitte des Fensters sichtbar sind.
 - a. Erweitern Sie das Fenster **ViewSalesOrders**, bis alle Kontrollkästchen sichtbar sind.
 - b. Klicken Sie auf **Region**.
 - c. Klicken Sie auf **Quantity**.
 - d. Klicken Sie auf **ProductID**.

- e. Im Raster unterhalb des Fensters **ViewSalesOrders** entfernen Sie das Häkchen im Kontrollkästchen in der Spalte **Ausgabe** für die ProductID-Spalte.
- f. Bei der ProductID-Spalte geben Sie ein Fragezeichen (?) in der **Filter**-Zelle ein. Dadurch wird eine WHERE-Klausel für ProductID generiert.

Es wurde eine SQL-Abfrage aufgebaut, die wie die folgende aussieht:

```
SELECT    Region, Quantity
FROM      GROUPO.ViewSalesOrders
WHERE     (ProductID = :Param1)
```

6. Ändern Sie die SQL-Abfrage folgendermaßen:
 - a. Ändern Sie Quantity zu SUM(Quantity) AS TotalSales.
 - b. Fügen Sie GROUP BY Region am Ende der Abfrage im Anschluss an die WHERE-Klausel hinzu.

Die geänderte SQL-Abfrage sieht nun folgendermaßen aus:

```
SELECT    Region, SUM(Quantity) as TotalSales
FROM      GROUPO.ViewSalesOrders
WHERE     (ProductID = :Param1)
GROUP BY Region
```

7. Klicken Sie auf **OK**.
8. Klicken Sie auf **Fertig stellen**.

Ein neuer **TableAdapter** wurde dem Fenster **DataSet-Designer** hinzugefügt.

9. Klicken Sie auf die Formulardesign-Registerkarte (Form1).
 - Erweitern Sie das Formular nach rechts, um Platz für ein neues Steuerelement zu schaffen.
10. Erweitern Sie ViewSalesOrders im Fenster **Datenquellen**.
 - a. Klicken Sie auf ViewSalesOrders und klicken Sie auf **DataGridView** in der Dropdown-Liste.
 - b. Klicken Sie auf ViewSalesOrders und ziehen Sie es auf Ihr Formular (Form1).

Form1

0 of {0}

Param1: Fill

ID:

Name:

Description:

Size:

Color:

Quantity:

Unit Price:

Photo:

	Region	TotalSales
*		

Ein Datenrasteransicht-Steuerelement erscheint auf dem Formular.

11. Das Projekt kompilieren und ausführen

- Klicken Sie auf **Erstellen** » **Projektmappe erstellen**.
- Klicken Sie auf **Debuggen** » **Debuggen starten**.
- Im Textfeld **Param1** oder **ProductID** (VS 2010) geben Sie eine Produkt-ID-Nummer (z.B. 300) ein und klicken auf **Fill**.

Die Datenrasteransicht zeigt eine Zusammenfassung der Verkäufe pro Region für die eingegebene Produkt-ID an.

Region	TotalSales
Eastern	876
Central	708
Western	324
Canada	216
South	240
*	

Sie können auch das andere Steuerelement auf dem Formular verwenden, um durch die Zeilen der Ergebnismenge zu navigieren.

Es wäre allerdings ideal, wenn beide Steuerelemente miteinander synchronisiert wären. Die nächsten Schritte zeigen, wie Sie dies erreichen.

12. Beenden Sie die Anwendung und speichern Sie anschließend Ihr Projekt.

13. Löschen Sie die Leiste "Fill" (Füllen) vom Formular, weil sie nicht mehr benötigt wird.

- Auf dem Designformular (Form1) rechtsklicken Sie auf der Leiste "Fill" rechts neben das Wort **Fill (Füllen)** und klicken Sie auf **Löschen**.

Die Leiste "Fill" wird vom Formular entfernt.

14. Synchronisieren Sie die zwei Steuerelemente folgendermaßen.

- Auf dem Designformular (Form1) rechtsklicken Sie auf das ID-Textfeld und klicken Sie auf **Eigenschaften**.
- Klicken Sie auf die Schaltfläche **Ereignisse** (als Blitz dargestellt).
- Blättern Sie bis zum Ereignis **TextChanged**.

- d. Klicken Sie auf **TextChanged** und anschließend in der Dropdown-Liste auf **fillToolStripButton_Click**. Wenn Sie Visual Basic verwenden, heißt das Ereignis **FillToolStripButton_Click**.
 - e. Doppelklicken Sie auf **fillToolStripButton_Click**. Das Codefenster des Formulars wird im Event-Handler **fillToolStripButton_Click** geöffnet.
 - f. Suchen Sie die Referenz zu **param1ToolStripTextBox** oder **productIDToolStripTextBox** (VS2010) und ändern Sie dies in **IDTextBox**. Wenn Sie Visual Basic verwenden, heißt das Textfeld **IDTextBox**.
 - g. Kompilieren Sie das Projekt neu und führen Sie es aus.
15. Das Anwendungsformular erscheint nun mit einem einzigen Navigationssteuerelement.
- Die Datenrasteransicht zeigt eine aktualisierte Zusammenfassung von Verkäufen pro Region in Bezug auf das aktuelle Produkt an, während Sie durch die Ergebnismenge navigieren.

	Region	TotalSales
▶	Eastern	1130
	Central	1116
	Western	360
	Canada	252
	South	420
*		

16. Beenden Sie die Anwendung und speichern Sie anschließend Ihr Projekt.

Ergebnisse

Sie haben nun ein Steuerelement hinzugefügt, das automatisch aktualisiert wird, während Sie durch die Ergebnismenge navigieren.

In dieser praktischen Einführung haben Sie gesehen, wie die leistungsstarke Kombination aus Microsoft Visual Studio, dem Server-Explorer und dem SQL Anywhere-.NET-Datenprovider verwendet werden kann, um Datenbankanwendungen zu erstellen.

SQL Anywhere ASP.NET-Provider

Die SQL Anywhere ASP.NET-Provider ersetzen die Standard-ASP.NET-Provider für SQL Server und ermöglichen es Ihnen, Ihre Website mit einer SQL Anywhere-Datenbank auszuführen. Es gibt fünf Provider:

- **Mitgliedschaftsprovider** Der Mitgliedschaftsprovider stellt Authentifizierungs- und Autorisierungsdienste bereit. Verwenden Sie den Mitgliedschaftsprovider, um neue Benutzer und Kennwörter zu erstellen und die Identität von Benutzern zu validieren.
- **Rollenprovider** Der Rollenprovider stellt Methoden bereit, um Rollen zu erstellen, Rollen weitere Benutzer hinzuzufügen und Rollen zu löschen. Verwenden Sie den Rollenprovider, um Benutzer Gruppen zuzuordnen und Berechtigungen zu verwalten.
- **Profilprovider** Der Profilprovider stellt Methoden bereit, um Benutzerdaten zu lesen, zu speichern und abzurufen. Verwenden Sie den Profilprovider, um Benutzervoreinstellungen zu speichern.
- **Provider für Webseiten-Personalisierung** Der Provider für Webseiten-Personalisierung stellt Methoden bereit, um den personalisierten Inhalt bzw. das Layout von Webseiten zu speichern. Verwenden Sie den Provider für Webseiten-Personalisierung, um es Benutzern zu ermöglichen, personalisierte Ansichten Ihrer Website zu erstellen.
- **Systemüberwachungsprovider** Der Systemüberwachungsprovider stellt Methoden zur Überwachung des Status von bereitgestellten Webanwendungen bereit. Verwenden Sie den Systemüberwachungsprovider, um die Performance der Anwendung zu überwachen, fehlerhafte Anwendungen und Systeme zu identifizieren und signifikante Ereignisse zu protokollieren und zu prüfen.

Das SQL Anywhere-Datenbankerverschema, das von den SQL Anywhere ASP.NET-Providern verwendet wird, ist mit dem Schema identisch, das die Standard-ASP.NET-Provider verwenden. Die zum Ändern und Speichern der Daten verwendeten Methoden sind identisch.

Wenn Sie das Einrichten der SQL Anywhere ASP.NET-Provider abgeschlossen haben, können Sie das Visual Studio ASP.NET-Websiteverwaltungs-Tool verwenden, um Benutzer und Rollen zu erstellen und zu verwalten. Sie können auch die Visual Studio-Tools Login, LoginView und PasswordRecovery verwenden, um Ihrer Website Sicherheitsfunktionen hinzuzufügen. Verwenden Sie die statischen Wrapper-Klassen, um auf erweiterte Funktionen der Provider zuzugreifen oder eigene Login-Steuerelemente zu erstellen.

Hinweis

Ein Whitepaper namens *Tutorial: Creating an ASP.NET Web Page Using SQL Anywhere* veranschaulicht die Verwendung von SQL Anywhere und Visual Studio 2010 zum Erstellen einer datenbankgestützten ASP.NET-Website. Weitere Hinweise finden Sie unter www.sybase.com/detail?id=1080238.

ASP.NET-Provider – Datenbankkonfiguration

Um die SQL Anywhere ASP.NET-Provider zu implementieren, können Sie das erforderliche Schema zu einer neuen oder vorhandenen Datenbank hinzufügen.

Wenn Sie das Schema zu einer SQL Anywhere-Datenbank hinzufügen möchten, führen Sie die Datei *SASetupAspNet.exe* aus. Bei der Ausführung stellt die Datei *SASetupAspNet.exe* eine Verbindung mit der SQL Anywhere-Datenbank her und erstellt Tabellen und gespeicherte Prozeduren, die von den SQL Anywhere ASP.NET-Providern benötigt werden. Alle SQL Anywhere ASP.NET-Providerressourcen haben das Präfix *aspnet_*. Um Namenskonflikte mit vorhandenen Datenbankressourcen zu minimieren, können Sie Datenbankressourcen der Provider unter Verwendung der ID eines beliebigen Datenbankbenutzers installieren.

Sie können einen Assistenten oder die Befehlszeile verwenden, um *SASetupAspNet.exe* auszuführen. Um auf den Assistenten zuzugreifen, führen Sie die Anwendung aus oder verwenden eine Befehlszeilenanweisung ohne Argumente. Wenn Sie die Eingabe über eine Befehlszeile für den Zugriff auf *SASetupAspNet.exe* verwenden, benutzen Sie das Fragezeichen-Argument (-?), um eine detaillierte Hilfe zur Konfiguration der Datenbank anzuzeigen.

Die Datenbankverbindung einrichten

Es wird empfohlen, eine Verbindungszeichenfolge mit einem Benutzer anzugeben, der die entsprechenden Privilegien für die zu erstellenden Datenbankobjekte hat. Die folgenden Privilegien sind mindestens erforderlich.

- CREATE ANY TABLE
- SELECT ANY TABLE
- DELETE ANY TABLE
- INSERT ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- CREATE ANY PROCEDURE
- ALTER ANY PROCEDURE
- DROP ANY PROCEDURE
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX

Ressourceneigentümer angeben

Der Assistent und die Befehlszeile (-U <**Benutzer**>) ermöglichen es Ihnen, den Eigentümer der neuen Ressourcen (Tabellen und Prozeduren) anzugeben. Standardmäßig ist der Eigentümer DBA. Sie können

mit dem Assistenten einen anderen Eigentümer angeben. Wenn Sie Verbindungszeichenfolgen für die SQL Anywhere ASP.NET-Provider angeben, muss die UserID (UID) zu dem Benutzer passen, den Sie hier festlegen. Sie müssen dem angegebenen Benutzer keine zusätzlichen Privilegien erteilen. Der Benutzer ist Eigentümer der Ressourcen und hat volle Privilegien für die Tabellen und gespeicherten Prozeduren, die für diesen Benutzer vom Assistenten erstellt werden.

Funktionen auswählen und Daten speichern

Sie können bestimmte Funktionen hinzufügen oder entfernen. Übliche Komponenten werden automatisch installiert. Wenn Sie für eine nicht installierte Funktion **Entfernen** wählen, bleibt dies ohne Wirkung, bei einer bereits installierten Funktion können Sie mit **Hinzufügen** diese Funktion erneut installieren. Standardmäßig werden die Daten in Tabellen gespeichert, die der ausgewählten Funktion zugeordnet sind. Wenn ein Benutzer das Schema einer Tabelle signifikant ändert, ist es vielleicht nicht möglich, die in ihr gespeicherten Daten automatisch zu speichern. Wenn eine komplette Neuinstallation erforderlich ist, kann die Datenspeicherung deaktiviert werden.

Es wird empfohlen, die Mitgliedschafts- und Rollenprovider zusammen zu installieren. Die Effizienz des Visual Studio ASP.NET-Website-Verwaltungstools wird vermindert, wenn der Mitgliedschaftsprovider nicht zusammen mit dem Rollenprovider installiert wird.

Weitere Hinweise

Ein Whitepaper namens *Tutorial: Creating an ASP.NET Web Page Using SQL Anywhere* veranschaulicht die Verwendung von SQL Anywhere und Visual Studio 2010 zum Erstellen einer datenbankgestützten ASP.NET-Website. Weitere Hinweise finden Sie unter www.sybase.com/detail?id=1080238.

Registrierung der Verbindungszeichenfolge

Es gibt zwei Methoden zum Registrieren der Verbindungszeichenfolge:

- Sie können eine ODBC-Datenquelle mithilfe des ODBC-Datenquellenadministrators registrieren und mit Namen referenzieren.
- Sie können eine vollständige SQL Anywhere-Verbindungszeichenfolge angeben, zum Beispiel:

```
connectionString="SERVER=MyServer;DBN=MyDatabase;UID=DBA;PWD=sql"
```

Wenn Sie das <connectionStrings>-Element der Datei *web.config* hinzufügen, kann die Verbindungszeichenfolge und ihr Provider durch die Anwendung referenziert werden. Aktualisierungen können an einem einzigen Speicherort implementiert werden.

XML-Codebeispiel für die Registrierung der Verbindungszeichenfolge

```
<connectionStrings>
  <add name="MyConnectionString"
    connectionString="DSN=MyDataSource"
    providerName="iAnywhere.Data.SQLAnywhere" />
</connectionStrings>
```

Registrieren des SQL Anywhere ASP.NET-Providers

Ihre Webanwendung muss für die Verwendung mit den SQL Anywhere ASP.NET-Providern und nicht den Standard-Providern konfiguriert sein. So registrieren Sie SQL Anywhere ASP.NET-Provider:

- Fügen Sie Ihrer Website eine Referenz zur `iAnywhere.Web.Security-Assembly` hinzu.
- Fügen Sie für jeden Provider dem `<system.web>`-Element in der Datei `web.config` einen Eintrag hinzu.
- Fügen Sie den Namen des SQL Anywhere ASP.NET-Providers dem Attribut `defaultProvider` in der Anwendung hinzu.

Die Providerdatenbank kann Daten für mehrere Anwendungen speichern. Bei jeder Anwendung muss das Attribut `applicationName` für jeden SQL Anywhere ASP.NET-Provider gleich sein. Wenn Sie keinen `applicationName`-Wert angeben, wird jedem Provider in der Providerdatenbank ein identischer Name zugeordnet.

Um eine vorher registrierte Verbindungszeichenfolge zu referenzieren, ersetzen Sie das Attribut `connectionString` durch das Attribut `connectionStringName`.

XML-Codebeispiel für die Registrierung des Mitgliedschaftsproviders

```
<membership defaultProvider="SAMembershipProvider">
  <providers>
    <add name="SAMembershipProvider"
      type="iAnywhere.Web.Security.SAMembershipProvider"
      connectionStringName="MyConnectionString"
      applicationName="MyApplication"
      commandTimeout="30"
      enablePasswordReset="true"
      enablePasswordRetrieval="false"
      maxInvalidPasswordAttempts="5"
      minRequiredNonalphanumericCharacters="1"
      minRequiredPasswordLength="7"
      passwordAttemptWindow="10"
      passwordFormat="Hashed"
      requiresQuestionAndAnswer="true"
      requiresUniqueEmail="true"
      passwordStrengthRegularExpression="" />
  </providers>
</membership>
```

Spaltenbeschreibungen finden Sie unter „[XML-Attribute von Mitgliedschafts Providern](#)“ auf Seite 102.

XML-Codebeispiel für die Registrierung des Rollenproviders

```
<roleManager enabled="true" defaultProvider="SARoleProvider">
  <providers>
    <add name="SARoleProvider"
      type="iAnywhere.Web.Security.SARoleProvider"
      connectionStringName="MyConnectionString"
      applicationName="MyApplication"
      commandTimeout="30" />
  </providers>
</roleManager>
```

Spaltenbeschreibungen finden Sie unter „[Tabellenschema des Rollenproviders](#)“ auf Seite 102.

XML-Codebeispiel für die Registrierung des Profilproviders

```
<profile defaultProvider="SAProfileProvider">
  <providers>
    <add name="SAProfileProvider"
      type="iAnywhere.Web.Security.SAProfileProvider"
      connectionStringName="MyConnectionString"
      applicationName="MyApplication"
      commandTimeout="30" />
  </providers>
  <properties>
    <add name="UserString" type="string"
      serializeAs="Xml" />
    <add name="UserObject" type="object"
      serializeAs="Binary" />
  </properties>
</profile>
```

Spaltenbeschreibungen finden Sie unter „[Profilprovider-Tabellenschema](#)“ auf Seite 103.

XML-Codebeispiel für die Registrierung des Personalisierungsproviders

```
<webParts>
  <personalization defaultProvider="SAPersonalizationProvider">
    <providers>
      <add name="SAPersonalizationProvider"
        type="iAnywhere.Web.Security.SAPersonalizationProvider"
        connectionStringName="MyConnectionString"
        applicationName="MyApplication"
        commandTimeout="30" />
    </providers>
  </personalization>
</webParts>
```

Spaltenbeschreibungen finden Sie unter „[Tabellenschema des Webseiten-Personalisierungsproviders](#)“ auf Seite 104.

XML-Codebeispiel für die Registrierung eines Systemüberwachungsproviders

Weitere Hinweise zum Einrichten der Systemüberwachung finden Sie unter "How To: Use Health Monitoring in ASP.NET 2.0" unter <http://msdn.microsoft.com/de-de/library/ms998306.aspx>.

```
<healthMonitoring enabled="true">
  ...
  <providers>
    <add name="SAWebEventProvider"
      type="iAnywhere.Web.Security.SAWebEventProvider"
      connectionStringName="MyConnectionString"
      commandTimeout="30"
      bufferMode="Notification"
      maxEventDetailsLength="Infinite" />
  </providers>
  ...
</healthMonitoring>
```

Spaltenbeschreibungen finden Sie unter „[Tabellenschema des Systemüberwachungsproviders](#)“ auf Seite 104.

XML-Attribute von Mitgliedschafts Providern

Spaltenname	Beschreibung
name	Der Name des Providers
type	<code>iAnywhere.Web.Security.SAMembershipProvider</code>
connectionStringName	Der Name einer Verbindungszeichenfolge, die im Element <code><connectionStrings></code> angegeben ist.
connectionString	Die Verbindungszeichenfolge Optional. Erforderlich, falls <code>connectionStringName</code> nicht angegeben ist.
applicationName	Der Name der Anwendung, der Providerdaten zugeordnet werden sollen.
commandTimeout	Der Zeitablaufwert, in Sekunden, für Datenbankserveraufrufe
enablePasswordReset	Gültige Einträge sind TRUE oder FALSE.
enablePasswordRetrieval	Gültige Einträge sind TRUE oder FALSE.
maxInvalidPasswordAttempts	Gültige Einträge sind TRUE oder FALSE.
minRequiredNonalphanumericCharacters	Die Mindestanzahl von Sonderzeichen, die in einem gültigen Kennwort enthalten sein müssen.
minRequiredPasswordLength	Die minimale Länge, die ein Kennwort haben muss.
passwordAttemptWindow	Die Zeitspanne, in der aufeinanderfolgende fehlgeschlagene Versuche, ein gültiges Kennwort bzw. Kennwortbestätigung einzugeben, verfolgt werden.
passwordFormat	Gültige Einträge sind Clear, Hashed oder Encrypted.
requiresQuestionAndAnswer	Gültige Einträge sind TRUE oder FALSE.
requiresUniqueEmail	Gültige Einträge sind TRUE oder FALSE.
passwordStrengthRegularExpression	Der reguläre Ausdruck, der verwendet wird, um ein Kennwort auszuwerten.

Tabellenschema des Rollenproviders

SARoleProvider speichert Rolleninformationen in der `aspnet_Roles`-Tabelle der Providerdatenbank. Der SARoleProvider zugeordnete Namespace lautet `iAnywhere.Web.Security`. Jeder Datensatz in der Roles-Tabelle entspricht einer Rolle.

SARoleProvider verwendet die aspnet_UsersInRoles-Tabelle, um Benutzern Rollen zuzuordnen.

Spaltenname	Beschreibung
name	Der Name des Providers
type	<code>iAnywhere.Web.Security.SARoleProvider</code>
connectionStringName	Der Name einer Verbindungszeichenfolge, die im Element <code><connectionStrings></code> angegeben ist.
connectionString	Die Verbindungszeichenfolge Optional. Erforderlich, falls <code>connectionStringName</code> nicht angegeben ist.
applicationName	Der Name der Anwendung, der Providerdaten zugeordnet werden sollen.
commandTimeout	Der Timeoutwert, in Sekunden, für Serveraufrufe.

Profilprovider-Tabellenschema

SAProfileProvider speichert Profildaten in der aspnet_Profile-Tabelle der Providerdatenbank. Der dem SAProfileProvider zugeordnete Namespace lautet `iAnywhere.Web.Security`. Jeder Datensatz in der Profile-Tabelle entspricht den beständigen Profil-Eigenschaften eines Benutzers.

Spaltenname	Beschreibung
name	Der Name des Providers
type	<code>iAnywhere.Web.Security.SAProfileProvider</code>
connectionStringName	Der Name einer Verbindungszeichenfolge, die im Element <code><connectionStrings></code> angegeben ist.
connectionString	Die Verbindungszeichenfolge Optional. Erforderlich, falls <code>connectionStringName</code> nicht angegeben ist.
applicationName	Der Name der Anwendung, der Providerdaten zugeordnet werden sollen.
commandTimeout	Der Timeoutwert, in Sekunden, für Serveraufrufe.

Tabellenschema des Webseiten-Personalisierungsproviders

SAPersonalizationProvider speichert personalisierte Benutzerinhalte in der aspnet_Paths-Tabelle der Providerdatenbank. Der SAPersonalizationProvider zugeordnete Namespace lautet iAnywhere.Web.Security.

SARoleProvider verwendet die Tabellen aspnet_PersonalizationPerUser und die aspnet_PersonalizationAllUsers, um den Pfad festzulegen, für den der Status der Webparts-Personalisierung gespeichert wurde. Die PathID-Spalten zeigen auf die Spalte gleichen Namens in der aspnet_Paths-Tabelle.

Spaltenname	Beschreibung
name	Der Name des Providers
type	iAnywhere.Web.Security.SAPersonalizationProvider
connectionStringName	Der Name einer Verbindungszeichenfolge, die im Element <connectionStrings> angegeben ist.
connectionString	Die Verbindungszeichenfolge Optional. Erforderlich, falls connectionStringName nicht angegeben ist.
applicationName	Der Name der Anwendung, der Providerdaten zugeordnet werden sollen.
commandTimeout	Der Timeoutwert, in Sekunden, für Serveraufrufe.

Tabellenschema des Systemüberwachungsproviders

SAWebEventProvider protokolliert Web-Ereignisse in der aspnet_WebEvent_Events-Tabelle der Provider-Datenbank. Der SAWebEventProvider zugeordnete Namespace lautet iAnywhere.Web.Security. Jeder Datensatz in der WebEvents_Events-Tabelle entspricht einem Webereignis.

Weitere Hinweise zum Einrichten der Systemüberwachung finden Sie unter "How To: Use Health Monitoring in ASP.NET 2.0" unter <http://msdn.microsoft.com/de-de/library/ms998306.aspx>.

Spaltenname	Beschreibung
name	Der Name des Providers
type	iAnywhere.Web.Security.SAWebEventProvider
connectionStringName	Der Name einer Verbindungszeichenfolge, die im Element <connectionStrings> angegeben ist.

Spaltenname	Beschreibung
connectionString	Die Verbindungszeichenfolge Optional. Erforderlich, falls <code>connectionStringName</code> nicht angegeben ist.
commandTimeout	Der Timeoutwert, in Sekunden, für Serveraufrufe.
maxEventDetailsLength	Die maximale Länge der Details-Zeichenfolge für jedes Ereignis oder "Infinite"

SQL Anywhere-.NET-API-Referenz

Namespace

- `iAnywhere.Data.SQLAnywhere`
- `iAnywhere.SQLAnywhere.Server`

SABulkCopy-Klasse

Eine SQL Anywhere-Tabelle mit großen Datenmengen aus einer anderen Quelle effizient laden.

Visual Basic-Syntax

```
Public NotInheritable Class SABulkCopy Implements System.IDisposable
```

C#-Syntax

```
public sealed class SABulkCopy : System.IDisposable
```

Basisklassen

- [System.IDisposable](#)

Mitglieder

Alle Mitglieder der SABulkCopy-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SABulkCopy-Konstruktor	Initialisiert ein SABulkCopy-Objekt.
Close-Methode	Schließt die SABulkCopy-Instanz.
Dispose-Methode	Beseitigt die SABulkCopy-Instanz.

Name	Beschreibung
WriteToServer-Methode	Kopiert alle Zeilen im bereitgestellten Array von System.Data.DataRow-Objekten in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.
BatchSize-Eigenschaft	Ruft die Anzahl der Zeilen in den einzelnen Batches ab oder setzt sie fest.
BulkCopyTimeout-Eigenschaft	Ruft die Anzahl der Sekunden ab bzw. legt die Anzahl der Sekunden fest, nach deren Ablauf der Vorgang abgebrochen wird.
ColumnMappings-Eigenschaft	Gibt eine Sammlung von SABulkCopyColumnMapping-Einträgen zurück.
DestinationTableName-Eigenschaft	Ruft den Namen der Zieltabelle auf dem Server ab bzw. legt ihn fest.
NotifyAfter-Eigenschaft	Ruft die Anzahl der Zeilen ab bzw. legt die Anzahl der Zeilen fest, die vor der Erzeugung eines Benachrichtigungsereignisses verarbeitet werden sollen.
SARowsCopied-Ereignis	Dieses Ereignis tritt ein, wenn die von der Eigenschaft NotifyAfter angegebene Anzahl der Zeilen verarbeitet wurde.

Bemerkungen

Die Klasse SABulkCopy ist in .NET Compact Framework 2.0 nicht verfügbar.

Implements: System.IDisposable

SABulkCopy-Konstruktor

Initialisiert ein SABulkCopy-Objekt.

Überladungsliste

Name	Beschreibung
SABulkCopy(SAConnection)-Konstruktor	Initialisiert ein SABulkCopy-Objekt.
SABulkCopy(SAConnection, SABulkCopyOptions, SATransaction)-Konstruktor	Initialisiert ein SABulkCopy-Objekt.
SABulkCopy(String)-Konstruktor	Initialisiert ein SABulkCopy-Objekt.
SABulkCopy(String, SABulkCopyOptions)-Konstruktor	Initialisiert ein SABulkCopy-Objekt.

SABulkCopy(SAConnection)-Konstruktor

Initialisiert ein SABulkCopy-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal connection As SAConnection)
```

C#-Syntax

```
public SABulkCopy(SAConnection connection)
```

Parameter

- **connection** Die bereits geöffnete SAConnection-Verbindung, die für den Vorgang zum Kopieren großer Datenmengen verwendet wird. Wenn die Verbindung nicht geöffnet ist, wird eine Ausnahmebedingung in WriteToServer ausgegeben.

Bemerkungen

Die Klasse SABulkCopy ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopy(SAConnection, SABulkCopyOptions, SATransaction)-Konstruktor

Initialisiert ein SABulkCopy-Objekt.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal connection As SAConnection,  
    ByVal copyOptions As SABulkCopyOptions,  
    ByVal externalTransaction As SATransaction  
)
```

C#-Syntax

```
public SABulkCopy(  
    SAConnection connection,  
    SABulkCopyOptions copyOptions,  
    SATransaction externalTransaction  
)
```

Parameter

- **connection** Die bereits geöffnete SAConnection-Verbindung, die für den Vorgang zum Kopieren großer Datenmengen verwendet wird. Wenn die Verbindung nicht geöffnet ist, wird eine Ausnahmebedingung in WriteToServer ausgegeben.
- **copyOptions** Eine Kombination von Werten aus der Enumeration SABulkCopyOptions, die festlegt, welche Datenquellenzeilen in die Zieltabelle kopiert werden.
- **externalTransaction** Eine vorhandene SATransaction-Instanz, unter der der Massenexport/-import der Daten ausgeführt wird. Wenn externalTransaction nicht NULL ist, wird der

Massenexport/-import innerhalb der Transaktion ausgeführt. Es wird ein Fehler ausgegeben, wenn sowohl eine externe Transaktion als auch die Option `UseInternalTransaction` festgelegt wird.

Bemerkungen

Die Klasse `SABulkCopy` ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopy(String)-Konstruktor

Initialisiert ein `SABulkCopy`-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal connectionString As String)
```

C#-Syntax

```
public SABulkCopy(string connectionString)
```

Parameter

- **connectionString** Die Zeichenfolge, die die Verbindung festlegt und für die Verwendung durch die `SABulkCopy`-Instanz geöffnet wird. Eine Verbindungszeichenfolge ist eine durch Semikola getrennte Liste von Schlüsselwort=Wert-Paaren.

Bemerkungen

Diese Syntax öffnet während `WriteToServer` unter Verwendung der Verbindungszeichenfolge eine Verbindung. Die Verbindung wird am Ende von `WriteToServer` geschlossen.

Die Klasse `SABulkCopy` ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopy(String, SABulkCopyOptions)-Konstruktor

Initialisiert ein `SABulkCopy`-Objekt.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal connectionString As String,  
    ByVal copyOptions As SABulkCopyOptions  
)
```

C#-Syntax

```
public SABulkCopy(  
    string connectionString,  
    SABulkCopyOptions copyOptions  
)
```

Parameter

- **connectionString** Die Zeichenfolge, die die Verbindung festlegt und für die Verwendung durch die `SABulkCopy`-Instanz geöffnet wird. Eine Verbindungszeichenfolge ist eine durch Semikola getrennte Liste von Schlüsselwort=Wert-Paaren.

- **copyOptions** Eine Kombination von Werten aus der Enumeration `SABulkCopyOptions`, die festlegt, welche Datenquellenzeilen in die Zieltabelle kopiert werden.

Bemerkungen

Diese Syntax öffnet während `WriteToServer` unter Verwendung der Verbindungszeichenfolge eine Verbindung. Die Verbindung wird am Ende von `WriteToServer` geschlossen. Der `copyOptions`-Parameter hat die oben beschriebenen Auswirkungen.

Die Klasse `SABulkCopy` ist in .NET Compact Framework 2.0 nicht verfügbar.

Close-Methode

Schließt die `SABulkCopy`-Instanz.

Visual Basic-Syntax

```
Public Sub Close()
```

C#-Syntax

```
public void Close()
```

Dispose-Methode

Beseitigt die `SABulkCopy`-Instanz.

Visual Basic-Syntax

```
Public Sub Dispose()
```

C#-Syntax

```
public void Dispose()
```

WriteToServer-Methode

Kopiert alle Zeilen im bereitgestellten Array von `System.Data.DataRow`-Objekten in eine Zieltabelle, die von der `DestinationTableName`-Eigenschaft des `SABulkCopy`-Objekts angegeben wird.

Überladungsliste

Name	Beschreibung
WriteToServer(DataRow[])-Methode	Kopiert alle Zeilen im bereitgestellten Array von <code>System.Data.DataRow</code> -Objekten in eine Zieltabelle, die von der <code>DestinationTableName</code> -Eigenschaft des <code>SABulkCopy</code> -Objekts angegeben wird.

Name	Beschreibung
WriteToServer(DataTable)-Methode	Kopiert alle Zeilen in der bereitgestellten Tabelle System.Data.DataTable in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.
WriteToServer(DataTable, DataRowState)-Methode	Kopiert alle Zeilen in der bereitgestellten Tabelle System.Data.DataTable mit dem angegebenen Zeilenstatus in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.
WriteToServer(IDataReader)-Methode	Kopiert alle Zeilen im bereitgestellten Leser System.Data.IDataReader in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.

WriteToServer(DataRow[])-Methode

Kopiert alle Zeilen im bereitgestellten Array von System.Data.DataRow-Objekten in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.

Visual Basic-Syntax

```
Public Sub WriteToServer(ByVal rows As DataRow())
```

C#-Syntax

```
public void WriteToServer(DataRow[] rows)
```

Parameter

- **rows** Ein Array von System.Data.DataRow-Objekten, die in die Zieltabelle kopiert werden

Bemerkungen

Die Klasse SABulkCopy ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SABulkCopy.DestinationTableName-Eigenschaft \[SQL Anywhere .NET\] auf Seite 113](#)

WriteToServer(DataTable)-Methode

Kopiert alle Zeilen in der bereitgestellten Tabelle System.Data.DataTable in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.

Visual Basic-Syntax

```
Public Sub WriteToServer(ByVal table As DataTable)
```

C#-Syntax

```
public void WriteToServer(DataTable table)
```

Parameter

- **table** Eine System.Data.DataTable-Tabelle, deren Zeilen in die Zieltabelle kopiert werden.

Bemerkungen

Die Klasse SABulkCopy ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SABulkCopy.DestinationTableName-Eigenschaft \[SQL Anywhere .NET\] auf Seite 113](#)

WriteToServer(DataTable, DataRowState)-Methode

Kopiert alle Zeilen in der bereitgestellten Tabelle System.Data.DataTable mit dem angegebenen Zeilenstatus in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.

Visual Basic-Syntax

```
Public Sub WriteToServer(  
    ByVal table As DataTable,  
    ByVal rowState As DataRowState  
)
```

C#-Syntax

```
public void WriteToServer(DataTable table, DataRowState rowState)
```

Parameter

- **table** Eine System.Data.DataTable-Tabelle, deren Zeilen in die Zieltabelle kopiert werden.
- **rowState** Ein Wert aus der System.Data.DataRowState-Enumeration. Nur Zeilen mit diesem Zeilenstatus werden in die Zieltabelle kopiert.

Bemerkungen

Nur Zeilen mit entsprechendem Zeilenstatus werden kopiert.

Die Klasse SABulkCopy ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SABulkCopy.DestinationTableName-Eigenschaft \[SQL Anywhere .NET\] auf Seite 113](#)

WriteToServer(IDataReader)-Methode

Kopiert alle Zeilen im bereitgestellten Leser System.Data.IDataReader in eine Zieltabelle, die von der DestinationTableName-Eigenschaft des SABulkCopy-Objekts angegeben wird.

Visual Basic-Syntax

```
Public Sub WriteToServer(ByVal reader As IDataReader)
```

C#-Syntax

```
public void WriteToServer(IDataReader reader)
```

Parameter

- **reader** Ein System.Data.IDataReader-Objekt, dessen Zeilen in die Zieltabelle kopiert werden

Bemerkungen

Die Klasse SABulkCopy ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SABulkCopy.DestinationTableName-Eigenschaft \[SQL Anywhere .NET\] auf Seite 113](#)

BatchSize-Eigenschaft

Ruft die Anzahl der Zeilen in den einzelnen Batches ab oder setzt sie fest.

Visual Basic-Syntax

```
Public Property BatchSize As Integer
```

C#-Syntax

```
public int BatchSize {get;set;}
```

Bemerkungen

Am Ende der einzelnen Batches werden die enthaltenen Zeilen an den Server gesendet.

Die Anzahl von Zeilen in jedem Batch. Standardwert ist "0".

Wenn Sie die Eigenschaft auf Null setzen, werden alle Zeilen in einem Batch gesendet.

Die Verwendung eines negativen Werts für diese Eigenschaft wird als Fehler behandelt.

Wenn dieser Wert geändert wird, während ein Batch ausgeführt wird, wird der aktuelle Batch abgeschlossen und alle folgenden Batches verwenden den neuen Wert.

BulkCopyTimeout-Eigenschaft

Ruft die Anzahl der Sekunden ab bzw. legt die Anzahl der Sekunden fest, nach deren Ablauf der Vorgang abgebrochen wird.

Visual Basic-Syntax

```
Public Property BulkCopyTimeout As Integer
```

C#-Syntax

```
public int BulkCopyTimeout {get;set;}
```

Bemerkungen

Standardwert ist 30 Sekunden.

Der Wert Null zeigt an, dass kein Limit gesetzt ist. Dies sollte vermieden werden, da es zu endlosen Wartezeiten führen kann.

Wenn der Vorgang abgebrochen wird, werden alle Zeilen in der aktuellen Transaktion zurückgesetzt, und es wird eine `SAException`-Ausnahmebedingung generiert.

Die Verwendung eines negativen Werts für diese Eigenschaft wird als Fehler behandelt.

ColumnMappings-Eigenschaft

Gibt eine Sammlung von `SABulkCopyColumnMapping`-Einträgen zurück.

Visual Basic-Syntax

```
Public ReadOnly Property ColumnMappings As  
SABulkCopyColumnMappingCollection
```

C#-Syntax

```
public SABulkCopyColumnMappingCollection ColumnMappings {get;}
```

Bemerkungen

Spaltenzuordnungen definieren die Beziehungen zwischen Spalten in der Datenquelle und Spalten im Datenziel.

Standardmäßig ist dies eine leere Sammlung.

Die Eigenschaft kann nicht geändert werden, während `WriteToServer` ausgeführt wird.

Wenn `ColumnMappings` leer ist, während `WriteToServer` ausgeführt wird, wird die erste Spalte in der Quelle der ersten Spalte im Ziel zugewiesen, die zweite Spalte der zweiten Spalte usw. Dieser Vorgang wird ausgeführt, sofern die Spaltentypen konvertierbar sind, mindestens ebenso viele Zielspalten wie Quellspalten vorhanden sind und eventuelle zusätzliche Zielspalten nullwertfähig sein können.

DestinationTableName-Eigenschaft

Ruft den Namen der Zieltabelle auf dem Server ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Property DestinationTableName As String
```

C#-Syntax

```
public string DestinationTableName {get;set;}
```

Bemerkungen

Der Standardwert ist eine Nullreferenz. In Visual Basic ist er "Nothing".

Wenn der Wert während der Ausführung von WriteToServer geändert wird, hat die Änderung keine Wirkung.

Wenn der Wert vor einem Aufruf von WriteToServer nicht bereits gesetzt wurde, wird die Ausnahmereignis InvalidOperationException generiert.

Es wird als Fehler behandelt, wenn der Wert auf NULL oder eine leere Zeichenfolge gesetzt wird.

NotifyAfter-Eigenschaft

Ruft die Anzahl der Zeilen ab bzw. legt die Anzahl der Zeilen fest, die vor der Erzeugung eines Benachrichtigungsereignisses verarbeitet werden sollen.

Visual Basic-Syntax

```
Public Property NotifyAfter As Integer
```

C#-Syntax

```
public int NotifyAfter {get;set;}
```

Bemerkungen

Wenn die Eigenschaft nicht festgelegt wurde, wird Null zurückgegeben.

Änderungen an NotifyAfter, die während der Ausführung von WriteToServer vorgenommen werden, werden erst nach der nächsten Benachrichtigung wirksam.

Die Verwendung eines negativen Werts für diese Eigenschaft wird als Fehler behandelt.

Die Werte von NotifyAfter und BulkCopyTimeout schließen einander aus, sodass das Ereignis auch dann ausgelöst werden kann, wenn keine Zeilen an die Datenbank gesendet bzw. festgeschrieben wurden.

Siehe auch

- [SABulkCopy.BulkCopyTimeout-Eigenschaft \[SQL Anywhere .NET-API\] auf Seite 112](#)

SARowsCopied-Ereignis

Dieses Ereignis tritt ein, wenn die von der Eigenschaft NotifyAfter angegebene Anzahl der Zeilen verarbeitet wurde.

Visual Basic-Syntax

```
Public Event SARowsCopied As SARowsCopiedEventHandler
```

C#-Syntax

```
public event SARowsCopiedEventHandler SARowsCopied;
```

Bemerkungen

Der Empfang eines SARowsCopied-Ereignisses impliziert nicht, dass irgendwelche Zeilen an den Datenbankserver gesendet oder festgeschrieben wurden. Es ist nicht möglich, die Methode "Close" von diesem Ereignis aus aufzurufen.

Siehe auch

- [SABulkCopy.NotifyAfter-Eigenschaft \[SQL Anywhere .NET\] auf Seite 114](#)

SABulkCopyColumnMapping-Klasse

Definiert die Zuordnung zwischen einer Spalte in der Datenquelle einer SABulkCopy-Instanz und einer Spalte in der Zieltabelle der Instanz.

Visual Basic-Syntax

```
Public NotInheritable Class SABulkCopyColumnMapping
```

C#-Syntax

```
public sealed class SABulkCopyColumnMapping
```

Mitglieder

Alle Mitglieder der SABulkCopyColumnMapping-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SABulkCopyColumnMapping-Konstruktor	Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen oder Namen zur Angabe von Quell- und Zielspalten verwendet werden.
DestinationColumn-Eigenschaft	Ruft den Namen der Spalte ab bzw. legt den Namen der Spalte der Tabelle der Zieldatenbank fest, der zugeordnet wird.
DestinationOrdinal-Eigenschaft	Ruft den Ordinalwert der Spalte ab bzw. legt den Ordinalnamen der Spalte fest, die in der Zieltabelle zugeordnet wird.
SourceColumn-Eigenschaft	Ruft den Namen der in der Datenquelle zugeordneten Spalte ab bzw. legt ihn fest.
SourceOrdinal-Eigenschaft	Ruft die Ordinalposition der Quellspalte innerhalb der Datenquelle ab bzw. legt sie fest.

Bemerkungen

Die Klasse SABulkCopyColumnMapping ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopyColumnMapping-Konstruktor

Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen oder Namen zur Angabe von Quell- und Zielspalten verwendet werden.

Überladungsliste

Name	Beschreibung
SABulkCopyColumnMapping()-Konstruktor	Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen oder Namen zur Angabe von Quell- und Zielspalten verwendet werden.
SABulkCopyColumnMapping(Int, Int)-Konstruktor	Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen zur Angabe von Quell- und Zielspalten verwendet werden.
SABulkCopyColumnMapping(Int, String)-Konstruktor	Erstellt eine neue Spaltenzuordnung unter Verwendung einer Spaltenordinalzahl, um auf die Quellspalte zu verweisen, und eines Spaltennamens, um auf die Zielspalte zu verweisen.
SABulkCopyColumnMapping(String, Int)-Konstruktor	Erstellt eine neue Spaltenzuordnung unter Verwendung eines Spaltennamens, um auf die Quellspalte zu verweisen, und einer Spaltenordinalzahl, um auf die Zielspalte zu verweisen.
SABulkCopyColumnMapping(String, String)-Konstruktor	Erstellt eine neue Spaltenzuordnung, wobei Spaltennamen zur Angabe von Quell- und Zielspalten verwendet werden.

SABulkCopyColumnMapping()-Konstruktor

Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen oder Namen zur Angabe von Quell- und Zielspalten verwendet werden.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SABulkCopyColumnMapping()
```

Bemerkungen

Die Klasse SABulkCopyColumnMapping ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopyColumnMapping(Int, Int)-Konstruktor

Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen zur Angabe von Quell- und Zielspalten verwendet werden.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumnOrdinal As Integer  
)
```

C#-Syntax

```
public SABulkCopyColumnMapping(  
    int sourceColumnOrdinal,  
    int destinationColumnOrdinal  
)
```

Parameter

- **sourceColumnOrdinal** Die Ordinalposition der Quellspalte innerhalb der Datenquelle. Die erste Spalte in einer Datenquelle hat die Ordinalposition Null.
- **destinationColumnOrdinal** Die Ordinalposition der Zielspalte innerhalb der Zieltabelle. Die erste Spalte in einer Tabelle hat die Ordinalposition Null.

Bemerkungen

Die Klasse **SABulkCopyColumnMapping** ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopyColumnMapping(Int, String)-Konstruktor

Erstellt eine neue Spaltenzuordnung unter Verwendung einer Spaltenordinalzahl, um auf die Quellspalte zu verweisen, und eines Spaltennamens, um auf die Zielspalte zu verweisen.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumn As String  
)
```

C#-Syntax

```
public SABulkCopyColumnMapping(  
    int sourceColumnOrdinal,  
    string destinationColumn  
)
```

Parameter

- **sourceColumnOrdinal** Die Ordinalposition der Quellspalte innerhalb der Datenquelle. Die erste Spalte in einer Datenquelle hat die Ordinalposition Null.
- **destinationColumn** Der Name der Zielspalte innerhalb der Zieltabelle.

Bemerkungen

Die Klasse **SABulkCopyColumnMapping** ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopyColumnMapping(String, Int)-Konstruktor

Erstellt eine neue Spaltenzuordnung unter Verwendung eines Spaltennamens, um auf die Quellspalte zu verweisen, und einer Spaltenordinalzahl, um auf die Zielspalte zu verweisen.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal sourceColumn As String,  
    ByVal destinationColumnOrdinal As Integer  
)
```

C#-Syntax

```
public SABulkCopyColumnMapping(  
    string sourceColumn,  
    int destinationColumnOrdinal  
)
```

Parameter

- **sourceColumn** Der Name der Quellspalte innerhalb der Datenquelle.
- **destinationColumnOrdinal** Die Ordinalposition der Zielspalte innerhalb der Zieltabelle. Die erste Spalte in einer Tabelle hat die Ordinalposition Null.

Bemerkungen

Die Klasse SABulkCopyColumnMapping ist in .NET Compact Framework 2.0 nicht verfügbar.

SABulkCopyColumnMapping(String, String)-Konstruktor

Erstellt eine neue Spaltenzuordnung, wobei Spaltennamen zur Angabe von Quell- und Zielspalten verwendet werden.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal sourceColumn As String,  
    ByVal destinationColumn As String  
)
```

C#-Syntax

```
public SABulkCopyColumnMapping(  
    string sourceColumn,  
    string destinationColumn  
)
```

Parameter

- **sourceColumn** Der Name der Quellspalte innerhalb der Datenquelle.
- **destinationColumn** Der Name der Zielspalte innerhalb der Zieltabelle.

Bemerkungen

Die Klasse `SABulkCopyColumnMapping` ist in .NET Compact Framework 2.0 nicht verfügbar.

DestinationColumn-Eigenschaft

Ruft den Namen der Spalte ab bzw. legt den Namen der Spalte der Tabelle der Zieldatenbank fest, der zugeordnet wird.

Visual Basic-Syntax

```
Public Property DestinationColumn As String
```

C#-Syntax

```
public string DestinationColumn {get;set;}
```

Bemerkungen

Eine Zeichenfolge, die den Namen der Spalte in der Zieltabelle oder eine Nullreferenz ("Nothing" in Visual Basic) angibt, wenn die `DestinationOrdinal`-Eigenschaft Priorität hat.

Die Eigenschaften `DestinationColumn` und `DestinationOrdinal` schließen einander aus. Der zuletzt festgelegte Wert hat Vorrang.

Die Festlegung der Eigenschaft `DestinationColumn` bewirkt, dass die Eigenschaft `DestinationOrdinal` auf -1 gesetzt wird. Wenn die Eigenschaft `DestinationOrdinal` festgelegt wird, erhält die Eigenschaft `DestinationColumn` den Wert einer Nullreferenz ("Nothing" in Visual Basic).

Es wird als Fehler behandelt, wenn `DestinationColumn` auf Null oder eine leere Zeichenfolge gesetzt wird.

Siehe auch

- [SABulkCopyColumnMapping.DestinationOrdinal-Eigenschaft \[SQL Anywhere .NET\] auf Seite 119](#)

DestinationOrdinal-Eigenschaft

Ruft den Ordinalwert der Spalte ab bzw. legt den Ordinalnamen der Spalte fest, die in der Zieltabelle zugeordnet wird.

Visual Basic-Syntax

```
Public Property DestinationOrdinal As Integer
```

C#-Syntax

```
public int DestinationOrdinal {get;set;}
```

Bemerkungen

Eine Ganzzahl, die den Ordinalwert der Spalte angibt, die in der Zieltabelle zugeordnet wird, oder -1, wenn die Eigenschaft nicht festgelegt wurde.

Die Eigenschaften `DestinationColumn` und `DestinationOrdinal` schließen einander aus. Der zuletzt festgelegte Wert hat Vorrang.

Die Festlegung der Eigenschaft `DestinationColumn` bewirkt, dass die Eigenschaft `DestinationOrdinal` auf -1 gesetzt wird. Wenn die Eigenschaft `DestinationOrdinal` festgelegt wird, erhält die Eigenschaft `DestinationColumn` den Wert einer Nullreferenz ("Nothing" in Visual Basic).

Siehe auch

- [SABulkCopyColumnMapping.DestinationColumn-Eigenschaft \[SQL Anywhere .NET\] auf Seite 119](#)

SourceColumn-Eigenschaft

Ruft den Namen der in der Datenquelle zugeordneten Spalte ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Property SourceColumn As String
```

C#-Syntax

```
public string SourceColumn {get;set;}
```

Bemerkungen

Eine Zeichenfolge, die den Namen der Spalte in der Datenquelle oder eine Nullreferenz ("Nothing" in Visual Basic) angibt, wenn die `SourceOrdinal`-Eigenschaft Priorität hat.

Die Eigenschaften `SourceColumn` und `SourceOrdinal` schließen einander aus. Der zuletzt festgelegte Wert hat Vorrang.

Die Festlegung der Eigenschaft `SourceColumn` bewirkt, dass die Eigenschaft `SourceOrdinal` auf -1 gesetzt wird. Wenn die Eigenschaft `SourceOrdinal` festgelegt wird, erhält die Eigenschaft `SourceColumn` den Wert einer Nullreferenz ("Nothing" in Visual Basic).

Es wird als Fehler behandelt, wenn `SourceColumn` auf Null oder eine leere Zeichenfolge gesetzt wird.

Siehe auch

- [SABulkCopyColumnMapping.SourceOrdinal-Eigenschaft \[SQL Anywhere .NET\] auf Seite 120](#)

SourceOrdinal-Eigenschaft

Ruft die Ordinalposition der Quellspalte innerhalb der Datenquelle ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property SourceOrdinal As Integer
```

C#-Syntax

```
public int SourceOrdinal {get;set;}
```

Bemerkungen

Eine Ganzzahl, die den Ordinalwert der Spalte in der Datenquelle angibt, oder -1, wenn die Eigenschaft nicht festgelegt wurde.

Die Eigenschaften SourceColumn und SourceOrdinal schließen einander aus. Der zuletzt festgelegte Wert hat Vorrang.

Die Festlegung der Eigenschaft SourceColumn bewirkt, dass die Eigenschaft SourceOrdinal auf -1 gesetzt wird. Wenn die Eigenschaft SourceOrdinal festgelegt wird, erhält die Eigenschaft SourceColumn den Wert einer Nullreferenz ("Nothing" in Visual Basic).

Siehe auch

- [SABulkCopyColumnMapping.SourceColumn-Eigenschaft \[SQL Anywhere .NET\] auf Seite 120](#)

SABulkCopyColumnMappingCollection-Klasse

Eine Sammlung von SABulkCopyColumnMapping-Objekten, die Werte aus System.Collections.CollectionBase erbt.

Visual Basic-Syntax

```
Public NotInheritable Class SABulkCopyColumnMappingCollection
    Inherits System.Collections.CollectionBase
```

C#-Syntax

```
public sealed class SABulkCopyColumnMappingCollection :
    System.Collections.CollectionBase
```

Basisklassen

- [System.Collections.CollectionBase](#)

Mitglieder

Alle Mitglieder der SABulkCopyColumnMappingCollection-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Add-Methode	Fügt das angegebene SABulkCopyColumnMapping-Objekt der Sammlung hinzu.
Clear-Methode (geerbt aus System.Collections.CollectionBase)	Entfernt alle Objekte aus der CollectionBase -Instanz.
Contains-Methode	Ruft einen Wert ab, der anzeigt, ob ein angegebenes SABulkCopyColumnMapping-Objekt in der Sammlung vorhanden ist.

Name	Beschreibung
CopyTo-Methode	Kopiert die Elemente der Sammlung SABulkCopyColumnMappingCollection in ein Array von SABulkCopyColumnMapping -Elementen, wobei an einem bestimmten Index begonnen wird.
GetEnumerator-Methode (geerbt aus System.Collections.CollectionBase)	Gibt einen Enumerator zurück, der die Objekte der System.Collections.CollectionBase -Instanz durchläuft.
IndexOf-Methode	Ruft den Index des angegebenen SABulkCopyColumnMapping -Objekts in der Sammlung ab oder legt ihn fest.
OnClear-Methode (geerbt aus System.Collections.CollectionBase)	Führt beim Löschen des Inhalts der System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnClearComplete-Methode (geerbt aus System.Collections.CollectionBase)	Führt nach dem Löschen des Inhalts der System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnInsert-Methode (geerbt aus System.Collections.CollectionBase)	Führt vor dem Einfügen eines neuen Elements in die System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnInsertComplete-Methode (geerbt aus System.Collections.CollectionBase)	Führt nach dem Einfügen eines neuen Elements in die System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnRemove-Methode (geerbt aus System.Collections.CollectionBase)	Führt beim Entfernen eines Elements aus der System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnRemoveComplete-Methode (geerbt aus System.Collections.CollectionBase)	Führt nach dem Entfernen eines Elements aus der System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnSet-Methode (geerbt aus System.Collections.CollectionBase)	Führt vor dem Einstellen eines Werts in der System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnSetComplete-Methode (geerbt aus System.Collections.CollectionBase)	Führt nach Einstellung eines Werts in der System.Collections.CollectionBase -Instanz zusätzliche benutzerdefinierte Prozesse aus.
OnValidate-Methode (geerbt aus System.Collections.CollectionBase)	Führt beim Validieren eines Werts zusätzliche benutzerdefinierte Prozesse aus.

Name	Beschreibung
Remove-Methode	Entfernt das angegebene SABulkCopyColumnMapping-Element aus der Sammlung SABulkCopyColumnMappingCollection.
RemoveAt-Methode	Entfernt die Zuordnung am angegebenen Index aus der Sammlung.
Capacity-Eigenschaft (geerbt aus System.Collections.CollectionBase)	Ruft die Anzahl der Elemente ab, die System.Collections.CollectionBase enthalten kann, oder legt sie fest.
Count-Eigenschaft (geerbt aus System.Collections.CollectionBase)	Ruft die Anzahl der Elemente ab, die in der System.Collections.CollectionBase -Instanz enthalten sind.
InnerList-Eigenschaft (geerbt aus System.Collections.CollectionBase)	Bezieht eine System.Collections.ArrayList mit der Liste der Elemente in der System.Collections.CollectionBase -Instanz.
List-Eigenschaft (geerbt aus System.Collections.CollectionBase)	Bezieht eine System.Collections.IList mit der Liste der Elemente in der System.Collections.CollectionBase -Instanz.
this-Eigenschaft	Ruft das SABulkCopyColumnMapping-Objekt am angegebenen Index ab.

Bemerkungen

Die Klasse SABulkCopyColumnMappingCollection ist in .NET Compact Framework 2.0 nicht verfügbar.

Implements: ICollection, IEnumerable, IList

Add-Methode

Fügt das angegebene SABulkCopyColumnMapping-Objekt der Sammlung hinzu.

Überladungsliste

Name	Beschreibung
Add(Int, Int)-Methode	Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung der Ordinalzahlen für die Quell- und Zielspalte. Die Zuordnung wird der Sammlung hinzugefügt.
Add(Int, String)-Methode	Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung einer Spaltenordinalzahl, um auf die Quellspalte zu verweisen, und eines Spaltennamens, um auf die Zielspalte zu verweisen. Die Zuordnung wird der Sammlung hinzugefügt.

Name	Beschreibung
Add(SABulkCopyColumnMapping)-Methode	Fügt das angegebene SABulkCopyColumnMapping-Objekt der Sammlung hinzu.
Add(String, Int)-Methode	Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung einer Spaltennamens, um auf die Quellspalte zu verweisen, und einer Spaltenordinalzahl, um auf die Zielspalte zu verweisen. Die Zuordnung wird der Sammlung hinzugefügt.
Add(String, String)-Methode	Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung der Spaltennamen für die Quell- und Zielspalte. Die Zuordnung wird der Sammlung hinzugefügt.

Add(Int, Int)-Methode

Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung der Ordinalzahlen für die Quell- und Zielspalte. Die Zuordnung wird der Sammlung hinzugefügt.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumnOrdinal As Integer  
) As SABulkCopyColumnMapping
```

C#-Syntax

```
public SABulkCopyColumnMapping Add(  
    int sourceColumnOrdinal,  
    int destinationColumnOrdinal  
)
```

Parameter

- **sourceColumnOrdinal** Die Ordinalposition der Quellspalte innerhalb der Datenquelle.
- **destinationColumnOrdinal** Die Ordinalposition der Zielspalte innerhalb der Zieltabelle.

Bemerkungen

Die Klasse SABulkCopyColumnMappingCollection ist in .NET Compact Framework 2.0 nicht verfügbar.

Add(Int, String)-Methode

Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung einer Spaltenordinalzahl, um auf die Quellspalte zu verweisen, und eines Spaltennamens, um auf die Zielspalte zu verweisen. Die Zuordnung wird der Sammlung hinzugefügt.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumn As String  
) As SABulkCopyColumnMapping
```

C#-Syntax

```
public SABulkCopyColumnMapping Add(  
    int sourceColumnOrdinal,  
    string destinationColumn  
)
```

Parameter

- **sourceColumnOrdinal** Die Ordinalposition der Quellspalte innerhalb der Datenquelle.
- **destinationColumn** Der Name der Zielspalte innerhalb der Zieltabelle.

Bemerkungen

Die Klasse SABulkCopyColumnMappingCollection ist in .NET Compact Framework 2.0 nicht verfügbar.

Add(SABulkCopyColumnMapping)-Methode

Fügt das angegebene SABulkCopyColumnMapping-Objekt der Sammlung hinzu.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal bulkCopyColumnMapping As SABulkCopyColumnMapping  
) As SABulkCopyColumnMapping
```

C#-Syntax

```
public SABulkCopyColumnMapping Add(  
    SABulkCopyColumnMapping bulkCopyColumnMapping  
)
```

Parameter

- **bulkCopyColumnMapping** Das Objekt SABulkCopyColumnMapping, das die Zuordnung beschreibt, die zur Sammlung hinzugefügt werden soll.

Bemerkungen

Die Klasse SABulkCopyColumnMappingCollection ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SABulkCopyColumnMapping-Klasse \[SQL Anywhere .NET\] auf Seite 115](#)

Add(String, Int)-Methode

Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung einer Spaltennamens, um auf die Quellspalte zu verweisen, und einer Spaltenordinalzahl, um auf die Zielspalte zu verweisen. Die Zuordnung wird der Sammlung hinzugefügt.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal sourceColumn As String,  
    ByVal destinationColumnOrdinal As Integer  
) As SABulkCopyColumnMapping
```

C#-Syntax

```
public SABulkCopyColumnMapping Add(  
    string sourceColumn,  
    int destinationColumnOrdinal  
)
```

Parameter

- **sourceColumn** Der Name der Quellspalte innerhalb der Datenquelle.
- **destinationColumnOrdinal** Die Ordinalposition der Zielspalte innerhalb der Zieltabelle.

Bemerkungen

Erstellt eine neue Spaltenzuordnung, wobei Ordinalzahlen oder Namen zur Angabe von Quell- und Zielspalten verwendet werden.

Die Klasse SABulkCopyColumnMappingCollection ist in .NET Compact Framework 2.0 nicht verfügbar.

Add(String, String)-Methode

Erstellt ein neues SABulkCopyColumnMapping-Objekt unter Verwendung der Spaltennamen für die Quell- und Zielspalte. Die Zuordnung wird der Sammlung hinzugefügt.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal sourceColumn As String,  
    ByVal destinationColumn As String  
) As SABulkCopyColumnMapping
```

C#-Syntax

```
public SABulkCopyColumnMapping Add(  
    string sourceColumn,  
    string destinationColumn  
)
```

Parameter

- **sourceColumn** Der Name der Quellspalte innerhalb der Datenquelle.

- **destinationColumn** Der Name der Zielspalte innerhalb der Zieltabelle.

Bemerkungen

Die Klasse SABulkCopyColumnMappingCollection ist in .NET Compact Framework 2.0 nicht verfügbar.

Contains-Methode

Ruft einen Wert ab, der anzeigt, ob ein angegebenes SABulkCopyColumnMapping-Objekt in der Sammlung vorhanden ist.

Visual Basic-Syntax

```
Public Function Contains(  
    ByVal value As SABulkCopyColumnMapping  
) As Boolean
```

C#-Syntax

```
public bool Contains(SABulkCopyColumnMapping value)
```

Parameter

- **value** Ein gültiges SABulkCopyColumnMapping-Objekt.

Rückgabe

TRUE, wenn die angegebene Zuordnung in der Sammlung vorhanden ist, anderenfalls FALSE.

CopyTo-Methode

Kopiert die Elemente der Sammlung SABulkCopyColumnMappingCollection in ein Array von SABulkCopyColumnMapping-Elementen, wobei an einem bestimmten Index begonnen wird.

Visual Basic-Syntax

```
Public Sub CopyTo(  
    ByVal array As SABulkCopyColumnMapping(),  
    ByVal index As Integer  
)
```

C#-Syntax

```
public void CopyTo(SABulkCopyColumnMapping[] array, int index)
```

Parameter

- **array** Das eindimensionale SABulkCopyColumnMapping-Array, das das Ziel der aus SABulkCopyColumnMappingCollection kopierten Elemente ist. Das Array muss eine auf Null basierende Indizierung haben.
- **index** Der auf Null basierende Index im Array, an dem der Kopiervorgang beginnt

IndexOf-Methode

Ruft den Index des angegebenen SABulkCopyColumnMapping-Objekts in der Sammlung ab oder legt ihn fest.

Visual Basic-Syntax

```
Public Function IndexOf(  
    ByVal value As SABulkCopyColumnMapping  
) As Integer
```

C#-Syntax

```
public int IndexOf(SABulkCopyColumnMapping value)
```

Parameter

- **value** Das SABulkCopyColumnMapping-Objekt, das gesucht werden soll.

Rückgabe

Der Rückgabewert ist der auf Null basierende Index der Spaltenzuordnung. Wenn die Spaltenzuordnung nicht in der Sammlung gefunden wird, wird -1 zurückgegeben.

Remove-Methode

Entfernt das angegebene SABulkCopyColumnMapping-Element aus der Sammlung SABulkCopyColumnMappingCollection.

Visual Basic-Syntax

```
Public Sub Remove(ByVal value As SABulkCopyColumnMapping)
```

C#-Syntax

```
public void Remove(SABulkCopyColumnMapping value)
```

Parameter

- **value** Das SABulkCopyColumnMapping-Objekt, das aus der Sammlung entfernt werden soll.

RemoveAt-Methode

Entfernt die Zuordnung am angegebenen Index aus der Sammlung.

Visual Basic-Syntax

```
Public Shadows Sub RemoveAt(ByVal index As Integer)
```

C#-Syntax

```
public new void RemoveAt(int index)
```

Parameter

- **index** Der auf Null basierende Index des SABulkCopyColumnMapping-Objekts, das aus der Sammlung entfernt werden soll.

this-Eigenschaft

Ruft das SABulkCopyColumnMapping-Objekt am angegebenen Index ab.

Visual Basic-Syntax

```
Public ReadOnly Property Item(  
    ByVal index As Integer  
) As SABulkCopyColumnMapping
```

C#-Syntax

```
public SABulkCopyColumnMapping this[int index] {get;}
```

Parameter

- **index** Der auf Null basierende Index des zu suchenden SABulkCopyColumnMapping-Objekts

Rückgabe

Es wird ein SABulkCopyColumnMapping-Objekt zurückgegeben.

SACommand-Klasse

Eine SQL-Anweisung oder eine gespeicherte Prozedur, die in einer SQL Anywhere-Datenbank ausgeführt wird.

Visual Basic-Syntax

```
Public NotInheritable Class SACommand  
    Inherits System.Data.Common.DbCommand  
    Implements System.ICloneable
```

C#-Syntax

```
public sealed class SACommand :  
    System.Data.Common.DbCommand,  
    System.ICloneable
```

Basisklassen

- [System.Data.Common.DbCommand](#)
- [System.ICloneable](#)

Mitglieder

Alle Mitglieder der SACommand-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SACommand-Konstruktor	Initialisiert ein SACommand-Objekt.
BeginExecuteNonQuery-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird.
BeginExecuteReader-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Datenbankserver ab.
Cancel-Methode	Bricht die Ausführung eines SACommand-Objekts ab.
CreateParameter-Methode	Stellt ein SAParameter-Objekt bereit, um Parameter an SACommand-Objekte übergeben zu können.
EndExecuteNonQuery-Methode	Beendet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur.
EndExecuteReader-Methode	Beendet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur und gibt den angeforderten SADataReader zurück.
ExecuteDbDataReader-Methode (geerbt aus System.Data.Common.DbCommand)	Führt den Befehltext für die Verbindung aus.
ExecuteDbDataReaderAsync-Methode (geerbt aus System.Data.Common.DbCommand)	Provider sollten diese Methode implementieren, um eine Nicht-Standardimplementierung für Overload: System.Data.Common.DbCommand.ExecuteReader-Überladungen bereitzustellen. Die Standardimplementierung ruft die synchrone System.Data.Common.DbCommand.ExecuteReader -Methode auf und gibt eine abgeschlossene Aufgabe zurück, die den aufrufenden Thread blockiert.
ExecuteNonQuery-Methode	Führt eine Anweisung aus, die keine Ergebnismenge zurückgibt, z.B. eine INSERT-, UPDATE-, DELETE- oder Datendefinitionsanweisung.
ExecuteNonQueryAsync-Methode (geerbt aus System.Data.Common.DbCommand)	Eine asynchrone Version von System.Data.Common.DbCommand.ExecuteNonQuery , die eine SQL-Anweisung in einem Verbindungsobjekt ausführt.
ExecuteReader-Methode	Führt eine SQL-Anweisung aus, die eine Ergebnismenge zurückgibt.

Name	Beschreibung
ExecuteReaderAsync-Methode (geerbt aus System.Data.Common.DbCommand)	Eine asynchrone Version von Overload: System.Data.Common.DbCommand.ExecuteReader , die System.Data.Common.DbCommand.CommandText auf System.Data.Common.DbCommand.Connection ausführt und ein System.Data.Common.DbDataReader -Objekt zurückgibt. Ruft System.Data.Common.DbCommand.ExecuteReaderAsync(System.Data.CommandBehavior, System.Threading.CancellationToken) mit CancellationTokens.None auf.
ExecuteScalar-Methode	Führt eine Anweisung aus, die einen einzelnen Wert zurückgibt.
ExecuteScalarAsync-Methode (geerbt aus System.Data.Common.DbCommand)	Eine asynchrone Version von System.Data.Common.DbCommand.ExecuteScalar , die die Abfrage ausführt und die erste Spalte der ersten Zeile in der von der Abfrage zurückgegebenen Ergebnismenge zurückgibt.
Prepare-Methode	Bereitet das SACommand -Objekt für die Datenquelle vor oder kompiliert den Befehl.
ResetCommandTimeout-Methode	Setzt die CommandTimeout -Eigenschaft auf ihren Standardwert von 30 Sekunden zurück.
CommandText-Eigenschaft	Ruft den Text einer SQL-Anweisung oder einer gespeicherten Prozedur ab bzw. legt ihn fest.
CommandTimeout-Eigenschaft	Diese Funktion wird nicht vom SQL Anywhere .NET-Datenprovider unterstützt.
CommandType-Eigenschaft	Ruft den Befehlstyp ab, der durch ein SACommand -Objekt repräsentiert wird, bzw. legt den Befehlstyp fest.
Connection-Eigenschaft	Ruft das Verbindungsobjekt ab, auf das das SACommand -Objekt anzuwenden ist, bzw. legt das Verbindungsobjekt fest.
DesignTimeVisible-Eigenschaft	Ruft einen Wert ab bzw. legt einen Wert fest, der angibt, ob das SACommand -Objekt in einem Steuerelement eines Windows Form Designers sichtbar sein soll.
Parameters-Eigenschaft	Eine Sammlung von Parametern für die aktuelle Anweisung.
Transaction-Eigenschaft	Gibt das SATransaction -Objekt an, in der das SACommand -Objekt ausgeführt wird.
UpdatedRowSource-Eigenschaft	Legt fest oder ruft ab, wie Ergebnisse von Befehlen auf das DataRow -Objekt angewendet werden, wenn sie von der Update -Methode des SADataAdapter -Objekts verwendet werden.

Bemerkungen

Implements: IDbCommand, ICloneable

Weitere Hinweise finden Sie unter [„Datenzugriff und Datenverarbeitung“](#) auf Seite 47.

SACommand-Konstruktor

Initialisiert ein SACommand-Objekt.

Überladungsliste

Name	Beschreibung
SACommand()-Konstruktor	Initialisiert ein SACommand-Objekt.
SACommand(String)-Konstruktor	Initialisiert ein SACommand-Objekt.
SACommand(String, SAConnection)-Konstruktor	Eine SQL-Anweisung oder eine gespeicherte Prozedur, die in einer SQL Anywhere-Datenbank ausgeführt wird.
SACommand(String, SAConnection, SATransaction)-Konstruktor	Eine SQL-Anweisung oder eine gespeicherte Prozedur, die in einer SQL Anywhere-Datenbank ausgeführt wird.

SACommand()-Konstruktor

Initialisiert ein SACommand-Objekt.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SACommand()
```

SACommand(String)-Konstruktor

Initialisiert ein SACommand-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal cmdText As String)
```

C#-Syntax

```
public SACommand(string cmdText)
```

Parameter

- **cmdText** Der Text der SQL-Anweisung oder der gespeicherten Prozedur. Bei Anweisungen mit Parametern benutzen Sie als Platzhalter ein Fragezeichen (?), um Parameter zu übergeben.

SACommand(String, SAConnection)-Konstruktor

Eine SQL-Anweisung oder eine gespeicherte Prozedur, die in einer SQL Anywhere-Datenbank ausgeführt wird.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal cmdText As String,  
    ByVal connection As SAConnection  
)
```

C#-Syntax

```
public SACommand(string cmdText, SAConnection connection)
```

Parameter

- **cmdText** Der Text der SQL-Anweisung oder der gespeicherten Prozedur. Bei Anweisungen mit Parametern benutzen Sie als Platzhalter ein Fragezeichen (?), um Parameter zu übergeben.
- **connection** Die aktuelle Verbindung.

SACommand(String, SAConnection, SATransaction)-Konstruktor

Eine SQL-Anweisung oder eine gespeicherte Prozedur, die in einer SQL Anywhere-Datenbank ausgeführt wird.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal cmdText As String,  
    ByVal connection As SAConnection,  
    ByVal transaction As SATransaction  
)
```

C#-Syntax

```
public SACommand(  
    string cmdText,  
    SAConnection connection,  
    SATransaction transaction  
)
```

Parameter

- **cmdText** Der Text der SQL-Anweisung oder der gespeicherten Prozedur. Bei Anweisungen mit Parametern benutzen Sie als Platzhalter ein Fragezeichen (?), um Parameter zu übergeben.
- **connection** Die aktuelle Verbindung.
- **transaction** Das SATransaction-Objekt, in dem das SAConnection-Objekt ausgeführt wird.

Siehe auch

- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)

BeginExecuteNonQuery-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses `SACommand`-Objekt beschrieben wird.

Überladungsliste

Name	Beschreibung
BeginExecuteNonQuery()-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses <code>SACommand</code> -Objekt beschrieben wird.
BeginExecuteNonQuery(AsyncCallback, Object)-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses <code>SACommand</code> -Objekt beschrieben wird, sofern eine Callback-Prozedur und Statusinformationen vorhanden sind.

BeginExecuteNonQuery()-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses `SACommand`-Objekt beschrieben wird.

Visual Basic-Syntax

```
Public Function BeginExecuteNonQuery() As IAsyncResult
```

C#-Syntax

```
public IAsyncResult BeginExecuteNonQuery()
```

Rückgabe

Ein `System.IAsyncResult`-Wert, der für Abrufe, zum Warten auf Ergebnisse oder für beides verwendet werden kann. Dieser Wert ist auch für den Aufruf von `EndExecuteNonQuery(IAsyncResult)` erforderlich, der die Anzahl der betroffenen Zeilen zurückgibt

Ausnahmebedingungen

- **[SAException-Klasse](#)** Jeder Fehler, der während der Ausführung des Befehlstexts aufgetreten ist.

Bemerkungen

Bei asynchronen Befehlen muss die Reihenfolge der Parameter mit `CommandText` konsistent sein.

Siehe auch

- [SACommand.EndExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 140](#)

BeginExecuteNonQuery(AsyncCallback, Object)-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, sofern eine Callback-Prozedur und Statusinformationen vorhanden sind.

Visual Basic-Syntax

```
Public Function BeginExecuteNonQuery(  
    ByVal callback As AsyncCallback,  
    ByVal stateObject As Object  
) As IAsyncResult
```

C#-Syntax

```
public IAsyncResult BeginExecuteNonQuery(  
    AsyncCallback callback,  
    object stateObject  
)
```

Parameter

- **callback** Ein System.AsyncCallback-Delegat, der aufgerufen wird, wenn die Ausführung des Befehls abgeschlossen ist. Mit der Übergabe von Null (in Microsoft Visual Basic "Nothing") legen Sie fest, dass keine Callback-Prozedur erforderlich ist.
- **stateObject** Ein benutzerdefiniertes Statusobjekt, das an die Callback-Prozedur übergeben wird. Sie rufen dieses Objekt innerhalb der Callback-Prozedur unter Verwendung der System.IAsyncResult.AsyncState-Eigenschaft ab.

Rückgabe

Ein System.IAsyncResult-Wert, der für Abrufe, zum Warten auf Ergebnisse oder für beides verwendet werden kann. Dieser Wert ist auch für den Aufruf von EndExecuteNonQuery(IAsyncResult) erforderlich, der die Anzahl der betroffenen Zeilen zurückgibt

Ausnahmebedingungen

- **SAException-Klasse** Jeder Fehler, der während der Ausführung des Befehlstexts aufgetreten ist.

Bemerkungen

Bei asynchronen Befehlen muss die Reihenfolge der Parameter mit CommandText konsistent sein.

Siehe auch

- [SACommand.EndExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 140](#)

BeginExecuteReader-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Datenbankserver ab.

Überladungsliste

Name	Beschreibung
BeginExecuteReader()-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Datenbankserver ab.
BeginExecuteReader(AsyncCallback, Object)-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung, die durch das SACommand-Objekt beschrieben wird, und ruft die Ergebnismenge ab, sofern eine Callback-Prozedur und Statusinformationen vorhanden sind.
BeginExecuteReader(AsyncCallback, Object, CommandBehavior)-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Server ab.
BeginExecuteReader(CommandBehavior)-Methode	Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Server ab.

BeginExecuteReader()-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Datenbankserver ab.

Visual Basic-Syntax

```
Public Function BeginExecuteReader() As IAsyncResult
```

C#-Syntax

```
public IAsyncResult BeginExecuteReader()
```

Rückgabe

Ein System.IAsyncResult-Wert, der für Abrufe, zum Warten auf Ergebnisse oder für beides verwendet werden kann. Dieser Wert ist auch für den Aufruf des EndExecuteReader(IAsyncResult)-Objekts erforderlich, der ein SADATAReader-Objekt zurückgibt, das für die Abfrage der zurückgegebenen Zeilen verwendet werden kann.

Ausnahmebedingungen

- **SAException-Klasse** Jeder Fehler, der während der Ausführung des Befehlstexts aufgetreten ist.

Bemerkungen

Bei asynchronen Befehlen muss die Reihenfolge der Parameter mit CommandText konsistent sein.

Siehe auch

- [SACCommand.EndExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 142](#)
- [SADatReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)

BeginExecuteReader(AsyncCallback, Object)-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung, die durch das SACCommand-Objekt beschrieben wird, und ruft die Ergebnismenge ab, sofern eine Callback-Prozedur und Statusinformationen vorhanden sind.

Visual Basic-Syntax

```
Public Function BeginExecuteReader(  
    ByVal callback As AsyncCallback,  
    ByVal stateObject As Object  
) As IAsyncResult
```

C#-Syntax

```
public IAsyncResult BeginExecuteReader(  
    AsyncCallback callback,  
    object stateObject  
)
```

Parameter

- **callback** Ein System.AsyncCallback-Delegat, der aufgerufen wird, wenn die Ausführung des Befehls abgeschlossen ist. Mit der Übergabe von Null (in Microsoft Visual Basic "Nothing") legen Sie fest, dass keine Callback-Prozedur erforderlich ist.
- **stateObject** Ein benutzerdefiniertes Statusobjekt, das an die Callback-Prozedur übergeben wird. Sie rufen dieses Objekt innerhalb der Callback-Prozedur unter Verwendung der System.IAsyncResult.AsyncState-Eigenschaft ab.

Rückgabe

Ein System.IAsyncResult-Wert, der für Abrufe, zum Warten auf Ergebnisse oder für beides verwendet werden kann. Dieser Wert ist auch für den Aufruf des EndExecuteReader(IAsyncResult)-Objekts erforderlich, der ein SADatReader-Objekt zurückgibt, das für die Abfrage der zurückgegebenen Zeilen verwendet werden kann.

Ausnahmebedingungen

- **SAException-Klasse** Jeder Fehler, der während der Ausführung des Befehlstexts aufgetreten ist.

Bemerkungen

Bei asynchronen Befehlen muss die Reihenfolge der Parameter mit CommandText konsistent sein.

Siehe auch

- [SACCommand.EndExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 142](#)
- [SADatReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)

BeginExecuteReader(AsyncCallback, Object, CommandBehavior)-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Server ab.

Visual Basic-Syntax

```
Public Function BeginExecuteReader(  
    ByVal callback As AsyncCallback,  
    ByVal stateObject As Object,  
    ByVal behavior As CommandBehavior  
) As IAsyncResult
```

C#-Syntax

```
public IAsyncResult BeginExecuteReader(  
    AsyncCallback callback,  
    object stateObject,  
    CommandBehavior behavior  
)
```

Parameter

- **callback** Ein System.AsyncCallback-Delegat, der aufgerufen wird, wenn die Ausführung des Befehls abgeschlossen ist. Mit der Übergabe von Null (in Microsoft Visual Basic "Nothing") legen Sie fest, dass keine Callback-Prozedur erforderlich ist.
- **stateObject** Ein benutzerdefiniertes Statusobjekt, das an die Callback-Prozedur übergeben wird. Sie rufen dieses Objekt innerhalb der Callback-Prozedur unter Verwendung der System.IAsyncResult.AsyncState-Eigenschaft ab.
- **behavior** Eine bitweise Kombination von System.Data.CommandBehavior-Parametern, die das Ergebnis der Abfrage und ihre Auswirkung auf die Verbindung beschreibt.

Rückgabe

Ein System.IAsyncResult-Wert, der für Abrufe, zum Warten auf Ergebnisse oder für beides verwendet werden kann. Dieser Wert ist auch für den Aufruf des EndExecuteReader(IAsyncResult)-Objekts erforderlich, der ein SADataReader-Objekt zurückgibt, das für die Abfrage der zurückgegebenen Zeilen verwendet werden kann.

Ausnahmebedingungen

- **SAException-Klasse** Jeder Fehler, der während der Ausführung des Befehlstexts aufgetreten ist.

Bemerkungen

Bei asynchronen Befehlen muss die Reihenfolge der Parameter mit CommandText konsistent sein.

Siehe auch

- [SACommand.EndExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 142](#)
- [SADataReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)

BeginExecuteReader(CommandBehavior)-Methode

Startet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur, die durch dieses SACommand-Objekt beschrieben wird, und ruft mindestens eine Ergebnismenge vom Server ab.

Visual Basic-Syntax

```
Public Function BeginExecuteReader(  
    ByVal behavior As CommandBehavior  
) As IAsyncResult
```

C#-Syntax

```
public IAsyncResult BeginExecuteReader(CommandBehavior behavior)
```

Parameter

- **behavior** Eine bitweise Kombination von System.Data.CommandBehavior-Parametern, die das Ergebnis der Abfrage und ihre Auswirkung auf die Verbindung beschreibt.

Rückgabe

Ein System.IAsyncResult-Wert, der für Abrufe, zum Warten auf Ergebnisse oder für beides verwendet werden kann. Dieser Wert ist auch für den Aufruf des EndExecuteReader(IAsyncResult)-Objekts erforderlich, der ein SADATAReader-Objekt zurückgibt, das für die Abfrage der zurückgegebenen Zeilen verwendet werden kann.

Ausnahmebedingungen

- **SAException-Klasse** Jeder Fehler, der während der Ausführung des Befehlstexts aufgetreten ist.

Bemerkungen

Bei asynchronen Befehlen muss die Reihenfolge der Parameter mit CommandText konsistent sein.

Siehe auch

- [SACommand.EndExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 142](#)
- [SADATAReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)

Cancel-Methode

Bricht die Ausführung eines SACommand-Objekts ab.

Visual Basic-Syntax

```
Public Overrides Sub Cancel()
```

C#-Syntax

```
public override void Cancel()
```

Bemerkungen

Wenn kein Vorgang läuft, der abgebrochen werden könnte, passiert nichts. Wenn die Ausführung eines Befehls läuft, wird die Ausnahmebedingung "Anweisung vom Benutzer unterbrochen" generiert.

CreateParameter-Methode

Stellt ein SAParameter-Objekt bereit, um Parameter an SACommand-Objekte übergeben zu können.

Visual Basic-Syntax

```
Public Shadows Function CreateParameter() As SAParameter
```

C#-Syntax

```
public new SAParameter CreateParameter()
```

Rückgabe

Ein neuer Parameter als SAParameter-Objekt.

Bemerkungen

Gespeicherte Prozeduren und einige andere SQL-Anweisungen können Parameter übernehmen, die im Text einer Anweisung durch ein Fragezeichen (?) angegeben werden.

Die CreateParameter-Methode stellt ein SAParameter-Objekt bereit. Sie können Eigenschaften für das SAParameter-Objekt angeben, um den Wert, den Datentyp und andere Eigenschaften für den Parameter festzulegen.

Siehe auch

- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)

EndExecuteNonQuery-Methode

Beendet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur.

Visual Basic-Syntax

```
Public Function EndExecuteNonQuery(  
    ByVal asyncResult As IAsyncResult  
) As Integer
```

C#-Syntax

```
public int EndExecuteNonQuery(IAsyncResult asyncResult)
```

Parameter

- **asyncResult** Das vom Aufruf von SACommand.BeginExecuteNonQuery zurückgegebene IAsyncResult-Ergebnis.

Rückgabe

Die Anzahl der betroffenen Zeilen (dasselbe Verhalten wie SACommand.ExecuteNonQuery).

Ausnahmebedingungen

- **ArgumentException** Der asyncResult-Parameter ist Null ("Nothing" in Microsoft Visual Basic).

- **InvalidOperationException** `SACCommand.EndExecuteNonQuery(IAsyncResult)` wurde für eine einzelne Befehlsausführung mehrmals aufgerufen oder die Methode stimmte nicht mit der Ausführungsmethode überein.

Bemerkungen

`EndExecuteNonQuery` muss für jeden Aufruf von `BeginExecuteNonQuery` ein Mal aufgerufen werden. Der Aufruf muss erfolgen, nachdem `BeginExecuteNonQuery` ausgeführt wurde. ADO.NET ist nicht threadsicher. Sie müssen dafür Sorge tragen, dass `BeginExecuteNonQuery` abgeschlossen wurde. Das `IAsyncResult`-Ergebnis, das an `EndExecuteNonQuery` übergeben wird, muss mit dem des ausgeführten Aufrufs `BeginExecuteNonQuery` übereinstimmen. Es wird ein Fehler erzeugt, wenn `EndExecuteNonQuery` aufgerufen wird, um einen Aufruf von `BeginExecuteReader` zu beenden, und umgekehrt.

Wenn während der Ausführung des Befehls ein Fehler auftritt, wird eine Ausnahmebedingung generiert, wenn `EndExecuteNonQuery` aufgerufen wird.

Es gibt vier Möglichkeiten, auf den Abschluss der Ausführung zu warten:

(1) Aufruf von `EndExecuteNonQuery`.

Der Aufruf von `EndExecuteNonQuery` wird blockiert, bis der Befehl ausgeführt wurde. Beispiel:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
// this will block until the command completes
int rowCount = cmd.EndExecuteNonQuery( res );
```

(2) Abruf der Eigenschaft `IsCompleted` von `IAsyncResult`.

Sie können die Eigenschaft `IsCompleted` von `IAsyncResult` abrufen. Beispiel:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
    // do other work
}
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );
```

(3) Verwendung der Eigenschaft `IAsyncResult.AsyncWaitHandle`, um ein Synchronisationsobjekt zu erhalten.

Sie können die Eigenschaft `IAsyncResult.AsyncWaitHandle` verwenden, um ein Synchronisationsobjekt abzurufen und darauf zu warten. Beispiel:

```
SAConnection conn = new SAConnection("DSN=SQL Anywhere 16 Demo");
conn.Open();
SACommand cmd = new SACommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );
```

(4) Angabe einer Callback-Funktion beim Aufruf von `BeginExecuteNonQuery`.

Sie können beim Aufruf von `BeginExecuteNonQuery` eine Callback-Funktion angeben. Beispiel:

```
private void callbackFunction( IAsyncResult ar ) {
    SACommand cmd = (SACommand) ar.AsyncState;
    // this won't block since the command has completed
    int rowCount = cmd.EndExecuteNonQuery( ar );
}

// elsewhere in the code
private void DoStuff() {
    SAConnection conn = new SAConnection("DSN=SQL Anywhere 16 Demo");
    conn.Open();
    SACommand cmd = new SACommand(
        "UPDATE Departments"
        + " SET DepartmentName = 'Engineering'"
        + " WHERE DepartmentID=100",
        conn );
    IAsyncResult res = cmd.BeginExecuteNonQuery( callbackFunction, cmd );
    // perform other work. The callback function will be
    // called when the command completes
}
```

Die Callback-Funktion wird in einem separaten Thread ausgeführt, sodass die üblichen Warnungen bezüglich der Aktualisierung der Benutzeroberfläche in einem Programm mit mehreren Threads gelten.

Siehe auch

- [SACommand.BeginExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 134](#)

EndExecuteReader-Methode

Beendet die asynchrone Ausführung einer SQL-Anweisung oder gespeicherten Prozedur und gibt den angeforderten `SADataReader` zurück.

Visual Basic-Syntax

```
Public Function EndExecuteReader(
    ByVal asyncResult As IAsyncResult
) As SADataReader
```

C#-Syntax

```
public SDataReader EndExecuteReader(IAsyncResult asyncResult)
```

Parameter

- **asyncResult** Das vom Aufruf von `SACCommand.BeginExecuteReader` zurückgegebene `IAsyncResult`-Ergebnis.

Rückgabe

Ein `SDataReader`-Objekt, das für die Abfrage der angeforderten Zeilen verwendet werden kann (dasselbe Verhalten wie `SACCommand.ExecuteReader`).

Ausnahmebedingungen

- **ArgumentException** Der `asyncResult`-Parameter ist Null ("Nothing" in Microsoft Visual Basic).
- **InvalidOperationException** `SACCommand.EndExecuteReader(IAsyncResult)` wurde für eine einzelne Befehlsausführung mehrmals aufgerufen oder die Methode stimmt nicht mit der Ausführungsmethode überein.

Bemerkungen

`EndExecuteReader` muss für jeden Aufruf von `BeginExecuteReader` ein Mal aufgerufen werden. Der Aufruf muss erfolgen, nachdem `BeginExecuteReader` ausgeführt wurde. ADO.NET ist nicht threadsicher. Sie müssen dafür Sorge tragen, dass `BeginExecuteReader` abgeschlossen wurde. Das `IAsyncResult`-Ergebnis, das an `EndExecuteReader` übergeben wurde, muss mit dem des ausgeführten Aufrufs `BeginExecuteReader` übereinstimmen. Es wird ein Fehler erzeugt, wenn `EndExecuteReader` aufgerufen wird, um einen Aufruf von `BeginExecuteNonQuery` zu beenden, und umgekehrt.

Wenn während der Ausführung des Befehls ein Fehler auftritt, wird eine Ausnahmebedingung generiert, wenn `EndExecuteReader` aufgerufen wird.

Es gibt vier Möglichkeiten, auf den Abschluss der Ausführung zu warten:

(1) Aufruf von `EndExecuteReader`.

Der Aufruf von `EndExecuteReader` wird blockiert, bis der Befehl ausgeführt wurde. Beispiel:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
// this blocks until the command completes
SDataReader reader = cmd.EndExecuteReader( res );
```

(2) Abruf der Eigenschaft `IsCompleted` von `IAsyncResult`.

Sie können die Eigenschaft `IsCompleted` von `IAsyncResult` abrufen. Beispiel:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 16 Demo");
conn.Open();
SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
```

```
while( !res.IsCompleted ) {  
    // do other work  
}  
// this does not block because the command is finished  
SADataReader reader = cmd.EndExecuteReader( res );
```

(3) Verwendung der Eigenschaft `IAsyncResult.AsyncWaitHandle`, um ein Synchronisationsobjekt zu erhalten.

Sie können die Eigenschaft `IAsyncResult.AsyncWaitHandle` verwenden, um ein Synchronisationsobjekt abzurufen und darauf zu warten. Beispiel:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 16 Demo");  
conn.Open();  
SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );  
IAsyncResult res = cmd.BeginExecuteReader();  
// perform other work  
WaitHandle wh = res.AsyncWaitHandle;  
wh.WaitOne();  
// this does not block because the command is finished  
SADataReader reader = cmd.EndExecuteReader( res );
```

(4) Angabe einer Callback-Funktion beim Aufruf von `BeginExecuteReader`.

Sie können beim Aufruf von `BeginExecuteReader` eine Callback-Funktion angeben. Beispiel:

```
private void callbackFunction( IAsyncResult ar ) {  
    SACCommand cmd = (SACCommand) ar.AsyncState;  
    // this does not block since the command has completed  
    SADataReader reader = cmd.EndExecuteReader();  
}  
  
// elsewhere in the code  
private void DoStuff() {  
    SACConnection conn = new SACConnection("DSN=SQL Anywhere 16 Demo");  
    conn.Open();  
    SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );  
    IAsyncResult res = cmd.BeginExecuteReader( callbackFunction, cmd );  
    // perform other work. The callback function will be  
    // called when the command completes  
}
```

Die Callback-Funktion wird in einem separaten Thread ausgeführt, sodass die üblichen Warnungen bezüglich der Aktualisierung der Benutzeroberfläche in einem Programm mit mehreren Threads gelten.

Siehe auch

- [SACCommand.BeginExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 135](#)
- [SADataReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)

ExecuteNonQuery-Methode

Führt eine Anweisung aus, die keine Ergebnismenge zurückgibt, z.B. eine INSERT-, UPDATE-, DELETE- oder Datendefinitionsanweisung.

Visual Basic-Syntax

```
Public Overrides Function ExecuteNonQuery() As Integer
```

C#-Syntax

```
public override int ExecuteNonQuery()
```

Rückgabe

Die Anzahl der betroffenen Zeilen

Bemerkungen

Sie können ExecuteNonQuery verwenden, um die Daten in einer Datenbank ohne Einsatz eines DataSet zu ändern. Verwenden Sie dafür UPDATE-, INSERT- oder DELETE-Anweisungen.

Obwohl ExecuteNonQuery keine Zeilen zurückgibt, werden die einem Parameter zugewiesenen Ausgabeparameter oder Rückgabewerte mit Daten gefüllt.

Für UPDATE-, INSERT- und DELETE-Anweisungen ist der Rückgabewert die Anzahl von Zeilen, die vom Befehl betroffen sind. Für alle anderen Anweisungstypen und für Rollback-Anweisungen ist der Rückgabewert -1.

Siehe auch

- [SACommand.ExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 145](#)

ExecuteReader-Methode

Führt eine SQL-Anweisung aus, die eine Ergebnismenge zurückgibt.

Überladungsliste

Name	Beschreibung
ExecuteReader()-Methode	Führt eine SQL-Anweisung aus, die eine Ergebnismenge zurückgibt.
ExecuteReader(CommandBehavior)-Methode	Führt eine SQL-Anweisung aus, die eine Ergebnismenge zurückgibt.

ExecuteReader()-Methode

Führt eine SQL-Anweisung aus, die eine Ergebnismenge zurückgibt.

Visual Basic-Syntax

```
Public Shadows Function ExecuteReader() As SDataReader
```

C#-Syntax

```
public new SDataReader ExecuteReader()
```

Rückgabe

Die Ergebnismenge als SDataReader-Objekt.

Bemerkungen

Die Anweisung ist das aktuelle `SACommand`-Objekt, erforderlichenfalls mit den `CommandText`- und `Parameters`-Objekten. Das `SADataReader`-Objekt ist eine schreibgeschützte Ergebnismenge, die nur vorwärts gelesen wird. Wenn Sie änderbare Ergebnismengen benötigen, verwenden Sie ein `SDataAdapter`-Objekt.

Siehe auch

- [SACommand.ExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 144](#)
- [SADataReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)
- [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)
- [SACommand.CommandText-Eigenschaft \[SQL Anywhere .NET\] auf Seite 148](#)
- [SACommand.Parameters-Eigenschaft \[SQL Anywhere .NET\] auf Seite 150](#)

ExecuteReader(CommandBehavior)-Methode

Führt eine SQL-Anweisung aus, die eine Ergebnismenge zurückgibt.

Visual Basic-Syntax

```
Public Shadows Function ExecuteReader(  
    ByVal behavior As CommandBehavior  
) As SADataReader
```

C#-Syntax

```
public new SADataReader ExecuteReader(CommandBehavior behavior)
```

Parameter

- **behavior** Einer der Werte `CloseConnection`, `Default`, `KeyInfo`, `SchemaOnly`, `SequentialAccess`, `SingleResult` oder `SingleRow`. Weitere Hinweise zu diesem Parameter finden Sie in der Dokumentation zum .NET Framework unter der Rubrik "CommandBehavior Enumeration".

Rückgabe

Die Ergebnismenge als `SADataReader`-Objekt.

Bemerkungen

Die Anweisung ist das aktuelle `SACommand`-Objekt, erforderlichenfalls mit den `CommandText`- und `Parameters`-Objekten. Das `SADataReader`-Objekt ist eine schreibgeschützte Ergebnismenge, die nur vorwärts gelesen wird. Wenn Sie änderbare Ergebnismengen benötigen, verwenden Sie ein `SDataAdapter`-Objekt.

Siehe auch

- [SACommand.ExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 144](#)
- [SADataReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)
- [SDataAdapter-Klasse \[SQL Anywhere .NET\] auf Seite 213](#)
- [SACommand.CommandText-Eigenschaft \[SQL Anywhere .NET\] auf Seite 148](#)
- [SACommand.Parameters-Eigenschaft \[SQL Anywhere .NET\] auf Seite 150](#)

ExecuteScalar-Methode

Führt eine Anweisung aus, die einen einzelnen Wert zurückgibt.

Visual Basic-Syntax

```
Public Overrides Function ExecuteScalar() As Object
```

C#-Syntax

```
public override object ExecuteScalar()
```

Rückgabe

Die erste Spalte der ersten Zeile in der Ergebnismenge oder eine NULL-Referenz, wenn die Ergebnismenge leer ist.

Bemerkungen

Wenn diese Methode in einer Abfrage aufgerufen wird, die mehrere Zeilen und Spalten zurückgibt, wird nur die erste Spalte der ersten Zeile zurückgegeben.

Prepare-Methode

Bereitet das SACommand-Objekt für die Datenquelle vor oder kompiliert den Befehl.

Visual Basic-Syntax

```
Public Overrides Sub Prepare()
```

C#-Syntax

```
public override void Prepare()
```

Bemerkungen

Wenn Sie die Methode `ExecuteNonQuery`, `ExecuteReader` oder `ExecuteScalar` aufrufen, nachdem Sie `Prepare` aufgerufen haben, wird jeder Parameterwert, der länger als der in der `Size`-Eigenschaft angegebene Wert ist, automatisch auf die ursprünglich angegebene Parametergröße gekürzt, und es werden keine diesbezüglichen Fehler zurückgegeben.

Nur die folgenden Datentypen werden gekürzt:

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR

- LONG NVARCHAR
- NTEXT
- BINARY
- LONG BINARY
- VARBINARY
- IMAGE

Wenn die Größeneigenschaft nicht festgelegt wurde, sondern der Standardwert verwendet wird, werden die Daten nicht gekürzt.

Siehe auch

- [SACommand.ExecuteNonQuery-Methode \[SQL Anywhere .NET\] auf Seite 144](#)
- [SACommand.ExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 145](#)
- [SACommand.ExecuteScalar-Methode \[SQL Anywhere .NET\] auf Seite 147](#)

ResetCommandTimeout-Methode

Setzt die CommandTimeout-Eigenschaft auf ihren Standardwert von 30 Sekunden zurück.

Visual Basic-Syntax

```
Public Sub ResetCommandTimeout()
```

C#-Syntax

```
public void ResetCommandTimeout()
```

CommandText-Eigenschaft

Ruft den Text einer SQL-Anweisung oder einer gespeicherten Prozedur ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Overrides Property CommandText As String
```

C#-Syntax

```
public override string CommandText {get;set;}
```

Bemerkungen

Die SQL-Anweisung oder der Name der gespeicherten Prozedur, die ausgeführt werden soll. Standardwert ist eine leere Zeichenfolge.

Siehe auch

- [SACommand.SACommand-Konstruktor \[SQL Anywhere .NET\] auf Seite 132](#)

CommandTimeout-Eigenschaft

Diese Funktion wird nicht vom SQL Anywhere .NET-Datenprovider unterstützt.

Visual Basic-Syntax

```
Public Overrides Property CommandTimeout As Integer
```

C#-Syntax

```
public override int CommandTimeout {get;set;}
```

Bemerkungen

Verwenden Sie das folgende Beispiel, um einen Anforderungs-Timeout festzulegen.

```
cmd.CommandText = "SET OPTION request_timeout = 30";
cmd.ExecuteNonQuery();
```

CommandType-Eigenschaft

Ruft den Befehlstyp ab, der durch ein SACommand-Objekt repräsentiert wird, bzw. legt den Befehlstyp fest.

Visual Basic-Syntax

```
Public Overrides Property CommandType As CommandType
```

C#-Syntax

```
public override CommandType CommandType {get;set;}
```

Bemerkungen

Einer der System.Data.CommandType-Werte. Der Standardwert ist System.Data.CommandType.Text.

Folgende Befehlstypen werden unterstützt:

- System.Data.CommandType.StoredProcedure - Wenn Sie dieses CommandType-Objekt angeben, muss der Befehlstext der Name einer gespeicherten Prozedur sein und Sie müssen Argumente als SAParameter-Objekte übergeben.
- System.Data.CommandType.Text - Dies ist der Standardwert.

Wenn die CommandType-Eigenschaft mit StoredProcedure definiert ist, muss die CommandText-Eigenschaft auf den Namen der gespeicherten Prozedur eingestellt werden. Der Befehl führt diese gespeicherte Prozedur aus, wenn Sie eine der Execute-Methoden aufrufen.

Benutzen Sie als Platzhalter ein Fragezeichen (?), um Parameter zu übergeben. Beispiel:

```
SELECT * FROM Customers WHERE ID = ?
```

Die Reihenfolge, in der SAParameter-Objekte der SAParameterCollection hinzugefügt werden, muss direkt der Position des Fragezeichen-Platzhalters für den Parameter entsprechen.

Connection-Eigenschaft

Ruft das Verbindungsobjekt ab, auf das das SACCommand-Objekt anzuwenden ist, bzw. legt das Verbindungsobjekt fest.

Visual Basic-Syntax

```
Public Shadows Property Connection As SAConnection
```

C#-Syntax

```
public new SAConnection Connection {get;set;}
```

Bemerkungen

Der Standardwert ist eine Nullreferenz. In Visual Basic ist er "Nothing".

DesignTimeVisible-Eigenschaft

Ruft einen Wert ab bzw. legt einen Wert fest, der angibt, ob das SACCommand-Objekt in einem Steuerelement eines Windows Form Designers sichtbar sein soll.

Visual Basic-Syntax

```
Public Overrides Property DesignTimeVisible As Boolean
```

C#-Syntax

```
public override bool DesignTimeVisible {get;set;}
```

Bemerkungen

Der Standardwert ist TRUE.

TRUE, wenn diese SACCommand-Instanz sichtbar sein soll, andernfalls FALSE. Der Standardwert ist FALSE.

Parameters-Eigenschaft

Eine Sammlung von Parametern für die aktuelle Anweisung.

Visual Basic-Syntax

```
Public ReadOnly Shadows Property Parameters As SAParameterCollection
```

C#-Syntax

```
public new SAParameterCollection Parameters {get;}
```

Bemerkungen

Benutzen Sie zur Angabe von Parametern im CommandText-Objekt Fragezeichen.

Die Parameter der SQL-Anweisung oder der gespeicherten Prozedur. Der Standardwert ist eine leere Sammlung.

Wenn CommandType auf Text gesetzt ist, übergeben Sie Parameter mit dem Fragezeichen-Platzhalter.
Beispiel:

```
SELECT * FROM Customers WHERE ID = ?
```

Die Reihenfolge, in der SAParameter-Objekte der SAParameterCollection-Sammlung hinzugefügt werden, muss direkt der Position des Fragezeichen-Platzhalters für den Parameter im Befehlstext entsprechen.

Wenn die Parameter in der Sammlung den Anforderungen der auszuführenden Abfrage nicht entsprechen, kann ein Fehler auftreten oder eine Ausnahmebedingung generiert werden.

Siehe auch

- [SAParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)

Transaction-Eigenschaft

Gibt das SATransaction-Objekt an, in der das SACommand-Objekt ausgeführt wird.

Visual Basic-Syntax

```
Public Shadows Property Transaction As SATransaction
```

C#-Syntax

```
public new SATransaction Transaction {get;set;}
```

Bemerkungen

Der Standardwert ist eine Nullreferenz. In Visual Basic ist dieser Wert "Nothing".

Sie können die Transaction-Eigenschaft nicht definieren, wenn sie bereits auf einen bestimmten Wert eingestellt ist und der Befehl ausgeführt wird. Wenn Sie die Transaction-Eigenschaft für ein SATransaction-Objekt definieren, das nicht mit demselben SAConnection-Objekt verbunden ist wie das SACommand-Objekt, wird eine Ausnahmebedingung generiert, wenn Sie das nächste Mal versuchen, eine Anweisung auszuführen.

Weitere Hinweise finden Sie unter „[Transaktionsverarbeitung](#)“ auf Seite 64.

Siehe auch

- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)

UpdatedRowSource-Eigenschaft

Legt fest oder ruft ab, wie Ergebnisse von Befehlen auf das DataRow-Objekt angewendet werden, wenn sie von der Update-Methode des SDataAdapter-Objekts verwendet werden.

Visual Basic-Syntax

```
Public Overrides Property UpdatedRowSource As UpdateRowSource
```

C#-Syntax

```
public override UpdateRowSource UpdatedRowSource {get;set;}
```

Bemerkungen

Einer der UpdatedRowSource-Werte. Der Standardwert ist UpdateRowSource.OutputParameters. Wenn der Befehl automatisch generiert wird, hat diese Eigenschaft den Wert UpdateRowSource.None.

UpdatedRowSource.Both, mit dem sowohl Ergebnismenge als auch Ausgabeparameter zurückgegeben werden, wird nicht unterstützt.

SACommandBuilder-Klasse

Eine Möglichkeit zur Generierung von SQL-Anweisungen mit einzelnen Tabellen, bei der Änderungen an einem DataSet mit den Daten in der entsprechenden Datenbank abgeglichen werden.

Visual Basic-Syntax

```
Public NotInheritable Class SACommandBuilder  
    Inherits System.Data.Common.DbCommandBuilder
```

C#-Syntax

```
public sealed class SACommandBuilder :  
    System.Data.Common.DbCommandBuilder
```

Basisklassen

- [System.Data.Common.DbCommandBuilder](#)

Mitglieder

Alle Mitglieder der SACommandBuilder-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SACommandBuilder-Konstruktor	Initialisiert ein SACommandBuilder-Objekt.
ApplyParameterInfo-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Ermöglicht es der Provider-Implementierung der Klasse System.Data.Common.DbCommandBuilder , zusätzliche Parametereigenschaften zu verarbeiten.
DeriveParameters-Methode	Füllt die Parametersammlung des angegebenen SACommand-Objekts auf.
Dispose-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Gibt die unverwalteten Ressourcen frei, die von System.Data.Common.DbCommandBuilder verwendet werden, und optional auch die verwalteten Ressourcen.

Name	Beschreibung
GetDeleteCommand-Methode	Gibt das generierte SCommand-Objekt zurück, das DELETE-Vorgänge in der Datenbank ausführt, wenn SDataAdapter.Update aufgerufen wird.
GetInsertCommand-Methode	Gibt das generierte SCommand-Objekt zurück, das INSERT-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.
GetParameterName-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Gibt den Namen des angegebenen Parameters im Format @p# zurück.
GetParameterPlaceholder-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Gibt den Platzhalter für den Parameter in der zugeordneten SQL-Anweisung zurück.
GetSchemaTable-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Gibt die Schematabelle für System.Data.Common.DbCommandBuilder zurück.
GetUpdateCommand-Methode	Gibt das generierte SCommand-Objekt zurück, das UPDATE-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.
InitializeCommand-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Setzt die Eigenschaften System.Data.Common.DbCommand.CommandTimeout , System.Data.Common.DbCommand.Transaction , System.Data.Common.DbCommand.CommandType und System.Data.UpdateRowSource für das System.Data.Common.DbCommand -Objekt zurück.
QuoteIdentifier-Methode	Gibt einen Bezeichner ohne Anführungszeichen in der richtigen Schreibweise mit Anführungszeichen zurück, wobei auch Escapezeichen für eingebettete Anführungszeichen in den Bezeichner aufgenommen werden.
RefreshSchema-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Löscht die zugeordneten Befehle für dieses System.Data.Common.DbCommandBuilder -Objekt.
RowUpdatingHandler-Methode (geerbt aus System.Data.Common.DbCommandBuilder)	Fügt einen Event-Handler für das System.Data.OleDb.OleDbDataAdapter.RowUpdating -Ereignis hinzu.

Name	Beschreibung
SetRowUpdatingHandler-Methode (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Registriert die <code>System.Data.Common.DbCommandBuilder</code> für die Verarbeitung des <code>System.Data.OleDb.OleDbDataAdapter.RowUpdating</code> -Ereignisses für ein <code>System.Data.Common.DbDataAdapter</code> -Objekt
UnquoteIdentifier-Methode	Gibt einen Bezeichner mit Anführungszeichen in der richtigen Schreibweise ohne Anführungszeichen zurück, wobei auch Escapezeichen und eingebettete Anführungszeichen entsprechend entfernt werden.
CatalogLocation-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Legt das <code>CatalogLocation</code> -Objekt für eine Instanz der Klasse <code>System.Data.Common.DbCommandBuilder</code> fest bzw. ruft es ab.
CatalogSeparator-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Legt eine Zeichenfolge fest, die als Katalogtrennzeichen für eine Instanz der Klasse <code>System.Data.Common.DbCommandBuilder</code> verwendet wird, bzw. ruft sie ab.
ConflictOption-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Gibt an, welches <code>System.Data.ConflictOption</code> -Objekt von <code>System.Data.Common.DbCommandBuilder</code> verwendet werden soll.
DataAdapter-Eigenschaft	Legt das <code>SADDataAdapter</code> -Objekt fest, für das Anweisungen zu generieren sind.
QuotePrefix-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Ruft das oder die Anfangszeichen ab, die bei der Angabe von Datenbankobjekten (wie z.B. Tabellen oder Spalten), deren Namen z.B. Leerzeichen oder reservierte Tokens enthalten, verwendet werden sollen, bzw. legt die betreffenden Zeichen fest.
QuoteSuffix-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Ruft die Endzeichen ab, die beim Angeben von Datenbankobjekten (z.B. Tabellen oder Spalten), deren Namen Zeichen wie Leerzeichen oder reservierte Tokens enthalten, verwendet werden sollen, bzw. legt diese Zeichen fest.
SchemaSeparator-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Ruft das Zeichen ab, das als Trennzeichen zwischen dem Schema-bezeichner und anderen Bezeichnern verwendet werden soll, oder legt es fest.
SetAllValues-Eigenschaft (geerbt aus <code>System.Data.Common.DbCommandBuilder</code>)	Gibt an, ob alle Spaltenwerte in einer UPDATE-Anweisung oder nur die geänderten Werte aufgenommen werden.

SACommandBuilder-Konstruktor

Initialisiert ein SACommandBuilder-Objekt.

Überladungsliste

Name	Beschreibung
SACommandBuilder()-Konstruktor	Initialisiert ein SACommandBuilder-Objekt.
SACommandBuilder(SADataAdapter)-Konstruktor	Initialisiert ein SACommandBuilder-Objekt.

SACommandBuilder()-Konstruktor

Initialisiert ein SACommandBuilder-Objekt.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SACommandBuilder()
```

SACommandBuilder(SADataAdapter)-Konstruktor

Initialisiert ein SACommandBuilder-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal adapter As SADataAdapter)
```

C#-Syntax

```
public SACommandBuilder(SADataAdapter adapter)
```

Parameter

- **adapter** Ein SADataAdapter-Objekt, für das Anweisungen zum Datenabgleich generiert werden.

DeriveParameters-Methode

Füllt die Parametersammlung des angegebenen SACommand-Objekts auf.

Visual Basic-Syntax

```
Public Shared Sub DeriveParameters(ByVal command As SACommand)
```

C#-Syntax

```
public static void DeriveParameters(SACommand command)
```

Parameter

- **command** Ein SACCommand-Objekt, für das Parameter abgeleitet werden.

Bemerkungen

Dies wird für die gespeicherte Prozedur verwendet, die im SACCommand-Objekt angegeben wird.

DeriveParameters überschreibt etwaige bestehende Parameterinformationen für das SACCommand-Objekt.

DeriveParameters erfordert einen eigenen Aufruf an den Datenbankserver. Wenn die Parameterinformationen von vornherein bekannt sind, dürfte es sinnvoller sein, die Parametersammlung mit Daten zu versehen, indem die Informationen explizit definiert werden.

GetDeleteCommand-Methode

Gibt das generierte SACCommand-Objekt zurück, das DELETE-Vorgänge in der Datenbank ausführt, wenn SDataAdapter.Update aufgerufen wird.

Überladungsliste

Name	Beschreibung
GetDeleteCommand()-Methode	Gibt das generierte SACCommand-Objekt zurück, das DELETE-Vorgänge in der Datenbank ausführt, wenn SDataAdapter.Update aufgerufen wird.
GetDeleteCommand(bool)-Methode	Gibt das generierte SACCommand-Objekt zurück, das DELETE-Vorgänge in der Datenbank ausführt, wenn SDataAdapter.Update aufgerufen wird.

GetDeleteCommand()-Methode

Gibt das generierte SACCommand-Objekt zurück, das DELETE-Vorgänge in der Datenbank ausführt, wenn SDataAdapter.Update aufgerufen wird.

Visual Basic-Syntax

```
Public Shadows Function GetDeleteCommand() As SACCommand
```

C#-Syntax

```
public new SACCommand GetDeleteCommand()
```

Rückgabe

Das automatisch generierte SACCommand-Objekt, das für die Durchführung von Löschvorgängen erforderlich ist.

Bemerkungen

Die GetDeleteCommand-Methode gibt das auszuführende SACCommand-Objekt zurück und kann daher für Informationszwecke oder die Fehlersuche eingesetzt werden.

Sie können `GetDeleteCommand` auch als Basis eines modifizierten Befehls verwenden. Beispielsweise können Sie `GetDeleteCommand` aufrufen, den `CommandTimeout`-Wert ändern und dann diesen Wert explizit für das `SADDataAdapter`-Objekt definieren.

SQL-Anweisungen werden erst generiert, wenn die Anwendung `Update` oder `GetDeleteCommand` aufruft. Nachdem die SQL-Anweisung generiert wurde, muss die Anwendung explizit `RefreshSchema` aufrufen, sofern sie die Anweisung ändert. Sonst verwendet `GetDeleteCommand` Informationen aus der vorhergehenden Anweisung.

Siehe auch

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

GetDeleteCommand(bool)-Methode

Gibt das generierte `SACommand`-Objekt zurück, das `DELETE`-Vorgänge in der Datenbank ausführt, wenn `SADDataAdapter.Update` aufgerufen wird.

Visual Basic-Syntax

```
Public Shadows Function GetDeleteCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As SACommand
```

C#-Syntax

```
public new SACommand GetDeleteCommand(bool useColumnsForParameterNames)
```

Parameter

- **useColumnsForParameterNames** Wenn der Wert `TRUE` ergibt, werden Parameternamen generiert, die mit dem Spaltennamen übereinstimmen, sofern möglich. Wenn der Wert `FALSE` ergibt, werden `@p1`, `@p2` usw. generiert.

Rückgabe

Das automatisch generierte `SACommand`-Objekt, das für die Durchführung von Löschvorgängen erforderlich ist.

Bemerkungen

Die `GetDeleteCommand`-Methode gibt das auszuführende `SACommand`-Objekt zurück und kann daher für Informationszwecke oder die Fehlersuche eingesetzt werden.

Sie können `GetDeleteCommand` auch als Basis eines modifizierten Befehls verwenden. Beispielsweise können Sie `GetDeleteCommand` aufrufen, den `CommandTimeout`-Wert ändern und dann diesen Wert explizit für das `SADDataAdapter`-Objekt definieren.

SQL-Anweisungen werden erst generiert, wenn die Anwendung `Update` oder `GetDeleteCommand` aufruft. Nachdem die SQL-Anweisung generiert wurde, muss die Anwendung explizit `RefreshSchema` aufrufen, sofern sie die Anweisung ändert. Sonst verwendet `GetDeleteCommand` Informationen aus der vorhergehenden Anweisung.

Siehe auch

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

GetInsertCommand-Methode

Gibt das generierte SCommand-Objekt zurück, das INSERT-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

Überladungsliste

Name	Beschreibung
GetInsertCommand()-Methode	Gibt das generierte SCommand-Objekt zurück, das INSERT-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.
GetInsertCommand(bool)-Methode	Gibt das generierte SCommand-Objekt zurück, das INSERT-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

GetInsertCommand()-Methode

Gibt das generierte SCommand-Objekt zurück, das INSERT-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

Visual Basic-Syntax

```
Public Shadows Function GetInsertCommand() As SCommand
```

C#-Syntax

```
public new SCommand GetInsertCommand()
```

Rückgabe

Das automatisch generierte SCommand-Objekt, das für die Durchführung von Einfügungen erforderlich ist.

Bemerkungen

Die GetInsertCommand-Methode gibt das auszuführende SCommand-Objekt zurück und kann daher für Informationszwecke oder die Fehlersuche eingesetzt werden.

Sie können GetInsertCommand auch als Basis eines modifizierten Befehls verwenden. Beispielsweise können Sie GetInsertCommand aufrufen, den CommandTimeout-Wert ändern und dann diesen Wert explizit für den SDataAdapter definieren.

SQL-Anweisungen werden erst generiert, wenn die Anwendung Update oder GetInsertCommand aufruft. Nachdem die SQL-Anweisung generiert wurde, muss die Anwendung explizit RefreshSchema aufrufen,

sofern sie die Anweisung ändert. Sonst verwendet `GetInsertCommand` Informationen aus der vorhergehenden Anweisung, die nicht unbedingt richtig sein müssen.

Siehe auch

- [SACommandBuilder.GetDeleteCommand-Methode \[SQL Anywhere .NET\] auf Seite 156](#)

GetInsertCommand(bool)-Methode

Gibt das generierte `SACommand`-Objekt zurück, das INSERT-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

Visual Basic-Syntax

```
Public Shadows Function GetInsertCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As SACommand
```

C#-Syntax

```
public new SACommand GetInsertCommand(bool useColumnsForParameterNames)
```

Parameter

- **useColumnsForParameterNames** Wenn der Wert `TRUE` ergibt, werden Parameternamen generiert, die mit dem Spaltennamen übereinstimmen, sofern möglich. Wenn der Wert `FALSE` ergibt, werden `@p1`, `@p2` usw. generiert.

Rückgabe

Das automatisch generierte `SACommand`-Objekt, das für die Durchführung von Einfügungen erforderlich ist.

Bemerkungen

Die `GetInsertCommand`-Methode gibt das auszuführende `SACommand`-Objekt zurück und kann daher für Informationszwecke oder die Fehlersuche eingesetzt werden.

Sie können `GetInsertCommand` auch als Basis eines modifizierten Befehls verwenden. Beispielsweise können Sie `GetInsertCommand` aufrufen, den `CommandTimeout`-Wert ändern und dann diesen Wert explizit für den `SADDataAdapter` definieren.

SQL-Anweisungen werden erst generiert, wenn die Anwendung `Update` oder `GetInsertCommand` aufruft. Nachdem die SQL-Anweisung generiert wurde, muss die Anwendung explizit `RefreshSchema` aufrufen, sofern sie die Anweisung ändert. Sonst verwendet `GetInsertCommand` Informationen aus der vorhergehenden Anweisung, die nicht unbedingt richtig sein müssen.

Siehe auch

- [SACommandBuilder.GetDeleteCommand-Methode \[SQL Anywhere .NET\] auf Seite 156](#)

GetUpdateCommand-Methode

Gibt das generierte SCommand-Objekt zurück, das UPDATE-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

Überladungsliste

Name	Beschreibung
GetUpdateCommand()-Methode	Gibt das generierte SCommand-Objekt zurück, das UPDATE-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.
GetUpdateCommand(bool)-Methode	Gibt das generierte SCommand-Objekt zurück, das UPDATE-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

GetUpdateCommand()-Methode

Gibt das generierte SCommand-Objekt zurück, das UPDATE-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

Visual Basic-Syntax

```
Public Shadows Function GetUpdateCommand() As SCommand
```

C#-Syntax

```
public new SCommand GetUpdateCommand()
```

Rückgabe

Das automatisch generierte SCommand-Objekt, das für die Durchführung von Aktualisierungen erforderlich ist.

Bemerkungen

Die GetUpdateCommand-Methode gibt das auszuführende SCommand-Objekt zurück und kann daher für Informationszwecke oder die Fehlersuche eingesetzt werden.

Sie können GetUpdateCommand auch als Basis eines modifizierten Befehls verwenden. Beispielsweise können Sie GetUpdateCommand aufrufen, den CommandTimeout-Wert ändern und dann diesen Wert explizit für den SDataAdapter definieren.

SQL-Anweisungen werden erst generiert, wenn die Anwendung Update oder GetUpdateCommand aufruft. Nachdem die SQL-Anweisung generiert wurde, muss die Anwendung explizit RefreshSchema aufrufen, sofern sie die Anweisung ändert. Sonst verwendet GetUpdateCommand Informationen aus der vorhergehenden Anweisung, die nicht unbedingt richtig sein müssen.

Siehe auch

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

GetUpdateCommand(bool)-Methode

Gibt das generierte SACommand-Objekt zurück, das UPDATE-Vorgänge für die Datenbank ausführt, wenn eine Update-Anweisung aufgerufen wird.

Visual Basic-Syntax

```
Public Shadows Function GetUpdateCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As SACommand
```

C#-Syntax

```
public new SACommand GetUpdateCommand(bool useColumnsForParameterNames)
```

Parameter

- **useColumnsForParameterNames** Wenn der Wert TRUE ergibt, werden Parameternamen generiert, die mit dem Spaltennamen übereinstimmen, sofern möglich. Wenn der Wert FALSE ergibt, werden @p1, @p2 usw. generiert.

Rückgabe

Das automatisch generierte SACommand-Objekt, das für die Durchführung von Aktualisierungen erforderlich ist.

Bemerkungen

Die GetUpdateCommand-Methode gibt das auszuführende SACommand-Objekt zurück und kann daher für Informationszwecke oder die Fehlersuche eingesetzt werden.

Sie können GetUpdateCommand auch als Basis eines modifizierten Befehls verwenden. Beispielsweise können Sie GetUpdateCommand aufrufen, den CommandTimeout-Wert ändern und dann diesen Wert explizit für den SDataAdapter definieren.

SQL-Anweisungen werden erst generiert, wenn die Anwendung Update oder GetUpdateCommand aufruft. Nachdem die SQL-Anweisung generiert wurde, muss die Anwendung explizit RefreshSchema aufrufen, sofern sie die Anweisung ändert. Sonst verwendet GetUpdateCommand Informationen aus der vorhergehenden Anweisung, die nicht unbedingt richtig sein müssen.

Siehe auch

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

Quoteldentifizier-Methode

Gibt einen Bezeichner ohne Anführungszeichen in der richtigen Schreibweise mit Anführungszeichen zurück, wobei auch Escapezeichen für eingebettete Anführungszeichen in den Bezeichner aufgenommen werden.

Visual Basic-Syntax

```
Public Overrides Function QuoteIdentifier(  
    ByVal unquotedIdentifier As String  
    ) As String
```

C#-Syntax

```
public override string QuoteIdentifier(string unquotedIdentifier)
```

Parameter

- **unquotedIdentifier** Die Zeichenfolge, die den Bezeichner ohne Anführungszeichen darstellt, dem Anführungszeichen zugeordnet werden.

Rückgabe

Gibt eine Zeichenfolge zurück, die einen Bezeichner ohne Anführungszeichen in der Schreibweise mit Anführungszeichen darstellt, wobei die Escapezeichen der eingebetteten Anführungszeichen aufgenommen werden.

UnquoteIdentifier-Methode

Gibt einen Bezeichner mit Anführungszeichen in der richtigen Schreibweise ohne Anführungszeichen zurück, wobei auch Escapezeichen und eingebettete Anführungszeichen entsprechend entfernt werden.

Visual Basic-Syntax

```
Public Overrides Function UnquoteIdentifier(  
    ByVal quotedIdentifier As String  
    ) As String
```

C#-Syntax

```
public override string UnquoteIdentifier(string quotedIdentifier)
```

Parameter

- **quotedIdentifier** Die Zeichenfolge, die den zwischen Anführungszeichen stehenden Bezeichner darstellt, dessen eingebettete Anführungszeichen entfernt werden.

Rückgabe

Gibt eine Zeichenfolge zurück, die einen zwischen Anführungszeichen stehenden Bezeichner in der Schreibweise ohne Anführungszeichen darstellt, wobei die Escapezeichen der eingebetteten Anführungszeichen entsprechend entfernt werden.

DataAdapter-Eigenschaft

Legt das SqlDataAdapter-Objekt fest, für das Anweisungen zu generieren sind.

Visual Basic-Syntax

```
Public Shadows Property DataAdapter As SqlDataAdapter
```

C#-Syntax

```
public new SDataAdapter DataAdapter {get;set;}
```

Bemerkungen

Ein SDataAdapter-Objekt.

Wenn Sie eine neue Instanz der Klasse SACommandBuilder erstellen, wird jeder bestehende SACommandBuilder, der mit diesem SDataAdapter verbunden ist, freigegeben.

SACommLinksOptionsBuilder-Klasse

Bietet eine einfache Möglichkeit zum Erstellen und Verwalten des Teils der CommLinks-Optionen von Verbindungszeichenfolgen, die von der Klasse SAConnection verwendet werden.

Visual Basic-Syntax

```
Public NotInheritable Class SACommLinksOptionsBuilder
```

C#-Syntax

```
public sealed class SACommLinksOptionsBuilder
```

Mitglieder

Alle Mitglieder der SACommLinksOptionsBuilder-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SACommLinksOptionsBuilder-Konstruktor	Initialisiert ein SACommLinksOptionsBuilder-Objekt.
GetUseLongNameAsKey-word-Methode	Ruft einen booleschen Wert ab, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
SetUseLongNameAsKey-word-Methode	Legt einen booleschen Wert fest, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
ToString-Methode	Konvertiert das SACommLinksOptionsBuilder-Objekt in eine Zeichenfolgedarstellung.
All-Eigenschaft	Ruft die CommLinks-Option ALL ab bzw. legt sie fest.
ConnectionString-Eigenschaft	Ruft die zu erstellende Verbindungszeichenfolge ab bzw. legt sie fest.
SharedMemory-Eigenschaft	Ruft das SharedMemory-Protokoll ab bzw. legt es fest.
TcpOptionsBuilder-Eigenschaft	Ruft ein SATcpOptionsBuilder-Objekt ab bzw. legt das SATcpOptionsBuilder-Objekt fest, das zum Erstellen einer TCP-Optionszeichenfolge verwendet wird.

Name	Beschreibung
TcpOptionsString-Eigenschaft	Ruft eine Zeichenfolge von TCP-Optionen ab bzw. legt sie fest.

Bemerkungen

Die SACommLinksOptionsBuilder-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

Eine Liste der Verbindungsparameter finden Sie unter „[Verbindungsparameter](#)“ [*SQL Anywhere Server - Datenbankadministration*].

SACommLinksOptionsBuilder-Konstruktor

Initialisiert ein SACommLinksOptionsBuilder-Objekt.

Überladungsliste

Name	Beschreibung
SACommLinksOptionsBuilder()-Konstruktor	Initialisiert ein SACommLinksOptionsBuilder-Objekt.
SACommLinksOptionsBuilder(String)-Konstruktor	Initialisiert ein SACommLinksOptionsBuilder-Objekt.

SACommLinksOptionsBuilder()-Konstruktor

Initialisiert ein SACommLinksOptionsBuilder-Objekt.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SACommLinksOptionsBuilder()
```

Bemerkungen

Die SACommLinksOptionsBuilder-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

Beispiel

Die nachstehende Anweisung initialisiert ein SACommLinksOptionsBuilder-Objekt.

```
SACommLinksOptionsBuilder commLinks = new SACommLinksOptionsBuilder();
```

SACommLinksOptionsBuilder(String)-Konstruktor

Initialisiert ein SACommLinksOptionsBuilder-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal options As String)
```

C#-Syntax

```
public SACommLinksOptionsBuilder(string options)
```

Parameter

- **options** Eine Zeichenfolge von SQL Anywhere-Verbindungsparametern. Eine Liste der Verbindungsparameter finden Sie unter „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)].

Bemerkungen

Die SACommLinksOptionsBuilder-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

Beispiel

Die nachstehende Anweisung initialisiert ein SACommLinksOptionsBuilder-Objekt.

```
SACommLinksOptionsBuilder commLinks =  
    new SACommLinksOptionsBuilder("TCPIP(DoBroadcast=ALL;Timeout=20)");
```

GetUseLongNameAsKeyword-Methode

Ruft einen booleschen Wert ab, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.

Visual Basic-Syntax

```
Public Function GetUseLongNameAsKeyword() As Boolean
```

C#-Syntax

```
public bool GetUseLongNameAsKeyword()
```

Rückgabe

TRUE, wenn zur Erstellung von Verbindungszeichenfolgen lange Verbindungsparameternamen verwendet werden, andernfalls FALSE.

Bemerkungen

Für die Namen der SQL Anywhere-Verbindungsparameter gibt es sowohl lange als auch kurze Formen. Um z.B. den Namen einer ODBC-Datenquelle in der Verbindungszeichenfolge anzugeben, können Sie einen der beiden folgenden Werte verwenden: DataSourceName oder DSN. Standardmäßig werden zur Erstellung von Verbindungszeichenfolgen lange Verbindungsparameternamen verwendet.

Siehe auch

- [SACommLinksOptionsBuilder.SetUseLongNameAsKeyword-Methode](#) [SQL Anywhere .NET] auf Seite 166

SetUseLongNameAsKeyword-Methode

Legt einen booleschen Wert fest, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.

Visual Basic-Syntax

```
Public Sub SetUseLongNameAsKeyword(  
    ByVal useLongNameAsKeyword As Boolean  
)
```

C#-Syntax

```
public void SetUseLongNameAsKeyword(bool useLongNameAsKeyword)
```

Parameter

- **useLongNameAsKeyword** Ein boolescher Wert, der angibt, ob in der Verbindungszeichenfolge ein langer Verbindungsparameternamen verwendet wird.

Bemerkungen

Lange Verbindungsparameternamen werden standardmäßig verwendet.

Siehe auch

- [SACommLinksOptionsBuilder.GetUseLongNameAsKeyword-Methode \[SQL Anywhere .NET\] auf Seite 165](#)

ToString-Methode

Konvertiert das SACommLinksOptionsBuilder-Objekt in eine Zeichenfolgedarstellung.

Visual Basic-Syntax

```
Public Overrides Function ToString() As String
```

C#-Syntax

```
public override string ToString()
```

Rückgabe

Die erstellte Optionszeichenfolge.

All-Eigenschaft

Ruft die CommLinks-Option ALL ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property All As Boolean
```

C#-Syntax

```
public bool All {get;set;}
```

Bemerkungen

Erster Versuch einer Shared Memory-Verbindung, danach mit den restlichen Kommunikationsprotokollen. Benutzen Sie diese Einstellung, wenn Sie nicht wissen, welche Verbindungsprotokolle zu benutzen sind.

Die SACommLinksOptionsBuilder-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

ConnectionString-Eigenschaft

Ruft die zu erstellende Verbindungszeichenfolge ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ConnectionString As String
```

C#-Syntax

```
public string ConnectionString {get;set;}
```

Bemerkungen

Die SACommLinksOptionsBuilder-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

SharedMemory-Eigenschaft

Ruft das SharedMemory-Protokoll ab bzw. legt es fest.

Visual Basic-Syntax

```
Public Property SharedMemory As Boolean
```

C#-Syntax

```
public bool SharedMemory {get;set;}
```

Bemerkungen

Die SACommLinksOptionsBuilder-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

TcpOptionsBuilder-Eigenschaft

Ruft ein SATcpOptionsBuilder-Objekt ab bzw. legt das SATcpOptionsBuilder-Objekt fest, das zum Erstellen einer TCP-Optionszeichenfolge verwendet wird.

Visual Basic-Syntax

```
Public Property TcpOptionsBuilder As SATcpOptionsBuilder
```

C#-Syntax

```
public SATcpOptionsBuilder TcpOptionsBuilder {get;set;}
```

TcpOptionsString-Eigenschaft

Ruft eine Zeichenfolge von TCP-Optionen ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property TcpOptionsString As String
```

C#-Syntax

```
public string TcpOptionsString {get;set;}
```

SACConnection-Klasse

Repräsentiert eine Verbindung zu einer SQL Anywhere-Datenbank.

Visual Basic-Syntax

```
Public NotInheritable Class SACConnection  
    Inherits System.Data.Common.DbConnection
```

C#-Syntax

```
public sealed class SACConnection : System.Data.Common.DbConnection
```

Basisklassen

- [System.Data.Common.DbConnection](#)

Mitglieder

Alle Mitglieder der SACConnection-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SACConnection-Konstruktor	Initialisiert ein SACConnection-Objekt.
BeginDbTransaction-Methode (geerbt aus System.Data.Common.DbConnection)	Startet eine Datenbanktransaktion.
BeginTransaction-Methode	Gibt ein Transaktionsobjekt zurück.
ChangeDatabase-Methode	Ändert die aktuelle Datenbank für ein geöffnetes SACConnection-Objekt.

Name	Beschreibung
ChangePassword-Methode	Ändert das Kennwort für den Benutzer, der in der Verbindungszeichenfolge für das bereitgestellte neue Kennwort angegeben ist.
ClearAllPools-Methode	Leert alle Verbindungspools.
ClearPool-Methode	Leert den Verbindungspool, der der angegebenen Verbindung zugeordnet ist.
Close-Methode	Schließt eine Datenbankverbindung.
CreateCommand-Methode	Initialisiert ein <code>SACCommand</code> -Objekt.
EnlistDistributedTransaction-Methode	Nimmt die angegebene Transaktion als verteilte Transaktion in die Liste auf.
EnlistTransaction-Methode	Nimmt die angegebene Transaktion als verteilte Transaktion in die Liste auf.
GetSchema-Methode	Gibt die Liste der unterstützten Schemasammlungen zurück.
OnStateChange-Methode (geerbt aus <code>System.Data.Common.DbConnection</code>)	Löst das System.Data.Common.DbConnection.StateChange -Ereignis aus.
Open-Methode	Öffnet eine Datenbankverbindung mit den durch das <code>SACconnection.ConnectionString</code> -Objekt angegebenen Eigenschaftseinstellungen.
OpenAsync-Methode (geerbt aus <code>System.Data.Common.DbConnection</code>)	Eine asynchrone Version von System.Data.Common.DbConnection.Open , die eine Datenbankverbindung mit den durch System.Data.Common.DbConnection.ConnectionString angegebenen Einstellungen öffnet.
ConnectionString-Eigenschaft	Stellt die Datenbank-Verbindungszeichenfolge bereit.
ConnectionTimeout-Eigenschaft	Ruft die Zeitspanne in Sekunden ab, nach deren Ablauf der Verbindungsversuch mit einer Fehlermeldung abgebrochen wird.
Database-Eigenschaft	Ruft den Namen der aktuellen Datenbank ab.
DataSource-Eigenschaft	Ruft den Namen des Datenbankservers ab.
DbProviderFactory-Eigenschaft (geerbt aus <code>System.Data.Common.DbConnection</code>)	Ruft das System.Data.Common.DbProviderFactory -Objekt für diese System.Data.Common.DbConnection ab.

Name	Beschreibung
InitString-Eigenschaft	Ein Befehl, der unmittelbar nach Herstellung der Verbindung ausgeführt wird.
ServerVersion-Eigenschaft	Ruft eine Zeichenfolge ab, die die Version der Instanz von SQL Anywhere enthält, mit der der Client verbunden ist.
State-Eigenschaft	Gibt den Status des SAConnection-Objekts an.
InfoMessage-Ereignis	Tritt ein, wenn der SQL Anywhere-Datenbankserver eine Warnung oder eine Informationsmeldung zurückgibt.
StateChange-Ereignis	Tritt ein, wenn sich der Status des SAConnection-Objekts ändert.

Bemerkungen

Eine Liste der Verbindungsparameter finden Sie unter „[Verbindungsparameter](#)“ [*SQL Anywhere Server - Datenbankadministration*].

SAConnection-Konstruktor

Initialisiert ein SAConnection-Objekt.

Überladungsliste

Name	Beschreibung
SAConnection()-Konstruktor	Initialisiert ein SAConnection-Objekt.
SAConnection(String)-Konstruktor	Initialisiert ein SAConnection-Objekt.

SAConnection()-Konstruktor

Initialisiert ein SAConnection-Objekt.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SAConnection()
```

Bemerkungen

Die Verbindung muss geöffnet werden, damit Sie Vorgänge mit der Datenbank ausführen können.

SACConnection(String)-Konstruktor

Initialisiert ein SACConnection-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal connectionString As String)
```

C#-Syntax

```
public SACConnection(string connectionString)
```

Parameter

- **connectionString** Eine SQL Anywhere-Verbindungszeichenfolge. Eine Verbindungszeichenfolge ist eine durch Semikola getrennte Liste von Schlüsselwort=Wert-Paaren. Eine Liste der Verbindungsparameter finden Sie unter „[Verbindungsparameter](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

Bemerkungen

Die Verbindung muss geöffnet werden, damit Sie Vorgänge mit der Datenbank ausführen können.

Siehe auch

- [SACConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

Beispiel

Die folgende Anweisung initialisiert ein SACConnection-Objekt für eine Verbindung mit der Datenbank "policies", die auf einem SQL Anywhere-Datenbankserver mit dem Namen "hr" läuft. Die Verbindung verwendet die Benutzer-ID "admin" und das Kennwort "money".

```
SACConnection conn = new SACConnection(
    "UID=admin;PWD=money;SERVER=hr;DBN=policies" );
conn.Open();
```

BeginTransaction-Methode

Gibt ein Transaktionsobjekt zurück.

Überladungsliste

Name	Beschreibung
BeginTransaction()-Methode	Gibt ein Transaktionsobjekt zurück.
BeginTransaction(IsolationLevel)-Methode	Gibt ein Transaktionsobjekt zurück.
BeginTransaction(SAIsolationLevel)-Methode	Gibt ein Transaktionsobjekt zurück.

BeginTransaction()-Methode

Gibt ein Transaktionsobjekt zurück.

Visual Basic-Syntax

```
Public Shadows Function BeginTransaction() As SATransaction
```

C#-Syntax

```
public new SATransaction BeginTransaction()
```

Rückgabe

Ein SATransaction-Objekt, das die neue Transaktion darstellt.

Bemerkungen

Befehle, die mit einem Transaction-Objekt verbunden sind, werden als einzelne Transaktion ausgeführt. Die Transaktion wird mit einem Aufruf der Methode Commit oder Rollback abgeschlossen.

Mit der SACommand.Transaction-Eigenschaft können Sie einem Befehl ein Transaktionsobjekt zuordnen.

Siehe auch

- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)
- [SACommand.Transaction-Eigenschaft \[SQL Anywhere .NET\] auf Seite 151](#)

BeginTransaction(IsolationLevel)-Methode

Gibt ein Transaktionsobjekt zurück.

Visual Basic-Syntax

```
Public Shadows Function BeginTransaction(  
    ByVal isolationLevel As IsolationLevel  
) As SATransaction
```

C#-Syntax

```
public new SATransaction BeginTransaction(IsolationLevel isolationLevel)
```

Parameter

- **isolationLevel** Ein Mitglied der SAIsolationLevel-Enumeration. Der Standardwert ist ReadCommitted.

Rückgabe

Ein SATransaction-Objekt, das die neue Transaktion darstellt.

Bemerkungen

Befehle, die mit einem Transaction-Objekt verbunden sind, werden als einzelne Transaktion ausgeführt. Die Transaktion wird mit einem Aufruf der Methode Commit oder Rollback abgeschlossen.

Mit der `SACommand.Transaction`-Eigenschaft können Sie einem Befehl ein Transaktionsobjekt zuordnen.

Siehe auch

- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)
- [SACommand.Transaction-Eigenschaft \[SQL Anywhere .NET\] auf Seite 151](#)
- [SAIsolationLevel-Enumeration \[SQL Anywhere .NET\] auf Seite 333](#)

Beispiel

```
SATransaction tx =  
    conn.BeginTransaction( SAIsolationLevel.ReadUncommitted );
```

BeginTransaction(SAIsolationLevel)-Methode

Gibt ein Transaktionsobjekt zurück.

Visual Basic-Syntax

```
Public Function BeginTransaction(  
    ByVal isolationLevel As SAIsolationLevel  
) As SATransaction
```

C#-Syntax

```
public SATransaction BeginTransaction(SAIsolationLevel isolationLevel)
```

Parameter

- **isolationLevel** Ein Mitglied der `SAIsolationLevel`-Enumeration. Der Standardwert ist `ReadCommitted`.

Rückgabe

Ein `SATransaction`-Objekt, das die neue Transaktion darstellt.

Bemerkungen

Befehle, die mit einem Transaction-Objekt verbunden sind, werden als einzelne Transaktion ausgeführt. Die Transaktion wird mit einem Aufruf der Methode `Commit` oder `Rollback` abgeschlossen.

Mit der `SACommand.Transaction`-Eigenschaft können Sie einem Befehl ein Transaktionsobjekt zuordnen.

Weitere Hinweise finden Sie unter „[Transaktionsverarbeitung](#)“ auf Seite 64.

Weitere Hinweise finden Sie unter „[Typische Arten von Inkonsistenz](#)“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*].

Siehe auch

- [SATransaction-Klasse \[SQL Anywhere .NET\] auf Seite 321](#)
- [SACommand.Transaction-Eigenschaft \[SQL Anywhere .NET\] auf Seite 151](#)
- [SAIsolationLevel-Enumeration \[SQL Anywhere .NET\] auf Seite 333](#)
- [SATransaction.Commit-Methode \[SQL Anywhere .NET\] auf Seite 322](#)
- [SATransaction.Rollback-Methode \[SQL Anywhere .NET\] auf Seite 322](#)

ChangeDatabase-Methode

Ändert die aktuelle Datenbank für ein geöffnetes SAConnection-Objekt.

Visual Basic-Syntax

```
Public Overrides Sub ChangeDatabase(ByVal database As String)
```

C#-Syntax

```
public override void ChangeDatabase(string database)
```

Parameter

- **database** Der Name der Datenbank, die anstelle der aktuellen Datenbank benutzt werden soll

ChangePassword-Methode

Ändert das Kennwort für den Benutzer, der in der Verbindungszeichenfolge für das bereitgestellte neue Kennwort angegeben ist.

Visual Basic-Syntax

```
Public Shared Sub ChangePassword(  
    ByVal connectionString As String,  
    ByVal newPassword As String  
)
```

C#-Syntax

```
public static void ChangePassword(  
    string connectionString,  
    string newPassword  
)
```

Parameter

- **connectionString** Die Verbindungszeichenfolge, die genügend Informationen enthält, damit eine Verbindung mit dem gewünschten Datenbankserver erstellt werden kann. Die Verbindungszeichenfolge kann die Benutzer-ID und das aktuelle Kennwort enthalten.
- **newPassword** Das gewünschte neue Kennwort. Dieses Kennwort muss alle Sicherheitsrichtlinien für die Kennwörter auf dem Server erfüllen, einschließlich der Mindestlänge, Anforderungen spezieller Zeichen usw.

Ausnahmebedingungen

- **ArgumentNullException** Entweder der connectionString-Parameter oder der newPassword-Parameter ist Null.
- **ArgumentException** Die Verbindungszeichenfolge umfasst die Option zur Verwendung der integrierten Sicherheit.

ClearAllPools-Methode

Leert alle Verbindungspools.

Visual Basic-Syntax

```
Public Shared Sub ClearAllPools()
```

C#-Syntax

```
public static void ClearAllPools()
```

ClearPool-Methode

Leert den Verbindungspool, der der angegebenen Verbindung zugeordnet ist.

Visual Basic-Syntax

```
Public Shared Sub ClearPool(ByVal connection As SAConnection)
```

C#-Syntax

```
public static void ClearPool(SAConnection connection)
```

Parameter

- **connection** Das SAConnection-Objekt, das aus dem Pool entfernt werden soll.

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

Close-Methode

Schließt eine Datenbankverbindung.

Visual Basic-Syntax

```
Public Overrides Sub Close()
```

C#-Syntax

```
public override void Close()
```

Bemerkungen

Die Close-Methode setzt noch nicht festgeschriebene Transaktionen zurück. Danach gibt sie die Verbindung an den Verbindungspool zurück oder schließt die Verbindung, wenn die Verbindungspoolfunktion deaktiviert ist. Wenn Close während der Verarbeitung eines StateChange-Ereignisses aufgerufen wird, werden keine zusätzlichen StateChange-Ereignisse ausgelöst. Eine Anwendung kann Close mehrmals aufrufen.

CreateCommand-Methode

Initialisiert ein SACommand-Objekt.

Visual Basic-Syntax

```
Public Shadows Function CreateCommand() As SACommand
```

C#-Syntax

```
public new SACommand CreateCommand()
```

Rückgabe

Ein SACommand-Objekt.

Bemerkungen

Das Befehlsobjekt ist dem SAConnection-Objekt zugeordnet.

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)
- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

EnlistDistributedTransaction-Methode

Nimmt die angegebene Transaktion als verteilte Transaktion in die Liste auf.

Visual Basic-Syntax

```
Public Sub EnlistDistributedTransaction(  
    ByVal transaction As System.EnterpriseServices.ITransaction  
)
```

C#-Syntax

```
public void EnlistDistributedTransaction(  
    System.EnterpriseServices.ITransaction transaction  
)
```

Parameter

- **transaction** Eine Referenz zu einer bestehenden System.EnterpriseServices.ITransaction, an der teilgenommen werden soll.

EnlistTransaction-Methode

Nimmt die angegebene Transaktion als verteilte Transaktion in die Liste auf.

Visual Basic-Syntax

```
Public Overrides Sub EnlistTransaction(  
    ByVal transaction As System.Transactions.Transaction  
)
```

C#-Syntax

```
public override void EnlistTransaction(  
    System.Transactions.Transaction transaction  
)
```

Parameter

- **transaction** Eine Referenz zu einer bestehenden System.Transactions.Transaction, an der teilgenommen werden soll.

GetSchema-Methode

Gibt die Liste der unterstützten Schemasammlungen zurück.

Überladungsliste

Name	Beschreibung
GetSchema()-Methode	Gibt die Liste der unterstützten Schemasammlungen zurück.
GetSchema(String)-Methode	Gibt Informationen für die angegebene Metadatensammlung für dieses SAConnection-Objekt zurück
GetSchema(String, String[])-Methode	Gibt Schemainformationen für die Datenquelle dieses SAConnection-Objekts zurück und verwendet ggf. die angegebene Zeichenfolge für den Schemanamen und das angegebene Zeichenfolgen-Array für die Einschränkungswerte.

GetSchema()-Methode

Gibt die Liste der unterstützten Schemasammlungen zurück.

Visual Basic-Syntax

```
Public Overrides Function GetSchema() As DataTable
```

C#-Syntax

```
public override DataTable GetSchema()
```

Bemerkungen

Eine Beschreibung der verfügbaren Metadaten finden Sie unter `GetSchema(string,string[])`.

GetSchema(String)-Methode

Gibt Informationen für die angegebene Metadatensammlung für dieses `SACConnection`-Objekt zurück

Visual Basic-Syntax

```
Public Overrides Function GetSchema(  
    ByVal collection As String  
) As DataTable
```

C#-Syntax

```
public override DataTable GetSchema(string collection)
```

Parameter

- **collection** Name der Metadatensammlung. Wenn kein Name angegeben ist, wird `MetaDataCollections` verwendet.

Bemerkungen

Eine Beschreibung der verfügbaren Metadaten finden Sie unter `GetSchema(string,string[])`.

Siehe auch

- [SACConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

GetSchema(String, String[])-Methode

Gibt Schemainformationen für die Datenquelle dieses `SACConnection`-Objekts zurück und verwendet ggf. die angegebene Zeichenfolge für den Schemanamen und das angegebene Zeichenfolgen-Array für die Einschränkungswerte.

Visual Basic-Syntax

```
Public Overrides Function GetSchema(  
    ByVal collection As String,  
    ByVal restrictions As String()  
) As DataTable
```

C#-Syntax

```
public override DataTable GetSchema(  
    string collection,  
    string[] restrictions  
)
```

Rückgabe

Eine Datentabelle, die Schemainformationen enthält

Bemerkungen

Diese Methoden werden verwendet, um den Datenbankserver nach Metadaten abzufragen. Jeder Metadaten-Typ erhält einen Sammlungsnamen, der für den Empfang der Daten übergeben werden muss. Der Standard-Sammlungsname ist `MetaDataCollections`.

Sie können den SQL Anywhere .NET-Datenprovider abfragen, um die Liste der unterstützten Schemasammlungen zu erhalten, indem Sie die Methode `GetSchema` ohne Argumente oder mit dem Schemasammlungsnamen **`MetaDataCollections`** übergeben. Sie erhalten eine `DataTable`-Tabelle mit einer Liste der unterstützten Schemasammlungen (`CollectionName`), der Anzahl der Einschränkungen, die sie jeweils unterstützen (`NumberOfRestrictions`) und der Anzahl der ID-Teile, die sie verwenden, (`NumberOfIdentifierParts`) zurück.

Sammlung	Metadaten
Columns	Gibt Informationen zu allen Spalten in der Datenbank zurück.
DataSourceInformation	Gibt Informationen über den Datenbankserver zurück.
DataTypes	Gibt eine Liste der unterstützten Datentypen zurück.
ForeignKeys	Gibt Informationen zu allen Fremdschlüsseln in der Datenbank zurück.
IndexColumns	Gibt Informationen zu allen Indexspalten in der Datenbank zurück.
Indizes	Gibt Informationen zu allen Indizes in der Datenbank zurück.
MetaDataCollections	Gibt eine Liste aller Sammlungsnamen zurück.
ProcedureParameters	Gibt Informationen zu allen Prozedurparametern in der Datenbank zurück.
Procedures	Gibt Informationen zu allen Prozeduren in der Datenbank zurück.
ReservedWords	Gibt eine Liste der von SQL Anywhere verwendeten reservierten Wörter zurück.
Restrictions	Gibt Informationen zu in <code>GetSchema</code> verwendeten Einschränkungen zurück.
Tables	Gibt Informationen zu allen Tabellen in der Datenbank zurück.
UserDefinedTypes	Gibt Informationen zu allen benutzerdefinierten Datentypen in der Datenbank zurück.
User	Gibt Informationen zu allen Benutzern in der Datenbank zurück.
ViewColumns	Gibt Informationen zu allen Spalten in Ansichten in der Datenbank zurück.
Views	Gibt Informationen zu allen Ansichten in der Datenbank zurück.

Diese Sammlungsnamen sind auch als schreibgeschützte Eigenschaften in der Klasse `SAMetaDataCollectionNames` verfügbar.

Die zurückgegebenen Ergebnisse können durch die Angabe eines Arrays von Einschränkungen im Aufruf von `GetSchema` gefiltert werden.

Die für jede Sammlung verfügbaren Einschränkungen können mit folgendem Aufruf abgefragt werden:

```
GetSchema( "Restrictions" )
```

Wenn die Sammlung vier Einschränkungen erfordert, muss der `Restrictions`-Parameter `NULL` oder eine Zeichenfolge von 4 Werten sein.

Zum Filtern einer bestimmten Einschränkung positionieren Sie die Zeichenfolge, anhand der gefiltert werden soll, an die betreffende Position im Array und geben für alle nicht verwendeten Stellen `NULL` an. Die Sammlung `Tables` hat z.B. drei Einschränkungen: `Owner`, `Table` und `TableType`.

So filtern Sie die Sammlung `Table` anhand von `table_name`:

```
GetSchema( "Tables", new string[ ] { NULL, "my_table", NULL } )
```

Dies gibt Informationen zu allen Tabellen mit dem Namen `my_table` zurück.

```
GetSchema( "Tables", new string[ ] { "DBA", "my_table", NULL } )
```

Dies gibt Informationen zu allen Tabellen mit Namen `my_table` zurück, die dem Benutzer `DBA` gehören.

Das Folgende ist eine Zusammenfassung der Spalten, die von den jeweiligen Sammlungen zurückgegeben werden. Wenn die Anzahl der Zeilen, die in einer Sammlung zurückgegeben werden, durch die Angabe einer Einschränkung vermindert werden kann, wird der Einschränkungsname für die Spalte in Klammern angezeigt. Einschränkungen werden in der Reihenfolge angegeben, in der sie in der folgenden Liste aufgeführt sind.

Columns-Sammlung

- `table_schema` (`Owner`)
- `table_name` (`Table`)
- `column_name` (`Column`)
- `ordinal_position`
- `column_default`
- `is_nullable`
- `data_type`
- `precision`
- `scale`

- column_size

DataSourceInformation-Sammlung

- CompositeIdentifierSeparatorPattern
- DataSourceProductName
- DataSourceProductVersion
- DataSourceProductVersionNormalized
- GroupByBehavior
- IdentifierPattern
- IdentifierCase
- OrderByColumnsInSelect
- ParameterMarkerFormat
- ParameterMarkerPattern
- ParameterNameMaxLength
- ParameterNamePattern
- QuotedIdentifierPattern
- QuotedIdentifierCase
- StatementSeparatorPattern
- StringLiteralPattern
- SupportedJoinOperators

DataTypes-Sammlung

- TypeName
- ProviderDbType
- ColumnSize
- CreateFormat
- CreateParameters
- DataType

- IsAutoIncrementable
- IsBestMatch
- IsCaseSensitive
- IsFixedLength
- IsFixedPrecisionScale
- IsLong
- IsNullable
- IsSearchable
- IsSearchableWithLike
- IsUnsigned
- MaximumScale
- MinimumScale
- IsConcurrencyType
- IsLiteralSupported
- LiteralPrefix
- LiteralSuffix

ForeignKeys-Sammlung

- table_schema (Owner)
- table_name (Table)
- column_name (Column)

IndexColumns-Sammlung

- table_schema (Owner)
- table_name (Table)
- index_name (Name)
- column_name (Column)
- order

Indexes-Sammlung

- table_schema (Owner)
- table_name (Table)
- index_name (Name)
- primary_key
- is_unique

MetaDataCollections-Sammlung

- CollectionName
- NumberOfRestrictions
- NumberOfIdentifierParts

ProcedureParameters-Sammlung

- procedure_schema (Owner)
- procedure_name (Name)
- parameter_name (Parameter)
- data_type
- parameter_type
- is_input
- is_output

Procedures-Sammlung

- procedure_schema (Owner)
- procedure_name (Name)

ReservedWords-Sammlung

- reserved_word

Restrictions-Sammlung

- CollectionName
- RestrictionName
- RestrictionDefault

- RestrictionNumber

Tables-Sammlung

- table_schema (Owner)
- table_name (Table)
- table_type (TableType) - "BASE", "VIEW", "MAT VIEW", "LCL TEMP", "GBL TEMP", "TEXT" oder "TEXT GBL TEMP"

UserDefinedTypes-Sammlung

- data_type
- default
- precision
- scale

Users-Sammlung

- user_name (UserName)
- resource_auth
- database_auth
- schedule_auth
- user_group

ViewColumns-Sammlung

- view_schema (Owner)
- view_name (Name)
- column_name (Column)

Views-Sammlung

- view_schema (Owner)
- view_name (Name)

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

Open-Methode

Öffnet eine Datenbankverbindung mit den durch das `SACConnection.ConnectionString`-Objekt angegebenen Eigenschaftseinstellungen.

Visual Basic-Syntax

```
Public Overrides Sub Open()
```

C#-Syntax

```
public override void Open()
```

Siehe auch

- [SACConnection.ConnectionString-Eigenschaft \[SQL Anywhere .NET\] auf Seite 185](#)

ConnectionString-Eigenschaft

Stellt die Datenbank-Verbindungszeichenfolge bereit.

Visual Basic-Syntax

```
Public Overrides Property ConnectionString As String
```

C#-Syntax

```
public override string ConnectionString {get;set;}
```

Bemerkungen

Die `ConnectionString`-Eigenschaft ist so ausgelegt, dass sie so eng wie möglich an das Format der SQL Anywhere-Verbindungszeichenfolge angepasst ist, wobei aber folgende Ausnahme gilt: Wenn der `Persist Security Info`-Wert auf `FALSE` gesetzt ist (Standardwert), entspricht die zurückgegebene Verbindungszeichenfolge der vom Benutzer definierten `ConnectionString`-Eigenschaft abzüglich der Sicherheitsinformationen. Der SQL Anywhere .NET-Datenprovider speichert das Kennwort nur dann dauerhaft in einer zurückgegebenen Verbindungszeichenfolge, wenn `Persist Security Info` auf `TRUE` gesetzt ist.

Sie können die `ConnectionString`-Eigenschaft verwenden, um eine Verbindung mit einer Reihe von Datenquellen herzustellen.

Sie können die `ConnectionString`-Eigenschaft nur dann definieren, wenn die Verbindung geschlossen ist. Viele Werte für die Verbindungszeichenfolge haben zugehörige schreibgeschützte Eigenschaften. Wenn die Verbindungszeichenfolge definiert ist, werden alle diese Eigenschaften aktualisiert, sofern nicht ein Fehler erkannt wird. Falls ein Fehler festgestellt wird, wird keine der Eigenschaften aktualisiert. `SACConnection`-Eigenschaften geben nur die Einstellungen zurück, die im `ConnectionString`-Objekt enthalten sind.

Wenn Sie `ConnectionString` bei einer geschlossenen Verbindung zurücksetzen, werden alle Werte für die Verbindungszeichenfolge und die dazugehörigen Eigenschaften zurückgesetzt, einschließlich des Kennworts.

Wenn die Eigenschaft gesetzt wird, wird eine vorbereitende Validierung der Verbindungszeichenfolge durchgeführt. Wenn eine Anwendung die Open-Methode aufruft, wird die Verbindungszeichenfolge vollständig validiert. Eine Laufzeit-Ausnahmebedingung wird generiert, wenn die Verbindungszeichenfolge ungültige oder nicht unterstützte Eigenschaften enthält.

Werte können in Apostrophe oder Anführungszeichen gesetzt werden. Apostrophe oder Anführungszeichen können innerhalb einer Verbindungszeichenfolge benutzt werden, in der das jeweils andere Begrenzungszeichen benutzt wird, z.B. `name="Wert'berichtigung"` oder `name='Wert"berichtigung'`, aber nicht `name="Wert'berichtigung"` oder `name=" ""Wertberichtigung""`. Leerzeichen werden ignoriert, außer wenn sie innerhalb eines Werts oder zwischen Anführungszeichen bzw. Apostrophen gestellt werden. Schlüsselwert=Wert-Paare müssen durch Semikolon getrennt werden. Wenn ein Semikolon Teil eines Werts ist, muss es ebenfalls durch Anführungszeichen begrenzt werden. Escape-Sequenzen werden nicht unterstützt und der Werttyp ist nicht relevant. Bei Namen wird die Groß- und Kleinschreibung nicht berücksichtigt. Wenn ein Eigenschaftsname mehr als einmal in der Verbindungszeichenfolge vorkommt, wird der Wert verwendet, der mit dem zuletzt angegebenen Eigenschaftsnamen verbunden ist.

Beachten Sie gewisse Vorsichtsmaßnahmen, wenn Sie eine Verbindungszeichenfolge erstellen, die auf Benutzereingaben beruht, etwa wenn eine Benutzer-ID und ein Kennwort aus einem Fenster abgerufen und an die Verbindungszeichenfolge angehängt wird. Die Anwendung darf nicht zulassen, dass ein Benutzer zusätzliche Parameter für die Verbindungszeichenfolge in diese Werte einbettet.

Der Standardwert für die Verbindungspoolfunktion ist TRUE (`pooling=TRUE`).

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)
- [SAConnection.Open-Methode \[SQL Anywhere .NET\] auf Seite 185](#)

Beispiel

Die folgenden Anweisungen legen eine Verbindungszeichenfolge für eine ODBC-Datenquelle namens "SQL Anywhere 16 Demo" fest und öffnen die Verbindung.

```
SAConnection conn = new SAConnection();  
conn.ConnectionString = "DSN=SQL Anywhere 16 Demo";  
conn.Open();
```

ConnectionTimeout-Eigenschaft

Ruft die Zeitspanne in Sekunden ab, nach deren Ablauf der Verbindungsversuch mit einer Fehlermeldung abgebrochen wird.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property ConnectionTimeout As Integer
```

C#-Syntax

```
public override int ConnectionTimeout {get;}
```

Bemerkungen

Der Standardwert für ConnectionTimeout beträgt 15 Sekunden.

Beispiel

Die folgende Anweisung zeigt den Wert der ConnectionTimeout-Eigenschaft.

```
MessageBox.Show( conn.ConnectionTimeout.ToString( ) );
```

Database-Eigenschaft

Ruft den Namen der aktuellen Datenbank ab.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Database As String
```

C#-Syntax

```
public override string Database {get;}
```

Bemerkungen

Wenn die Verbindung geöffnet ist, gibt SAConnection den Namen der aktuellen Datenbank zurück. Anderfalls durchsucht SAConnection die Verbindungszeichenfolge in dieser Reihenfolge: DatabaseName, DBN, DataSourceName, Data Source, DSN, DatabaseFile, DBF, FileDataSourceName, FileDSN.

DataSource-Eigenschaft

Ruft den Namen des Datenbankservers ab.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property DataSource As String
```

C#-Syntax

```
public override string DataSource {get;}
```

Bemerkungen

Wenn die Verbindung geöffnet wird, gibt SAConnection die Servereigenschaft ServerName zurück. Andernfalls sucht das SAConnection-Objekt in der Verbindungszeichenfolge nach dem ServerName-Verbindungsparameter.

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

InitString-Eigenschaft

Ein Befehl, der unmittelbar nach Herstellung der Verbindung ausgeführt wird.

Visual Basic-Syntax

```
Public Property InitString As String
```

C#-Syntax

```
public string InitString {get;set;}
```

Bemerkungen

Die InitString-Eigenschaft wird ausgeführt, sobald die Verbindung geöffnet wurde.

ServerVersion-Eigenschaft

Ruft eine Zeichenfolge ab, die die Version der Instanz von SQL Anywhere enthält, mit der der Client verbunden ist.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property ServerVersion As String
```

C#-Syntax

```
public override string ServerVersion {get;}
```

Rückgabe

Die Version der Instanz von SQL Anywhere.

Bemerkungen

Die Version ist **##.##.####**, wobei die ersten beiden Stellen die Hauptversion, die nächsten beiden Stellen die Unterversion und die letzten vier Stellen die Release-Version angeben. Die angehängte Zeichenfolge hat das Format major.minor.build, wobei gilt: major und minor haben zwei Ziffern und build hat vier Ziffern.

State-Eigenschaft

Gibt den Status des SAConnection-Objekts an.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property State As ConnectionState
```

C#-Syntax

```
public override ConnectionState state {get;}
```

Rückgabe

Eine System.Data.ConnectionState-Enumeration.

InfoMessage-Ereignis

Tritt ein, wenn der SQL Anywhere-Datenbankserver eine Warnung oder eine Informationsmeldung zurückgibt.

Visual Basic-Syntax

```
Public Event InfoMessage As SAInfoMessageEventHandler
```

C#-Syntax

```
public event SAInfoMessageEventHandler InfoMessage;
```

Bemerkungen

Der Event-Handler empfängt ein Argument vom Typ SAInfoMessageEventArgs, das Daten im Zusammenhang mit diesem Ereignis enthält. Die nachstehenden SAInfoMessageEventArgs-Eigenschaften übermitteln für dieses Ereignis spezifische Informationen: NativeError, Errors, Message, MessageType und Source.

Weitere Hinweise finden Sie in der Dokumentation zum .NET Framework unter der Rubrik "OleDbConnection.InfoMessage Event".

StateChange-Ereignis

Tritt ein, wenn sich der Status des SAConnection-Objekts ändert.

Visual Basic-Syntax

```
Public Event StateChange As StateChangeEventHandler
```

C#-Syntax

```
public event override StateChangeEventHandler StateChange;
```

Bemerkungen

Der Event-Handler empfängt ein Argument vom Typ StateChangeEventArgs, das Daten im Zusammenhang mit diesem Ereignis enthält. Die nachfolgenden StateChangeEventArgs-Eigenschaften enthalten Informationen, die dieses Ereignis betreffen: CurrentState und OriginalState.

Weitere Hinweise finden Sie in der Dokumentation zum .NET Framework unter der Rubrik OleDbConnection.StateChange Event.

SAConnectionStringBuilder-Klasse

Bietet eine einfache Möglichkeit zum Erstellen und Verwalten des Inhalts von Verbindungszeichenfolgen, die von der Klasse SAConnection verwendet werden.

Visual Basic-Syntax

```
Public NotInheritable Class SAConnectionStringBuilder  
    Inherits SAConnectionStringBuilderBase
```

C#-Syntax

```
public sealed class SAConnectionStringBuilder :  
    SAConnectionStringBuilderBase
```

Basisklassen

- [SAConnectionStringBuilderBase-Klasse \[SQL Anywhere .NET\] auf Seite 207](#)

Mitglieder

Alle Mitglieder der SAConnectionStringBuilder-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SAConnectionStringBuilder-Konstruktor	Initialisiert eine neue Instanz der Klasse SAConnectionStringBuilder.
Add-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Fügt dem System.Data.Common.DbConnectionStringBuilder -Objekt einen Eintrag mit dem angegebenen Schlüssel und Wert hinzu.
AppendKeyValuePair-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Stellt eine effiziente und sichere Möglichkeit dar, einen Schlüssel und Wert an ein vorhandenes System.Text.StringBuilder -Objekt anzuhängen.
Clear-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Löscht den Inhalt der System.Data.Common.DbConnectionStringBuilder -Instanz.
ClearPropertyDescriptors-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Löscht die Sammlung von System.ComponentModel.PropertyDescriptor -Objekte auf dem verknüpften System.Data.Common.DbConnectionStringBuilder -Objekt.
ContainsKey-Methode	Ermittelt, ob das SAConnectionStringBuilder-Objekt ein spezifisches Schlüsselwort enthält.
EquivalentTo-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Vergleicht die Verbindungsinformationen in diesem System.Data.Common.DbConnectionStringBuilder -Objekt mit den Verbindungsinformationen im angegebenen Objekt
GetKeyword-Methode	Ruft das Schlüsselwort für die angegebene SAConnectionStringBuilder-Eigenschaft ab.
GetProperties-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Füllt ein angegebenes System.Collections.Hashtable -Objekt mit Informationen zu allen Eigenschaften dieses System.Data.Common.DbConnectionStringBuilder -Objekts.
GetUseLongNameAsKeyword-Methode	Ruft einen booleschen Wert ab, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.

Name	Beschreibung
Remove-Methode	Löscht den Eintrag mit dem angegebenen Schlüssel aus der <code>SACConnectionStringBuilder</code> -Instanz.
SetUseLongNameAsKeyword-Methode	Legt einen booleschen Wert fest, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
ShouldSerialize-Methode	Gibt an, ob der angegebene Schlüssel in dieser <code>SACConnectionStringBuilder</code> -Instanz vorhanden ist.
ToString-Methode (geerbt aus <code>System.Data.Common.DbConnectionStringBuilder</code>)	Gibt die Verbindungszeichenfolge zurück, die diesem System.Data.Common.DbConnectionStringBuilder -Objekt zugeordnet ist.
TryGetValue-Methode	Ruft einen Wert ab, der dem bereitgestellten Schlüssel aus diesem <code>SACConnectionStringBuilder</code> -Objekt entspricht.
AppInfo-Eigenschaft	Ruft die <code>AppInfo</code> -Verbindungseigenschaft ab bzw. legt sie fest.
AutoStart-Eigenschaft	Ruft die <code>AutoStart</code> -Verbindungseigenschaft ab bzw. legt sie fest.
AutoStop-Eigenschaft	Ruft die <code>AutoStop</code> -Verbindungseigenschaft ab bzw. legt sie fest.
BrowsableConnectionString-Eigenschaft (geerbt aus <code>System.Data.Common.DbConnectionStringBuilder</code>)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder.ConnectionString -Objekt in den Visual Studio-Designern sichtbar ist, bzw. legt ihn fest.
Charset-Eigenschaft	Ruft die <code>Charset</code> -Verbindungseigenschaft ab bzw. legt sie fest.
CommBufferSize-Eigenschaft	Ruft die <code>CommBufferSize</code> -Verbindungseigenschaft ab bzw. legt sie fest.
CommLinks-Eigenschaft	Ruft die <code>CommLinks</code> -Eigenschaft ab bzw. legt sie fest.
Compress-Eigenschaft	Ruft die <code>Compress</code> -Verbindungseigenschaft ab bzw. legt sie fest.
CompressionThreshold-Eigenschaft	Ruft die <code>CompressionThreshold</code> -Verbindungseigenschaft ab bzw. legt sie fest.
ConnectionLifetime-Eigenschaft	Ruft die <code>ConnectionLifetime</code> -Verbindungseigenschaft ab bzw. legt sie fest.
ConnectionName-Eigenschaft	Ruft die <code>ConnectionName</code> -Verbindungseigenschaft ab bzw. legt sie fest.
ConnectionPool-Eigenschaft	Ruft die <code>ConnectionPool</code> -Eigenschaft ab bzw. legt sie fest.

Name	Beschreibung
ConnectionReset-Eigenschaft	Ruft die ConnectionReset-Verbindungseigenschaft ab bzw. legt sie fest.
ConnectionString-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft die Verbindungszeichenfolge ab, die dem DbConnectionStringBuilder -Objekt zugeordnet ist, bzw. legt sie fest.
ConnectionTimeout-Eigenschaft	Ruft die ConnectionTimeout-Verbindungseigenschaft ab bzw. legt sie fest.
Count-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft die aktuelle Anzahl von Schlüsseln ab, die in der System.Data.Common.DbConnectionStringBuilder.ConnectionString-Eigenschaft enthalten sind.
DatabaseFile-Eigenschaft	Ruft die DatabaseFile-Verbindungseigenschaft ab bzw. legt sie fest.
DatabaseKey-Eigenschaft	Ruft die DatabaseKey-Verbindungseigenschaft ab bzw. legt sie fest.
DatabaseName-Eigenschaft	Ruft die DatabaseName-Verbindungseigenschaft ab bzw. legt sie fest.
DatabaseSwitches-Eigenschaft	Ruft die DatabaseSwitches-Verbindungseigenschaft ab bzw. legt sie fest.
DataSourceName-Eigenschaft	Ruft die DataSourceName-Verbindungseigenschaft ab bzw. legt sie fest.
DisableMultiRowFetch-Eigenschaft	Ruft die DisableMultiRowFetch-Verbindungseigenschaft ab bzw. legt sie fest.
Elevate-Eigenschaft	Ruft die Elevate-Verbindungseigenschaft ab bzw. legt sie fest.
EncryptedPassword-Eigenschaft	Ruft die EncryptedPassword-Verbindungseigenschaft ab bzw. legt sie fest.
Encryption-Eigenschaft	Ruft die Encryption-Verbindungseigenschaft ab bzw. legt sie fest.
Enlist-Eigenschaft	Ruft die Enlist-Verbindungseigenschaft ab bzw. legt sie fest.
FileDataSourceName-Eigenschaft	Ruft die FileDataSourceName-Verbindungseigenschaft ab bzw. legt sie fest.
ForceStart-Eigenschaft	Ruft die ForceStart-Verbindungseigenschaft ab bzw. legt sie fest.
Host-Eigenschaft	Ruft die Host-Eigenschaft ab bzw. legt sie fest.

Name	Beschreibung
IdleTimeout-Eigenschaft	Ruft die IdleTimeout-Verbindungseigenschaft ab bzw. legt sie fest.
InitString-Eigenschaft	Ruft die InitString-Verbindungseigenschaft ab bzw. legt sie fest.
Integrated-Eigenschaft	Ruft die Integrated-Verbindungseigenschaft ab bzw. legt sie fest.
IsFixedSize-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder -Objekt eine feste Größe hat.
IsReadOnly-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder -Objekt schreibgeschützt ist.
Kerberos-Eigenschaft	Ruft die Kerberos-Verbindungseigenschaft ab bzw. legt sie fest.
Keys-Eigenschaft	Ruft ein System.Collections.ICollection-Objekt ab, das die Schlüssel im SAConnectionStringBuilder-Objekt enthält.
Language-Eigenschaft	Ruft die Language-Verbindungseigenschaft ab bzw. legt sie fest.
LazyClose-Eigenschaft	Ruft die LazyClose-Verbindungseigenschaft ab bzw. legt sie fest.
LivenessTimeout-Eigenschaften	Ruft die LivenessTimeout-Verbindungseigenschaft ab bzw. legt sie fest.
LogFile-Eigenschaft	Ruft die LogFile-Verbindungseigenschaft ab bzw. legt sie fest.
MaxPoolSize-Eigenschaft	Ruft die MaxPoolSize-Verbindungseigenschaft ab bzw. legt sie fest.
MinPoolSize-Eigenschaft	Ruft die MinPoolSize-Verbindungseigenschaft ab bzw. legt sie fest.
NewPassword-Eigenschaft	Ruft die NewPassword-Verbindungseigenschaft ab bzw. legt sie fest.
NodeType-Eigenschaft	Ruft die NodeType-Eigenschaft ab bzw. legt sie fest.
Password-Eigenschaft	Ruft die Password-Verbindungseigenschaft ab bzw. legt sie fest.
PersistSecurityInfo-Eigenschaft	Ruft die PersistSecurityInfo-Verbindungseigenschaft ab bzw. legt sie fest.
Pooling-Eigenschaft	Ruft die Pooling-Verbindungseigenschaft ab bzw. legt sie fest.

Name	Beschreibung
PrefetchBuffer-Eigenschaft	Ruft die PrefetchBuffer-Verbindungseigenschaft ab bzw. legt sie fest.
PrefetchRows-Eigenschaft	Ruft die PrefetchRows-Verbindungseigenschaft ab bzw. legt sie fest.
RetryConnectionTimeout-Eigenschaft	Ruft die RetryConnectionTimeout-Eigenschaft ab bzw. legt sie fest.
ServerName-Eigenschaft	Ruft die ServerName-Verbindungseigenschaft ab bzw. legt sie fest.
StartLine-Eigenschaft	Ruft die StartLine-Verbindungseigenschaft ab bzw. legt sie fest.
this-Eigenschaft	Ruft den Wert des Verbindungsschlüsselworts ab bzw. legt ihn fest.
Unconditional-Eigenschaft	Ruft die Unconditional-Verbindungseigenschaft ab bzw. legt sie fest.
UserID-Eigenschaft	Ruft die UserID-Verbindungseigenschaft ab bzw. legt sie fest.
Values-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft ein System.Collections.ICollection -Objekt ab, das die Werte aus dem System.Data.Common.DbConnectionStringBuilder -Objekt enthält.

Bemerkungen

Die Klasse `SAConnectionStringBuilder` erbt das `SAConnectionStringBuilderBase`-Objekt, das wiederum das `DbConnectionStringBuilder`-Objekt erbt.

Die Klasse `SAConnectionStringBuilder` ist in .NET Compact Framework 2.0 nicht verfügbar.

Eine Liste der Verbindungsparameter finden Sie unter „[Verbindungsparameter](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

SAConnectionStringBuilder-Konstruktor

Initialisiert eine neue Instanz der Klasse `SAConnectionStringBuilder`.

Überladungsliste

Name	Beschreibung
SAConnectionStringBuilder()-Konstruktor	Initialisiert eine neue Instanz der Klasse <code>SAConnectionStringBuilder</code> .

Name	Beschreibung
SAConnectionStringBuilder(String)-Konstruktor	Initialisiert eine neue Instanz der Klasse SAConnectionStringBuilder.

SAConnectionStringBuilder()-Konstruktor

Initialisiert eine neue Instanz der Klasse SAConnectionStringBuilder.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SAConnectionStringBuilder()
```

Bemerkungen

Die Klasse SAConnectionStringBuilder ist in .NET Compact Framework 2.0 nicht verfügbar.

SAConnectionStringBuilder(String)-Konstruktor

Initialisiert eine neue Instanz der Klasse SAConnectionStringBuilder.

Visual Basic-Syntax

```
Public Sub New(ByVal connectionString As String)
```

C#-Syntax

```
public SAConnectionStringBuilder(string connectionString)
```

Parameter

- connectionString** Die Basis für die internen Verbindungsinformationen des Objekts. Wird syntaktisch als Schlüsselwort=Wert-Paare analysiert. Eine Liste der Verbindungsparameter finden Sie unter „[Verbindungsparameter](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

Bemerkungen

Die Klasse SAConnectionStringBuilder ist in .NET Compact Framework 2.0 nicht verfügbar.

Beispiel

Die folgende Anweisung initialisiert ein SAConnection-Objekt für eine Verbindung mit der Datenbank "policies", die auf einem SQL Anywhere-Datenbankserver mit dem Namen "hr" läuft. Die Verbindung verwendet die Benutzer-ID "admin" und das Kennwort "money".

```
SAConnectionStringBuilder conn = new SAConnectionStringBuilder(
    "UID=admin;PWD=money;SERVER=hr;DBN=policies" );
```

AppInfo-Eigenschaft

Ruft die AppInfo-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property AppInfo As String
```

C#-Syntax

```
public string AppInfo {get;set;}
```

AutoStart-Eigenschaft

Ruft die AutoStart-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property AutoStart As String
```

C#-Syntax

```
public string AutoStart {get;set;}
```

AutoStop-Eigenschaft

Ruft die AutoStop-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property AutoStop As String
```

C#-Syntax

```
public string AutoStop {get;set;}
```

Charset-Eigenschaft

Ruft die Charset-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Charset As String
```

C#-Syntax

```
public string Charset {get;set;}
```

CommBufferSize-Eigenschaft

Ruft die CommBufferSize-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property CommBufferSize As Integer
```

C#-Syntax

```
public int CommBufferSize {get;set;}
```

CommLinks-Eigenschaft

Ruft die CommLinks-Eigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property CommLinks As String
```

C#-Syntax

```
public string CommLinks {get;set;}
```

Compress-Eigenschaft

Ruft die Compress-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Compress As String
```

C#-Syntax

```
public string Compress {get;set;}
```

CompressionThreshold-Eigenschaft

Ruft die CompressionThreshold-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property CompressionThreshold As Integer
```

C#-Syntax

```
public int CompressionThreshold {get;set;}
```

ConnectionLifetime-Eigenschaft

Ruft die ConnectionLifetime-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ConnectionLifetime As Integer
```

C#-Syntax

```
public int ConnectionLifetime {get;set;}
```

ConnectionString-Eigenschaft

Ruft die ConnectionName-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ConnectionName As String
```

C#-Syntax

```
public string ConnectionName {get;set;}
```

ConnectionPool-Eigenschaft

Ruft die ConnectionPool-Eigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ConnectionPool As String
```

C#-Syntax

```
public string ConnectionPool {get;set;}
```

ConnectionReset-Eigenschaft

Ruft die ConnectionReset-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ConnectionReset As Boolean
```

C#-Syntax

```
public bool ConnectionReset {get;set;}
```

Rückgabe

Eine Datentabelle, die Schemainformationen enthält

ConnectionTimeout-Eigenschaft

Ruft die ConnectionTimeout-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ConnectionTimeout As Integer
```

C#-Syntax

```
public int ConnectionTimeout {get;set;}
```

Beispiel

Die folgende Anweisung zeigt den Wert der ConnectionTimeout-Eigenschaft.

```
MessageBox.Show( connString.ConnectionTimeout.ToString() );
```

DatabaseFile-Eigenschaft

Ruft die DatabaseFile-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DatabaseFile As String
```

C#-Syntax

```
public string DatabaseFile {get;set;}
```

DatabaseKey-Eigenschaft

Ruft die DatabaseKey-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DatabaseKey As String
```

C#-Syntax

```
public string DatabaseKey {get;set;}
```

DatabaseName-Eigenschaft

Ruft die DatabaseName-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DatabaseName As String
```

C#-Syntax

```
public string DatabaseName {get;set;}
```

DatabaseSwitches-Eigenschaft

Ruft die DatabaseSwitches-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DatabaseSwitches As String
```

C#-Syntax

```
public string DatabaseSwitches {get;set;}
```

DataSourceName-Eigenschaft

Ruft die DataSourceName-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DataSourceName As String
```

C#-Syntax

```
public string DataSourceName {get;set;}
```

DisableMultiRowFetch-Eigenschaft

Ruft die DisableMultiRowFetch-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DisableMultiRowFetch As String
```

C#-Syntax

```
public string DisableMultiRowFetch {get;set;}
```

Elevate-Eigenschaft

Ruft die Elevate-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Elevate As String
```

C#-Syntax

```
public string Elevate {get;set;}
```

EncryptedPassword-Eigenschaft

Ruft die EncryptedPassword-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property EncryptedPassword As String
```

C#-Syntax

```
public string EncryptedPassword {get;set;}
```

Encryption-Eigenschaft

Ruft die Encryption-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Encryption As String
```

C#-Syntax

```
public string Encryption {get;set;}
```

Enlist-Eigenschaft

Ruft die Enlist-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Enlist As Boolean
```

C#-Syntax

```
public bool Enlist {get;set;}
```

FileDataSourceName-Eigenschaft

Ruft die FileDataSourceName-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property FileDataSourceName As String
```

C#-Syntax

```
public string FileDataSourceName {get;set;}
```

ForceStart-Eigenschaft

Ruft die ForceStart-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ForceStart As String
```

C#-Syntax

```
public string ForceStart {get;set;}
```

Host-Eigenschaft

Ruft die Host-Eigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Host As String
```

C#-Syntax

```
public string Host {get;set;}
```

IdleTimeout-Eigenschaft

Ruft die IdleTimeout-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property IdleTimeout As Integer
```

C#-Syntax

```
public int IdleTimeout {get;set;}
```

InitString-Eigenschaft

Ruft die InitString-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property InitString As String
```

C#-Syntax

```
public string InitString {get;set;}
```

Integrated-Eigenschaft

Ruft die Integrated-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Integrated As String
```

C#-Syntax

```
public string Integrated {get;set;}
```

Kerberos-Eigenschaft

Ruft die Kerberos-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Kerberos As String
```

C#-Syntax

```
public string Kerberos {get;set;}
```

Language-Eigenschaft

Ruft die Language-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Language As String
```

C#-Syntax

```
public string Language {get;set;}
```

LazyClose-Eigenschaft

Ruft die LazyClose-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property LazyClose As String
```

C#-Syntax

```
public string LazyClose {get;set;}
```

LivenessTimeout-Eigenschaften

Ruft die LivenessTimeout-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property LivenessTimeout As Integer
```

C#-Syntax

```
public int LivenessTimeout {get;set;}
```

LogFile-Eigenschaft

Ruft die LogFile-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property LogFile As String
```

C#-Syntax

```
public string LogFile {get;set;}
```

MaxPoolSize-Eigenschaft

Ruft die MaxPoolSize-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property MaxPoolSize As Integer
```

C#-Syntax

```
public int MaxPoolSize {get;set;}
```

MinPoolSize-Eigenschaft

Ruft die MinPoolSize-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property MinPoolSize As Integer
```

C#-Syntax

```
public int MinPoolSize {get;set;}
```

NewPassword-Eigenschaft

Ruft die NewPassword-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property NewPassword As String
```

C#-Syntax

```
public string NewPassword {get;set;}
```

NodeType-Eigenschaft

Ruft die NodeType-Eigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property NodeType As String
```

C#-Syntax

```
public string NodeType {get;set;}
```

Password-Eigenschaft

Ruft die Password-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Password As String
```

C#-Syntax

```
public string Password {get;set;}
```

PersistSecurityInfo-Eigenschaft

Ruft die PersistSecurityInfo-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property PersistSecurityInfo As Boolean
```

C#-Syntax

```
public bool PersistSecurityInfo {get;set;}
```

Pooling-Eigenschaft

Ruft die Pooling-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Pooling As Boolean
```

C#-Syntax

```
public bool Pooling {get;set;}
```

PrefetchBuffer-Eigenschaft

Ruft die PrefetchBuffer-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property PrefetchBuffer As Integer
```

C#-Syntax

```
public int PrefetchBuffer {get;set;}
```

PrefetchRows-Eigenschaft

Ruft die PrefetchRows-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property PrefetchRows As Integer
```

C#-Syntax

```
public int PrefetchRows {get;set;}
```

Bemerkungen

Der Standardwert ist 200.

RetryConnectionTimeout-Eigenschaft

Ruft die RetryConnectionTimeout-Eigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property RetryConnectionTimeout As Integer
```

C#-Syntax

```
public int RetryConnectionTimeout {get;set;}
```

ServerName-Eigenschaft

Ruft die ServerName-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ServerName As String
```

C#-Syntax

```
public string ServerName {get;set;}
```

StartLine-Eigenschaft

Ruft die StartLine-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property StartLine As String
```

C#-Syntax

```
public string StartLine {get;set;}
```

Unconditional-Eigenschaft

Ruft die Unconditional-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Unconditional As String
```

C#-Syntax

```
public string Unconditional {get;set;}
```

UserID-Eigenschaft

Ruft die UserID-Verbindungseigenschaft ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property UserID As String
```

C#-Syntax

```
public string UserID {get;set;}
```

SACConnectionStringBuilderBase-Klasse

Basisklasse der Klasse SACConnectionStringBuilder.

Visual Basic-Syntax

```
Public MustInherit Class SACConnectionStringBuilderBase  
Inherits System.Data.Common.DbConnectionStringBuilder
```

C#-Syntax

```
public abstract class SACConnectionStringBuilderBase :  
System.Data.Common.DbConnectionStringBuilder
```

Basisklassen

- [System.Data.Common.DbConnectionStringBuilder](#)

Abgeleitete Klassen.

- [SACConnectionStringBuilder-Klasse \[SQL Anywhere .NET\] auf Seite 189](#)
- [SATcpOptionsBuilder-Klasse \[SQL Anywhere .NET\] auf Seite 313](#)

Mitglieder

Alle Mitglieder der SACConnectionStringBuilderBase-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Add-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Fügt dem System.Data.Common.DbConnectionStringBuilder -Objekt einen Eintrag mit dem angegebenen Schlüssel und Wert hinzu.
AppendKeyValuePair-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Stellt eine effiziente und sichere Möglichkeit dar, einen Schlüssel und Wert an ein vorhandenes System.Text.StringBuilder -Objekt anzuhängen.

Name	Beschreibung
Clear-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Löscht den Inhalt der System.Data.Common.DbConnectionStringBuilder -Instanz.
ClearPropertyDescriptors-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Löscht die Sammlung von System.ComponentModel.PropertyDescriptor -Objekte auf dem verknüpften System.Data.Common.DbConnectionStringBuilder -Objekt.
ContainsKey-Methode	Ermittelt, ob das SAConnectionStringBuilder -Objekt ein spezifisches Schlüsselwort enthält.
EquivalentTo-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Vergleicht die Verbindungsinformationen in diesem System.Data.Common.DbConnectionStringBuilder -Objekt mit den Verbindungsinformationen im angegebenen Objekt
GetKeyword-Methode	Ruft das Schlüsselwort für die angegebene SAConnectionStringBuilder -Eigenschaft ab.
GetProperties-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Füllt ein angegebenes System.Collections.Hashtable -Objekt mit Informationen zu allen Eigenschaften dieses System.Data.Common.DbConnectionStringBuilder -Objekts.
GetUseLongNameAsKeyword-Methode	Ruft einen booleschen Wert ab, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
Remove-Methode	Löscht den Eintrag mit dem angegebenen Schlüssel aus der SAConnectionStringBuilder -Instanz.
SetUseLongNameAsKeyword-Methode	Legt einen booleschen Wert fest, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
ShouldSerialize-Methode	Gibt an, ob der angegebene Schlüssel in dieser SAConnectionStringBuilder -Instanz vorhanden ist.
ToString-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Gibt die Verbindungszeichenfolge zurück, die diesem System.Data.Common.DbConnectionStringBuilder -Objekt zugeordnet ist.
TryGetValue-Methode	Ruft einen Wert ab, der dem bereitgestellten Schlüssel aus diesem SAConnectionStringBuilder -Objekt entspricht.

Name	Beschreibung
BrowsableConnectionString-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder.ConnectionString -Objekt in den Visual Studio-Designern sichtbar ist, bzw. legt ihn fest.
ConnectionString-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft die Verbindungszeichenfolge ab, die dem DbConnectionStringBuilder -Objekt zugeordnet ist, bzw. legt sie fest.
Count-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft die aktuelle Anzahl von Schlüsseln ab, die in der System.Data.Common.DbConnectionStringBuilder.ConnectionString -Eigenschaft enthalten sind.
IsFixedSize-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder -Objekt eine feste Größe hat.
IsReadOnly-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder -Objekt schreibgeschützt ist.
Keys-Eigenschaft	Ruft ein System.Collections.ICollection-Objekt ab, das die Schlüssel im SAConnectionStringBuilder-Objekt enthält.
this-Eigenschaft	Ruft den Wert des Verbindungsschlüsselworts ab bzw. legt ihn fest.
Values-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft ein System.Collections.ICollection -Objekt ab, das die Werte aus dem System.Data.Common.DbConnectionStringBuilder -Objekt enthält.

ContainsKey-Methode

Ermittelt, ob das SAConnectionStringBuilder-Objekt ein spezifisches Schlüsselwort enthält.

Visual Basic-Syntax

```
Public Overrides Function ContainsKey(  
    ByVal keyword As String  
) As Boolean
```

C#-Syntax

```
public override bool ContainsKey(string keyword)
```

Parameter

- **keyword** Das Schlüsselwort, das im SAConnectionStringBuilder-Objekt gesucht werden soll.

Rückgabe

TRUE, wenn der dem Schlüsselwort zugewiesene Wert angegeben wurde, andernfalls FALSE.

Beispiel

Die folgende Anweisung ermittelt, ob das `SACConnectionStringBuilder`-Objekt das `UserID`-Schlüsselwort enthält.

```
connectString.ContainsKey( "UserID" )
```

GetKeyword-Methode

Ruft das Schlüsselwort für die angegebene `SACConnectionStringBuilder`-Eigenschaft ab.

Visual Basic-Syntax

```
Public Function GetKeyword(ByVal propName As String) As String
```

C#-Syntax

```
public string GetKeyword(string propName)
```

Parameter

- **propName** Der Name der Eigenschaft `SACConnectionStringBuilder`.

Rückgabe

Das Schlüsselwort für die angegebene `SACConnectionStringBuilder`-Eigenschaft.

GetUseLongNameAsKeyword-Methode

Ruft einen booleschen Wert ab, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.

Visual Basic-Syntax

```
Public Function GetUseLongNameAsKeyword() As Boolean
```

C#-Syntax

```
public bool GetUseLongNameAsKeyword()
```

Rückgabe

TRUE, wenn zur Erstellung von Verbindungszeichenfolgen lange Verbindungsparameternamen verwendet werden, andernfalls FALSE.

Bemerkungen

Für die Namen der SQL Anywhere-Verbindungsparameter gibt es sowohl lange als auch kurze Formen. Um z.B. den Namen einer ODBC-Datenquelle in der Verbindungszeichenfolge anzugeben, können Sie einen der beiden folgenden Werte verwenden: `DataSourceName` oder `DSN`. Standardmäßig werden zur Erstellung von Verbindungszeichenfolgen lange Verbindungsparameternamen verwendet.

Siehe auch

- [SAConnectionStringBuilderBase.SetUseLongNameAsKeyword-Methode \[SQL Anywhere .NET\] auf Seite 211](#)

Remove-Methode

Löscht den Eintrag mit dem angegebenen Schlüssel aus der SAConnectionStringBuilder-Instanz.

Visual Basic-Syntax

```
Public Overrides Function Remove(ByVal keyword As String) As Boolean
```

C#-Syntax

```
public override bool Remove(string keyword)
```

Parameter

- **keyword** Der Schlüssel des Schlüssel/Wert-Paares, das aus der Verbindungszeichenfolge in dieser SAConnectionStringBuilder-Instanz entfernt werden soll.

Rückgabe

TRUE, wenn der Schlüssel in der Verbindungszeichenfolge vorhanden war und entfernt wurde, andernfalls FALSE.

SetUseLongNameAsKeyword-Methode

Legt einen booleschen Wert fest, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.

Visual Basic-Syntax

```
Public Sub SetUseLongNameAsKeyword(  
    ByVal useLongNameAsKeyword As Boolean  
)
```

C#-Syntax

```
public void SetUseLongNameAsKeyword(bool useLongNameAsKeyword)
```

Parameter

- **useLongNameAsKeyword** Ein boolescher Wert, der angibt, ob in der Verbindungszeichenfolge ein langer Verbindungsparameternamen verwendet wird.

Bemerkungen

Lange Verbindungsparameternamen werden standardmäßig verwendet.

Siehe auch

- [SAConnectionStringBuilderBase.GetUseLongNameAsKeyword-Methode \[SQL Anywhere .NET\] auf Seite 210](#)

ShouldSerialize-Methode

Gibt an, ob der angegebene Schlüssel in dieser `SACConnectionStringBuilder`-Instanz vorhanden ist.

Visual Basic-Syntax

```
Public Overrides Function ShouldSerialize(  
    ByVal keyword As String  
) As Boolean
```

C#-Syntax

```
public override bool ShouldSerialize(string keyword)
```

Parameter

- **keyword** Der Schlüssel, der im `SACConnectionStringBuilder`-Objekt gesucht werden soll.

Rückgabe

TRUE, wenn die `SACConnectionStringBuilder`-Instanz einen Eintrag mit dem angegebenen Schlüssel enthält, andernfalls FALSE.

TryGetValue-Methode

Ruft einen Wert ab, der dem bereitgestellten Schlüssel aus diesem `SACConnectionStringBuilder`-Objekt entspricht.

Visual Basic-Syntax

```
Public Overrides Function TryGetValue(  
    ByVal keyword As String,  
    ByVal value As Object  
) As Boolean
```

C#-Syntax

```
public override bool TryGetValue(string keyword, out object value)
```

Parameter

- **keyword** Der Schlüssel des abzurufenden Elements.
- **value** Der Wert des Schlüsselworts.

Rückgabe

TRUE, wenn das Schlüsselwort in der Verbindungszeichenfolge gefunden wurde, andernfalls FALSE.

Keys-Eigenschaft

Ruft ein `System.Collections.ICollection`-Objekt ab, das die Schlüssel im `SACConnectionStringBuilder`-Objekt enthält.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Keys As ICollection
```

C#-Syntax

```
public override ICollection Keys {get;}
```

Rückgabe

Ein System.Collections.ICollection-Wert, der die Schlüssel im SAConnectionStringBuilder-Objekt enthält.

this-Eigenschaft

Ruft den Wert des Verbindungsschlüsselworts ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Overrides Property Item(ByVal keyword As String) As Object
```

C#-Syntax

```
public override object this[string keyword] {get;set;}
```

Parameter

- **keyword** Der Name des Verbindungsschlüsselworts.

Bemerkungen

Ein Objekt, das den Wert des angegebenen Verbindungsschlüsselworts repräsentiert.

Wenn das Schlüsselwort oder der Typ unzulässig ist, wird eine Ausnahmebedingung generiert. Das Schlüsselwort unterscheidet nicht zwischen Groß- und Kleinschreibung.

Wenn der Wert festgelegt wird, wird der Wert bei der Übergabe von NULL gelöscht.

SADataAdapter-Klasse

Stellt eine Gruppe von Befehlen und eine Datenbankverbindung dar, die verwendet werden, um ein System.Data.DataSet-Objekt aufzufüllen und eine Datenbank zu aktualisieren.

Visual Basic-Syntax

```
Public NotInheritable Class SADataAdapter  
    Inherits System.Data.Common.DbDataAdapter  
    Implements System.ICloneable
```

C#-Syntax

```
public sealed class SADataAdapter :  
    System.Data.Common.DbDataAdapter,  
    System.ICloneable
```

Basisklassen

- [System.Data.Common.DbDataAdapter](#)
- [System.ICloneable](#)

Mitglieder

Alle Mitglieder der SDataAdapter-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SDataAdapter-Konstruktor	Initialisiert ein SDataAdapter-Objekt.
AddToBatch-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Fügt dem aktuellen Batch ein System.Data.IDbCommand -Objekt hinzu.
CreateRowUpdatedEvent-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Initialisiert eine neue Instanz der Klasse System.Data.Common.RowUpdatedEventArgs .
CreateRowUpdatingEvent-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Initialisiert eine neue Instanz der Klasse System.Data.Common.RowUpdatingEventArgs .
Dispose-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Gibt die unverwalteten Ressourcen frei, die von System.Data.Common.DbDataAdapter verwendet werden, und optional auch die verwalteten Ressourcen.
ExecuteBatch-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Führt den aktuellen Batch aus.
Fill-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Fügt Zeilen zum System.Data.DataSet -Objekt hinzu bzw. aktualisiert sie.
FillSchema-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Fügt ein System.Data.DataTable -Objekt namens "Table" zum angegebenen System.Data.DataSet -Objekt hinzu und konfiguriert das Schema, damit es dem Schema in der Datenquelle entspricht, basierend auf dem angegebenen System.Data.SchemaType -Wert.
GetBatchedParameter-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Gibt einen System.Data.IDataParameter -Wert aus einem der Befehle im aktuellen Batch zurück.
GetBatchedRecordsAffected-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Gibt Informationen über einen einzelnen Aktualisierungsversuche innerhalb einer größeren Batch-Aktualisierung zurück.

Name	Beschreibung
GetFillParameters-Methode	Gibt die Parameter zurück, die Sie bei der Ausführung einer SELECT-Anweisung angeben.
OnRowUpdated-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Löst das RowUpdated-Ereignis in eines .NET Framework-Datenproviders aus.
OnRowUpdating-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Löst das RowUpdating-Ereignis in eines .NET Framework-Datenproviders aus.
Update-Methode (geerbt aus System.Data.Common.DbDataAdapter)	Ruft die entsprechenden INSERT-, UPDATE- oder DELETE-Anweisungen für jede eingefügte, aktualisierte oder gelöschte Zeile im angegebenen Array von System.Data.DataRow -Objekten auf.
DeleteCommand-Eigenschaft	Legt ein SACommand-Objekt fest, das in der Datenbank ausgeführt wird, wenn die Update-Methode aufgerufen wird, um Zeilen aus der Datenbank zu löschen, die gelöschten Zeilen im DataSet entsprechen.
FillCommandBehavior-Eigenschaft (geerbt aus System.Data.Common.DbDataAdapter)	Ruft das Verhalten des Befehls ab, der zum Füllen des Datenadapters verwendet wird, bzw. legt es fest.
InsertCommand-Eigenschaft	Gibt ein SACommand-Objekt an, das in der Datenbank beim Aufruf der Update-Methode ausgeführt wird, die der Datenbank Zeilen hinzufügt, welche den in das DataSet-Objekt eingefügten Zeilen entsprechen.
SelectCommand-Eigenschaft	Gibt ein SACommand-Objekt an, das während Fill oder FillSchema verwendet wird, um aus der Datenbank eine Ergebnismenge abzufragen, die in ein DataSet-Objekt kopiert werden soll.
TableMappings-Eigenschaft	Gibt eine Sammlung an, die die Hauptzuordnung zwischen einer Quelltable und einem DataTable-Objekt enthält.
UpdateBatchSize-Eigenschaft	Ruft die Anzahl von Zeilen ab bzw. legt die Anzahl von Zeilen fest, die bei jedem Zugriff auf den Server verarbeitet werden sollen.
UpdateCommand-Eigenschaft	Gibt ein SACommand-Objekt an, das in der Datenbank ausgeführt wird, wenn die Update-Methode aufgerufen wird, um Zeilen in der Datenbank zu aktualisieren, die aktualisierten Zeilen im DataSet-Objekt entsprechen.

Name	Beschreibung
RowUpdated-Ereignis	Tritt während einer Aktualisierung auf, nachdem ein Befehl in einer Datenquelle ausgeführt wurde.
RowUpdating-Ereignis	Tritt während einer Aktualisierung auf, bevor ein Befehl in einer Datenquelle ausgeführt wurde.

Bemerkungen

Das System.Data.DataSet-Objekt bietet eine Möglichkeit, offline mit Daten zu arbeiten. Der SDataAdapter bietet Methoden, um ein DataSet-Objekt einer Gruppe von SQL-Anweisungen zuzuordnen.

Implements: IDbDataAdapter, IDataAdapter, ICloneable

Weitere Hinweise finden Sie unter [„SDataAdapter: Übersicht“ auf Seite 53](#) und [„Datenzugriff und Datenverarbeitung“ auf Seite 47](#).

SDataAdapter-Konstruktor

Initialisiert ein SDataAdapter-Objekt.

Überladungsliste

Name	Beschreibung
SDataAdapter()-Konstruktor	Initialisiert ein SDataAdapter-Objekt.
SDataAdapter(SACommand)-Konstruktor	Initialisiert ein SDataAdapter-Objekt mit der angegebenen SELECT-Anweisung.
SDataAdapter(String, SAConnection)-Konstruktor	Initialisiert ein SDataAdapter-Objekt mit der angegebenen SELECT-Anweisung und Verbindung.
SDataAdapter(String, String)-Konstruktor	Initialisiert ein SDataAdapter-Objekt mit der angegebenen SELECT-Anweisung und Verbindungszeichenfolge.

SDataAdapter()-Konstruktor

Initialisiert ein SDataAdapter-Objekt.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SDataAdapter()
```

Siehe auch

- [SDataAdapter.SDataAdapter-Konstruktor \[SQL Anywhere .NET\] auf Seite 216](#)

SDataAdapter(SACommand)-Konstruktor

Initialisiert ein SDataAdapter-Objekt mit der angegebenen SELECT-Anweisung.

Visual Basic-Syntax

```
Public Sub New(ByVal selectCommand As SACommand)
```

C#-Syntax

```
public SDataAdapter(SACommand selectCommand)
```

Parameter

- **selectCommand** Ein SACommand-Objekt, das während der Ausführung von System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet) verwendet wird, um Datensätze aus der Datenquelle für die Platzierung im System.Data.DataSet-Objekt auszuwählen

Siehe auch

- [SDataAdapter.SDataAdapter-Konstruktor \[SQL Anywhere .NET\] auf Seite 216](#)

SDataAdapter(String, SAConnection)-Konstruktor

Initialisiert ein SDataAdapter-Objekt mit der angegebenen SELECT-Anweisung und Verbindung.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal selectCommandText As String,  
    ByVal selectConnection As SAConnection  
)
```

C#-Syntax

```
public SDataAdapter(  
    string selectCommandText,  
    SAConnection selectConnection  
)
```

Parameter

- **selectCommandText** Eine SELECT-Anweisung zum Festlegen der SDataAdapter.SelectCommand-Eigenschaft des SDataAdapter-Objekts.
- **selectConnection** Ein SAConnection-Objekt, das eine Verbindung mit einer Datenbank definiert.

Siehe auch

- [SDataAdapter.SDataAdapter-Konstruktor \[SQL Anywhere .NET\] auf Seite 216](#)
- [SDataAdapter.SelectCommand-Eigenschaft \[SQL Anywhere .NET\] auf Seite 220](#)
- [SACConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

SDataAdapter(String, String)-Konstruktor

Initialisiert ein SDataAdapter-Objekt mit der angegebenen SELECT-Anweisung und Verbindungszeichenfolge.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal selectCommandText As String,  
    ByVal selectConnectionString As String  
)
```

C#-Syntax

```
public SDataAdapter(  
    string selectCommandText,  
    string selectConnectionString  
)
```

Parameter

- **selectCommandText** Eine SELECT-Anweisung zum Festlegen der SDataAdapter.SelectCommand-Eigenschaft des SDataAdapter-Objekts.
- **selectConnectionString** Eine Verbindungszeichenfolge für eine SQL Anywhere-Datenbank.

Siehe auch

- [SDataAdapter.SDataAdapter-Konstruktor \[SQL Anywhere .NET\] auf Seite 216](#)
- [SDataAdapter.SelectCommand-Eigenschaft \[SQL Anywhere .NET\] auf Seite 220](#)

GetFillParameters-Methode

Gibt die Parameter zurück, die Sie bei der Ausführung einer SELECT-Anweisung angeben.

Visual Basic-Syntax

```
Public Shadows Function GetFillParameters() As SAParameter()
```

C#-Syntax

```
public new SAParameter[] GetFillParameters()
```

Rückgabe

Ein Array von IDataParameter-Objekten, das die vom Benutzer definierten Parameter enthält.

DeleteCommand-Eigenschaft

Legt ein SACommand-Objekt fest, das in der Datenbank ausgeführt wird, wenn die Update-Methode aufgerufen wird, um Zeilen aus der Datenbank zu löschen, die gelöschten Zeilen im DataSet entsprechen.

Visual Basic-Syntax

```
Public Shadows Property DeleteCommand As SACommand
```

C#-Syntax

```
public new SACommand DeleteCommand {get;set;}
```

Bemerkungen

Wenn diese Eigenschaft nicht aktiviert ist und Primärschlüsselinformationen beim Update im DataSet-Objekt vorhanden sind, kann DeleteCommand automatisch generiert werden, indem SelectCommand definiert und das SACommandBuilder-Objekt benutzt wird. In diesem Fall generiert SACommandBuilder zusätzliche Befehle, die Sie nicht selbst definiert haben. Diese Generierungslogik setzt voraus, dass Informationen zur Spalte im SelectCommand-Objekt vorhanden sind.

Wenn DeleteCommand einem bestehenden SACommand-Objekt zugewiesen wird, wird das SACommand-Objekt nicht geklont. Die DeleteCommand-Eigenschaft enthält eine Referenz auf das bestehende SACommand-Objekt.

Siehe auch

- [SDataAdapter.SelectCommand-Eigenschaft \[SQL Anywhere .NET\] auf Seite 220](#)

InsertCommand-Eigenschaft

Gibt ein SACommand-Objekt an, das in der Datenbank beim Aufruf der Update-Methode ausgeführt wird, die der Datenbank Zeilen hinzufügt, welche den in das DataSet-Objekt eingefügten Zeilen entsprechen.

Visual Basic-Syntax

```
Public Shadows Property InsertCommand As SACommand
```

C#-Syntax

```
public new SACommand InsertCommand {get;set;}
```

Bemerkungen

Der SACommandBuilder verlangt nicht, dass Spalten ein InsertCommand-Objekt generieren.

Wenn das InsertCommand-Objekt einem bestehenden SACommand-Objekt zugewiesen wurde, wird das SACommand-Objekt nicht geklont. Das InsertCommand-Objekt behält einen Verweis auf das bestehende SACommand-Objekt.

Wenn dieser Befehl Zeilen zurückgibt, können die Zeilen dem DataSet-Objekt hinzugefügt werden, sofern Sie die UpdatedRowSource-Eigenschaft des SACommand-Objekts entsprechend definiert haben.

SelectCommand-Eigenschaft

Gibt ein SCommand-Objekt an, das während Fill oder FillSchema verwendet wird, um aus der Datenbank eine Ergebnismenge abzufragen, die in ein DataSet-Objekt kopiert werden soll.

Visual Basic-Syntax

```
Public Shadows Property SelectCommand As SCommand
```

C#-Syntax

```
public new SCommand SelectCommand {get;set;}
```

Bemerkungen

Wenn SelectCommand einem vorher erstellten SCommand-Objekt zugewiesen wurde, wird das SCommand-Objekt nicht geklont. SelectCommand behält einen Verweis auf ein früher erstelltes SCommand-Objekt.

Wenn die SelectCommand-Eigenschaft keine Zeilen zurückgibt, werden dem DataSet-Objekt keine Zeilen hinzugefügt und es wird keine Ausnahmebedingung generiert.

Die Select-Anweisung kann auch im SDataAdapter-Konstruktor angegeben werden.

TableMappings-Eigenschaft

Gibt eine Sammlung an, die die Hauptzuordnung zwischen einer Quelltable und einem DataTable-Objekt enthält.

Visual Basic-Syntax

```
Public ReadOnly Shadows Property TableMappings As  
DataTableMappingCollection
```

C#-Syntax

```
public new DataTableMappingCollection TableMappings {get;}
```

Bemerkungen

Der Standardwert ist eine leere Sammlung.

Wenn die Änderungen abgeglichen werden, benutzt das SDataAdapter-Objekt die DataTableMappingCollection-Sammlung, um die von der Datenquelle benutzten Spaltennamen den vom DataSet-Objekt benutzten Spaltennamen zuzuordnen.

Die TableMappings-Eigenschaft ist in .NET Compact Framework 2.0 nicht verfügbar.

UpdateBatchSize-Eigenschaft

Ruft die Anzahl von Zeilen ab bzw. legt die Anzahl von Zeilen fest, die bei jedem Zugriff auf den Server verarbeitet werden sollen.

Visual Basic-Syntax

```
Public Overrides Property UpdateBatchSize As Integer
```

C#-Syntax

```
public override int UpdateBatchSize {get;set;}
```

Bemerkungen

Der Standardwert ist 1.

Wenn der Wert auf einen größeren Wert als 1 gesetzt wird, führt `SADDataAdapter.Update` alle Insert-Anweisungen in Batches aus. Die Lösch- und Aktualisierungsvorgänge werden wie zuvor sequenziell ausgeführt, aber Einfügungen werden anschließend in Batches in der Größe von `UpdateBatchSize` ausgeführt. Wenn der Wert auf 0 gesetzt wird, sendet `Update` die Insert-Anweisungen in einem einzigen Batch.

Wenn der Wert auf einen größeren Wert als 1 gesetzt wird, wird `SADDataAdapter.Fill` veranlasst, alle Insert-Anweisungen in Batches auszuführen. Die Lösch- und Aktualisierungsvorgänge werden wie zuvor sequenziell ausgeführt, aber Einfügungen werden anschließend in Batches in der Größe von `UpdateBatchSize` ausgeführt.

Wenn der Wert auf 0 gesetzt wird, sendet `Fill` die Insert-Anweisungen in einem einzigen Batch.

Die Verwendung eines negativen Werts wird als Fehler behandelt.

Wenn `UpdateBatchSize` auf einen anderen Wert als 1 gesetzt wird und die `InsertCommand`-Eigenschaft auf eine andere Anweisung als `INSERT`, dann wird beim Aufruf von `Fill` eine Ausnahmebedingung generiert.

Dieses Verhalten unterscheidet sich von `SqlDataAdapter`. `SqlDataAdapter` verwendet für alle Befehlstypen Anweisungsfolgen.

UpdateCommand-Eigenschaft

Gibt ein `SACommand`-Objekt an, das in der Datenbank ausgeführt wird, wenn die `Update`-Methode aufgerufen wird, um Zeilen in der Datenbank zu aktualisieren, die aktualisierten Zeilen im `DataSet`-Objekt entsprechen.

Visual Basic-Syntax

```
Public Shadows Property UpdateCommand As SACommand
```

C#-Syntax

```
public new SACommand UpdateCommand {get;set;}
```

Bemerkungen

Wenn die Aktualisierung läuft, diese Eigenschaft nicht definiert ist und Primärschlüsselinformationen im `SelectCommand`-Objekt zur Verfügung stehen, kann `UpdateCommand` automatisch generiert werden, sofern Sie die `SelectCommand`-Eigenschaft angeben und das `SACommandBuilder`-Objekt verwenden. Danach werden etwaige zusätzliche Befehle, die Sie nicht definiert haben, vom `SACommandBuilder`-

Objekt generiert. Diese Generierungslogik setzt voraus, dass Informationen zur Spalte im SelectCommand-Objekt vorhanden sind.

Wenn UpdateCommand einem vorher erstellten SCommand-Objekt zugewiesen wurde, wird das SCommand-Objekt nicht geklont. UpdateCommand behält einen Verweis auf ein früher erstelltes SCommand-Objekt.

Wenn dieser Befehl Zeilen zurückgibt, können diese Zeilen mit dem DataSet-Objekt fusioniert werden, sofern Sie die UpdatedRowSource-Eigenschaft im SCommand-Objekt entsprechend eingestellt haben.

RowUpdated-Ereignis

Tritt während einer Aktualisierung auf, nachdem ein Befehl in einer Datenquelle ausgeführt wurde.

Visual Basic-Syntax

```
Public Event RowUpdated As SRowUpdatedEventHandler
```

C#-Syntax

```
public event SRowUpdatedEventHandler RowUpdated;
```

Bemerkungen

Das Ereignis wird ausgelöst, wenn eine Aktualisierung versucht wird.

Der Event-Handler erhält ein Argument des Typs SRowUpdatedEventArgs, das Daten im Zusammenhang mit diesem Ereignis enthält.

Weitere Hinweise finden Sie in der Dokumentation zum .NET Framework unter der Rubrik OleDbDataAdapter.RowUpdated Event.

RowUpdating-Ereignis

Tritt während einer Aktualisierung auf, bevor ein Befehl in einer Datenquelle ausgeführt wurde.

Visual Basic-Syntax

```
Public Event RowUpdating As SRowUpdatingEventHandler
```

C#-Syntax

```
public event SRowUpdatingEventHandler RowUpdating;
```

Bemerkungen

Das Ereignis wird ausgelöst, wenn eine Aktualisierung versucht wird.

Der Event-Handler erhält ein Argument des Typs SRowUpdatingEventArgs, das Daten im Zusammenhang mit diesem Ereignis enthält.

Weitere Hinweise finden Sie in der Dokumentation zum .NET Framework unter der Rubrik OleDbDataAdapter.RowUpdating Event.

SADataReader-Klasse

Eine schreibgeschützte Ergebnismenge nur zum Weitergeben aus einer Abfrage oder einer gespeicherten Prozedur.

Visual Basic-Syntax

```
Public NotInheritable Class SADataReader
    Inherits System.Data.Common.DbDataReader
    Implements System.ComponentModel.IListSource
```

C#-Syntax

```
public sealed class SADataReader :
    System.Data.Common.DbDataReader,
    System.ComponentModel.IListSource
```

Basisklassen

- [System.Data.Common.DbDataReader](#)
- [System.ComponentModel.IListSource](#)

Mitglieder

Alle Mitglieder der SADataReader-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Close-Methode	Schließt das SADataReader-Objekt.
Dispose-Methode (geerbt aus System.Data.Common.DbDataReader)	Gibt alle Ressourcen frei, die von der aktuellen Instanz der Klasse System.Data.Common.DbDataReader verwendet werden.
GetBoolean-Methode	Gibt den Wert der angegebenen Spalte als booleschen Wert zurück.
GetByte-Methode	Gibt den Wert der angegebenen Spalte als Byte-Wert zurück.
GetBytes-Methode	Liest einen Bytestrom aus dem angegebenen Spaltenoffset als Array in den Puffer ein und beginnt dabei am angegebenen Pufferoffset.
GetChar-Methode	Gibt den Wert der angegebenen Spalte als Zeichen zurück.
GetChars-Methode	Liest einen Zeichenstrom aus dem angegebenen Spaltenoffset als Array in den Puffer ein und beginnt dabei am angegebenen Pufferoffset.
GetDate-Methode	Diese Methode wird nicht unterstützt.
GetDataTypeName-Methode	Gibt den Namen des Quelldatentyps zurück.

Name	Beschreibung
GetDateTime-Methode	Gibt den Wert der angegebenen Spalte als DateTime-Objekt zurück.
GetDateTimeOffset-Methode	Gibt den Wert der angegebenen Spalte als DateTimeOffset-Objekt zurück.
GetDbDataReader-Methode (geerbt aus System.Data.Common.DbDataReader)	Gibt ein System.Data.Common.DbDataReader -Objekt für die angeforderte Spaltenordinalzahl zurück, das mit einer providerspezifischen Implementierung aufgehoben werden kann.
GetDecimal-Methode	Gibt den Wert der angegebenen Spalte als Decimal-Objekt zurück.
GetDouble-Methode	Gibt den Wert der angegebenen Spalte als doppelgenaue Gleitkommazahl zurück.
GetEnumerator-Methode	Gibt einen System.Collections.IEnumerator zurück, der das SAdtaReader-Objekt durchläuft.
GetFieldType-Methode	Gibt den Datentyp des Objekts zurück.
GetFieldValue(T)-Methode (geerbt aus System.Data.Common.DbDataReader)	Ruft synchron den Wert der angegebenen Spalte als Typ ab.
GetFieldValueAsync-Methode (geerbt aus System.Data.Common.DbDataReader)	Ruft asynchron den Wert der angegebenen Spalte als Typ ab.
GetFloat-Methode	Gibt den Wert der angegebenen Spalte als einfachgenaue Gleitkommazahl zurück.
GetGuid-Methode	Gibt den Wert der angegebenen Spalte als globalen eindeutigen Bezeichner (Global Unique Identifier, GUID) zurück.
GetInt16-Methode	Gibt den Wert der angegebenen Spalte als 16-Bit-Ganzzahl mit Vorzeichen zurück.
GetInt32-Methode	Gibt den Wert der angegebenen Spalte als 32-Bit-Ganzzahl mit Vorzeichen zurück.
GetInt64-Methode	Gibt den Wert der angegebenen Spalte als 64-Bit-Ganzzahl mit Vorzeichen zurück.
GetName-Methode	Gibt den Namen der angegebenen Spalte zurück.
GetOrdinal-Methode	Gibt die Ordinalzahl der Spalte für den betreffenden Spaltennamen zurück.

Name	Beschreibung
GetProviderSpecificFieldType-Methode (geerbt aus <code>System.Data.Common.DbDataReader</code>)	Gibt den providerspezifischen Feldtyp der angegebenen Spalte zurück.
GetProviderSpecificValue-Methode (geerbt aus <code>System.Data.Common.DbDataReader</code>)	Ruft den Wert der angegebenen Spalte als Instanz von System.Object ab.
GetProviderSpecificValues-Methode (geerbt aus <code>System.Data.Common.DbDataReader</code>)	Ruft alle providerspezifischen Attributspalten in der Sammlung für die aktuelle Zeile ab.
GetSchemaTable-Methode	Gibt ein <code>DataTable</code> -Objekt zurück, das die Spalten-Metadaten des <code>SADataReader</code> -Objekts beschreibt.
GetStream-Methode (geerbt aus <code>System.Data.Common.DbDataReader</code>)	Ruft Daten als System.IO.Stream -Objekt ab.
GetString-Methode	Gibt den Wert der angegebenen Spalte als Zeichenfolge zurück.
GetTextReader-Methode (geerbt aus <code>System.Data.Common.DbDataReader</code>)	Ruft Daten als System.IO.TextReader -Objekt ab.
GetTimeSpan-Methode	Gibt den Wert der angegebenen Spalte als <code>TimeSpan</code> -Objekt zurück.
GetUInt16-Methode	Gibt den Wert der angegebenen Spalte als 16-Bit-Ganzzahl ohne Vorzeichen zurück.
GetUInt32-Methode	Gibt den Wert der angegebenen Spalte als 32-Bit-Ganzzahl ohne Vorzeichen zurück.
GetUInt64-Methode	Gibt den Wert der angegebenen Spalte als 64-Bit-Ganzzahl ohne Vorzeichen zurück.
GetValue-Methode	Gibt den Wert der angegebenen Spalte als Objekt zurück.
GetValues-Methode	Ruft alle Spalten in der aktuellen Zeile ab.
IsDBNull-Methode	Gibt einen Wert zurück, der anzeigt, ob die Spalte NULL enthält.

Name	Beschreibung
IsDBNullAsync-Methode (geerbt aus System.Data.Common.DbDataReader)	Eine asynchrone Version von System.Data.Common.DbDataReader.IsDBNull(System.Int32) , die einen Wert abrufen, der anzeigt, ob die Spalte nicht vorhandene oder fehlende Werte enthält.
myDispose-Methode	Gibt dem Objekt zugeordneten Ressourcen frei.
NextResult-Methode	Verschiebt das SDataReader-Objekt zur nächsten Ergebnismenge, wenn Abfragen verarbeitet werden, die mehrere Ergebnismengen zurückgeben.
NextResultAsync-Methode (geerbt aus System.Data.Common.DbDataReader)	Eine asynchrone Version von System.Data.Common.DbDataReader.NextResult , die den Reader zum nächsten Ergebnis verschiebt, wenn die Ergebnisse für einen Anweisungsbatch gelesen werden. Ruft System.Data.Common.DbDataReader.NextResultAsync(System.Threading.CancellationToken) mit CancellationToken.None auf.
Read-Methode	Liest die nächste Zeile der Ergebnismenge und verschiebt das SDataReader-Objekt in diese Zeile.
ReadAsync-Methode (geerbt aus System.Data.Common.DbDataReader)	Eine asynchrone Version von System.Data.Common.DbDataReader.Read , die das Reader-Objekt zum nächsten Datensatz in einer Ergebnismenge verschiebt.
Depth-Eigenschaft	Ruft einen Wert ab, der die Tiefe der Verschachtelung für die aktuelle Zeile anzeigt.
FieldCount-Eigenschaft	Ruft die Anzahl der Spalten in der Ergebnismenge ab
HasRows-Eigenschaft	Ruft einen Wert ab, der angibt, ob SDataReader mindestens eine Zeile enthält.
IsClosed-Eigenschaft	Ruft einen Wert ab, der angibt, ob das SDataReader-Objekt geschlossen ist.
RecordsAffected-Eigenschaft	Die Anzahl der durch eine SQL-Anweisung geänderten, eingefügten oder gelöschten Zeilen.
this-Eigenschaft	Gibt den Wert einer Spalte in ihrem nativen Format zurück.
VisibleFieldCount-Eigenschaft (geerbt aus System.Data.Common.DbDataReader)	Ruft die Anzahl der Felder in System.Data.Common.DbDataReader ab, die nicht ausgeblendet sind.

Bemerkungen

Es gibt keinen Konstruktor für `SADataReader`. Um ein `SADataReader`-Objekt zu erhalten, führen Sie ein `SACCommand`-Objekt aus:

```
SACCommand cmd = new SACCommand(  
    "SELECT EmployeeID FROM Employees", conn );  
SADataReader reader = cmd.ExecuteReader();
```

Sie können sich durch ein `SADataReader`-Objekt nur vorwärts bewegen. Wenn Sie ein flexibleres Objekt für die Verarbeitung von Ergebnissen benötigen, benutzen Sie ein `SADataAdapter`-Objekt.

Das `SADataReader`-Objekt ruft Zeilen nach Bedarf ab, während das `SADataAdapter`-Objekt alle Zeilen einer Ergebnismenge abrufen, bevor Sie mit dem Objekt weitere Vorgänge durchführen können. Bei großen Ergebnismengen bietet das `SADataReader`-Objekt daher viel kürzere Antwortzeiten.

Implements: `IDataReader`, `IDisposable`, `IDataRecord`, `IListSource`

Weitere Hinweise finden Sie unter [„Datenzugriff und Datenverarbeitung“ auf Seite 47](#).

Siehe auch

- [SACCommand.ExecuteReader-Methode \[SQL Anywhere .NET\] auf Seite 145](#)

Close-Methode

Schließt das `SADataReader`-Objekt.

Visual Basic-Syntax

```
Public Overrides Sub Close()
```

C#-Syntax

```
public override void Close()
```

Bemerkungen

Sie müssen explizit die `Close`-Methode aufrufen, wenn Sie das `SADataReader`-Objekt nicht mehr benötigen.

Im Autocommit-Modus wird beim Schließen des `SADataReader`-Objekts als Nebeneffekt ein `COMMIT` ausgegeben.

GetBoolean-Methode

Gibt den Wert der angegebenen Spalte als Booleschen Wert zurück.

Visual Basic-Syntax

```
Public Overrides Function GetBoolean(  
    ByVal ordinal As Integer  
) As Boolean
```

C#-Syntax

```
public override bool GetBoolean(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Booleschen Format vorliegen.

Siehe auch

- [SADataReader.GetOrdinal-Methode \[SQL Anywhere .NET\] auf Seite 238](#)
- [SADataReader.GetFieldType-Methode \[SQL Anywhere .NET\] auf Seite 235](#)

GetByte-Methode

Gibt den Wert der angegebenen Spalte als Byte-Wert zurück.

Visual Basic-Syntax

```
Public Overrides Function GetByte(ByVal ordinal As Integer) As Byte
```

C#-Syntax

```
public override byte GetByte(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Byteformat vorliegen.

GetBytes-Methode

Liest einen Bytestrom aus dem angegebenen Spaltenoffset als Array in den Puffer ein und beginnt dabei am angegebenen Pufferoffset.

Visual Basic-Syntax

```
Public Overrides Function GetBytes(  
    ByVal ordinal As Integer,  
    ByVal dataIndex As Long,  
    ByVal buffer As Byte(),  
    ByVal bufferIndex As Integer,  
    ByVal length As Integer  
) As Long
```

C#-Syntax

```
public override long GetBytes(  
    int ordinal,  
    long dataIndex,  
    byte[] buffer,  
    int bufferIndex,  
    int length  
)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.
- **dataIndex** Der Index innerhalb des Spaltenwerts, aus dem Byte ausgelesen werden.
- **buffer** Ein Array, in dem Daten gespeichert werden.
- **bufferIndex** Der Index im Array zum Beginn des Kopierens von Daten.
- **length** Die maximale Länge der Daten, die in den angegebenen Puffer kopiert werden sollen

Rückgabe

Die Anzahl der gelesenen Byte

Bemerkungen

GetBytes gibt die Anzahl der verfügbaren Byte im Feld zurück. In den meisten Fällen ist dies die exakte Länge des Feldes. Allerdings kann die zurückgegebene Zahl geringer sein als die wahre Länge des Feldes, wenn GetBytes bereits benutzt wurde, um Byte aus dem Feld abzurufen. Dies kann beispielsweise vorkommen, wenn das SDataReader-Objekt eine umfangreiche Datenstruktur in den Puffer einliest.

Wenn Sie einen Puffer übergeben, der eine Nullreferenz ist ("Nothing" in Visual Basic), gibt GetBytes die Länge des Feldes in Byte zurück.

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten als Byte-Array vorliegen.

GetChar-Methode

Gibt den Wert der angegebenen Spalte als Zeichen zurück.

Visual Basic-Syntax

```
Public Overrides Function GetChar(ByVal ordinal As Integer) As Char
```

C#-Syntax

```
public override char GetChar(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Zeichenformat vorliegen.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetChars-Methode

Liest einen Zeichenstrom aus dem angegebenen Spaltenoffset als Array in den Puffer ein und beginnt dabei am angegebenen Pufferoffset.

Visual Basic-Syntax

```
Public Overrides Function GetChars(  
    ByVal ordinal As Integer,  
    ByVal dataIndex As Long,  
    ByVal buffer As Char(),  
    ByVal bufferIndex As Integer,  
    ByVal length As Integer  
) As Long
```

C#-Syntax

```
public override long GetChars(  
    int ordinal,  
    long dataIndex,  
    char[] buffer,  
    int bufferIndex,  
    int length  
)
```

Parameter

- **ordinal** Die auf Null basierende Spalten-Ordinalzahl.
- **dataIndex** Der Index in der Zeile, ab der der Lesevorgang durchgeführt werden soll.
- **buffer** Der Puffer, in den die Daten kopiert werden sollen.
- **bufferIndex** Der Index für den Puffer, ab dem der Lesevorgang beginnt.
- **length** Die Anzahl der zu lesenden Zeichen.

Rückgabe

Die tatsächliche Anzahl der gelesenen Zeichen.

Bemerkungen

GetChars gibt die Anzahl der verfügbaren Zeichen im Feld zurück. In den meisten Fällen ist dies die exakte Länge des Feldes. Allerdings kann die zurückgegebene Anzahl geringer sein als die wahre Länge des Feldes, wenn GetChars bereits benutzt wurde, um Zeichen aus dem Feld zu beziehen. Dies kann beispielsweise vorkommen, wenn das SADataReader-Objekt eine umfangreiche Datenstruktur in den Puffer einliest.

Wenn Sie einen Puffer übergeben, der eine Nullreferenz ist ("Nothing" in Visual Basic), gibt GetChars die Länge des Feldes in Zeichen zurück.

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als Zeichen-Array vorliegen.

Weitere Hinweise zur BLOB-Verarbeitung finden Sie unter [„BLOBs“ auf Seite 62](#).

GetDate-Methode

Diese Methode wird nicht unterstützt.

Visual Basic-Syntax

```
Public Shadows Function GetData(ByVal i As Integer) As IDataReader
```

C#-Syntax

```
public new IDataReader GetData(int i)
```

Bemerkungen

Wenn sie aufgerufen wird, gibt sie eine InvalidOperationException-Ausnahmebedingung aus.

Siehe auch

- [System.InvalidOperationException](#)

GetDataTypeName-Methode

Gibt den Namen des Quelldatentyps zurück.

Visual Basic-Syntax

```
Public Overrides Function GetDataTypeName(  
    ByVal index As Integer  
    ) As String
```

C#-Syntax

```
public override string GetDataTypeName(int index)
```

Parameter

- **index** Die auf Null basierende Spalten-Ordinalzahl.

Rückgabe

Der Name des Quelldatentyps.

GetDateTime-Methode

Gibt den Wert der angegebenen Spalte als DateTime-Objekt zurück.

Visual Basic-Syntax

```
Public Overrides Function GetDateTime(ByVal ordinal As Integer) As Date
```

C#-Syntax

```
public override DateTime GetDateTime(int ordinal)
```

Parameter

- **ordinal** Die auf Null basierende Spalten-Ordinalzahl.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als DateTime-Objekt vorliegen.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetDateTimeOffset-Methode

Gibt den Wert der angegebenen Spalte als DateTimeOffset-Objekt zurück.

Visual Basic-Syntax

```
Public Function GetDateTimeOffset(  
    ByVal ordinal As Integer  
) As DateTimeOffset
```

C#-Syntax

```
public DateTimeOffset GetDateTimeOffset(int ordinal)
```

Parameter

- **ordinal** Die auf Null basierende Spalten-Ordinalzahl.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als DateTimeOffset-Objekt vorliegen.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetDecimal-Methode

Gibt den Wert der angegebenen Spalte als Decimal-Objekt zurück.

Visual Basic-Syntax

```
Public Overrides Function GetDecimal(  
    ByVal ordinal As Integer  
) As Decimal
```

C#-Syntax

```
public override decimal GetDecimal(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als Decimal-Objekt vorliegen.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetDouble-Methode

Gibt den Wert der angegebenen Spalte als doppelgenaue Gleitkommazahl zurück.

Visual Basic-Syntax

```
Public Overrides Function GetDouble(ByVal ordinal As Integer) As Double
```

C#-Syntax

```
public override double GetDouble(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als doppelgenaue Gleitkommazahl vorliegen.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetEnumerator-Methode

Gibt einen `System.Collections.IEnumerator` zurück, der das `SADataReader`-Objekt durchläuft.

Visual Basic-Syntax

```
Public Overrides Function GetEnumerator()  
    As System.Collections.IEnumerator
```

C#-Syntax

```
public override IEnumerator GetEnumerator()
```

Rückgabe

Ein System.Collections.IEnumerator für das SADATAReader-Objekt.

Siehe auch

- [SADATAReader-Klasse \[SQL Anywhere .NET\] auf Seite 223](#)

GetFieldType-Methode

Gibt den Datentyp des Objekts zurück.

Visual Basic-Syntax

```
Public Overrides Function GetFieldType(ByVal index As Integer) As Type
```

C#-Syntax

```
public override Type GetFieldType(int index)
```

Parameter

- **index** Die auf Null basierende Spalten-Ordinalzahl.

Rückgabe

Der Feldtyp, der als Datentyp des Objekts gilt.

GetFloat-Methode

Gibt den Wert der angegebenen Spalte als einfachgenaue Gleitkommazahl zurück.

Visual Basic-Syntax

```
Public Overrides Function GetFloat(ByVal ordinal As Integer) As Single
```

C#-Syntax

```
public override float GetFloat(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als einfachgenaue Gleitkommazahl vorliegen.

Rufen Sie die `SADaReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADaReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetGuid-Methode

Gibt den Wert der angegebenen Spalte als globalen eindeutigen Bezeichner (Global Unique Identifier, GUID) zurück.

Visual Basic-Syntax

```
Public Overrides Function GetGuid(ByVal ordinal As Integer) As Guid
```

C#-Syntax

```
public override Guid GetGuid(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Die abgerufenen Daten müssen bereits ein globaler eindeutiger Bezeichner oder binary(16) sein.

Rufen Sie die `SADaReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADaReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetInt16-Methode

Gibt den Wert der angegebenen Spalte als 16-Bit-Ganzzahl mit Vorzeichen zurück.

Visual Basic-Syntax

```
Public Overrides Function GetInt16(ByVal ordinal As Integer) As Short
```

C#-Syntax

```
public override short GetInt16(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Format 16-Bit-Ganzzahl mit Vorzeichen vorliegen.

GetInt32-Methode

Gibt den Wert der angegebenen Spalte als 32-Bit-Ganzzahl mit Vorzeichen zurück.

Visual Basic-Syntax

```
Public Overrides Function GetInt32(ByVal ordinal As Integer) As Integer
```

C#-Syntax

```
public override int GetInt32(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Format 32-Bit-Ganzzahl mit Vorzeichen vorliegen.

GetInt64-Methode

Gibt den Wert der angegebenen Spalte als 64-Bit-Ganzzahl mit Vorzeichen zurück.

Visual Basic-Syntax

```
Public Overrides Function GetInt64(ByVal ordinal As Integer) As Long
```

C#-Syntax

```
public override long GetInt64(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Format 64-Bit-Ganzzahl mit Vorzeichen vorliegen.

GetName-Methode

Gibt den Namen der angegebenen Spalte zurück.

Visual Basic-Syntax

```
Public Overrides Function GetName(ByVal index As Integer) As String
```

C#-Syntax

```
public override string GetName(int index)
```

Parameter

- **index** Der auf Null basierende Index der Spalte.

Rückgabe

Der Name der angegebenen Spalte.

GetOrdinal-Methode

Gibt die Ordinalzahl der Spalte für den betreffenden Spaltennamen zurück.

Visual Basic-Syntax

```
Public Overrides Function GetOrdinal(ByVal name As String) As Integer
```

C#-Syntax

```
public override int GetOrdinal(string name)
```

Parameter

- **name** Der Spaltenname.

Rückgabe

Die auf Null basierende Spalten-Ordinalzahl.

Bemerkungen

GetOrdinal führt erst eine Suche durch, bei der die Groß- und Kleinschreibung beachtet wird. Wenn sie fehlschlägt, wird eine zweite Suche durchgeführt, bei der die Groß- und Kleinschreibung nicht beachtet wird.

GetOrdinal berücksichtigt die japanische kana-Breite nicht.

Da Suchvorgänge auf Ordinalzahlbasis wirksamer sind als die Suche mit Namen, ist es ineffizient, GetOrdinal innerhalb einer Schleife aufzurufen. Sie können Zeit sparen, indem Sie GetOrdinal einmal aufrufen und die Ergebnisse einer Ganzzahlvariable zuweisen, die in der Schleife verwendet wird.

GetSchemaTable-Methode

Gibt ein DataTable-Objekt zurück, das die Spalten-Metadaten des SADATAReader-Objekts beschreibt.

Visual Basic-Syntax

```
Public Overrides Function GetSchemaTable() As DataTable
```

C#-Syntax

```
public override DataTable GetSchemaTable()
```

Rückgabe

Ein DataTable-Objekt, das die Spalten-Metadaten beschreibt.

Bemerkungen

Diese Methode gibt Metadaten über jede Spalte in der nachstehenden Reihenfolge zurück:

DataTable-Spalte	Beschreibung
ColumnName	Der Name der Spalte oder eine Nullreferenz ("Nothing" in Visual Basic), wenn die Spalte keinen Namen hat. Wenn die Spalte in der SQL-Abfrage einen Aliasnamen hat, wird der Aliasname zurückgegeben. Beachten Sie, dass in Ergebnismengen nicht alle Spalten Namen haben und nicht alle Spaltennamen eindeutig sind.
ColumnOrdinal	Die ID der Spalte. Der Wert ist im Bereich [0, FieldCount - 1].
ColumnSize	Für Spalten mit Größenangaben ist dies die maximale Länge eines Werts in der Spalte. Für andere Spalten ist dies die Größe des Datentyps in Bytes.
NumericPrecision	Die Gesamtstellenzahl einer numerischen Spalte oder DBNull, wenn die Spalte nicht numerisch ist.
NumericScale	Die Anzahl der Dezimalstellen einer numerischen Spalte oder DBNull, wenn die Spalte nicht numerisch ist.
IsUnique	TRUE, wenn die Spalte eine nicht berechnete eindeutige Spalte in der Tabelle (BaseTableName) ist, aus der sie stammt.

DataTable-Spalte	Beschreibung
IsKey	TRUE, wenn die Spalte Teil einer Gruppe von Spalten in der Ergebnismenge ist, die zusammen aus einem eindeutigen Schlüssel für die Ergebnismenge genommen werden. Die Gruppe der Spalten, für die IsKey den Wert TRUE hat, muss nicht die Mindestgruppe sein, die eine Zeile in der Ergebnismenge eindeutig kennzeichnet.
BaseServerName	Der Name des SQL Anywhere-Datenbankservers, der von SADATAReader verwendet wird.
BaseCatalogName	Der Name des Katalogs in der Datenbank, die die Spalte enthält. Dieser Wert ist stets DBNull.
BaseColumnName	Der ursprüngliche Name in der Tabelle BaseTableName der Datenbank oder DBNull, wenn die Spalte berechnet wird oder wenn die Informationen nicht ermittelt werden können.
BaseSchemaName	Der Name des Schemas in der Datenbank, das die Spalte enthält.
BaseTableName	Der Name der Tabelle in der Datenbank, die die Spalte enthält, oder DBNull, wenn die Spalte berechnet wird oder diese Information nicht ermittelt werden kann.
DataType	Der .NET-Datentyp, der für diesen Spaltentyp am besten geeignet ist.
AllowDBNull	TRUE, wenn die Spalte nullwertfähig ist. FALSE, wenn die Spalte nicht nullwertfähig ist oder diese Information nicht ermittelt werden kann.
ProviderType	Der Spaltentyp.
IsAliased	TRUE, wenn der Spaltenname ein Aliasname ist; andernfalls FALSE.
IsExpression	TRUE, wenn die Spalte ein Ausdruck ist. FALSE, wenn Sie einen Wert enthält.
IsIdentity	TRUE, wenn die Spalte eine Identity-Spalte ist, andernfalls FALSE.
IsAutoIncrement	TRUE, wenn die Spalte eine Autoincrement- oder globalAutoincrement-Spalte ist, andernfalls FALSE (oder wenn diese Information nicht ermittelt werden kann)
IsRowVersion	TRUE, wenn die Spalte einen beständigen Zeilenbezeichner enthält, der nicht beschrieben werden kann und einen Wert ohne Bedeutung hat, der lediglich die Zeile kennzeichnet.
IsHidden	TRUE, wenn die Spalte ausgeblendet ist, andernfalls FALSE.
IsLong	TRUE, wenn die Spalte vom Datentyp Long Varchar, Long NVarchar oder Long Binary ist, andernfalls FALSE.

DataTable-Spalte	Beschreibung
IsReadOnly	TRUE, wenn die Spalte schreibgeschützt ist. FALSE, wenn sie geändert werden kann oder die Zugriffsberechtigung nicht ermittelt werden kann.

Weitere Hinweise zu diesen Spalten finden Sie in der Dokumentation zum .NET Framework unter der Rubrik `SqlDataReader.GetSchemaTable`.

Weitere Hinweise finden Sie unter „[SACommand: Ergebnismengenschema mit GetSchemaTable abrufen](#)“ auf Seite 50.

GetString-Methode

Gibt den Wert der angegebenen Spalte als Zeichenfolge zurück.

Visual Basic-Syntax

```
Public Overrides Function GetString(ByVal ordinal As Integer) As String
```

C#-Syntax

```
public override string GetString(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits als Zeichenfolge vorliegen.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetTimeSpan-Methode

Gibt den Wert der angegebenen Spalte als TimeSpan-Objekt zurück.

Visual Basic-Syntax

```
Public Function GetTimeSpan(ByVal ordinal As Integer) As TimeSpan
```

C#-Syntax

```
public TimeSpan GetTimeSpan(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Die Spalte muss ein TIME-Datentyp von SQL Anywhere sein. Die Daten werden in TimeSpan konvertiert. Die Eigenschaft "Days" von TimeSpan ist immer auf 0 gesetzt.

Rufen Sie die `SADataReader.IsDBNull`-Methode auf, um eine Prüfung auf NULL durchzuführen, bevor Sie diese Methode aufrufen.

Weitere Hinweise finden Sie unter [„Zeitwerte“ auf Seite 63](#).

Siehe auch

- [SADataReader.IsDBNull-Methode \[SQL Anywhere .NET\] auf Seite 245](#)

GetUInt16-Methode

Gibt den Wert der angegebenen Spalte als 16-Bit-Ganzzahl ohne Vorzeichen zurück.

Visual Basic-Syntax

```
Public Function GetUInt16(ByVal ordinal As Integer) As UShort
```

C#-Syntax

```
public ushort GetUInt16(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Format 16-Bit-Ganzzahl ohne Vorzeichen vorliegen.

GetUInt32-Methode

Gibt den Wert der angegebenen Spalte als 32-Bit-Ganzzahl ohne Vorzeichen zurück.

Visual Basic-Syntax

```
Public Function GetUInt32(ByVal ordinal As Integer) As UInteger
```

C#-Syntax

```
public uint GetUInt32(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Format 32-Bit-Ganzzahl ohne Vorzeichen vorliegen.

GetUInt64-Methode

Gibt den Wert der angegebenen Spalte als 64-Bit-Ganzzahl ohne Vorzeichen zurück.

Visual Basic-Syntax

```
Public Function GetUInt64(ByVal ordinal As Integer) As ULong
```

C#-Syntax

```
public ulong GetUInt64(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte.

Bemerkungen

Es werden keine Konvertierungen durchgeführt, daher müssen die abgerufenen Daten bereits im Format 64-Bit-Ganzzahl ohne Vorzeichen vorliegen.

GetValue-Methode

Gibt den Wert der angegebenen Spalte als Objekt zurück.

Überladungsliste

Name	Beschreibung
GetValue(Int)-Methode	Gibt den Wert der angegebenen Spalte als Objekt zurück.
GetValue(Int, Long, Int)-Methode	Gibt eine Teilzeichenkette mit dem Wert der angegebenen Spalte als Objekt zurück.

GetValue(Int)-Methode

Gibt den Wert der angegebenen Spalte als Objekt zurück.

Visual Basic-Syntax

```
Public Overrides Function GetValue(ByVal ordinal As Integer) As Object
```

C#-Syntax

```
public override object GetValue(int ordinal)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

Rückgabe

Der Wert der angegebenen Spalte als Objekt.

Bemerkungen

Diese Methode gibt DBNull für Datenbankspalten mit NULL zurück.

GetValue(Int, Long, Int)-Methode

Gibt eine Teilzeichenkette mit dem Wert der angegebenen Spalte als Objekt zurück.

Visual Basic-Syntax

```
Public Function GetValue(  
    ByVal ordinal As Integer,  
    ByVal index As Long,  
    ByVal length As Integer  
) As Object
```

C#-Syntax

```
public object GetValue(int ordinal, long index, int length)
```

Parameter

- **ordinal** Eine Ordinalzahl, die die Spalte angibt, aus der der Wert bezogen wurde. Die Nummerierung erfolgt auf Nullbasis.

- **index** Der auf Null basierende Index der Teilzeichenkette des abzurufenden Werts.
- **length** Die Länge der Teilzeichenkette des abzurufenden Werts.

Rückgabe

Der Wert der Teilzeichenkette wird als Objekt zurückgegeben.

Bemerkungen

Diese Methode gibt DBNull für Datenbankspalten mit NULL zurück.

GetValues-Methode

Ruft alle Spalten in der aktuellen Zeile ab.

Visual Basic-Syntax

```
Public Overrides Function GetValues(ByVal values As Object()) As Integer
```

C#-Syntax

```
public override int GetValues(object[] values)
```

Parameter

- **values** Ein Array von Objekten, das eine komplette Zeile der Ergebnismenge enthält.

Rückgabe

Die Anzahl von Objekten im Array.

Bemerkungen

Für die meisten Anwendungen bietet die GetValues-Methode eine effiziente Möglichkeit für den Abruf aller Spalten anstelle des Abrufs jeweils einzelner Spalten.

Sie können ein Objekt-Array übergeben, das weniger Spalten enthält als in der Ergebniszeile vorhanden sind. Nur die Menge von Daten, die im Objekt-Array enthalten sind, wird in das Array kopiert. Sie können auch ein Objekt-Array übergeben, dessen Länge geringer ist als die Anzahl von Spalten in der Ergebniszeile.

Diese Methode gibt DBNull für Datenbankspalten mit NULL zurück.

IsDBNull-Methode

Gibt einen Wert zurück, der anzeigt, ob die Spalte NULL enthält.

Visual Basic-Syntax

```
Public Overrides Function IsDBNull(ByVal ordinal As Integer) As Boolean
```

C#-Syntax

```
public override bool IsDBNull(int ordinal)
```

Parameter

- **ordinal** Die auf Null basierende Spalten-Ordinalzahl.

Rückgabe

Gibt TRUE zurück, wenn der angegebene Spaltenwert gleich DBNull ist. Sonst FALSE.

Bemerkungen

Rufen Sie diese Methode auf, um das Vorhandensein von Spalten mit NULL zu prüfen, bevor die formatspezifischen get-Methoden aufgerufen werden (z.B. GetByte, GetChar usw.), sodass eventuelle Ausnahmebedingungen vermieden werden.

myDispose-Methode

Gibt die Ressourcen frei, die mit dem Objekt verbunden sind.

Visual Basic-Syntax

```
Public Sub myDispose()
```

C#-Syntax

```
public void myDispose()
```

NextResult-Methode

Verschiebt das SqlDataReader-Objekt zur nächsten Ergebnismenge, wenn Abfragen verarbeitet werden, die mehrere Ergebnismengen zurückgeben.

Visual Basic-Syntax

```
Public Overrides Function NextResult() As Boolean
```

C#-Syntax

```
public override bool NextResult()
```

Rückgabe

Gibt TRUE zurück, wenn mehrere Ergebnismengen vorhanden sind. Sonst FALSE.

Bemerkungen

Wird verwendet, um mehrere Ergebnismengen zu verarbeiten, die durch die Batch-Ausführung von SQL-Anweisungen oder gespeicherten Prozeduren generiert werden können.

Standardmäßig wird das Objekt auf der ersten Ergebnismenge positioniert.

Read-Methode

Liest die nächste Zeile der Ergebnismenge ein und setzt das SqlDataReader-Objekt auf diese Zeile.

Visual Basic-Syntax

```
Public Overrides Function Read() As Boolean
```

C#-Syntax

```
public override bool Read()
```

Rückgabe

Gibt TRUE zurück, wenn mehrere Zeilen vorhanden sind. Sonst FALSE.

Bemerkungen

Die Standardposition des SADataReader-Objekts ist unmittelbar vor dem ersten Datensatz. Daher müssen Sie Read aufrufen, um mit dem Zugriff auf Daten zu beginnen.

Beispiel

Der nachstehende Programmcode füllt ein Listefeld mit den Werten in einer einzelnen Ergebnisspalte.

```
while( reader.Read() ) {  
    listResults.Items.Add( reader.GetValue( 0 ).ToString() );  
}  
listResults.EndUpdate();  
reader.Close();
```

Depth-Eigenschaft

Ruft einen Wert ab, der die Tiefe der Verschachtelung für die aktuelle Zeile anzeigt.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Depth As Integer
```

C#-Syntax

```
public override int Depth {get;}
```

Bemerkungen

Die äußerste Tabelle hat die Tiefe Null.

Die Tiefe der Verschachtelung für die aktuelle Zeile.

FieldCount-Eigenschaft

Ruft die Anzahl der Spalten in der Ergebnismenge ab

Visual Basic-Syntax

```
Public ReadOnly Overrides Property FieldCount As Integer
```

C#-Syntax

```
public override int FieldCount {get;}
```

Bemerkungen

Die Anzahl der Spalten im aktuellen Datensatz

HasRows-Eigenschaft

Ruft einen Wert ab, der angibt, ob SDataReader mindestens eine Zeile enthält.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property HasRows As Boolean
```

C#-Syntax

```
public override bool HasRows {get;}
```

Bemerkungen

TRUE, wenn SDataReader mindestens eine Zeile enthält, andernfalls FALSE.

IsClosed-Eigenschaft

Ruft einen Wert ab, der angibt, ob das SDataReader-Objekt geschlossen ist.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property IsClosed As Boolean
```

C#-Syntax

```
public override bool IsClosed {get;}
```

Bemerkungen

TRUE, wenn das SDataReader-Objekt geschlossen ist, andernfalls FALSE.

IsClosed und RecordsAffected sind die einzigen Eigenschaften, die Sie aufrufen können, nachdem das SDataReader-Objekt geschlossen wurde.

RecordsAffected-Eigenschaft

Die Anzahl der durch eine SQL-Anweisung geänderten, eingefügten oder gelöschten Zeilen.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property RecordsAffected As Integer
```

C#-Syntax

```
public override int RecordsAffected {get;}
```

Bemerkungen

Die Anzahl der geänderten, eingefügten oder gelöschten Zeilen. Dieser Wert ist 0, wenn keine Zeilen betroffen sind bzw. die Anweisung fehlgeschlagen ist, oder -1 bei Select-Anweisungen.

Die Anzahl der geänderten, eingefügten oder gelöschten Zeilen. Dieser Wert ist 0, wenn keine Zeilen betroffen sind bzw. die Anweisung fehlgeschlagen ist, oder -1 bei Select-Anweisungen.

Der Wert dieser Eigenschaft ist kumulativ. Wenn beispielsweise zwei Datensätze im Batchmodus eingefügt werden, ist der Wert von RecordsAffected gleich 2.

IsClosed und RecordsAffected sind die einzigen Eigenschaften, die Sie aufrufen können, nachdem das SADATAReader-Objekt geschlossen wurde.

this-Eigenschaft

Gibt den Wert einer Spalte in ihrem nativen Format zurück.

Überladungsliste

Name	Beschreibung
this[Int]-Eigenschaft	Gibt den Wert einer Spalte in ihrem nativen Format zurück.
this[String]-Eigenschaft	Gibt den Wert einer Spalte in ihrem nativen Format zurück.

this[Int]-Eigenschaft

Gibt den Wert einer Spalte in ihrem nativen Format zurück.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Item(  
    ByVal index As Integer  
) As Object
```

C#-Syntax

```
public override object this[int index] {get;}
```

Parameter

- **index** Die Spalten-Ordinalzahl

Bemerkungen

In C# ist diese Eigenschaft der Indexersteller für die Klasse SADATAReader.

this[String]-Eigenschaft

Gibt den Wert einer Spalte in ihrem nativen Format zurück.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Item(ByVal name As String) As Object
```

C#-Syntax

```
public override object this[string name] {get;}
```

Parameter

- **name** Der Spaltenname.

Bemerkungen

In C# ist diese Eigenschaft der Indexersteller für die Klasse `SADataReader`.

SADataSourceEnumerator-Klasse

Stellt einen Mechanismus zur Enumeration aller verfügbaren Instanzen von SQL Anywhere-Datenbankservern innerhalb des lokalen Netzwerks bereit.

Visual Basic-Syntax

```
Public NotInheritable Class SADataSourceEnumerator  
    Inherits System.Data.Common.DbDataSourceEnumerator
```

C#-Syntax

```
public sealed class SADataSourceEnumerator :  
    System.Data.Common.DbDataSourceEnumerator
```

Basisklassen

- [System.Data.Common.DbDataSourceEnumerator](#)

Mitglieder

Alle Mitglieder der `SADataSourceEnumerator`-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
GetDataSources-Methode	Ruft ein <code>DataTable</code> -Objekt mit Informationen über alle sichtbaren SQL Anywhere-Datenbankserver ab.
Instance-Eigenschaft	Ruft eine Instanz des <code>SADataSourceEnumerator</code> -Objekts ab, die verwendet werden kann, um Informationen über alle sichtbaren SQL Anywhere-Datenbankserver abzufragen.

Bemerkungen

Es gibt keinen Konstruktor für `SADataSourceEnumerator`.

Die `SADataSourceEnumerator`-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

GetDataSources-Methode

Ruft ein DataTable-Objekt mit Informationen über alle sichtbaren SQL Anywhere-Datenbankserver ab.

Visual Basic-Syntax

```
Public Overrides Function GetDataSources() As DataTable
```

C#-Syntax

```
public override DataTable GetDataSources()
```

Bemerkungen

Die zurückgegebene Tabelle enthält vier Spalten: ServerName, IPAddress, PortNumber und DataBaseNames. Die Tabelle enthält für jeden verfügbaren Datenbankserver eine Zeile.

Beispiel

Der folgende Code füllt ein DataTable-Objekt mit Informationen über jeden verfügbaren Datenbankserver.

```
DataTable servers = SDataSourceEnumerator.Instance.GetDataSources();
```

Instance-Eigenschaft

Ruft eine Instanz des SDataSourceEnumerator-Objekts ab, die verwendet werden kann, um Informationen über alle sichtbaren SQL Anywhere-Datenbankserver abzufragen.

Visual Basic-Syntax

```
Public Shared ReadOnly Property Instance As SDataSourceEnumerator
```

C#-Syntax

```
public SDataSourceEnumerator Instance {get;}
```

SADefault-Klasse

Repräsentiert einen Parameter mit einem Standardwert.

Visual Basic-Syntax

```
Public NotInheritable Class SADefault
```

C#-Syntax

```
public sealed class SADefault
```

Mitglieder

Alle Mitglieder der SADefault-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Value-Feld	Ruft den Wert für einen Standardparameter ab.

Bemerkungen

Es gibt keinen Konstruktor für SADefault.

```
SAPparameter parm = new SAPparameter();  
parm.Value = SADefault.Value;
```

Value-Feld

Ruft den Wert für einen Standardparameter ab.

Visual Basic-Syntax

```
Public Shared ReadOnly Value As SADefault
```

C#-Syntax

```
public static readonly SADefault Value;
```

Bemerkungen

Dieses Feld ist schreibgeschützt und statisch.

SAError-Klasse

Sammelt Informationen, die für eine von der Datenquelle zurückgegebene Warn- oder Fehlermeldung von Bedeutung sind.

Visual Basic-Syntax

```
Public NotInheritable Class SAError
```

C#-Syntax

```
public sealed class SAError
```

Mitglieder

Alle Mitglieder der SAError-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
ToString-Methode	Der vollständige Text der Fehlermeldung.
Message-Eigenschaft	Gibt eine Kurzbeschreibung des Fehlers zurück.
NativeError-Eigenschaft	Gibt datenbankspezifische Fehlerinformationen zurück.

Name	Beschreibung
Source-Eigenschaft	Gibt den Namen des Providers zurück, der den Fehler generiert hat.
SqlState-Eigenschaft	Der fünfstellige SQLSTATE-Wert von SQL Anywhere gemäß dem ANSI SQL-Standard.

Bemerkungen

Es gibt keinen Konstruktor für `SAError`.

Hinweise zur Fehlerbehandlung finden Sie unter „[Fehlerbehandlung](#)“ auf Seite 66.

ToString-Methode

Der vollständige Text der Fehlermeldung.

Visual Basic-Syntax

```
Public Overrides Function ToString() As String
```

C#-Syntax

```
public override string ToString()
```

Beispiel

Der Rückgabewert ist eine Zeichenfolge in der Form **SAError:**, gefolgt von der Meldung. Beispiel:

```
SAError:UserId or Password not valid.
```

Message-Eigenschaft

Gibt eine Kurzbeschreibung des Fehlers zurück.

Visual Basic-Syntax

```
Public ReadOnly Property Message As String
```

C#-Syntax

```
public string Message {get;}
```

NativeError-Eigenschaft

Gibt datenbankspezifische Fehlerinformationen zurück.

Visual Basic-Syntax

```
Public ReadOnly Property NativeError As Integer
```

C#-Syntax

```
public int NativeError {get;}
```

Source-Eigenschaft

Gibt den Namen des Providers zurück, der den Fehler generiert hat.

Visual Basic-Syntax

```
Public ReadOnly Property Source As String
```

C#-Syntax

```
public string Source {get;}
```

SqlState-Eigenschaft

Der fünfstellige SQLSTATE-Wert von SQL Anywhere gemäß dem ANSI SQL-Standard.

Visual Basic-Syntax

```
Public ReadOnly Property SqlState As String
```

C#-Syntax

```
public string SqlState {get;}
```

SAErrorCollection-Klasse

Sammelt alle Fehler, die vom SQL Anywhere .NET-Datenprovider generiert werden.

Visual Basic-Syntax

```
Public NotInheritable Class SAErrorCollection  
    Implements System.Collections.ICollection  
    Implements System.Collections.IEnumerable
```

C#-Syntax

```
public sealed class SAErrorCollection :  
    System.Collections.ICollection,  
    System.Collections.IEnumerable
```

Basisklassen

- [System.Collections.ICollection](#)
- [System.Collections.IEnumerable](#)

Mitglieder

Alle Mitglieder der SAErrorCollection-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
CopyTo-Methode	Kopiert die Elemente der Klasse SAErrorCollection in ein Array, wobei am angegebenen Index im Array begonnen wird.
GetEnumerator-Methode	Gibt einen Enumerator zurück, der die Klasse SAErrorCollection durchläuft.
Count-Eigenschaft	Gibt die Anzahl der Fehler in der Sammlung zurück.
this-Eigenschaft	Gibt den Fehler an der Stelle des angegebenen Indexes zurück.

Bemerkungen

Es gibt keinen Konstruktor für SAErrorCollection. In der Regel wird eine SAErrorCollection-Klasse durch die SAException.Errors-Eigenschaft gebildet.

Implements: ICollection, IEnumerable

Hinweise zur Fehlerbehandlung finden Sie unter „[Fehlerbehandlung](#)“ auf Seite 66.

Siehe auch

- [SAException.Errors-Eigenschaft \[SQL Anywhere .NET\] auf Seite 258](#)

CopyTo-Methode

Kopiert die Elemente der Klasse SAErrorCollection in ein Array, wobei am angegebenen Index im Array begonnen wird.

Visual Basic-Syntax

```
Public Sub CopyTo(ByVal array As Array, ByVal index As Integer)
```

C#-Syntax

```
public void CopyTo(Array array, int index)
```

Parameter

- **array** Das Array, in das die Elemente kopiert werden sollen.
- **index** Der Startindex des Arrays.

GetEnumerator-Methode

Gibt einen Enumerator zurück, der die Klasse SAErrorCollection durchläuft.

Visual Basic-Syntax

```
Public Function GetEnumerator() As System.Collections.IEnumerator
```

C#-Syntax

```
public System.Collections.IEnumerator GetEnumerator()
```

Rückgabe

Ein System.Collections.IEnumerator für das SAErrorCollection-Objekt.

Count-Eigenschaft

Gibt die Anzahl der Fehler in der Sammlung zurück.

Visual Basic-Syntax

```
Public ReadOnly Property Count As Integer
```

C#-Syntax

```
public int Count {get;}
```

this-Eigenschaft

Gibt den Fehler an der Stelle des angegebenen Indexes zurück.

Visual Basic-Syntax

```
Public ReadOnly Property Item(ByVal index As Integer) As SAError
```

C#-Syntax

```
public SAError this[int index] {get;}
```

Parameter

- **index** Der auf Null basierende Index des abzurufenden Fehlers

Bemerkungen

Ein SAError-Objekt, das den Fehler an der Stelle des angegebenen Indexes enthält.

Siehe auch

- [SAError-Klasse \[SQL Anywhere .NET\] auf Seite 252](#)

SAException-Klasse

Die Ausnahmebedingung, die generiert wird, wenn SQL Anywhere eine Warnung oder einen Fehler zurückgibt.

Visual Basic-Syntax

```
Public Class SAException Inherits System.Exception
```

C#-Syntax

```
public class SAException : System.Exception
```

Basisklassen

- [System.Exception](#)

Mitglieder

Alle Mitglieder der SAException-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
GetBaseException-Methode (geerbt aus System.Exception)	Gibt, falls in einer abgeleiteten Klasse aufgehoben, den System.Exception -Wert zurück, der die Ursache für eine oder mehrere nachfolgende Ausnahmebedingungen ist.
GetObjectData-Methode	Legt das SerializationInfo-Objekt mit Informationen über die Ausnahmebedingung fest.
GetType-Methode (geerbt aus System.Exception)	Ruft den Laufzeit-Typ der aktuellen Instanz ab.
ToString-Methode (geerbt aus System.Exception)	Erstellt eine Zeichenfolgedarstellung der aktuellen Ausnahmebedingung und gibt die Zeichenfolgedarstellung zurück.
Data-Eigenschaft (geerbt aus System.Exception)	Ruft eine Sammlung von Schlüssel-/Wertpaaren ab, die zusätzliche benutzerdefinierte Informationen über die Ausnahmebedingung enthalten.
Errors-Eigenschaft	Gibt eine Sammlung mit mindestens einem SAError-Objekt zurück.
HelpLink-Eigenschaft (geerbt aus System.Exception)	Ruft eine Verknüpfung zur Hilfedatei ab, die dieser Ausnahmebedingung zugeordnet ist, oder legt die Verknüpfung fest.
HResult-Eigenschaft (geerbt aus System.Exception)	Ruft einen HRESULT-Wert ab oder legt ihn fest, einen kodierten numerischen Wert, der einer bestimmten Ausnahmebedingung zugewiesen ist.
InnerException-Eigenschaft (geerbt aus System.Exception)	Ruft die System.Exception -Instanz ab, die die aktuelle Ausnahmebedingung bewirkt hat.
Message-Eigenschaft	Gibt den Text zurück, der den Fehler beschreibt.
NativeError-Eigenschaft	Gibt datenbankspezifische Fehlerinformationen zurück.
Source-Eigenschaft	Gibt den Namen des Providers zurück, der den Fehler generiert hat.
StackTrace-Eigenschaft (geerbt aus System.Exception)	Ruft eine Zeichenfolgendarstellung der unmittelbaren Frames im Aufruf-Stack ab.

Name	Beschreibung
TargetSite-Eigenschaft (geerbt aus System.Exception)	Ruft die Methode ab, die die aktuelle Ausnahmebedingung ausgelöst hat.

Bemerkungen

Es gibt keinen Konstruktor für SAException. In der Regel wird ein SAException-Objekt in einem Catch deklariert. Beispiel:

```
...  
catch( SAException ex )  
{  
    MessageBox.Show( ex.Errors[0].Message, "Error" );  
}
```

Hinweise zur Fehlerbehandlung finden Sie unter [„Fehlerbehandlung“ auf Seite 66](#).

GetObjectData-Methode

Legt das SerializationInfo-Objekt mit Informationen über die Ausnahmebedingung fest.

Visual Basic-Syntax

```
Public Overrides Sub GetObjectData(  
    ByVal info As SerializationInfo,  
    ByVal context As StreamingContext  
)
```

C#-Syntax

```
public override void GetObjectData(  
    SerializationInfo info,  
    StreamingContext context  
)
```

Parameter

- **info** Die SerializationInfo, in der sich die serialisierten Objektdaten über die generierte Ausnahmebedingung befinden.
- **context** Der StreamingContext, der Kontextinformationen über die Quelle oder das Ziel enthält.

Bemerkungen

Setzt Exception.GetObjectData außer Kraft.

Errors-Eigenschaft

Gibt eine Sammlung mit mindestens einem SAError-Objekt zurück.

Visual Basic-Syntax

```
Public ReadOnly Property Errors As SAErrorCollection
```

C#-Syntax

```
public SAErrorCollection Errors {get;}
```

Bemerkungen

Das SAErrorCollection-Objekt enthält immer mindestens eine Instanz des SAError-Objekts.

Siehe auch

- [SAErrorCollection-Klasse \[SQL Anywhere .NET\] auf Seite 254](#)
- [SAError-Klasse \[SQL Anywhere .NET\] auf Seite 252](#)

Message-Eigenschaft

Gibt den Text zurück, der den Fehler beschreibt.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Message As String
```

C#-Syntax

```
public override string Message {get;}
```

Bemerkungen

Diese Methode gibt eine einzelne Zeichenfolge zurück, die eine Verkettung aller Message-Eigenschaften aller SAError-Objekte in der Errors-Sammlung enthält. Auf jede Nachricht, ausgenommen die letzte, folgt eine Zeilenschaltung.

Siehe auch

- [SAError-Klasse \[SQL Anywhere .NET\] auf Seite 252](#)

NativeError-Eigenschaft

Gibt datenbankspezifische Fehlerinformationen zurück.

Visual Basic-Syntax

```
Public ReadOnly Property NativeError As Integer
```

C#-Syntax

```
public int NativeError {get;}
```

Source-Eigenschaft

Gibt den Namen des Providers zurück, der den Fehler generiert hat.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Source As String
```

C#-Syntax

```
public override string Source {get;}
```

SAFactory-Klasse

Repräsentiert eine Gruppe von Methoden zum Erstellen von Instanzen der Implementierung der Datenquellenklassen des iAnywhere.Data.SQLAnywhere-Providers.

Visual Basic-Syntax

```
Public NotInheritable Class SAFactory  
    Inherits System.Data.Common.DbProviderFactory
```

C#-Syntax

```
public sealed class SAFactory : System.Data.Common.DbProviderFactory
```

Basisklassen

- [System.Data.Common.DbProviderFactory](#)

Mitglieder

Alle Mitglieder der SAFactory-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
CreateCommand-Methode	Gibt eine stark typisierte System.Data.Common.DbCommand-Instanz zurück
CreateCommandBuilder-Methode	Gibt eine stark typisierte System.Data.Common.DbCommandBuilder-Instanz zurück
CreateConnection-Methode	Gibt eine stark typisierte System.Data.Common.DbConnection-Instanz zurück
CreateConnectionStringBuilder-Methode	Gibt eine stark typisierte System.Data.Common.DbConnectionStringBuilder-Instanz zurück
CreateDataAdapter-Methode	Gibt eine stark typisierte System.Data.Common.DbDataAdapter-Instanz zurück
CreateDataSourceEnumerator-Methode	Gibt eine stark typisierte System.Data.Common.DbDataSourceEnumerator-Instanz zurück
CreateParameter-Methode	Gibt eine stark typisierte System.Data.Common.DbParameter-Instanz zurück
CreatePermission-Methode	Gibt eine stark typisierte CodeAccessPermission-Instanz zurück.

Name	Beschreibung
CanCreateDataSourceEnumerator-Eigenschaft	Gibt immer TRUE zurück, wodurch angezeigt wird, dass ein SADADataSourceEnumerator-Objekt erstellt werden kann.
Instance-Feld	Repräsentiert die einzige Instanz der Klasse SAFactory.

Bemerkungen

Es gibt keinen Konstruktor für SAFactory.

ADO.NET 2.0 führt die beiden neuen Klassen DbProviderFactories und DbProviderFactory ein, die das Schreiben von Provider-unabhängigem Code vereinfachen. Um sie mit SQL Anywhere zu verwenden, übergeben Sie iAnywhere.Data.SQLAnywhere als Provider-unveränderlichen Namen an GetFactory. Beispiel:

```
' Visual Basic
Dim factory As DbProviderFactory = _
    DbProviderFactories.GetFactory( "iAnywhere.Data.SQLAnywhere" )
Dim conn As DbConnection = _
    factory.CreateConnection()

// C#
DbProviderFactory factory =
    DbProviderFactories.GetFactory("iAnywhere.Data.SQLAnywhere" );
DbConnection conn = factory.CreateConnection();
```

In diesem Beispiel wird conn als ein SACConnection-Objekt erstellt.

Eine Beschreibung von Provider-Factories und der generischen Programmierung in ADO.NET 2.0 finden Sie unter [Generic Coding with the ADO.NET 2.0 Base Classes and Factories](#).

Die Klasse SAFactory ist in .NET Compact Framework 2.0 nicht verfügbar.

CreateCommand-Methode

Gibt eine stark typisierte System.Data.Common.DbCommand-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateCommand() As DbCommand
```

C#-Syntax

```
public override DbCommand CreateCommand()
```

Rückgabe

Ein neues SACCommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SACCommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreateCommandBuilder-Methode

Gibt eine stark typisierte System.Data.Common.DbCommandBuilder-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateCommandBuilder() As DbCommandBuilder
```

C#-Syntax

```
public override DbCommandBuilder CreateCommandBuilder()
```

Rückgabe

Ein neues SAcCommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreateConnection-Methode

Gibt eine stark typisierte System.Data.Common.DbConnection-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateConnection() As DbConnection
```

C#-Syntax

```
public override DbConnection CreateConnection()
```

Rückgabe

Ein neues SAcCommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreateConnectionStringBuilder-Methode

Gibt eine stark typisierte System.Data.Common.DbConnectionStringBuilder-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateConnectionStringBuilder()  
    As DbConnectionStringBuilder
```

C#-Syntax

```
public override DbConnectionStringBuilder CreateConnectionStringBuilder()
```

Rückgabe

Ein neues SCommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SCommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreateDataAdapter-Methode

Gibt eine stark typisierte System.Data.Common.DbDataAdapter-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateDataAdapter() As DbDataAdapter
```

C#-Syntax

```
public override DbDataAdapter CreateDataAdapter()
```

Rückgabe

Ein neues SCommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SCommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreateDataSourceEnumerator-Methode

Gibt eine stark typisierte System.Data.Common.DbDataSourceEnumerator-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateDataSourceEnumerator()  
    As DbDataSourceEnumerator
```

C#-Syntax

```
public override DbDataSourceEnumerator CreateDataSourceEnumerator()
```

Rückgabe

Ein neues SCommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SCommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreateParameter-Methode

Gibt eine stark typisierte System.Data.Common.DbParameter-Instanz zurück

Visual Basic-Syntax

```
Public Overrides Function CreateParameter() As DbParameter
```

C#-Syntax

```
public override DbParameter CreateParameter()
```

Rückgabe

Ein neues SACommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CreatePermission-Methode

Gibt eine stark typisierte CodeAccessPermission-Instanz zurück.

Visual Basic-Syntax

```
Public Overrides Function CreatePermission(  
    ByVal state As PermissionState  
) As CodeAccessPermission
```

C#-Syntax

```
public override CodeAccessPermission CreatePermission(  
    PermissionState state  
)
```

Parameter

- **state** Ein Mitglied der Enumeration System.Security.Permissions.PermissionState.

Rückgabe

Ein neues SACommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

CanCreateDataSourceEnumerator-Eigenschaft

Gibt immer TRUE zurück, wodurch angezeigt wird, dass ein SADATASourceEnumerator-Objekt erstellt werden kann.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property CanCreateDataSourceEnumerator As  
Boolean
```

C#-Syntax

```
public override bool CanCreateDataSourceEnumerator {get;}
```

Rückgabe

Ein neues SACommand-Objekt mit dem DbCommand-Datentyp.

Siehe auch

- [SADataSourceEnumerator-Klasse \[SQL Anywhere .NET\] auf Seite 250](#)
- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)

Instance-Feld

Repräsentiert die einzige Instanz der Klasse SAFactory.

Visual Basic-Syntax

```
Public Shared ReadOnly Instance As SAFactory
```

C#-Syntax

```
public static readonly SAFactory Instance;
```

Bemerkungen

SAFactory ist eine einzelne Klasse. Das heißt nur, dass diese Instanz dieser Klasse vorhanden sein kann.

Normalerweise wird dieses Feld nicht direkt verwendet. Stattdessen erhalten Sie eine Referenz dieser SAFactory-Instanz mithilfe von System.Data.Common.DbProviderFactories.GetFactory(String). Ein Beispiel finden Sie in der Beschreibung der Klasse SAFactory.

Die Klasse SAFactory ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SAFactory-Klasse \[SQL Anywhere .NET\] auf Seite 260](#)

SAInfoMessageEventArgs-Klasse

Stellt Daten für das InfoMessage-Ereignis bereit.

Visual Basic-Syntax

```
Public NotInheritable Class SAInfoMessageEventArgs  
    Inherits System.EventArgs
```

C#-Syntax

```
public sealed class SAInfoMessageEventArgs : System.EventArgs
```

Basisklassen

- [System.EventArgs](#)

Mitglieder

Alle Mitglieder der SAInfoMessageEventArgs-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
ToString-Methode	Ruft eine Darstellung des InfoMessage-Ereignisses als Zeichenfolge ab.
Errors-Eigenschaft	Gibt die Sammlung von Meldungen zurück, die von der Datenquelle gesendet wurden.
Message-Eigenschaft	Gibt den vollständigen Text der Fehlermeldung zurück, die von der Datenquelle gesendet wurde.
MessageType-Eigenschaft	Gibt den Typ der Meldung zurück.
NativeError-Eigenschaft	Gibt den von der Datenbank zurückgegebenen SQLCODE-Wert zurück
Source-Eigenschaft	Gibt den Namen des SQL Anywhere .NET-Datenproviders zurück.
Empty-Feld (geerbt aus System.EventArgs)	Repräsentiert ein Ereignis ohne Ereignisdaten.

Bemerkungen

Es gibt keinen Konstruktor für SAInfoMessageEventArgs.

ToString-Methode

Ruft eine Darstellung des InfoMessage-Ereignisses als Zeichenfolge ab.

Visual Basic-Syntax

```
Public Overrides Function ToString() As String
```

C#-Syntax

```
public override string ToString()
```

Rückgabe

Eine Zeichenfolge, die das InfoMessage-Ereignis darstellt.

Errors-Eigenschaft

Gibt die Sammlung von Meldungen zurück, die von der Datenquelle gesendet wurden.

Visual Basic-Syntax

```
Public ReadOnly Property Errors As SAErrorCollection
```

C#-Syntax

```
public SAErrorCollection Errors {get;}
```

Message-Eigenschaft

Gibt den vollständigen Text der Fehlermeldung zurück, die von der Datenquelle gesendet wurde.

Visual Basic-Syntax

```
Public ReadOnly Property Message As String
```

C#-Syntax

```
public string Message {get;}
```

MessageType-Eigenschaft

Gibt den Typ der Meldung zurück.

Visual Basic-Syntax

```
Public ReadOnly Property MessageType As SAMessageType
```

C#-Syntax

```
public SAMessageType MessageType {get;}
```

Bemerkungen

Dies kann einer der folgenden sein: Aktion, Info, Status oder Warnung.

NativeError-Eigenschaft

Gibt den von der Datenbank zurückgegebenen SQLCODE-Wert zurück

Visual Basic-Syntax

```
Public ReadOnly Property NativeError As Integer
```

C#-Syntax

```
public int NativeError {get;}
```

Source-Eigenschaft

Gibt den Namen des SQL Anywhere .NET-Datenproviders zurück.

Visual Basic-Syntax

```
Public ReadOnly Property Source As String
```

C#-Syntax

```
public string Source {get;}
```

SAMetaDataCollectionNames-Klasse

Stellt eine Liste von Konstanten für die `SACConnection.GetSchema(string)`-Methode bereit, um Metadaten-Sammlungen abzurufen.

Visual Basic-Syntax

```
Public NotInheritable Class SAMetaDataCollectionNames
```

C#-Syntax

```
public sealed class SAMetaDataCollectionNames
```

Mitglieder

Alle Mitglieder der `SAMetaDataCollectionNames`-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Columns-Feld	Stellt eine Konstante, welche die Sammlung <code>Columns</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
DataSourceInformation-Feld	Stellt eine Konstante, welche die Sammlung <code>DataSourceInformation</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
DataTypes-Feld	Stellt eine Konstante, welche die Sammlung <code>DataTypes</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
ForeignKeys-Feld	Stellt eine Konstante, welche die Sammlung <code>ForeignKeys</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
IndexColumns-Feld	Stellt eine Konstante, welche die Sammlung <code>IndexColumns</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
Indexes-Feld	Stellt eine Konstante, welche die Sammlung <code>Indexes</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
MetaDataCollections-Feld	Stellt eine Konstante, welche die Sammlung <code>MetaDataCollections</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
ProcedureParameters-Feld	Stellt eine Konstante, welche die Sammlung <code>ProcedureParameters</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
Procedures-Feld	Stellt eine Konstante, welche die Sammlung <code>Procedures</code> darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.

Name	Beschreibung
ReservedWords-Feld	Stellt eine Konstante, welche die Sammlung ReservedWords darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
Restrictions-Feld	Stellt eine Konstante, welche die Sammlung Restrictions darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
Tables-Feld	Stellt eine Konstante, welche die Sammlung Tables darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
UserDefinedTypes-Feld	Stellt eine Konstante, welche die Sammlung UserDefinedTypes darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
Users-Feld	Stellt eine Konstante, welche die Sammlung Users darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
ViewColumns-Feld	Stellt eine Konstante, welche die Sammlung ViewColumns darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.
Views-Feld	Stellt eine Konstante, welche die Sammlung Views darstellt, für die Verwendung mit der <code>SACConnection.GetSchema(string)</code> -Methode bereit.

Bemerkungen

Dieses Feld ist konstant und schreibgeschützt.

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Columns-Feld

Stellt eine Konstante, welche die Sammlung Columns darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Columns As String
```

C#-Syntax

```
public static readonly string Columns;
```

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Columns-Sammlung

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Columns );
```

DataSourceInformation-Feld

Stellt eine Konstante, welche die Sammlung DataSourceInformation darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly DataSourceInformation As String
```

C#-Syntax

```
public static readonly string DataSourceInformation;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der DataSourceInformation-Sammlung.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.DataSourceInformation );
```

DataTypes-Feld

Stellt eine Konstante, welche die Sammlung DataTypes darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly DataTypes As String
```

C#-Syntax

```
public static readonly string DataTypes;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der DataTypes-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.DataTypes );
```

ForeignKeys-Feld

Stellt eine Konstante, welche die Sammlung ForeignKeys darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly ForeignKeys As String
```

C#-Syntax

```
public static readonly string ForeignKeys;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der ForeignKeys-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.ForeignKeys );
```

IndexColumns-Feld

Stellt eine Konstante, welche die Sammlung IndexColumns darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly IndexColumns As String
```

C#-Syntax

```
public static readonly string IndexColumns;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der IndexColumns-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.IndexColumns );
```

Indexes-Feld

Stellt eine Konstante, welche die Sammlung Indexes darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Indexes As String
```

C#-Syntax

```
public static readonly string Indexes;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Indexes-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Indexes );
```

MetaDataCollections-Feld

Stellt eine Konstante, welche die Sammlung MetaDataCollections darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly MetaDataCollections As String
```

C#-Syntax

```
public static readonly string MetaDataCollections;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der MetaDataCollections-Sammlung.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.MetaDataCollections );
```

ProcedureParameters-Feld

Stellt eine Konstante, welche die Sammlung ProcedureParameters darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly ProcedureParameters As String
```

C#-Syntax

```
public static readonly string ProcedureParameters;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der ProcedureParameters-Sammlung.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.ProcedureParameters );
```

Procedures-Feld

Stellt eine Konstante, welche die Sammlung Procedures darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Procedures As String
```

C#-Syntax

```
public static readonly string Procedures;
```

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Procedures-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Procedures );
```

ReservedWords-Feld

Stellt eine Konstante, welche die Sammlung ReservedWords darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly ReservedWords As String
```

C#-Syntax

```
public static readonly string ReservedWords;
```

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der ReservedWords-Sammlung

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.ReservedWords );
```

Restrictions-Feld

Stellt eine Konstante, welche die Sammlung Restrictions darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Restrictions As String
```

C#-Syntax

```
public static readonly string Restrictions;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Restrictions-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Restrictions );
```

Tables-Feld

Stellt eine Konstante, welche die Sammlung Tables darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Tables As String
```

C#-Syntax

```
public static readonly string Tables;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Tables-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Tables );
```

UserDefinedTypes-Feld

Stellt eine Konstante, welche die Sammlung UserDefinedTypes darstellt, für die Verwendung mit der SAConnection.GetSchema(string)-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly UserDefinedTypes As String
```

C#-Syntax

```
public static readonly string UserDefinedTypes;
```

Siehe auch

- [SAConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt ein DataTable-Objekt mit der UserDefinedTypes-Sammlung.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.UserDefinedTypes );
```

Users-Feld

Stellt eine Konstante, welche die Sammlung Users darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Users As String
```

C#-Syntax

```
public static readonly string Users;
```

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Users-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Users );
```

ViewColumns-Feld

Stellt eine Konstante, welche die Sammlung ViewColumns darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly ViewColumns As String
```

C#-Syntax

```
public static readonly string ViewColumns;
```

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der ViewColumns-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.ViewColumns );
```

Views-Feld

Stellt eine Konstante, welche die Sammlung Views darstellt, für die Verwendung mit der `SACConnection.GetSchema(string)`-Methode bereit.

Visual Basic-Syntax

```
Public Shared ReadOnly Views As String
```

C#-Syntax

```
public static readonly string Views;
```

Siehe auch

- [SACConnection.GetSchema-Methode \[SQL Anywhere .NET\] auf Seite 177](#)

Beispiel

Der folgende Code füllt eine DataTable mit der Views-Sammlung.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Views );
```

SAParameter-Klasse

Stellt einen Parameter für ein SACommand-Objekt dar und optional auch dessen Zuordnung zu einer DataSet-Spalte.

Visual Basic-Syntax

```
Public NotInheritable Class SAParameter  
    Inherits System.Data.Common.DbParameter  
    Implements System.ICloneable
```

C#-Syntax

```
public sealed class SAParameter :  
    System.Data.Common.DbParameter ,  
    System.ICloneable
```

Basisklassen

- [System.Data.Common.DbParameter](#)
- [System.ICloneable](#)

Mitglieder

Alle Mitglieder der SAParameter-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SAParameter-Konstruktor	Initialisiert ein SAParameter-Objekt mit NULL ("Nothing" in Visual Basic).
ResetDbType-Methode	Setzt den Typ (die Werte von DbType und SADbType) zurück, der diesem SAParameter-Objekt zugewiesen ist.
ToString-Methode	Gibt eine Zeichenfolge zurück, die den Parameternamen enthält.
DbType-Eigenschaft	Ruft den DbType-Wert des Parameters ab bzw. legt ihn fest.

Name	Beschreibung
Direction-Eigenschaft	Ruft einen Wert ab bzw. legt einen Wert fest, der anzeigt, ob der Parameter nur für die Eingabe, nur für die Ausgabe, bidirektional oder ein Rückgabewertparameter einer gespeicherten Prozedur ist.
IsNullable-Eigenschaft	Ruft einen Wert ab bzw. legt einen Wert fest, der anzeigt, ob der Parameter NULL akzeptiert.
Offset-Eigenschaft	Ruft den Offset der Value-Eigenschaft ab bzw. legt ihn fest.
ParameterName-Eigenschaft	Ruft den Namen des SAParameter-Objekts ab bzw. legt ihn fest.
Precision-Eigenschaft	Ruft die maximale Anzahl von Stellen ab, die für die Darstellung der Value-Eigenschaft benutzt werden, bzw. legt den betreffenden Wert fest.
SADbType-Eigenschaft	Der SADbType des Parameters.
Scale-Eigenschaft	Gibt die Anzahl der Dezimalstellen an, mit denen Value aufgelöst wird, bzw. ruft sie ab.
Size-Eigenschaft	Legt die in Byte angegebene Maximalgröße der Daten in der Spalte fest bzw. ruft sie ab.
SourceColumn-Eigenschaft	Ruft den Namen der Quellspalte ab, die dem DataSet-Objekt zugewiesen ist und für das Laden und die Rückgabe des Werts verwendet wird, bzw. legt den betreffenden Spaltennamen fest.
SourceColumnNull-Mapping-Eigenschaft	Legt einen Wert fest bzw. ruft einen Wert ab, der angibt, ob die Spalte nullwertfähig ist.
SourceVersion-Eigenschaft	Gibt den DataRowVersion-Wert an, der beim Laden von Value zu verwenden ist, bzw. ruft ihn ab.
Value-Eigenschaft	Ruft den Wert des Parameters ab bzw. legt ihn fest.

Bemerkungen

Implements: IDbDataParameter, IDataParameter, ICloneable

SAParameter-Konstruktor

Initialisiert ein SAParameter-Objekt mit NULL ("Nothing" in Visual Basic).

Überladungsliste

Name	Beschreibung
SAPparameter()-Konstruktor	Initialisiert ein SAPparameter-Objekt mit NULL ("Nothing" in Visual Basic).
SAPparameter(String, Object)-Konstruktor	Initialisiert ein SAPparameter-Objekt mit dem angegebenen Parameternamen und -wert.
SAPparameter(String, SADB-Type)-Konstruktor	Initialisiert ein SAPparameter-Objekt mit dem angegebenen Parameternamen und -datentyp.
SAPparameter(String, SADB-Type, Int)-Konstruktor	Initialisiert ein SAPparameter-Objekt mit dem angegebenen Parameternamen und -datentyp.
SAPparameter(String, SADB-Type, Int, ParameterDirection, Bool, Byte, Byte, String, DataRowVersion, Object)-Konstruktor	Initialisiert ein SAPparameter-Objekt mit den Angaben für Name, Datentyp, Länge, Richtung, Nullwertfähigkeit, numerische Gesamtstellenzahl, Anzahl der Dezimalstellen, Quellspalte und Wert des Parameters.
SAPparameter(String, SADB-Type, Int, String)-Konstruktor	Initialisiert ein SAPparameter-Objekt mit den angegebenen Werten für Name, Datentyp und Länge des Parameters.

SAPparameter()-Konstruktor

Initialisiert ein SAPparameter-Objekt mit NULL ("Nothing" in Visual Basic).

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SAPparameter()
```

SAPparameter(String, Object)-Konstruktor

Initialisiert ein SAPparameter-Objekt mit dem angegebenen Parameternamen und -wert.

Visual Basic-Syntax

```
Public Sub New(ByVal parameterName As String, ByVal value As Object)
```

C#-Syntax

```
public SAPparameter(string parameterName, object value)
```

Parameter

- **parameterName** Der Name des Parameters.

- **value** Ein Objekt, das der Wert des Parameters ist.

Bemerkungen

Dieser Konstruktor wird nicht empfohlen. Er wird aus Gründen der Kompatibilität mit anderen Datenprovidern bereitgestellt.

SAPParameter(String, SADBType)-Konstruktor

Initialisiert ein SAPParameter-Objekt mit dem angegebenen Parameternamen und -datentyp.

Visual Basic-Syntax

```
Public Sub New(ByVal parameterName As String, ByVal dbType As SADBType)
```

C#-Syntax

```
public SAPParameter(string parameterName, SADBType dbType)
```

Parameter

- **parameterName** Der Name des Parameters.
- **dbType** Einer der SADBType-Werte.

Siehe auch

- [SAPParameter.SADBType-Eigenschaft \[SQL Anywhere .NET\] auf Seite 284](#)

SAPParameter(String, SADBType, Int)-Konstruktor

Initialisiert ein SAPParameter-Objekt mit dem angegebenen Parameternamen und -datentyp.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADBType,  
    ByVal size As Integer  
)
```

C#-Syntax

```
public SAPParameter(string parameterName, SADBType dbType, int size)
```

Parameter

- **parameterName** Der Name des Parameters.
- **dbType** Einer der SADBType-Werte.
- **size** Die Länge des Parameters.

SAPParameter(String, SADbType, Int, ParameterDirection, Bool, Byte, Byte, String, DataRowVersion, Object)-Konstruktor

Initialisiert ein SAPParameter-Objekt mit den Angaben für Name, Datentyp, Länge, Richtung, Nullwertfähigkeit, numerische Gesamtstellenzahl, Anzahl der Dezimalstellen, Quellspalte und Wert des Parameters.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADbType,  
    ByVal size As Integer,  
    ByVal direction As ParameterDirection,  
    ByVal isNullable As Boolean,  
    ByVal precision As Byte,  
    ByVal scale As Byte,  
    ByVal sourceColumn As String,  
    ByVal sourceVersion As DataRowVersion,  
    ByVal value As Object  
)
```

C#-Syntax

```
public SAPParameter(  
    string parameterName,  
    SADbType dbType,  
    int size,  
    ParameterDirection direction,  
    bool isNullable,  
    byte precision,  
    byte scale,  
    string sourceColumn,  
    DataRowVersion sourceVersion,  
    object value  
)
```

Parameter

- **parameterName** Der Name des Parameters.
- **dbType** Einer der SADbType-Werte.
- **size** Die Länge des Parameters.
- **direction** Einer der ParameterDirection-Werte.
- **isNullable** TRUE, wenn der Wert des Feldes NULL enthalten darf, sonst FALSE
- **precision** Die Gesamtanzahl der Stellen links und rechts vom Dezimalzeichen, die für die Auflösung von "Value" verwendet wird
- **scale** Die Gesamtanzahl der Dezimalstellen, mit denen "Value" aufgelöst wird
- **sourceColumn** Der Name der Quellspalte, die zuzuordnen ist.

- **sourceVersion** Einer der DataRowVersion-Werte.
- **value** Ein Objekt, das der Wert des Parameters ist.

SAParameter(String, SADBType, Int, String)-Konstruktor

Initialisiert ein SAParameter-Objekt mit den angegebenen Werten für Name, Datentyp und Länge des Parameters.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADBType,  
    ByVal size As Integer,  
    ByVal sourceColumn As String  
)
```

C#-Syntax

```
public SAParameter(  
    string parameterName,  
    SADBType dbType,  
    int size,  
    string sourceColumn  
)
```

Parameter

- **parameterName** Der Name des Parameters.
- **dbType** Einer der SADBType-Werte.
- **size** Die Länge des Parameters.
- **sourceColumn** Der Name der Quellspalte, die zuzuordnen ist.

ResetDbType-Methode

Setzt den Typ (die Werte von DbType und SADBType) zurück, der diesem SAParameter-Objekt zugewiesen ist.

Visual Basic-Syntax

```
Public Overrides Sub ResetDbType()
```

C#-Syntax

```
public override void ResetDbType()
```

ToString-Methode

Gibt eine Zeichenfolge zurück, die den Parameternamen enthält.

Visual Basic-Syntax

```
Public Overrides Function ToString() As String
```

C#-Syntax

```
public override string ToString()
```

Rückgabe

Der Name des Parameters.

DbType-Eigenschaft

Ruft den DbType-Wert des Parameters ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Overrides Property DbType As DbType
```

C#-Syntax

```
public override DbType DbType {get;set;}
```

Bemerkungen

SADbType und DbType sind miteinander verknüpft. Die Definition von DbType ändert daher SADbType auf einen unterstützenden SADbType.

Der Wert muss ein Mitglied des SADbType-Enumerators sein.

Direction-Eigenschaft

Ruft einen Wert ab bzw. legt einen Wert fest, der anzeigt, ob der Parameter nur für die Eingabe, nur für die Ausgabe, bidirektional oder ein Rückgabewertparameter einer gespeicherten Prozedur ist.

Visual Basic-Syntax

```
Public Overrides Property Direction As ParameterDirection
```

C#-Syntax

```
public override ParameterDirection Direction {get;set;}
```

Bemerkungen

Einer der ParameterDirection-Werte.

Wenn die durch ParameterDirection vorgegebene Richtung "output" ist und die Ausführung des zugehörigen SACommand-Objekts keinen Wert zurückgibt, enthält der SAParameter NULL. Nachdem die letzte Zeile aus der letzten Ergebnismenge gelesen wurde, werden die Parameter Output, InputOut und ReturnValue aktualisiert.

IsNullable-Eigenschaft

Ruft einen Wert ab bzw. legt einen Wert fest, der anzeigt, ob der Parameter NULL akzeptiert.

Visual Basic-Syntax

```
Public Overrides Property IsNullable As Boolean
```

C#-Syntax

```
public override bool IsNullable {get;set;}
```

Bemerkungen

Diese Eigenschaft ist TRUE, wenn NULL akzeptiert wird, sonst FALSE. Der Standardwert ist FALSE. NULL wird unter Verwendung der DBNull-Klasse verarbeitet.

Offset-Eigenschaft

Ruft den Offset der Value-Eigenschaft ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Property Offset As Integer
```

C#-Syntax

```
public int Offset {get;set;}
```

Bemerkungen

Der Offset des Werts. Standardwert ist "0".

ParameterName-Eigenschaft

Ruft den Namen des SAParameter-Objekts ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Overrides Property ParameterName As String
```

C#-Syntax

```
public override string ParameterName {get;set;}
```

Bemerkungen

Standardwert ist eine leere Zeichenfolge.

Der SQL Anywhere .NET-Datenprovider benutzt anstelle von benannten Parametern positionsbasierte Parameter, die mit einem Fragezeichen (?) markiert sind.

Precision-Eigenschaft

Ruft die maximale Anzahl von Stellen ab, die für die Darstellung der Value-Eigenschaft benutzt werden, bzw. legt den betreffenden Wert fest.

Visual Basic-Syntax

```
Public Property Precision As Byte
```

C#-Syntax

```
public byte Precision {get;set;}
```

Bemerkungen

Der Wert dieser Eigenschaft ist die maximale Anzahl von Stellen, die benutzt werden, um die Value-Eigenschaft darzustellen. Der Standardwert ist 0, was bedeutet, dass der Datenprovider die Gesamtstellenzahl für die Value-Eigenschaft definiert.

Die Precision-Eigenschaft wird nur für dezimale und numerische Eingabeparameter verwendet.

SADbType-Eigenschaft

Der SADbType des Parameters.

Visual Basic-Syntax

```
Public Property SADbType As SADbType
```

C#-Syntax

```
public SADbType SADbType {get;set;}
```

Bemerkungen

SADbType und DbType sind miteinander verknüpft. Die Definition von SADbType ändert daher DbType zu einem unterstützenden DbType.

Der Wert muss ein Mitglied des SADbType-Enumerators sein.

Scale-Eigenschaft

Gibt die Anzahl der Dezimalstellen an, mit denen Value aufgelöst wird, bzw. ruft sie ab.

Visual Basic-Syntax

```
Public Property Scale As Byte
```

C#-Syntax

```
public byte Scale {get;set;}
```

Bemerkungen

Die Anzahl der Dezimalstellen, mit denen "Value" aufgelöst wird. Standardwert ist "0".

Die Scale-Eigenschaft wird nur für dezimale und numerische Eingabeparameter verwendet.

Size-Eigenschaft

Legt die in Byte angegebene Maximalgröße der Daten in der Spalte fest bzw. ruft sie ab.

Visual Basic-Syntax

```
Public Overrides Property Size As Integer
```

C#-Syntax

```
public override int Size {get;set;}
```

Bemerkungen

Der Wert dieser Eigenschaft ist die in Byte angegebene Maximalgröße der Daten in der Spalte. Der Standardwert wird aus dem Parameterwert abgerufen.

Der Wert dieser Eigenschaft ist die in Byte angegebene Maximalgröße der Daten in der Spalte. Der Standardwert wird aus dem Parameterwert abgerufen.

Die Size-Eigenschaft wird für die Datentypen "binary" und "string" verwendet.

Für Datentypen mit variabler Länge beschreibt die Size-Eigenschaft die maximale Datenmenge, die an den Server übersandt werden kann. Beispielsweise kann die Size-Eigenschaft benutzt werden, um die Menge von Daten, die für einen Zeichenfolgenwert an den Server geschickt werden, auf die ersten hundert Byte zu begrenzen.

Wenn der Wert nicht explizit gesetzt wurde, wird der Size-Parameter aus der aktuellen Größe des angegebenen Parameterwerts bezogen. Bei Datentypen mit fester Breite wird der Wert von "Size" ignoriert. Der Wert kann für Informationszwecke abgerufen werden und gibt die maximale Anzahl von Byte zurück, die der Provider bei der Übertragung des Werts vom Parameter auf den Server benutzt.

SourceColumn-Eigenschaft

Ruft den Namen der Quellspalte ab, die dem DataSet-Objekt zugewiesen ist und für das Laden und die Rückgabe des Werts verwendet wird, bzw. legt den betreffenden Spaltennamen fest.

Visual Basic-Syntax

```
Public Overrides Property SourceColumn As String
```

C#-Syntax

```
public override string SourceColumn {get;set;}
```

Bemerkungen

Eine Zeichenfolge, die den Namen der Quellspalte angibt, die dem DataSet-Objekt zugewiesen ist und für das Laden und die Rückgabe des Werts verwendet wird

Wenn SourceColumn auf eine andere Größe als eine leere Zeichenfolge gesetzt ist, wird der Wert des Parameters mit dem SourceColumn-Namen aus der Spalte ausgelesen. Wenn die Richtung auf "Input" gesetzt ist, wird der Wert aus dem DataSet-Objekt bezogen. Wenn die Richtung auf "Output" gesetzt ist, wird der Wert aus der Datenquelle bezogen. Die Richtungsangabe "InputOutput" ist eine Kombination aus beiden.

SourceColumnNullMapping-Eigenschaft

Legt einen Wert fest bzw. ruft einen Wert ab, der angibt, ob die Spalte nullwertfähig ist.

Visual Basic-Syntax

```
Public Overrides Property SourceColumnNullMapping As Boolean
```

C#-Syntax

```
public override bool SourceColumnNullMapping {get;set;}
```

Bemerkungen

Damit kann das SACommandBuilder-Objekt korrekte Update-Anweisungen für nullwertfähige Spalten erzeugen.

Wenn die Quellspalte nullwertfähig ist, wird TRUE zurückgegeben, sonst FALSE.

SourceVersion-Eigenschaft

Gibt den DataRowVersion-Wert an, der beim Laden von Value zu verwenden ist, bzw. ruft ihn ab.

Visual Basic-Syntax

```
Public Overrides Property SourceVersion As DataRowVersion
```

C#-Syntax

```
public override DataRowVersion SourceVersion {get;set;}
```

Bemerkungen

Wird von UpdateCommand während eines Update-Vorgangs verwendet, um zu ermitteln, ob der Parameterwert auf "Current" oder "Original" gesetzt ist. Dadurch können Primärschlüssel aktualisiert werden. Diese Eigenschaft wird von InsertCommand und DeleteCommand ignoriert. Diese Eigenschaft wird auf die Version des DataRow-Objekts, die von der Item-Eigenschaft benutzt wird, oder die GetChildRows-Methode des DataRow-Objekts eingestellt.

Value-Eigenschaft

Ruft den Wert des Parameters ab bzw. legt ihn fest.

Visual Basic-Syntax

```
Public Overrides Property Value As Object
```

C#-Syntax

```
public override object value {get;set;}
```

Bemerkungen

Ein Objekt, das den Wert des Parameters angibt.

Für Eingabeparameter ist der Wert an das SACommand-Objekt gebunden, das an den Server geschickt wird. Bei Ausgabe- und Rückgabewertparametern wird der Wert nach Abschluss des SACommand-Objekts nach dem Schließen der SADATAReader-Klasse definiert.

Wenn ein NULL-Parameterwert an den Server geschickt wird, muss der Benutzer DBNull angeben, nicht NULL. Im System ist NULL ein leeres Objekt, das keinen Wert hat. DBNull wird benutzt, um NULL darzustellen.

Wenn die Anwendung den Datentyp angibt, wird der gebundene Wert in diesen Datentyp konvertiert, wenn der SQL Anywhere .NET-Datenprovider die Daten an den Server sendet. Der Provider versucht, einen beliebigen Werttyp zu konvertieren, wenn er die IConvertible-Schnittstelle unterstützt. Konvertierungsfehler können auftreten, wenn der angegebene Typ nicht mit dem Wert kompatibel ist.

Die beiden Eigenschaften DbType und SADBType können ausgelesen werden, indem "Value" definiert wird.

Die Value-Eigenschaft wird von Update überschrieben.

SAPParameterCollection-Klasse

Stellt alle Parameter für ein SACommand-Objekt und optional auch deren Zuordnung zu einer DataSet-Spalte dar.

Visual Basic-Syntax

```
Public NotInheritable Class SAPParameterCollection  
    Inherits System.Data.Common.DbParameterCollection
```

C#-Syntax

```
public sealed class SAPParameterCollection :  
    System.Data.Common.DbParameterCollection
```

Basisklassen

- [System.Data.Common.DbParameterCollection](#)

Mitglieder

Alle Mitglieder der SAPParameterCollection-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Add-Methode	Fügt dieser Sammlung ein SAPParameter-Objekt hinzu.

Name	Beschreibung
AddRange-Methode	Fügt ein Array von Werten am Ende der Sammlung SAParameterCollection hinzu.
AddWithValue-Methode	Fügt einen Wert am Ende dieser Sammlung hinzu.
Clear-Methode	Entfernt alle Elemente aus der Sammlung.
Contains-Methode	Gibt an, ob ein SAParameter-Objekt in der Sammlung vorhanden ist.
CopyTo-Methode	Kopiert SAParameter-Objekte aus der Sammlung SAParameterCollection in das angegebene Array.
GetEnumerator-Methode	Gibt einen Enumerator zurück, der die Klasse SAParameterCollection durchläuft.
GetParameter-Methode (geerbt aus System.Data.Common.DbParameterCollection)	Gibt das System.Data.Common.DbParameter -Objekt am angegebenen Index der Sammlung zurück.
IndexOf-Methode	Gibt die Position des SAParameter-Objekts in der Sammlung zurück.
Insert-Methode	Fügt ein SAParameter-Objekt am angegebenen Index in die Sammlung ein.
Remove-Methode	Entfernt das angegebene SAParameter-Objekt aus der Sammlung.
RemoveAt-Methode	Entfernt das angegebene SAParameter-Objekt aus der Sammlung.
SetParameter-Methode (geerbt aus System.Data.Common.DbParameterCollection)	Setzt das System.Data.Common.DbParameter -Objekt am angegebenen Index auf einen neuen Wert.
Count-Eigenschaft	Gibt die Anzahl der SAParameter-Objekte in der Sammlung zurück.
IsFixedSize-Eigenschaft	Ruft einen Wert ab, der angibt, ob die Sammlung SAParameterCollection eine feste Größe hat.
IsReadOnly-Eigenschaft	Ruft einen Wert ab, der angibt, ob die Sammlung SAParameterCollection schreibgeschützt ist.
IsSynchronized-Eigenschaft	Ruft einen Wert ab, der angibt, ob die Sammlung SAParameterCollection synchronisiert ist.
SyncRoot-Eigenschaft	Ruft ein Objekt ab, das verwendet werden kann, um den Zugriff auf die Sammlung SAParameterCollection zu synchronisieren.

Name	Beschreibung
this-Eigenschaft	Ruft das SAParameter-Objekt am angegebenen Index ab bzw. legt es fest.

Bemerkungen

Es gibt keinen Konstruktor für SAParameterCollection. Sie erhalten ein SAParameterCollection-Objekt aus der SACommand.Parameters-Eigenschaft eines SACommand-Objekts.

Siehe auch

- [SACommand-Klasse \[SQL Anywhere .NET\] auf Seite 129](#)
- [SACommand.Parameters-Eigenschaft \[SQL Anywhere .NET\] auf Seite 150](#)
- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)

Add-Methode

Fügt dieser Sammlung ein SAParameter-Objekt hinzu.

Überladungsliste

Name	Beschreibung
Add(Object)-Methode	Fügt dieser Sammlung ein SAParameter-Objekt hinzu.
Add(SAParameter)-Methode	Fügt dieser Sammlung ein SAParameter-Objekt hinzu.
Add(String, Object)-Methode	Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit dem angegebenen Parameternamen und -wert erstellt wurde.
Add(String, SADBType)-Methode	Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit dem angegebenen Parameternamen und Datentyp erstellt wurde.
Add(String, SADBType, Int)-Methode	Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit den angegebenen Werten für Parameternamen, Datentyp und Länge erstellt wurde.
Add(String, SADBType, Int, String)-Methode	Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit den angegebenen Werten für Parameternamen, Datentyp, Länge und Quellspaltenname erstellt wurde.

Add(Object)-Methode

Fügt dieser Sammlung ein SAParameter-Objekt hinzu.

Visual Basic-Syntax

```
Public Overrides Function Add(ByVal value As Object) As Integer
```

C#-Syntax

```
public override int Add(object value)
```

Parameter

- **value** Das SAPParameter-Objekt, das der Sammlung hinzugefügt werden soll.

Rückgabe

Der Index des neuen SAPParameter-Objekts.

Siehe auch

- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)

Add(SAPParameter)-Methode

Fügt dieser Sammlung ein SAPParameter-Objekt hinzu.

Visual Basic-Syntax

```
Public Function Add(ByVal value As SAPParameter) As SAPParameter
```

C#-Syntax

```
public SAPParameter Add(SAPParameter value)
```

Parameter

- **value** Das SAPParameter-Objekt, das der Sammlung hinzugefügt werden soll.

Rückgabe

Das neue SAPParameter-Objekt.

Add(String, Object)-Methode

Fügt der Sammlung ein neues SAPParameter-Objekt hinzu, das mit dem angegebenen Parameternamen und -wert erstellt wurde.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal value As Object  
) As SAPParameter
```

C#-Syntax

```
public SAPParameter Add(string parameterName, object value)
```

Parameter

- **parameterName** Der Name des Parameters.
- **value** Der Wert des Parameters, der zur Verbindung hinzugefügt werden soll.

Rückgabe

Das neue SAParameter-Objekt.

Bemerkungen

Aufgrund der speziellen Behandlung der 0- und 0.0-Konstanten und der Weise, wie überladene Methoden gelöst werden, wird dringend empfohlen, für Konstantenwerte explizit Typenobjekte festzulegen, wenn diese Methode verwendet wird.

Siehe auch

- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)

Add(String, SADBType)-Methode

Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit dem angegebenen Parameternamen und Datentyp erstellt wurde.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal sADBType As SADBType  
) As SAParameter
```

C#-Syntax

```
public SAParameter Add(string parameterName, SADBType sADBType)
```

Parameter

- **parameterName** Der Name des Parameters.
- **sADBType** Einer der SADBType-Werte.

Rückgabe

Das neue SAParameter-Objekt.

Siehe auch

- [SADBType-Enumeration \[SQL Anywhere .NET\] auf Seite 328](#)
- [SAParameter.Value-Eigenschaft \[SQL Anywhere .NET\] auf Seite 289](#)

Add(String, SADBType, Int)-Methode

Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit den angegebenen Werten für Parameternamen, Datentyp und Länge erstellt wurde.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal saDbType As SADBType,  
    ByVal size As Integer  
) As SAParameter
```

C#-Syntax

```
public SAParameter Add(  
    string parameterName,  
    SADBType saDbType,  
    int size  
)
```

Parameter

- **parameterName** Der Name des Parameters.
- **saDbType** Einer der SADBType-Werte.
- **size** Die Länge des Parameters.

Rückgabe

Das neue SAParameter-Objekt.

Siehe auch

- [SADBType-Enumeration \[SQL Anywhere .NET\] auf Seite 328](#)
- [SAParameter.Value-Eigenschaft \[SQL Anywhere .NET\] auf Seite 289](#)

Add(String, SADBType, Int, String)-Methode

Fügt der Sammlung ein neues SAParameter-Objekt hinzu, das mit den angegebenen Werten für Parametername, Datentyp, Länge und Quellspaltenname erstellt wurde.

Visual Basic-Syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal saDbType As SADBType,  
    ByVal size As Integer,  
    ByVal sourceColumn As String  
) As SAParameter
```

C#-Syntax

```
public SAParameter Add(  
    string parameterName,  
    SADBType saDbType,  
    int size,  
    string sourceColumn  
)
```

Parameter

- **parameterName** Der Name des Parameters.
- **sADBType** Einer der SADBType-Werte.
- **size** Die Länge der Spalte.
- **sourceColumn** Der Name der Quellspalte, die zuzuordnen ist.

Rückgabe

Das neue SAParameter-Objekt.

Siehe auch

- [SADBType-Enumeration \[SQL Anywhere .NET\] auf Seite 328](#)
- [SAParameter.Value-Eigenschaft \[SQL Anywhere .NET\] auf Seite 289](#)

AddRange-Methode

Fügt ein Array von Werten am Ende der Sammlung SAParameterCollection hinzu.

Überladungsliste

Name	Beschreibung
AddRange(Array)-Methode	Fügt ein Array von Werten am Ende der Sammlung SAParameterCollection hinzu.
AddRange(SAParameter[])-Methode	Fügt ein Array von Werten am Ende der Sammlung SAParameterCollection hinzu.

AddRange(Array)-Methode

Fügt ein Array von Werten am Ende der Sammlung SAParameterCollection hinzu.

Visual Basic-Syntax

```
Public Overrides Sub AddRange(ByVal values As Array)
```

C#-Syntax

```
public override void AddRange(Array values)
```

Parameter

- **values** Die hinzuzufügenden Werte.

AddRange(SAParameter[])-Methode

Fügt ein Array von Werten am Ende der Sammlung SAParameterCollection hinzu.

Visual Basic-Syntax

```
Public Sub AddRange(ByVal values As SAParameter())
```

C#-Syntax

```
public void AddRange(SAParameter[] values)
```

Parameter

- **values** Ein Array von SAParameter-Objekten, die an das Ende dieser Sammlung hinzugefügt werden sollen.

AddWithValue-Methode

Fügt einen Wert am Ende dieser Sammlung hinzu.

Visual Basic-Syntax

```
Public Function AddWithValue(  
    ByVal parameterName As String,  
    ByVal value As Object  
) As SAParameter
```

C#-Syntax

```
public SAParameter AddWithValue(string parameterName, object value)
```

Parameter

- **parameterName** Der Name des Parameters.
- **value** Der Wert, der hinzugefügt werden soll.

Rückgabe

Das neue SAParameter-Objekt.

Clear-Methode

Entfernt alle Elemente aus der Sammlung.

Visual Basic-Syntax

```
Public Overrides Sub Clear()
```

C#-Syntax

```
public override void Clear()
```

Contains-Methode

Gibt an, ob ein SAParameter-Objekt in der Sammlung vorhanden ist.

Überladungsliste

Name	Beschreibung
Contains(Object)-Methode	Gibt an, ob ein SAParameter-Objekt in der Sammlung vorhanden ist.
Contains(String)-Methode	Gibt an, ob ein SAParameter-Objekt in der Sammlung vorhanden ist.

Contains(Object)-Methode

Gibt an, ob ein SAParameter-Objekt in der Sammlung vorhanden ist.

Visual Basic-Syntax

```
Public Overrides Function Contains(ByVal value As Object) As Boolean
```

C#-Syntax

```
public override bool Contains(object value)
```

Parameter

- **value** Das SAParameter-Objekt, das gesucht werden soll

Rückgabe

TRUE, wenn die Sammlung das SAParameter-Objekt enthält. Sonst FALSE.

Siehe auch

- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAParameterCollection.Contains-Methode \[SQL Anywhere .NET\] auf Seite 294](#)

Contains(String)-Methode

Gibt an, ob ein SAParameter-Objekt in der Sammlung vorhanden ist.

Visual Basic-Syntax

```
Public Overrides Function Contains(ByVal value As String) As Boolean
```

C#-Syntax

```
public override bool Contains(string value)
```

Parameter

- **value** Der Name des zu suchenden Parameters.

Rückgabe

TRUE, wenn die Sammlung das SAParameter-Objekt enthält. Sonst FALSE.

Siehe auch

- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAPParameterCollection.Contains-Methode \[SQL Anywhere .NET\] auf Seite 294](#)

CopyTo-Methode

Kopiert SAPParameter-Objekte aus der Sammlung SAPParameterCollection in das angegebene Array.

Visual Basic-Syntax

```
Public Overrides Sub CopyTo(  
    ByVal array As Array,  
    ByVal index As Integer  
)
```

C#-Syntax

```
public override void CopyTo(Array array, int index)
```

Parameter

- **array** Das Array, in das die SAPParameter-Objekte kopiert werden sollen
- **index** Der Startindex des Arrays.

Siehe auch

- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAPParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)

GetEnumerator-Methode

Gibt einen Enumerator zurück, der die Klasse SAPParameterCollection durchläuft.

Visual Basic-Syntax

```
Public Overrides Function GetEnumerator()  
    As System.Collections.IEnumerator
```

C#-Syntax

```
public override IEnumerator GetEnumerator()
```

Rückgabe

Ein System.Collections.IEnumerator für das SAPParameterCollection-Objekt.

Siehe auch

- [SAPParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)

IndexOf-Methode

Gibt die Position des SAParameter-Objekts in der Sammlung zurück.

Überladungsliste

Name	Beschreibung
IndexOf(Object)-Methode	Gibt die Position des SAParameter-Objekts in der Sammlung zurück.
IndexOf(String)-Methode	Gibt die Position des SAParameter-Objekts in der Sammlung zurück.

IndexOf(Object)-Methode

Gibt die Position des SAParameter-Objekts in der Sammlung zurück.

Visual Basic-Syntax

```
Public Overrides Function IndexOf(ByVal value As Object) As Integer
```

C#-Syntax

```
public override int IndexOf(object value)
```

Parameter

- **value** Das SAParameter-Objekt, das gefunden werden soll

Rückgabe

Die auf Null basierende Position des SAParameter-Objekts in der Sammlung

Siehe auch

- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAParameterCollection.IndexOf-Methode \[SQL Anywhere .NET\] auf Seite 297](#)

IndexOf(String)-Methode

Gibt die Position des SAParameter-Objekts in der Sammlung zurück.

Visual Basic-Syntax

```
Public Overrides Function IndexOf(  
    ByVal parameterName As String  
) As Integer
```

C#-Syntax

```
public override int IndexOf(string parameterName)
```

Parameter

- **parameterName** Der Name des Parameters, der gefunden werden soll.

Rückgabe

Der auf Null basierende Index des SAParameter-Objekts in der Sammlung

Siehe auch

- [SAParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAParameterCollection.IndexOf-Methode \[SQL Anywhere .NET\] auf Seite 297](#)

Insert-Methode

Fügt ein SAParameter-Objekt am angegebenen Index in die Sammlung ein.

Visual Basic-Syntax

```
Public Overrides Sub Insert(  
    ByVal index As Integer,  
    ByVal value As Object  
)
```

C#-Syntax

```
public override void Insert(int index, object value)
```

Parameter

- **index** Der auf Null basierende Index, in den der Parameter innerhalb der Sammlung eingefügt werden soll.
- **value** Das SAParameter-Objekt, das der Sammlung hinzugefügt werden soll.

Remove-Methode

Entfernt das angegebene SAParameter-Objekt aus der Sammlung.

Visual Basic-Syntax

```
Public Overrides Sub Remove(ByVal value As Object)
```

C#-Syntax

```
public override void Remove(object value)
```

Parameter

- **value** Das SAParameter-Objekt, das aus der Sammlung entfernt werden soll

RemoveAt-Methode

Entfernt das angegebene SAParameter-Objekt aus der Sammlung.

Überladungsliste

Name	Beschreibung
RemoveAt(Int)-Methode	Entfernt das angegebene SAParameter-Objekt aus der Sammlung.
RemoveAt(String)-Methode	Entfernt das angegebene SAParameter-Objekt aus der Sammlung.

RemoveAt(Int)-Methode

Entfernt das angegebene SAParameter-Objekt aus der Sammlung.

Visual Basic-Syntax

```
Public Overrides Sub RemoveAt(ByVal index As Integer)
```

C#-Syntax

```
public override void RemoveAt(int index)
```

Parameter

- **index** Der auf Null basierende Index des zu entfernenden Parameters.

Siehe auch

- [SAParameterCollection.RemoveAt-Methode \[SQL Anywhere .NET\] auf Seite 298](#)

RemoveAt(String)-Methode

Entfernt das angegebene SAParameter-Objekt aus der Sammlung.

Visual Basic-Syntax

```
Public Overrides Sub RemoveAt(ByVal parameterName As String)
```

C#-Syntax

```
public override void RemoveAt(string parameterName)
```

Parameter

- **parameterName** Der Name des zu entfernenden SAParameter-Objekts

Siehe auch

- [SAParameterCollection.RemoveAt-Methode \[SQL Anywhere .NET\] auf Seite 298](#)

Count-Eigenschaft

Gibt die Anzahl der SAParameter-Objekte in der Sammlung zurück.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property Count As Integer
```

C#-Syntax

```
public override int Count {get;}
```

Bemerkungen

Die Anzahl von SAPParameter-Objekten in der Sammlung

Siehe auch

- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAPParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)

IsFixedSize-Eigenschaft

Ruft einen Wert ab, der angibt, ob die Sammlung SAPParameterCollection eine feste Größe hat.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property IsFixedSize As Boolean
```

C#-Syntax

```
public override bool IsFixedSize {get;}
```

Bemerkungen

TRUE, wenn diese Sammlung eine feste Größe hat, andernfalls FALSE.

IsReadOnly-Eigenschaft

Ruft einen Wert ab, der angibt, ob die Sammlung SAPParameterCollection schreibgeschützt ist.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property IsReadOnly As Boolean
```

C#-Syntax

```
public override bool IsReadOnly {get;}
```

Bemerkungen

TRUE, wenn diese Sammlung schreibgeschützt ist, andernfalls FALSE.

IsSynchronized-Eigenschaft

Ruft einen Wert ab, der angibt, ob die Sammlung SAPParameterCollection synchronisiert ist.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property IsSynchronized As Boolean
```

C#-Syntax

```
public override bool IsSynchronized {get;}
```

Bemerkungen

TRUE, wenn diese Sammlung synchronisiert ist, andernfalls FALSE.

SyncRoot-Eigenschaft

Ruft ein Objekt ab, das verwendet werden kann, um den Zugriff auf die Sammlung SAParameterCollection zu synchronisieren.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property SyncRoot As Object
```

C#-Syntax

```
public override object SyncRoot {get;}
```

this-Eigenschaft

Ruft das SAParameter-Objekt am angegebenen Index ab bzw. legt es fest.

Überladungsliste

Name	Beschreibung
this[Int]-Eigenschaft	Ruft das SAParameter-Objekt am angegebenen Index ab bzw. legt es fest.
this[String]-Eigenschaft	Ruft das SAParameter-Objekt am angegebenen Index ab bzw. legt es fest.

this[Int]-Eigenschaft

Ruft das SAParameter-Objekt am angegebenen Index ab bzw. legt es fest.

Visual Basic-Syntax

```
Public Shadows Property Item(ByVal index As Integer) As SAParameter
```

C#-Syntax

```
public new SAParameter this[int index] {get;set;}
```

Parameter

- **index** Der auf Null basierende Index des abzurufenden Parameters.

Rückgabe

Der SAPParameter an der Stelle des angegebenen Indexes.

Bemerkungen

Ein SAPParameter-Objekt.

In C# ist diese Eigenschaft der Indexersteller für das SAPParameterCollection-Objekt.

Siehe auch

- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAPParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)

this[String]-Eigenschaft

Ruft das SAPParameter-Objekt am angegebenen Index ab bzw. legt es fest.

Visual Basic-Syntax

```
Public Shadows Property Item(  
    ByVal parameterName As String  
) As SAPParameter
```

C#-Syntax

```
public new SAPParameter this[string parameterName] {get;set;}
```

Parameter

- **parameterName** Der Name des abzurufenden Parameters.

Rückgabe

Das SAPParameter-Objekt mit dem angegebenen Namen.

Bemerkungen

Ein SAPParameter-Objekt.

In C# ist diese Eigenschaft der Indexersteller für das SAPParameterCollection-Objekt.

Siehe auch

- [SAPParameter-Klasse \[SQL Anywhere .NET\] auf Seite 276](#)
- [SAPParameterCollection-Klasse \[SQL Anywhere .NET\] auf Seite 287](#)
- [SADataReader.GetOrdinal-Methode \[SQL Anywhere .NET\] auf Seite 238](#)
- [SADataReader.GetValue-Methode \[SQL Anywhere .NET\] auf Seite 243](#)
- [SADataReader.GetFieldType-Methode \[SQL Anywhere .NET\] auf Seite 235](#)

SAPermission-Klasse

Ermöglicht es dem SQL Anywhere .NET-Datenprovider sicherzustellen, dass ein Benutzer eine Sicherheitsstufe besitzt, die für den Zugriff auf eine SQL Anywhere-Datenquelle geeignet ist.

Visual Basic-Syntax

```
Public NotInheritable Class SAPermission
    Inherits System.Data.Common.DBDataPermission
```

C#-Syntax

```
public sealed class SAPermission : System.Data.Common.DBDataPermission
```

Basisklassen

- [System.Data.Common.DBDataPermission](#)

Mitglieder

Alle Mitglieder der SAPermission-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SAPermission-Konstruktor	Initialisiert eine neue Instanz der SAPermission-Klasse.
Add-Methode (geerbt aus System.Data.Common.DBDataPermission)	Fügt dem vorhandenen Status des DBDataPermission-Objekts den Zugriff auf die angegebene Verbindungszeichenfolge hinzu.
Clear-Methode (geerbt aus System.Data.Common.DBDataPermission)	Entfernt alle Berechtigungen, die zuvor unter Verwendung der System.Data.Common.DBDataPermission.Add(System.String, System.String, System.Data.KeyRestrictionBehavior) -Methode hinzugefügt wurden.
Copy-Methode (geerbt aus System.Data.Common.DBDataPermission)	Erstellt eine identische Kopie des aktuellen Berechtigungsobjekts und gibt sie zurück.
FromXml-Methode (geerbt aus System.Data.Common.DBDataPermission)	Baut ein Sicherheitsobjekt mit einem angegebenen Status aus einer XML-Kodierung neu auf.
Intersect-Methode (geerbt aus System.Data.Common.DBDataPermission)	Gibt ein neues Berechtigungsobjekt zurück, das die Schnittmenge zwischen dem aktuellen Berechtigungsobjekt und dem angegebenen Berechtigungsobjekt darstellt.
IsSubsetOf-Methode (geerbt aus System.Data.Common.DBDataPermission)	Gibt einen Wert zurück, der angibt, ob das aktuelle Berechtigungsobjekt eine Teilmenge des angegebenen Berechtigungsobjekts ist.

Name	Beschreibung
IsUnrestricted-Methode (geerbt aus System.Data.Common.DBDataPermission)	Gibt einen Wert zurück, der angibt, ob die Berechtigung ohne Kenntnis der Berechtigungssemantik als unbeschränkt dargestellt werden kann.
ToXml-Methode (geerbt aus System.Data.Common.DBDataPermission)	Erstellt eine XML-Kodierung des Sicherheitsobjekts und seines aktuellen Status.
Union-Methode (geerbt aus System.Data.Common.DBDataPermission)	Gibt ein neues Berechtigungsobjekt zurück, das die Vereinigung von aktuellem und angegebenem Berechtigungsobjekt darstellt.
AllowBlankPassword-Eigenschaft (geerbt aus System.Data.Common.DBDataPermission)	Ruft einen Wert ab, der angibt, ob ein leeres Kennwort zulässig ist.

SAPermission-Konstruktor

Initialisiert eine neue Instanz der SAPermission-Klasse.

Visual Basic-Syntax

```
Public Sub New(ByVal state As PermissionState)
```

C#-Syntax

```
public SAPermission(PermissionState state)
```

Parameter

- **state** Einer der PermissionState-Werte.

SAPermissionAttribute-Klasse

Verknüpft eine Sicherheitsaktion mit einem benutzerdefinierten Sicherheitsattribut.

Visual Basic-Syntax

```
Public NotInheritable Class SAPermissionAttribute  
    Inherits System.Data.Common.DBDataPermissionAttribute
```

C#-Syntax

```
public sealed class SAPermissionAttribute :  
    System.Data.Common.DBDataPermissionAttribute
```

Basisklassen

- [System.Data.Common.DBDataPermissionAttribute](#)

Mitglieder

Alle Mitglieder der SAPermissionAttribute-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SAPermissionAttribute-Konstruktor	Initialisiert eine neue Instanz der Klasse SAPermissionAttribute.
CreatePermission-Methode	Gibt ein SAPermission-Objekt zurück, das gemäß den Attributeigenschaften konfiguriert ist.
ShouldSerializeConnectionString-Methode (geerbt aus System.Data.Common.DBDataPermissionAttribute)	Gibt an, ob das Attribut die Verbindungszeichenfolge serialisieren soll.
ShouldSerializeKeyRestrictions-Methode (geerbt aus System.Data.Common.DBDataPermissionAttribute)	Gibt an, ob das Attribut die Menge der Schlüsseinschränkungen serialisieren soll.
AllowBlankPassword-Eigenschaft (geerbt aus System.Data.Common.DBDataPermissionAttribute)	Ruft einen Wert ab, der angibt, ob ein leeres Kennwort zulässig ist, bzw. legt ihn fest.
ConnectionString-Eigenschaft (geerbt aus System.Data.Common.DBDataPermissionAttribute)	Ruft eine zulässige Verbindungszeichenfolge ab oder legt sie fest.
KeyRestrictionBehavior-Eigenschaft (geerbt aus System.Data.Common.DBDataPermissionAttribute)	Legt fest, ob die Liste der Verbindungszeichenfolgen-Parameter, die durch die DBDataPermissionAttribute.KeyRestrictions -Eigenschaft angegeben sind, die einzigen zulässigen Verbindungszeichenfolgen-Parameter enthält.
KeyRestrictions-Eigenschaft (geerbt aus System.Data.Common.DBDataPermissionAttribute)	Ruft die zulässigen bzw. nicht zulässigen Verbindungszeichenfolgen-Parameter ab oder legt sie fest.

SAPermissionAttribute-Konstruktor

Initialisiert eine neue Instanz der Klasse SAPermissionAttribute.

Visual Basic-Syntax

```
Public Sub New(ByVal action As SecurityAction)
```

C#-Syntax

```
public SAPermissionAttribute(SecurityAction action)
```

Parameter

- **action** Einer der SecurityAction-Werte, der eine Aktion darstellt, die mithilfe der deklarativen Sicherheitsfunktion durchgeführt werden kann.

Rückgabe

Ein SAPermissionAttribute-Objekt.

CreatePermission-Methode

Gibt ein SAPermission-Objekt zurück, das gemäß den Attributeigenschaften konfiguriert ist.

Visual Basic-Syntax

```
Public Overrides Function CreatePermission() As IPermission
```

C#-Syntax

```
public override IPermission CreatePermission()
```

SARowsCopiedEventArgs-Klasse

Repräsentiert die Gruppe der Argumente, die an das SARowsCopiedEventHandler-Objekt übergeben werden.

Visual Basic-Syntax

```
Public NotInheritable Class SARowsCopiedEventArgs
```

C#-Syntax

```
public sealed class SARowsCopiedEventArgs
```

Mitglieder

Alle Mitglieder der SARowsCopiedEventArgs-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SARowsCopiedEventArgs-Konstruktor	Erstellt eine neue Instanz des SARowUpdatedEventArgs-Objekts.
Abort-Eigenschaft	Ruft einen Wert ab bzw. legt einen Wert fest, der angibt, ob der Massenexport/-import-Vorgang abgebrochen werden soll.
RowsCopied-Eigenschaft	Ruft die Anzahl der Zeilen ab, die während des aktuellen Vorgangs des Massenexports/-imports kopiert werden.

Bemerkungen

Die Klasse SARowsCopiedEventArgs ist in .NET Compact Framework 2.0 nicht verfügbar.

SARowsCopiedEventArgs-Konstruktor

Erstellt eine neue Instanz des SARowUpdatedEventArgs-Objekts.

Visual Basic-Syntax

```
Public Sub New(ByVal rowsCopied As Long)
```

C#-Syntax

```
public SARowsCopiedEventArgs(long rowsCopied)
```

Parameter

- **rowsCopied** Ein 64-Bit Ganzzahlwert, der die Anzahl der Zeilen angibt, die während des aktuellen Massensexport/-import-Vorgangs kopiert werden

Bemerkungen

Die Klasse SARowsCopiedEventArgs ist in .NET Compact Framework 2.0 nicht verfügbar.

Abort-Eigenschaft

Ruft einen Wert ab bzw. legt einen Wert fest, der angibt, ob der Massensexport/-import-Vorgang abgebrochen werden soll.

Visual Basic-Syntax

```
Public Property Abort As Boolean
```

C#-Syntax

```
public bool Abort {get;set;}
```

Bemerkungen

Die Klasse SARowsCopiedEventArgs ist in .NET Compact Framework 2.0 nicht verfügbar.

RowsCopied-Eigenschaft

Ruft die Anzahl der Zeilen ab, die während des aktuellen Vorgangs des Massensexports/-imports kopiert werden.

Visual Basic-Syntax

```
Public ReadOnly Property RowsCopied As Long
```

C#-Syntax

```
public long RowsCopied {get;}
```

Bemerkungen

Die Klasse SARowsCopiedEventArgs ist in .NET Compact Framework 2.0 nicht verfügbar.

SARowUpdatedEventArgs-Klasse

Stellt Daten für das RowUpdated-Ereignis bereit.

Visual Basic-Syntax

```
Public NotInheritable Class SARowUpdatedEventArgs  
    Inherits System.Data.Common.RowUpdatedEventArgs
```

C#-Syntax

```
public sealed class SARowUpdatedEventArgs :  
    System.Data.Common.RowUpdatedEventArgs
```

Basisklassen

- [System.Data.Common.RowUpdatedEventArgs](#)

Mitglieder

Alle Mitglieder der SARowUpdatedEventArgs-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SARowUpdatedEventArgs-Konstruktor	Initialisiert eine neue Instanz der Klasse SARowUpdatedEventArgs.
CopyToRows-Methode (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Kopiert Referenzen zu den geänderten Zeilen in das angegebene Array.
Command-Eigenschaft	Ruft das SACommand-Objekt ab, das ausgeführt wird, wenn DataAdapter.Update aufgerufen wird.
Errors-Eigenschaft (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Ruft alle Fehler ab, die beim Ausführen von System.Data.Common.RowUpdatedEventArgs.Command vom .NET Framework-Datenprovider generiert wurden.
RecordsAffected-Eigenschaft	Gibt die Anzahl der durch eine SQL-Anweisung geänderten, eingefügten oder gelöschten Zeilen zurück.
Row-Eigenschaft (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Ruft das System.Data.DataRow -Objekt ab, das durch System.Data.Common.DbDataAdapter.Update(System.Data.DataSet) gesendet wurde.
RowCount-Eigenschaft (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Ruft die Anzahl der Zeilen ab, die in einem Batch von aktualisierten Datensätzen abgearbeitet wurden.
StatementType-Eigenschaft (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Ruft den Typ einer ausgeführten SQL-Anweisung ab.

Name	Beschreibung
Status-Eigenschaft (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Ruft das System.Data.UpdateStatus -Objekt der System.Data.Common.RowUpdatedEventArgs.Command -Eigenschaft ab.
TableMapping-Eigenschaft (geerbt aus System.Data.Common.RowUpdatedEventArgs)	Ruft das System.Data.Common.DataTableMapping -Objekt ab, das durch System.Data.Common.DbDataAdapter.Update(System.Data.DataSet) gesendet wurde.

SARowUpdatedEventArgs-Konstruktor

Initialisiert eine neue Instanz der Klasse SARowUpdatedEventArgs.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal row As DataRow,  
    ByVal command As IDbCommand,  
    ByVal statementType As StatementType,  
    ByVal tableMapping As DataTableMapping  
)
```

C#-Syntax

```
public SARowUpdatedEventArgs(  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping  
)
```

Parameter

- **row** Das DataRow-Objekt, das durch ein Update gesendet wird.
- **command** Das IDbCommand-Objekt, das beim Aufruf von Update ausgeführt wird.
- **statementType** Einer der StatementType-Werte, die den Typ der angegebenen Abfrage definieren.
- **tableMapping** Die DataTableMapping-Eigenschaft, die über ein Update gesendet wird.

Command-Eigenschaft

Ruft das SACommand-Objekt ab, das ausgeführt wird, wenn DataAdapter.Update aufgerufen wird.

Visual Basic-Syntax

```
Public ReadOnly Shadows Property Command As SACommand
```

C#-Syntax

```
public new SACommand Command {get;}
```

RecordsAffected-Eigenschaft

Gibt die Anzahl der durch eine SQL-Anweisung geänderten, eingefügten oder gelöschten Zeilen zurück.

Visual Basic-Syntax

```
Public ReadOnly Shadows Property RecordsAffected As Integer
```

C#-Syntax

```
public new int RecordsAffected {get;}
```

Bemerkungen

Die Anzahl von Zeilen, die geändert, eingefügt oder gelöscht wurden, 0, wenn keine Zeilen betroffen waren oder die Anweisung fehlgeschlagen ist, und -1 bei SELECT-Anweisungen.

SARowUpdatingEventArgs-Klasse

Stellt Daten für das RowUpdating-Ereignis bereit.

Visual Basic-Syntax

```
Public NotInheritable Class SARowUpdatingEventArgs  
    Inherits System.Data.Common.RowUpdatingEventArgs
```

C#-Syntax

```
public sealed class SARowUpdatingEventArgs :  
    System.Data.Common.RowUpdatingEventArgs
```

Basisklassen

- [System.Data.Common.RowUpdatingEventArgs](#)

Mitglieder

Alle Mitglieder der SARowUpdatingEventArgs-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SARowUpdatingEventArgs-Konstruktor	Initialisiert eine neue Instanz der SARowUpdatingEventArgs-Klasse.
BaseCommand-Eigenschaft (geerbt aus System.Data.Common.RowUpdatingEventArgs)	Ruft das System.Data.IDbCommand -Objekt für eine Instanz dieser Klasse ab bzw. legt es fest.
Command-Eigenschaft	Gibt das SACommand-Objekt an, das bei der Durchführung des Update-Vorgangs auszuführen ist.

Name	Beschreibung
Errors-Eigenschaft (geerbt aus <code>System.Data.Common.RowUpdatingEventArgs</code>)	Ruft alle vom .NET Framework-Datenprovider generierten Fehler ab, wenn System.Data.Common.RowUpdatedEventArgs.Command ausgeführt wird
Row-Eigenschaft (geerbt aus <code>System.Data.Common.RowUpdatingEventArgs</code>)	Ruft das System.Data.DataRow -Objekt ab, das als Teil eines Einfügungs-, Aktualisierungs- oder Löschvorgangs an den Datenbankserver gesendet wird.
StatementType-Eigenschaft (geerbt aus <code>System.Data.Common.RowUpdatingEventArgs</code>)	Ruft den Typ einer auszuführenden SQL-Anweisung ab.
Status-Eigenschaft (geerbt aus <code>System.Data.Common.RowUpdatingEventArgs</code>)	Ruft das System.Data.UpdateStatus -Objekt der System.Data.Common.RowUpdatedEventArgs.Command -Eigenschaft ab bzw. legt es fest.
TableMapping-Eigenschaft (geerbt aus <code>System.Data.Common.RowUpdatingEventArgs</code>)	Ruft das System.Data.Common.DataTableMapping ausgeführt werden soll-Objekt ab, das über System.Data.Common.DbDataAdapter.Update(System.Data.DataSet) gesendet werden soll.

SARowUpdatingEventArgs-Konstruktor

Initialisiert eine neue Instanz der SARowUpdatingEventArgs-Klasse.

Visual Basic-Syntax

```
Public Sub New(  
    ByVal row As DataRow,  
    ByVal command As IDbCommand,  
    ByVal statementType As StatementType,  
    ByVal tableMapping As DataTableMapping  
)
```

C#-Syntax

```
public SARowUpdatingEventArgs(  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping  
)
```

Parameter

- **row** Das zu aktualisierende DataRow-Objekt.
- **command** Das IDbCommand-Objekt, das während des Update-Vorgangs auszuführen ist.

- **statementType** Einer der StatementType-Werte, die den Typ der angegebenen Abfrage definieren.
- **tableMapping** Die DataTableMapping-Eigenschaft, die über ein Update gesendet wird.

Command-Eigenschaft

Gibt das SACommand-Objekt an, das bei der Durchführung des Update-Vorgangs auszuführen ist.

Visual Basic-Syntax

```
Public Shadows Property Command As SACommand
```

C#-Syntax

```
public new SACommand Command {get;set;}
```

SAServerSideConnection-Klasse

Stellt eine serverseitige Verbindung mit einer SQL Anywhere-Datenbank dar.

Visual Basic-Syntax

```
Public NotInheritable Class SAServerSideConnection
```

C#-Syntax

```
public sealed class SAServerSideConnection
```

Mitglieder

Alle Mitglieder der SAServerSideConnection-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Connection-Eigenschaft	Stellt das SAConnection-Objekt bereit, das zum Ausführen von SQL-Anweisungen im Datenbankserver erforderlich ist.

Bemerkungen

Diese Klasse unterstützt die Ausführung von serverseitigen SQL-Anweisungen aus der externen CLR-Umgebung. Eine CLR-Funktion oder Methode, die vom Datenbankserver aufgerufen wird, kann einen Callback in den Server durchführen. Eine Verbindung mit dem Server ist bereits hergestellt. Es ist nicht notwendig, mit SAConnection eine Verbindung herzustellen.

Connection-Eigenschaft

Stellt das SAConnection-Objekt bereit, das zum Ausführen von SQL-Anweisungen im Datenbankserver erforderlich ist.

Visual Basic-Syntax

```
Public Shared ReadOnly Property Connection As SAConnection
```

C#-Syntax

```
public SAConnection Connection {get;}
```

Bemerkungen

Die Verbindung mit dem Server ist bereits hergestellt. Es ist nicht notwendig, mit SAConnection eine Verbindung herzustellen. Rufen Sie das Verbindungsobjekt mit einer Anweisung ähnlich der folgenden ab:

```
SAConnection _conn = SAServerSideConnection.Connection;
```

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)

Beispiel

Im folgenden Beispiel wird das SAConnection-Objekt abgerufen und anschließend eine SQL-Anweisung ausgeführt, um eine Tabelle zu erstellen.

```
private static void GetConnection() {  
    SAConnection _conn;  
    _conn = SAServerSideConnection.Connection;  
    SACommand cmd = _conn.CreateCommand();  
    cmd.CommandText = "CREATE TABLE Tab( c1 int, c2 char(128), c3 smallint, c4  
double, c5 numeric(30,6) )";  
    cmd.ExecuteNonQuery();  
    cmd.Dispose();  
}
```

SATcpOptionsBuilder-Klasse

Bietet eine einfache Möglichkeit zum Erstellen und Verwalten des Teils der TCP-Optionen von Verbindungszeichenfolgen, die vom SAConnection-Objekt verwendet werden.

Visual Basic-Syntax

```
Public NotInheritable Class SATcpOptionsBuilder  
    Inherits SAConnectionStringBuilderBase
```

C#-Syntax

```
public sealed class SATcpOptionsBuilder : SAConnectionStringBuilderBase
```

Basisklassen

- [SAConnectionStringBuilderBase-Klasse \[SQL Anywhere .NET\] auf Seite 207](#)

Mitglieder

Alle Mitglieder der SATcpOptionsBuilder-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SATcpOptionsBuilder-Konstruktor	Initialisiert ein SATcpOptionsBuilder-Objekt.
Add-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Fügt dem System.Data.Common.DbConnectionStringBuilder -Objekt einen Eintrag mit dem angegebenen Schlüssel und Wert hinzu.
AppendKeyValuePair-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Stellt eine effiziente und sichere Möglichkeit dar, einen Schlüssel und Wert an ein vorhandenes System.Text.StringBuilder -Objekt anzuhängen.
Clear-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Löscht den Inhalt der System.Data.Common.DbConnectionStringBuilder -Instanz.
ClearPropertyDescriptors-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Löscht die Sammlung von System.ComponentModel.PropertyDescriptor -Objekte auf dem verknüpften System.Data.Common.DbConnectionStringBuilder -Objekt.
ContainsKey-Methode	Ermittelt, ob das SAConnectionStringBuilder-Objekt ein spezifisches Schlüsselwort enthält.
EquivalentTo-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Vergleicht die Verbindungsinformationen in diesem System.Data.Common.DbConnectionStringBuilder -Objekt mit den Verbindungsinformationen im angegebenen Objekt
GetKeyword-Methode	Ruft das Schlüsselwort für die angegebene SAConnectionStringBuilder-Eigenschaft ab.
GetProperties-Methode (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Füllt ein angegebenes System.Collections.Hashtable -Objekt mit Informationen zu allen Eigenschaften dieses System.Data.Common.DbConnectionStringBuilder -Objekts.
GetUseLongNameAsKeyword-Methode	Ruft einen booleschen Wert ab, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
Remove-Methode	Löscht den Eintrag mit dem angegebenen Schlüssel aus der SAConnectionStringBuilder-Instanz.
SetUseLongNameAsKeyword-Methode	Legt einen booleschen Wert fest, der angibt, ob in der Verbindungszeichenfolge lange Verbindungsparameternamen verwendet werden.
ShouldSerialize-Methode	Gibt an, ob der angegebene Schlüssel in dieser SAConnectionStringBuilder-Instanz vorhanden ist.

Name	Beschreibung
ToString-Methode	Konvertiert das SATcpOptionsBuilder-Objekt in eine Zeichenfolgedarstellung.
TryGetValue-Methode	Ruft einen Wert ab, der dem bereitgestellten Schlüssel aus diesem SAConnectionStringBuilder-Objekt entspricht.
Broadcast-Eigenschaft	Ruft die Broadcast-Option ab bzw. legt sie fest.
BroadcastListener-Eigenschaft	Ruft die BroadcastListener-Option ab bzw. legt sie fest.
BrowsableConnectionString-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder.ConnectionString -Objekt in den Visual Studio-Designern sichtbar ist, bzw. legt ihn fest.
ClientPort-Eigenschaft	Ruft die ClientPort-Option ab bzw. legt sie fest.
ConnectionString-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft die Verbindungszeichenfolge ab, die dem DbConnectionStringBuilder -Objekt zugeordnet ist, bzw. legt sie fest.
Count-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft die aktuelle Anzahl von Schlüsseln ab, die in der System.Data.Common.DbConnectionStringBuilder.ConnectionString -Eigenschaft enthalten sind.
DoBroadcast-Eigenschaft	Ruft die DoBroadcast-Option ab bzw. legt sie fest.
Host-Eigenschaft	Ruft die Host-Option ab bzw. legt sie fest.
IPv6-Eigenschaft	Ruft die IPV6-Option ab bzw. legt sie fest.
IsFixedSize-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder -Objekt eine feste Größe hat.
IsReadOnly-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft einen Wert ab, der angibt, ob das System.Data.Common.DbConnectionStringBuilder -Objekt schreibgeschützt ist.
Keys-Eigenschaft	Ruft ein System.Collections.ICollection-Objekt ab, das die Schlüssel im SAConnectionStringBuilder-Objekt enthält.
LDAP-Eigenschaft	Ruft die LDAP-Option ab bzw. legt sie fest.
LocalOnly-Eigenschaft	Ruft die LocalOnly-Option ab bzw. legt sie fest.
MyIP-Eigenschaft	Ruft die MyIP-Option ab bzw. legt sie fest.

Name	Beschreibung
ReceiveBufferSize-Eigenschaft	Ruft die ReceiveBufferSize-Option ab bzw. legt sie fest.
SendBufferSize-Eigenschaft	Ruft die SendBufferSize-Option ab bzw. legt sie fest.
ServerPort-Eigenschaft	Ruft die ServerPort-Option ab bzw. legt sie fest.
TDS-Eigenschaft	Ruft die TDS-Option ab bzw. legt sie fest.
this-Eigenschaft	Ruft den Wert des Verbindungsschlüsselworts ab bzw. legt ihn fest.
Timeout-Eigenschaft	Ruft die Timeout-Option ab bzw. legt sie fest.
Values-Eigenschaft (geerbt aus System.Data.Common.DbConnectionStringBuilder)	Ruft ein System.Collections.ICollection -Objekt ab, das die Werte aus dem System.Data.Common.DbConnectionStringBuilder -Objekt enthält.
VerifyServerName-Eigenschaft	Ruft die VerifyServerName-Option ab bzw. legt sie fest.

Bemerkungen

Die Klasse SATcpOptionsBuilder ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\]](#) auf Seite 168

SATcpOptionsBuilder-Konstruktor

Initialisiert ein SATcpOptionsBuilder-Objekt.

Überladungsliste

Name	Beschreibung
SATcpOptionsBuilder()-Konstruktor	Initialisiert ein SATcpOptionsBuilder-Objekt.
SATcpOptionsBuilder(String)-Konstruktor	Initialisiert ein SATcpOptionsBuilder-Objekt.

SATcpOptionsBuilder()-Konstruktor

Initialisiert ein SATcpOptionsBuilder-Objekt.

Visual Basic-Syntax

```
Public Sub New()
```

C#-Syntax

```
public SATcpOptionsBuilder()
```

Bemerkungen

Die Klasse SATcpOptionsBuilder ist in .NET Compact Framework 2.0 nicht verfügbar.

Beispiel

Die nachstehende Anweisung initialisiert ein SATcpOptionsBuilder-Objekt.

```
SATcpOptionsBuilder options = new SATcpOptionsBuilder( );
```

SATcpOptionsBuilder(String)-Konstruktor

Initialisiert ein SATcpOptionsBuilder-Objekt.

Visual Basic-Syntax

```
Public Sub New(ByVal options As String)
```

C#-Syntax

```
public SATcpOptionsBuilder(string options)
```

Parameter

- **options** Eine Zeichenfolge von SQL Anywhere-TCP-Verbindungsparametern. Eine Liste der Verbindungsparameter finden Sie unter „[Verbindungsparameter](#)“ [*SQL Anywhere Server - Datenbankadministration*].

Bemerkungen

Die Klasse SATcpOptionsBuilder ist in .NET Compact Framework 2.0 nicht verfügbar.

Beispiel

Die nachstehende Anweisung initialisiert ein SATcpOptionsBuilder-Objekt.

```
SATcpOptionsBuilder options = new  
SATcpOptionsBuilder("localonly=yes;port=6873");
```

ToString-Methode

Konvertiert das SATcpOptionsBuilder-Objekt in eine Zeichenfolgedarstellung.

Visual Basic-Syntax

```
Public Overrides Function ToString() As String
```

C#-Syntax

```
public override string ToString()
```

Rückgabe

Die erstellte Optionszeichenfolge.

Broadcast-Eigenschaft

Ruft die Broadcast-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Broadcast As String
```

C#-Syntax

```
public string Broadcast {get;set;}
```

BroadcastListener-Eigenschaft

Ruft die BroadcastListener-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property BroadcastListener As String
```

C#-Syntax

```
public string BroadcastListener {get;set;}
```

ClientPort-Eigenschaft

Ruft die ClientPort-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ClientPort As String
```

C#-Syntax

```
public string ClientPort {get;set;}
```

DoBroadcast-Eigenschaft

Ruft die DoBroadcast-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property DoBroadcast As String
```

C#-Syntax

```
public string DoBroadcast {get;set;}
```

Host-Eigenschaft

Ruft die Host-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Host As String
```

C#-Syntax

```
public string Host {get;set;}
```

IPV6-Eigenschaft

Ruft die IPV6-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property IPV6 As String
```

C#-Syntax

```
public string IPV6 {get;set;}
```

LDAP-Eigenschaft

Ruft die LDAP-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property LDAP As String
```

C#-Syntax

```
public string LDAP {get;set;}
```

LocalOnly-Eigenschaft

Ruft die LocalOnly-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property LocalOnly As String
```

C#-Syntax

```
public string LocalOnly {get;set;}
```

MyIP-Eigenschaft

Ruft die MyIP-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property MyIP As String
```

C#-Syntax

```
public string MyIP {get;set;}
```

ReceiveBufferSize-Eigenschaft

Ruft die ReceiveBufferSize-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ReceiveBufferSize As Integer
```

C#-Syntax

```
public int ReceiveBufferSize {get;set;}
```

SendBufferSize-Eigenschaft

Ruft die SendBufferSize-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property SendBufferSize As Integer
```

C#-Syntax

```
public int SendBufferSize {get;set;}
```

ServerPort-Eigenschaft

Ruft die ServerPort-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property ServerPort As String
```

C#-Syntax

```
public string ServerPort {get;set;}
```

TDS-Eigenschaft

Ruft die TDS-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property TDS As String
```

C#-Syntax

```
public string TDS {get;set;}
```

Timeout-Eigenschaft

Ruft die Timeout-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property Timeout As Integer
```

C#-Syntax

```
public int Timeout {get;set;}
```

VerifyServerName-Eigenschaft

Ruft die VerifyServerName-Option ab bzw. legt sie fest.

Visual Basic-Syntax

```
Public Property VerifyServerName As String
```

C#-Syntax

```
public string VerifyServerName {get;set;}
```

SATransaction-Klasse

Repräsentiert eine SQL-Transaktion.

Visual Basic-Syntax

```
Public NotInheritable Class SATransaction
    Inherits System.Data.Common.DbTransaction
```

C#-Syntax

```
public sealed class SATransaction : System.Data.Common.DbTransaction
```

Basisklassen

- [System.Data.Common.DbTransaction](#)

Mitglieder

Alle Mitglieder der SATransaction-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Commit-Methode	Schreibt die Datenbanktransaktion fest.

Name	Beschreibung
Dispose-Methode (geerbt aus System.Data.Common.DbTransaction)	Gibt die nicht verwalteten Ressourcen frei, die von System.Data.Common.DbTransaction verwendet wurden
Rollback-Methode	Setzt eine Transaktion aus dem Status "Pending" zurück.
Save-Methode	Erstellt einen Savepoint in der Transaktion, der benutzt werden kann, um einen Teil der Transaktion zurückzusetzen, und definiert den Savepoint-Namen.
Connection-Eigenschaft	Das SAConnection-Objekt, das mit der Transaktion verbunden ist, oder eine Nullreferenz ("Nothing" in Visual Basic), wenn die Transaktion nicht mehr gültig ist.
IsolationLevel-Eigenschaft	Gibt die Isolationsstufe für diese Transaktion an.
SAIsolationLevel-Eigenschaft	Gibt die erweiterte Isolationsstufe für diese Transaktion an.

Bemerkungen

Es gibt keinen Konstruktor für SATransaction. Um ein SATransaction-Objekt zu erhalten, verwenden Sie eine der BeginTransaction-Methoden. Um einen Befehl mit einer Transaction zu verbinden, benutzen Sie die SACommand.Transaction-Eigenschaft.

Weitere Hinweise finden Sie unter „[Transaktionsverarbeitung](#)“ auf Seite 64 und „[SACommand: Zeilen mit ExecuteNonQuery einfügen, löschen und aktualisieren](#)“ auf Seite 50.

Siehe auch

- [SAConnection.BeginTransaction-Methode \[SQL Anywhere .NET\]](#) auf Seite 171
- [SACommand.Transaction-Eigenschaft \[SQL Anywhere .NET\]](#) auf Seite 151

Commit-Methode

Schreibt die Datenbanktransaktion fest.

Visual Basic-Syntax

```
Public Overrides Sub Commit()
```

C#-Syntax

```
public override void Commit()
```

Rollback-Methode

Setzt eine Transaktion aus dem Status "Pending" zurück.

Überladungsliste

Name	Beschreibung
Rollback()-Methode	Setzt eine Transaktion aus dem Status "Pending" zurück.
Rollback(String)-Methode	Setzt eine Transaktion aus dem Status "Pending" zurück.

Rollback()-Methode

Setzt eine Transaktion aus dem Status "Pending" zurück.

Visual Basic-Syntax

```
Public Overrides Sub Rollback()
```

C#-Syntax

```
public override void Rollback()
```

Bemerkungen

Die Transaktion kann nur aus dem Status "Pending" zurückgesetzt werden (nachdem BeginTransaction aufgerufen wurde, aber vor dem Aufruf von Commit).

Rollback(String)-Methode

Setzt eine Transaktion aus dem Status "Pending" zurück.

Visual Basic-Syntax

```
Public Sub Rollback(ByVal savePoint As String)
```

C#-Syntax

```
public void Rollback(string savePoint)
```

Parameter

- **savePoint** Der Name des Savepoints, zu dem zurückgesetzt werden soll.

Bemerkungen

Die Transaktion kann nur aus dem Status "Pending" zurückgesetzt werden (nachdem BeginTransaction aufgerufen wurde, aber vor dem Aufruf von Commit).

Save-Methode

Erstellt einen Savepoint in der Transaktion, der benutzt werden kann, um einen Teil der Transaktion zurückzusetzen, und definiert den Savepoint-Namen.

Visual Basic-Syntax

```
Public Sub Save(ByVal savePoint As String)
```

C#-Syntax

```
public void Save(string savePoint)
```

Parameter

- **savePoint** Der Name des Savepoints, bis zu dem zurückgesetzt werden soll.

Connection-Eigenschaft

Das SAConnection-Objekt, das mit der Transaktion verbunden ist, oder eine Nullreferenz ("Nothing" in Visual Basic), wenn die Transaktion nicht mehr gültig ist.

Visual Basic-Syntax

```
Public ReadOnly Shadows Property Connection As SAConnection
```

C#-Syntax

```
public new SAConnection Connection {get;}
```

Bemerkungen

Eine einzelne Anwendung kann mehrere Datenbankverbindungen aufweisen, über die jeweils null oder mehr Transaktionen durchgeführt werden. Mit dieser Eigenschaft können Sie das Verbindungsobjekt ermitteln, das mit einer bestimmten Transaktion verbunden ist, die von BeginTransaction erstellt wurde.

IsolationLevel-Eigenschaft

Gibt die Isolationsstufe für diese Transaktion an.

Visual Basic-Syntax

```
Public ReadOnly Overrides Property IsolationLevel As  
System.Data.IsolationLevel
```

C#-Syntax

```
public override System.Data.IsolationLevel IsolationLevel {get;}
```

Bemerkungen

Die IsolationLevel-Eigenschaft für diese Transaktion. Dies kann eine der folgenden sein:

- Unspecified
- Chaos
- ReadUncommitted
- ReadCommitted

- RepeatableRead
- Serialisierbar
- Snapshot

Standardwert ist ReadCommitted.

SAIsolationLevel-Eigenschaft

Gibt die erweiterte Isolationsstufe für diese Transaktion an.

Visual Basic-Syntax

```
Public ReadOnly Property SAIsolationLevel As SAIsolationLevel
```

C#-Syntax

```
public SAIsolationLevel SAIsolationLevel {get;}
```

Bemerkungen

Die SAIsolationLevel-Eigenschaft für diese Transaktion. Dies kann einer der folgenden Werte sein:

- Unspecified
- Chaos
- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serialisierbar
- Snapshot
- StatementSnapshot
- ReadOnlySnapshot

Standardwert ist ReadCommitted.

Parallele Transaktionen werden nicht unterstützt. Daher gilt die SAIsolationLevel-Eigenschaft für die gesamte Transaktion.

SAInfoMessageEventHandler-Delegat

Repräsentiert die Methode, die das SAConnection.InfoMessage-Ereignis eines SAConnection-Objekts verarbeitet.

Visual Basic-Syntax

```
Public Delegate Sub SAInfoMessageEventHandler(  
    ByVal obj As Object,  
    ByVal args As SAInfoMessageEventArgs  
)
```

C#-Syntax

```
public delegate void SAInfoMessageEventHandler(  
    object obj,  
    SAInfoMessageEventArgs args  
);
```

Siehe auch

- [SAConnection-Klasse \[SQL Anywhere .NET\] auf Seite 168](#)
- [SAConnection.InfoMessage-Ereignis \[SQL Anywhere .NET\] auf Seite 189](#)

SARowUpdatedEventHandler-Delegat

Repräsentiert die Methode, die das RowUpdated-Ereignis in einem SDataAdapter-Objekt verarbeitet.

Visual Basic-Syntax

```
Public Delegate Sub SARowUpdatedEventHandler(  
    ByVal sender As Object,  
    ByVal e As SARowUpdatedEventArgs  
)
```

C#-Syntax

```
public delegate void SARowUpdatedEventHandler(  
    object sender,  
    SARowUpdatedEventArgs e  
);
```

SARowUpdatingEventHandler-Delegat

Repräsentiert die Methode, die das RowUpdating-Ereignis in einem SDataAdapter-Objekt verarbeitet.

Visual Basic-Syntax

```
Public Delegate Sub SARowUpdatingEventHandler(  
    ByVal sender As Object,  
    ByVal e As SARowUpdatingEventArgs  
)
```

C#-Syntax

```
public delegate void SARowUpdatingEventHandler(  
    object sender,  
    SARowUpdatingEventArgs e  
);
```

SARowsCopiedEventHandler-Delegat

Repräsentiert die Methode, die das SABulkCopy.SARowsCopied-Ereignis einer SABulkCopy-Klasse bearbeitet.

Visual Basic-Syntax

```
Public Delegate Sub SARowsCopiedEventHandler(  
    ByVal sender As Object,  
    ByVal rowsCopiedEventArgs As SARowsCopiedEventArgs  
)
```

C#-Syntax

```
public delegate void SARowsCopiedEventHandler(  
    object sender,  
    SARowsCopiedEventArgs rowsCopiedEventArgs  
);
```

Bemerkungen

Der Delegat SARowsCopiedEventHandler ist in .NET Compact Framework 2.0 nicht verfügbar.

Siehe auch

- [SABulkCopy-Klasse \[SQL Anywhere .NET\] auf Seite 105](#)

SABulkCopyOptions-Enumeration

Ein bitweises Flag, das mindestens eine Option festlegt, die mit einer Instanz von SABulkCopy verwendet werden soll.

Visual Basic-Syntax

```
Public Enum SABulkCopyOptions
```

C#-Syntax

```
public enum SABulkCopyOptions
```

Mitglieder

Mitgliedsname	Beschreibung	Wert
default	Wenn nur dieser Wert angegeben wird, wird das Standardverhalten angewendet. Standardmäßig sind Trigger aktiviert.	0x0
DoNotFireTriggers	Wenn diese Option festgelegt wird, werden keine Trigger ausgelöst. Zum Deaktivieren von Triggern ist die DBA-Berechtigung erforderlich. Trigger werden zu Beginn von WriteToServer für die Verbindung deaktiviert und der Wert wird am Ende der Methode wiederhergestellt.	0x1

Mitgliedsname	Beschreibung	Wert
KeepIdentity	Wenn diese Option angegeben wird, werden die Quellwerte, die in eine Identity-Spalte kopiert werden sollen, beibehalten. Standardmäßig werden in der Zieltabelle neue Identity-Werte generiert.	0x2
TableLock	Wenn diese Option angegeben wird, wird die Tabelle mit dem Befehl "LOCK TABLE Tabellename WITH HOLD IN SHARE MODE" gesperrt. Diese Sperre bleibt erhalten, bis die Verbindung geschlossen wird.	0x4
UseInternalTransaction	Wenn diese Option angegeben wird, wird jeder Batch des Masseneimport/-export-Vorgangs innerhalb einer Transaktion ausgeführt. Wenn die Option nicht angegeben wird, werden keine Transaktionen verwendet. Wenn Sie diese Option verwenden und für den Konstruktor auch ein SATransaction-Objekt bereitstellen, tritt eine System.ArgumentException-Ausnahmebedingung auf.	0x8

Bemerkungen

Die SABulkCopyOptions-Enumeration wird verwendet, wenn ein SABulkCopy-Objekt erstellt werden soll, um festzulegen, wie sich die WriteToServer-Methoden verhalten.

Die SABulkCopyOptions-Klasse ist in .NET Compact Framework 2.0 nicht verfügbar.

Die Optionen CheckConstraints und KeepNulls werden nicht unterstützt.

Siehe auch

- [SABulkCopy-Klasse \[SQL Anywhere .NET\] auf Seite 105](#)

SADbType-Enumeration

Listet die Datentypen der SQL Anywhere .NET-Datenbanken auf.

Visual Basic-Syntax

```
Public Enum SADbType
```

C#-Syntax

```
public enum SADbType
```

Mitglieder

Mitgliedsname	Beschreibung
BigInt	64-Bit-Ganzzahl mit Vorzeichen.
Binary	Binärdaten mit einer festgelegten maximalen Länge. Die Enumerationswerte Binary und VarBinary sind Aliase füreinander.
Bit	1-Bit-Parameter.
Char	Zeichendaten mit einer definierten Länge. Dieser Typ unterstützt immer Unicode-Zeichen. Die Typen Char und Var-Char sind voll kompatibel.
Datum	Datumsinformationen.
Datum/Uhrzeit	Zeitstempelinformationen ("date", "time"). Die Enumerationswerte DateTime und TimeStamp sind Aliase füreinander.
DateTimeOffset	Zeitstempelinformationen-Offset ("date", "time").
Decimal	Exakte numerische Daten mit festgelegter Gesamtstellenzahl und festgelegten Dezimalstellen. Die Enumerationswerte Decimal und Numeric sind Aliase füreinander.
Double	Doppeltgenaue Gleitkommazahl (8 Byte).
Float	Einfachgenaue Gleitkommazahl (4 Byte). Die Enumerationswerte Float und Real sind Aliase füreinander.
Image	Speichert Binärdaten von beliebiger Länge.
Integer	32-Bit-Ganzzahl ohne Vorzeichen.
LongBinary	Binärdaten mit variabler Länge.
LongNVarChar	Zeichendaten im NCHAR-Zeichensatz mit variabler Länge. Dieser Typ unterstützt immer Unicode-Zeichen.
LongVarbit	Bit-Arrays mit variabler Länge.
LongVarchar	Zeichendaten mit variabler Länge. Dieser Typ unterstützt immer Unicode-Zeichen.

Mitgliedsname	Beschreibung
Money	Währungsdaten
NChar	Speichert Unicode-Zeichendaten, bis zu 32767 Zeichen.
NText	Speichert Unicode-Zeichendaten von beliebiger Länge.
Numeric	Exakte numerische Daten mit festgelegter Gesamtstellenzahl und festgelegten Dezimalstellen. Die Enumerationswerte Decimal und Numeric sind Aliase füreinander.
NVarChar	Speichert Unicode-Zeichendaten, bis zu 32767 Zeichen.
Real	Einfachgenaue Gleitkommazahl (4 Byte). Die Enumerationswerte Float und Real sind Aliase füreinander.
SmallDateTime	Eine Domäne, die als TIMESTAMP implementiert ist.
SmallInt	16-Bit-Ganzzahl mit Vorzeichen.
SmallMoney	Speichert Währungsdaten, die kleiner als eine Million Währungseinheiten sind.
SysName	Speichert Zeichendaten von beliebiger Länge.
Text	Speichert Zeichendaten von beliebiger Länge.
Time	Zeitinformationen.
TimeStamp	Zeitstempelinformationen ("date", "time"). Die Enumerationswerte DateTime und TimeStamp sind Aliase füreinander.
TimeStampWithTime-Zone	Zeitstempelinformationen (Datum, Zeit, Zeitzone). Die Enumerationswerte DateTime und TimeStamp sind Aliase füreinander.
TinyInt	8-Bit-Ganzzahl ohne Vorzeichen.
UniqueIdentifier	Universally Unique Identifier (UUID/GUID)
UniqueIdentifierStr	Eine Domäne, die als CHAR(36) implementiert ist. UniqueIdentifierStr wird beim entfernten Datenzugriff verwendet, wenn UniqueIdentifier-Spalten von Microsoft SQL Server zugeordnet werden.
UnsignedBigInt	64-Bit-Ganzzahl ohne Vorzeichen.

Mitgliedsname	Beschreibung
UnsignedInt	32-Bit-Ganzzahl ohne Vorzeichen.
UnsignedSmallInt	16-Bit-Ganzzahl ohne Vorzeichen.
VarBinary	Binärdaten mit einer festgelegten maximalen Länge. Die Enumerationswerte Binary und VarBinary sind Aliase füreinander.
VarBit	Bit-Arrays mit einer Länge von 1 bis 32767 Bit.
VarChar	Zeichendaten mit einer festgelegten maximalen Länge. Dieser Typ unterstützt immer Unicode-Zeichen. Die Typen Char und VarChar sind voll kompatibel.
Xml	XML-Daten. Dieser Typ speichert Zeichendaten von beliebiger Länge und wird verwendet, um XML-Dokumente zu speichern.

Bemerkungen

Die folgende Tabelle listet auf, welche .NET-Typen mit den einzelnen SADBType-Typen kompatibel sind. Bei integralen Typen können Tabellenspalten immer mit kleineren Integer-Typen und auch größeren Typen festgelegt werden, sofern der tatsächliche Wert innerhalb des Bereichs des Typs liegt.

SADBType	Kompatibler .NET-Typ	Integrierter C#-Typ	Integrierter Visual Basic-Typ
BigInt	System.Int64	long	Long
Binary, VarBinary	System.Byte[] oder System.Guid, wenn die Größe 16 ist	byte[]	Byte()
Bit	System.Boolean	bool	Boolean
Char, VarChar	System.String	String	String
Date	System.DateTime	DateTime (kein integrierter Typ)	Date
DateTime, Time-Stamp	System.DateTime	DateTime (kein integrierter Typ)	Date Time
DateTimeOffset	System.DateTimeOffset	DateTimeOffset (kein integrierter Typ)	DateTimeOffset

SADbType	Kompatibler .NET-Typ	Integrierter C#-Typ	Integrierter Visual Basic-Typ
Decimal, Numeric	System.String	decimal	Decimal
Double	System.Double	double	Double
Float, Real	System.Single	float	Single
Image	System.Byte[]	byte[]	Byte()
Integer	System.Int32	int	Integer
LongBinary	System.Byte[]	byte[]	Byte()
LongNVarChar	System.String	String	String
LongVarChar	System.String	String	String
Money	System.String	decimal	Decimal
NChar	System.String	String	String
NText	System.String	String	String
Numeric	System.String	decimal	Decimal
NVarChar	System.String	String	String
SmallDateTime	System.DateTime	DateTime (kein integrierter Typ)	Date
SmallInt	System.Int16	short	Short
SmallMoney	System.String	decimal	Decimal
SysName	System.String	String	String
Text	System.String	String	String
Time	System.TimeSpan	TimeSpan (kein integrierter Typ)	TimeSpan (kein integrierter Typ)
TimeStamp	System.DateTime	DateTime (kein integrierter Typ)	Date
TimeStampWithTimeZone	System.DateTimeOffset	DateTimeOffset (kein integrierter Typ)	DateTimeOffset
TinyInt	System.Byte	byte	Byte

SADbType	Kompatibler .NET-Typ	Integrierter C#-Typ	Integrierter Visual Basic-Typ
UniqueIdentifier	System.Guid	Guid (kein integrierter Typ)	Guid (kein integrierter Typ)
UniqueIdentifierStr	System.String	String	String
UnsignedBigInt	System.UInt64	ulong	UInt64 (kein integrierter Typ)
UnsignedInt	System.UInt32	uint	UInt64 (kein integrierter Typ)
UnsignedSmallInt	System.UInt16	ushort	UInt64 (kein integrierter Typ)
Xml	System.Xml	String	String

Binärspalten der Länge 16 sind voll mit dem Typ UniqueIdentifier kompatibel.

Siehe auch

- [SADbDataReader.GetFieldType-Methode \[SQL Anywhere .NET\] auf Seite 235](#)
- [SADbDataReader.GetDataTypeName-Methode \[SQL Anywhere .NET\] auf Seite 232](#)

SAIsolationLevel-Enumeration

Gibt SQL Anywhere-Isolationsstufen an.

Visual Basic-Syntax

```
Public Enum SAIsolationLevel
```

C#-Syntax

```
public enum SAIsolationLevel
```

Mitglieder

Mitgliedsname	Beschreibung
Chaos	<p>Diese Isolationsstufe wird nicht unterstützt.</p> <p>Wenn Sie <code>SACConnection.BeginTransaction</code> mit dieser Isolationsstufe aufrufen, wird eine Ausnahmebedingung generiert.</p> <p>Weitere Hinweise finden Sie unter SACConnection.BeginTransaction-Methode [SQL Anywhere .NET] auf Seite 171.</p>

Mitgliedsname	Beschreibung
ReadCommitted	<p>Setzt das Verhalten auf ein Äquivalent der Isolationsstufe 1.</p> <p>Diese Isolationsstufe verhindert Dirty Reads, lässt jedoch nicht wiederholbare Lesevorgänge und Phantomzeilen zu.</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>
ReadUncommitted	<p>Setzt das Verhalten auf ein Äquivalent der Isolationsstufe 0.</p> <p>Diese Isolationsstufe lässt Dirty Reads, nicht wiederholbare Lesevorgänge und Phantomzeilen zu.</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>
RepeatableRead	<p>Setzt das Verhalten auf ein Äquivalent der Isolationsstufe 2.</p> <p>Diese Isolationsstufe verhindert Dirty Reads und garantiert wiederholbare Lesevorgänge. Sie lässt jedoch Phantomzeilen zu.</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>
Serializable	<p>Setzt das Verhalten auf ein Äquivalent der Isolationsstufe 3.</p> <p>Diese Isolationsstufe verhindert Dirty Reads, garantiert wiederholbare Lesevorgänge und verhindert Phantomzeilen.</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>
Snapshot	<p>Verwendet einen Snapshot von festgeschriebenen Daten ab dem Zeitpunkt, an dem die erste Zeile von einer Transaktion gelesen, eingefügt, aktualisiert oder gelöscht wird.</p> <p>Diese Isolationsstufe verwendet einen Snapshot festgeschriebener Daten von dem Zeitpunkt, zu dem die erste Zeile durch die Transaktion gelesen oder aktualisiert wird.</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>

Mitgliedsname	Beschreibung
Unspecified	<p>Diese Isolationsstufe wird nicht unterstützt.</p> <p>Wenn Sie <code>SACConnection.BeginTransaction</code> mit dieser Isolationsstufe aufrufen, wird eine Ausnahmebedingung generiert.</p> <p>Weitere Hinweise finden Sie unter SACConnection.BeginTransaction-Methode [SQL Anywhere .NET] auf Seite 171.</p>
ReadOnlySnapshot	<p>Verwendet bei schreibgeschützten Anweisungen einen Snapshot von festgeschriebenen Daten ab dem Zeitpunkt, an dem die erste Zeile aus der Datenbank gelesen wird.</p> <p>Nicht wiederholbare Zeilen und Phantomzeilen können innerhalb einer Transaktion, aber nicht in einer einzelnen Anweisung vorkommen. Bei aktualisierbaren Anweisungen wird die Isolationsstufe verwendet, die durch die Option <code>updatable_statement_isolation</code> angegeben ist (kann 0 (Standardwert), 1, 2 oder 3 sein).</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>
StatementSnapshot	<p>Benutzt einen Snapshot der festgeschriebenen Daten von dem Zeitpunkt, zu dem die erste Zeile von der Anweisung gelesen wird.</p> <p>Jede Anweisung innerhalb der Transaktion sieht einen Snapshot von Daten von einem anderen Zeitpunkt.</p> <p>Verwendet bei jeder Anweisung einen Snapshot von festgeschriebenen Daten ab dem Zeitpunkt, an dem die erste Zeile aus der Datenbank gelesen wird. Nicht wiederholbare Zeilen und Phantomzeilen können innerhalb einer Transaktion, aber nicht in einer einzelnen Anweisung vorkommen.</p> <p>Weitere Hinweise finden Sie unter „Isolationsstufen und Konsistenz“ [SQL Anywhere Server - SQL-Benutzerhandbuch].</p>

Bemerkungen

Diese Klasse ergänzt die Klasse `System.Data.IsolationLevel`.

Der SQL Anywhere .NET-Datenprovider unterstützt alle SQL Anywhere-Isolationsstufen, einschließlich der Snapshot-Isolationsstufen. Um die Snapshot-Isolation zu verwenden, geben Sie `SAIsolationLevel.Snapshot`, `SAIsolationLevel.ReadOnlySnapshot` oder `SAIsolationLevel.StatementSnapshot` als Parameter für `BeginTransaction` an. `BeginTransaction` wurde überladen, sodass es entweder `IsolationLevel` oder `SAIsolationLevel` annehmen kann. Die Werte in den beiden Enumerationen sind gleich, abgesehen von `ReadOnlySnapshot` und `StatementSnapshot`, die nur in der Stufe `SAIsolationLevel` vorhanden sein können. In `SATransaction` gibt es die neue Eigenschaft `SAIsolationLevel`, die die `SAIsolationLevel`-Isolationsstufe abfragt.

Weitere Hinweise finden Sie unter „Snapshot-Isolation“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)].

SAMessageType-Enumeration

Identifiziert den Typ der Meldung.

Visual Basic-Syntax

```
Public Enum SAMessageType
```

C#-Syntax

```
public enum SAMessageType
```

Mitglieder

Mitgliedsname	Beschreibung	Wert
Action	Meldung vom Typ ACTION	2
Info	Meldung vom Typ INFO	0
Status	Meldung vom Typ STATUS	3
Warning	Meldung vom Typ WARNING	1

Bemerkungen

Dies kann einer der folgenden sein: Aktion, Info, Status oder Warnung.

OLE DB- und ADO-Entwicklung

SQL Anywhere umfasst einen OLE DB-Provider für OLE DB und ADO.

OLE DB ist eine Gruppe von Component Object Model-Schnittstellen (COM), die von Microsoft entwickelt wurden und Anwendungen einen gleichförmigen Zugriff auf Daten ermöglichen, die in unterschiedlichen Informationsquellen gespeichert sind. Sie bieten darüber hinaus die Möglichkeit, zusätzliche Datenbankdienste zu implementieren. Diese Schnittstellen unterstützen so viele DBMS-Funktionen, wie sie für die gemeinsame Nutzung des Datenspeichers erforderlich sind.

ADO ist ein Objektmodell, das es ermöglicht, über Programme und OLE DB-Systemschnittstellen auf eine große Anzahl von Datenquellen zuzugreifen, diese zu ändern und zu aktualisieren. ADO wurde ebenfalls von Microsoft entwickelt. Die meisten Entwickler, die die OLE DB-Programmierschnittstelle verwenden, tun dies, indem sie die ADO API einsetzen und nicht direkt die OLE DB API.

Verwechseln Sie die ADO-Schnittstelle nicht mit ADO.NET. ADO.NET ist eine eigene Schnittstelle.

In der Dokumentation zum Microsoft Developer Network finden Sie Hinweise zur OLE DB- und ADO-Programmierung. SQL Anywhere-spezifische Hinweise zur OLE DB- und ADO-Entwicklung finden Sie in diesem Dokument.

Siehe auch

- „[SQL Anywhere .NET-Unterstützung](#)“ auf Seite 41

OLE DB

OLE DB ist ein Datenzugriffsmodell von Microsoft. Es verwendet COM-Schnittstellen (Component Object Model). Anders als bei ODBC wird hier nicht vorausgesetzt, dass die Datenquelle einen SQL-Abfrageprozessor verwendet.

SQL Anywhere umfasst einen **OLE DB-Provider** namens **SAOLEDB**. Dieser Provider ist für aktuelle Windows-Plattformen verfügbar. Der Provider steht nicht für Windows Mobile-Plattformen zur Verfügung.

Sie können auf SQL Anywhere auch über den Microsoft OLE DB-Provider für ODBC (MSDASQL) zusammen mit dem ODBC-Treiber von SQL Anywhere zugreifen.

Die Verwendung des SQL Anywhere OLE DB-Provider bringt eine Reihe von Vorteilen:

- Einige Funktionen, wie z.B. die Aktualisierung über einen Cursor, sind bei Verwendung der OLE DB/ODBC Bridge nicht verfügbar.
- Wenn Sie den SQL Anywhere OLE DB-Provider verwenden, ist ODBC für Ihr Deployment nicht erforderlich.
- Mit MSDASQL können OLE DB-Clients mit jedem ODBC-Treiber arbeiten, wobei jedoch nicht garantiert wird, dass Sie sämtliche Funktionen jedes ODBC-Treibers nutzen können. Mit dem SQL

Anywhere-Provider können Sie in OLE DB-Programmierungsumgebungen vollen Zugriff auf die SQL Anywhere-Funktionen erhalten.

Unterstützte Plattformen

Der SQL Anywhere OLE DB-Provider dient für die Arbeit mit Microsoft Data Access Components (MDAC) 2.8 und späteren Versionen.

Eine Liste der unterstützten Plattformen finden Sie unter <http://www.sybase.com/detail?id=1061806>.

Verteilte Transaktionen in OLE DB

Der OLE DB-Treiber kann in einer Umgebung mit verteilten Transaktionen als Ressourcen-Manager eingesetzt werden.

Siehe auch

- „Dreischichtige Datenverarbeitung und verteilte Transaktionen“ auf Seite 865

ADO-Programmierung mit SQL Anywhere

Bei ADO (ActiveX-Datenobjekte) handelt es sich um ein Datenzugriffsmodell, das über eine Automationsschnittstelle zugänglich wird, die es Clientanwendungen ermöglicht, die Methoden und Eigenschaften von Objekten in Laufzeit zu erfassen, ohne das Objekt vorher zu kennen. Mithilfe von Automation können Skriptsprachen wie Visual Basic ein Standardmodell für Datenzugriffsobjekte verwenden. ADO verwendet OLE DB, um den Datenzugriff bereitzustellen.

Mit dem SQL Anywhere OLE DB-Provider erhalten Sie in einer ADO-Programmierungsumgebung vollen Zugriff auf SQL Anywhere-Funktionen.

In diesem Abschnitt wird beschrieben, wie grundlegende Aufgaben während der Verwendung von ADO aus Visual Basic ausgeführt werden. Er stellt keine vollständige Anleitung zur Programmierung mit ADO dar.

Codebeispiele für diesen Abschnitt finden Sie in der Projektdatei `%SQLANY%SAMP16%\SQLAnywhere\VB sampler\vbsampler.sln`.

Weitere Hinweise zur Programmierung in ADO finden Sie im Handbuch Ihres Entwicklungstools.

Verbindung mit einer Datenbank über das Connection-Objekt herstellen

In diesem Abschnitt wird eine einfache Visual Basic-Routine beschrieben, die eine Verbindung zu einer Datenbank herstellt.

Beispielcode

Sie können diese Routine ausprobieren, indem Sie eine Befehlsschaltfläche namens cmdTestConnection in einer Maske platzieren und die Routine in ihr Klick-Ereignis einfügen. Führen Sie das Programm aus und klicken Sie auf die Schaltfläche, um eine Verbindung herzustellen und trennen Sie die Verbindung dann.

```
Private Sub cmdTestConnection_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdTestConnection.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer

    On Error GoTo HandleError

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=SQL Anywhere 16 Demo"
    myConn.Open()
    MsgBox("Connection succeeded")
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

Hinweise

Das Beispiel führt die folgenden Aufgaben aus:

- Es deklariert die Variablen, die in der Routine verwendet werden.
- Es stellt mithilfe des OLE DB-Providers von SQL Anywhere OLE DB eine Verbindung zur Beispieldatenbank her.
- Es verwendet ein Befehlsobjekt zum Ausführen einer einfachen Anweisung, die eine Meldung im Meldungsfenster des Datenbankservers anzeigt.
- Es beendet die Verbindung.

Siehe auch

- „OLE DB-Verbindungsparameter“ auf Seite 345

Anweisungen mit dem Command-Objekt ausführen

In diesem Abschnitt wird eine einfache Routine beschrieben, die eine einfache SQL-Anweisung an die Datenbank sendet.

Beispielcode

Sie können diese Routine ausprobieren, indem Sie eine Befehlsschaltfläche namens cmdUpdate in einer Maske platzieren und die Routine in ihr Klick-Ereignis einfügen. Führen Sie das Programm aus und klicken Sie auf die Schaltfläche, um eine Verbindung herzustellen und eine Meldung im Meldungsfenster des Datenbankservers anzuzeigen. Trennen Sie anschließend die Verbindung.

```
Private Sub cmdUpdate_Click( _  
    ByVal eventSender As System.Object, _  
    ByVal eventArgs As System.EventArgs) _  
    Handles cmdUpdate.Click  
  
    ' Declare variables  
    Dim myConn As New ADODB.Connection  
    Dim myCommand As New ADODB.Command  
    Dim cAffected As Integer  
  
    On Error GoTo HandleError  
  
    ' Establish the connection  
    myConn.Provider = "SAOLEDB"  
    myConn.ConnectionString = _  
        "Data Source=SQL Anywhere 16 Demo"  
    myConn.Open()  
  
    'Execute a command  
    myCommand.CommandText = _  
        "UPDATE Customers SET GivenName='Liz' WHERE ID=102"  
    myCommand.ActiveConnection = myConn  
    myCommand.Execute(cAffected)  
    MsgBox(CStr(cAffected) & " rows affected.", _  
        MsgBoxStyle.Information)  
  
    myConn.Close()  
    Exit Sub  
  
HandleError:  
    MsgBox(ErrorToString(Err.Number))  
    Exit Sub  
End Sub
```

Hinweise

Nachdem eine Verbindung hergestellt wurde, erstellt der Beispielcode ein Befehlsobjekt und stellt die Eigenschaft CommandText auf eine Update-Anweisung und die Eigenschaft ActiveConnection auf die aktuelle Verbindung ein. Dann führt er die Update-Anweisung aus und zeigt die Anzahl der von der Aktualisierung betroffenen Zeilen in einem Fenster an.

In diesem Beispiel wird die Aktualisierung an die Datenbank gesendet und festgeschrieben, wenn sie ausgeführt wird.

Sie können auch Aktualisierungen über einen Cursor ausführen.

Siehe auch

- [„ADO-Transaktionen“ auf Seite 344](#)
- [„Zeilenaktualisierungen durch einen Cursor mit dem Recordset-Objekt“ auf Seite 343](#)

Ergebnismengen mit dem Recordset-Objekt abrufen

Das ADO-Recordset-Objekt stellt die Ergebnismenge einer Abfrage dar. Sie können es benutzen, um Daten aus einer Datenbank anzuzeigen.

Beispielcode

Sie können diese Routine ausprobieren, indem Sie eine Befehlsschaltfläche namens cmdQuery in einer Maske platzieren und die Routine in ihr Klick-Ereignis einfügen. Führen Sie das Programm aus und klicken Sie auf die Schaltfläche, um eine Verbindung herzustellen, eine Meldung im Meldungsfenster des Datenbankservers anzuzeigen, eine Abfrage auszuführen und die ersten Zeilen in Fenstern anzuzeigen. Trennen Sie anschließend die Verbindung.

```
Private Sub cmdQuery_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdQuery.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim myRS As New ADODB.Recordset

    On Error GoTo ErrorHandler

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=SQL Anywhere 16 Demo"
    myConn.CursorLocation = _
        ADODB.CursorLocationEnum.adUseServer
    myConn.Mode = _
        ADODB.ConnectModeEnum.adModeReadWrite
    myConn.IsolationLevel = _
        ADODB.IsolationLevelEnum.adXactCursorStability
    myConn.Open()

    'Execute a query
    myRS = New ADODB.Recordset
    myRS.CacheSize = 50
    myRS let_Source("SELECT * FROM Customers")
    myRS let_ActiveConnection(myConn)
    myRS.CursorType = ADODB.CursorTypeEnum.adOpenKeyset
    myRS.LockType = ADODB.LockTypeEnum.adLockOptimistic
    myRS.Open()

    'Scroll through the first few results
    myRS.MoveFirst()
    For i = 1 To 5
        MsgBox(myRS.Fields("CompanyName").Value, _
            MsgBoxStyle.Information)
        myRS.MoveNext()
    Next

    myRS.Close()
    myConn.Close()
    Exit Sub

ErrorHandler:
    MsgBox(ErrorToString(Err.Number))
```

```
Exit Sub  
End Sub
```

Hinweise

Das Recordset-Objekt in diesem Beispiel enthält die Ergebnisse aus einer Abfrage der Tabelle Customers. Die For-Schleife durchläuft die ersten Zeilen und zeigt für jede Zeile den Wert CompanyName an.

Hierbei handelt es sich um ein einfaches Beispiel für die Verwendung eines Cursors aus ADO.

Weiterführende Beispiele für die Verwendung eines Cursors aus ADO finden Sie unter [„Das Recordset-Objekt“ auf Seite 342](#).

Das Recordset-Objekt

Bei der Arbeit mit SQL Anywhere repräsentiert das ADO Recordset einen Cursor. Sie können den Cursortyp auswählen, indem Sie eine CursorType-Eigenschaft des Recordset-Objekts deklarieren, bevor Sie das Recordset öffnen. Die Auswahl des Cursortyps beeinflusst die Aktionen, die Sie am Recordset vornehmen können und hat Auswirkungen auf die Performance.

Cursortypen

ADO hat seine eigene Namenskonvention für Cursortypen. Die von SQL Anywhere unterstützten Cursortypen werden in [„Cursoreigenschaften“ auf Seite 15](#) beschrieben.

Die verfügbaren Cursortypen, die entsprechenden Cursortypenkonstanten und die SQL Anywhere-Typen, denen sie entsprechen, sind hier aufgelistet:

ADO-Cursortyp	ADO-Konstante	SQL Anywhere-Datentyp
Dynamischer Cursor	adOpenDynamic	Dynamisch abrollender Cursor
Keyset-gesteuerter Cursor	adOpenKeyset	Abrollender Cursor
Statischer Cursor	adOpenStatic	Insensitiver Cursor
Vorwärtscursor	adOpenForwardOnly	Nicht-abrollender Cursor

Beispielcode

Der folgende Code legt den Cursortyp für ein ADO-Recordset-Objekt fest:

```
Dim myRS As New ADODB.Recordset  
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic
```

Siehe auch

- [„Cursortypen“ auf Seite 15](#)

Zeilenaktualisierungen durch einen Cursor mit dem Recordset-Objekt

Mit dem SQL Anywhere OLE DB-Provider können Sie eine Ergebnismenge über einen Cursor aktualisieren. Diese Funktion ist über den MSDASQL-Provider nicht verfügbar.

Recordsets aktualisieren

Sie können die Datenbank über eine Datensatzgruppe (Recordset) aktualisieren.

```
Private Sub cmdUpdateThroughCursor_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdUpdateThroughCursor.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim SQLString As String

    On Error GoTo HandleError

    ' Connect
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=SQL Anywhere 16 Demo"
    myConn.Open()
    myConn.BeginTrans()
    SQLString = "SELECT * FROM Customers"
    myRS.Open(SQLString, myConn, _
        ADODB.CursorTypeEnum.adOpenDynamic, _
        ADODB.LockTypeEnum.adLockBatchOptimistic)

    If myRS.BOF And myRS.EOF Then
        MsgBox("Recordset is empty!", 16, "Empty Recordset")
    Else
        MsgBox("Cursor type: " & CStr(myRS.CursorType), _
            MsgBoxStyle.Information)
        myRS.MoveFirst()
        For i = 1 To 3
            MsgBox("Row: " & CStr(myRS.Fields("ID").Value), _
                MsgBoxStyle.Information)
            If i = 2 Then
                myRS.Update("City", "Toronto")
                myRS.UpdateBatch()
            End If
            myRS.MoveNext()
        Next i
        myRS.Close()
    End If
    myConn.CommitTrans()
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub

End Sub
```

Hinweise

Wenn Sie die Einstellung `adLockBatchOptimistic` für das Recordset verwenden, werden bei der `myRS.Update`-Methode keine Änderungen an der Datenbank selbst vorgenommen. stattdessen wird eine lokale Kopie des Recordset aktualisiert.

Die `myRS.UpdateBatch`-Methode nimmt die Aktualisierung am Datenbankserver vor, schreibt sie jedoch nicht fest, da sie sich innerhalb einer Transaktion befindet. Wenn eine `UpdateBatch`-Methode von außerhalb einer Transaktion aufgerufen wurde, werden die Änderungen festgeschrieben.

Die `myConn.CommitTrans`-Methode schreibt die Änderungen fest. Das Recordset-Objekt wurde zwischenzeitlich geschlossen, sodass das Problem, ob die lokale Kopie der Daten geändert wurde oder nicht, nicht mehr besteht.

ADO-Transaktionen

Standardmäßig werden sämtliche Änderungen, die Sie mit ADO an der Datenbank vornehmen, festgeschrieben, wenn sie ausgeführt werden. Hierzu gehören auch explizite Aktualisierungen und die `UpdateBatch`-Methode für ein Recordset. Im vorherigen Abschnitt wurde jedoch gezeigt, dass Sie die Methoden `BeginTrans` und `RollbackTrans` oder `CommitTrans` auf das Connection-Objekt anwenden können, um Transaktionen zu benutzen.

Die Transaktions-Isolationsstufe wird als eine Eigenschaft des Verbindungsobjekts festgelegt. Die Eigenschaft `IsolationLevel` kann einen der folgenden Werte annehmen:

ADO-Isolationsstufe	Konstante	SQL Anywhere-Ebene
Unspecified	<code>adXactUnspecified</code>	Nicht anwendbar. Auf 0 setzen.
Chaos	<code>adXactChaos</code>	Nicht unterstützt. Auf 0 setzen.
Durchsuchen	<code>adXactBrowse</code>	0
Nicht festgeschriebene Daten lesen	<code>adXactReadUncommitted</code>	0
Cursorstabilität (Cursor stability)	<code>adXactCursorStability</code>	1
Festgeschriebene Daten lesen	<code>adXactReadCommitted</code>	1
Wiederholbare Lesevorgänge	<code>adXactRepeatableRead</code>	2
Isoliert (Isolated)	<code>adXactIsolated</code>	3
Serialisierbar	<code>adXactSerializable</code>	3
Snapshot	2097152	4

ADO-Isolationsstufe	Konstante	SQL Anywhere-Ebene
Anweisungs-Snapshot (statement-snapshot)	4194304	5
Schreibgeschützter Anweisungs-Snapshot (readonly-statement-snapshot)	8388608	6

Siehe auch

- „Isolationsstufen und Konsistenz“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*]

OLE DB-Verbindungsparameter

OLE DB-Verbindungsparameter werden von Microsoft definiert. Der SQL Anywhere OLE DB-Provider unterstützt eine Teilmenge dieser Verbindungsparameter. Eine typische Verbindungszeichenfolge sieht wie folgt aus:

```
"Provider=SAOLEDB;Data Source=myDsn;Initial Catalog=myDbn;
  User ID=myUid;Password=myPwd"
```

Im Folgenden finden Sie die vom Provider unterstützten OLE DB-Verbindungsparameter. In einigen Fällen sind OLE DB-Verbindungsparameter mit SQL Anywhere-Verbindungsparametern identisch (z.B. "Password") oder ähneln diesen (z.B. "User ID"). Beachten Sie die Verwendung von Leerzeichen in vielen dieser Verbindungsparameter.

- **Provider** Dieser Parameter wird verwendet, um den SQL Anywhere OLE DB-Provider (SAOLEDB) zu identifizieren.
- **User ID** Dieser Verbindungsparameter wird direkt dem SQL Anywhere-Verbindungsparameter UserID (UID) zugeordnet. Beispiel: User ID=DBA.
- **Password** Dieser Verbindungsparameter wird direkt dem SQL Anywhere-Verbindungsparameter Password (PWD) zugeordnet. Beispiel: Password=sql.
- **Data Source** Dieser Verbindungsparameter wird direkt dem SQL Anywhere-Verbindungsparameter DataSourceName (DSN) zugeordnet. Beispiel: Data Source=SQL Anywhere 16 Demo.
- **Initial Catalog** Dieser Verbindungsparameter wird direkt dem SQL Anywhere-Verbindungsparameter DatabaseName (DBN) zugeordnet. Beispiel: Initial Catalog=demo.
- **Location** Dieser Verbindungsparameter wird direkt dem SQL Anywhere-Verbindungsparameter "Host" zugeordnet. Der Parameterwert hat dieselbe Form wie der Host-Parameterwert. Beispiel: Location=localhost:4444.

- **Extended Properties** Dieser Verbindungsparameter wird von OLE DB verwendet, um alle SQL Anywhere-spezifischen Verbindungsparameter zu übergeben. Beispiel: `Extended Properties="UserID=DBA;DBKEY=V3moj3952B;DBF=demo.db"`.

ADO verwendet diese Verbindungsparameter, um alle Verbindungsparameter zu sammeln und zu übergeben, die es nicht erkennt.

Einige Microsoft-Verbindungsfenster haben ein Feld namens **Anbieterzeichenfolge** oder **Provider-Zeichenfolge**. Der Inhalt dieses Felds wird als Wert in die erweiterten Eigenschaften übergeben.

- **OLE DB Services** Dieser Verbindungsparameter wird nicht direkt vom SQL Anywhere OLE DB-Provider verarbeitet. Er steuert das Verbindungspooling in ADO.
- **Prompt** Dieser Verbindungsparameter legt fest, wie Fehler bei einem Verbindungsversuch behandelt werden. Die möglichen Werte für die Eingabeaufforderung sind 1, 2, 3 und 4. Diese haben folgende Bedeutungen: `DBPROMPT_PROMPT` (1) `DBPROMPT_COMPLETE` (2) `DBPROMPT_COMPLETEREQUIRED` (3) und `DBPROMPT_NOPROMPT` (4).

Der Standardwert für die Eingabeaufforderung ist 4, was bedeutet, dass der Provider kein Verbindungsfenster anzeigt. Der Wert 1 für die Eingabeaufforderung bewirkt, dass immer ein Verbindungsfenster angezeigt wird. Der Wert 2 für die Eingabeaufforderung bewirkt, dass ein Verbindungsfenster angezeigt wird, wenn der anfängliche Verbindungsversuch fehlschlägt. Der Wert 3 für die Eingabeaufforderung bewirkt, dass ein Verbindungsfenster angezeigt wird, wenn der anfängliche Verbindungsversuch fehlschlägt, aber der Provider die Steuerelemente für alle Informationen deaktiviert, die nicht erforderlich sind, um eine Verbindung zur Datenquelle herzustellen.

- **Window Handle** Die Anwendung kann das Handle des übergeordneten Fenster übergeben, sofern vorhanden, oder einen Null-Zeiger, wenn kein Fenster-Handle (Window Handle) vorhanden ist oder der Provider keine Fenster anzeigt. Der Wert für das Fenster-Handle ist in der Regel 0 (NULL).

Andere OLE DB-Verbindungsparameter können angegeben werden, werden aber vom OLE DB-Provider ignoriert.

Wenn der SQL Anywhere OLE DB-Provider aufgerufen wird, ruft er die Eigenschaftswerte für die OLE DB-Verbindungsparameter ab. Hier sehen Sie eine typische Menge von Eigenschaftswerten aus der RowsetViewer-Anwendung von Microsoft.

```
User ID 'DBA'  
Password 'sql'  
Location 'localhost:4444'  
Initial Catalog 'demo'  
Data Source 'testds'  
Extended Properties 'appinfo=api=oledb'  
Prompt 2  
Window Handle 0
```

Die Verbindungszeichenfolge, die der Provider aus dieser Menge von Parameterwerten erstellt, lautet:

```
'DSN=testds;HOST=localhost:4444;DBN=demo;UID=DBA;PWD=sql;appinfo=api=oledb'
```

Der SQL Anywhere OLE DB-Provider benutzt die Verbindungszeichenfolge sowie die Werte für "Window Handle" und "Prompt" als Parameter für seinen Datenbankserver-Verbindungsaufruf.

Dies ist ein einfaches Beispiel einer ADO-Verbindungszeichenfolge.

```
connection.Open "Provider=SAOLEDB;UserID=DBA;Location=localhost:4444;Pwd=sql "
```

ADO analysiert die Verbindungszeichenfolge syntaktisch und übergibt alle nicht erkannten Verbindungsparameter in die erweiterten Eigenschaften. Wenn der SQL Anywhere OLE DB-Provider aufgerufen wird, ruft er die Eigenschaftswerte für die OLE DB-Verbindungsparameter ab. Hier sehen Sie die Menge von Eigenschaftswerten aus der ADO-Anwendung, welche die oben gezeigte Verbindungszeichenfolge benutzt hat.

```
User ID ''
Password ''
Location 'localhost:4444'
Initial Catalog ''
Data Source ''
Extended Properties 'UserID=DBA;Pwd=sql '
Prompt 4
Window Handle 0
```

Die Verbindungszeichenfolge, die der Provider aus dieser Menge von Parameterwerten erstellt, lautet:

```
'HOST=localhost:4444;UserID=DBA;Pwd=sql '
```

Der Provider benutzt die Verbindungszeichenfolge sowie die Werte für "Window Handle" und "Prompt" als Parameter für seinen Datenbankserver-Verbindungsaufwurf.

Siehe auch

- „Verbindungsparameter Userid (UID)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Verbindungsparameter Password (PWD)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Verbindungsparameter DataSourceName (DSN)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Verbindungsparameter DatabaseName (DBN)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Host-Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „OLE DB-Verbindungspooling“ auf Seite 347

OLE DB-Verbindungspooling

Der .NET Framework-Datenprovider für OLE DB fasst Verbindungen automatisch per OLE DB-Sitzungspooling zusammen. Wenn die Anwendung die Verbindung schließt, ist sie nicht wirklich geschlossen. Stattdessen wird die Verbindung für einen bestimmten Zeitraum aufrechterhalten. Wenn Ihre Anwendung eine Verbindung erneut öffnet, erkennt ADO/OLE DB, dass die Anwendung eine identische Verbindungszeichenfolge benutzt und verwendet die offene Verbindung erneut. Beispiel: Wenn die Anwendung 100 Mal geöffnet, ausgeführt und geschlossen wird, wird sie tatsächlich nur 1 Mal geöffnet und 1 Mal geschlossen. Das letzte Schließen tritt nach ca. 1 Minute Leerlaufzeit ein.

Wenn die Verbindung zur Datenbank durch externe Ursachen geschlossen wird (wie etwa eine erzwungene Trennung der Verbindung mittels eines administrativen Tools wie etwa Sybase Central), weiß ADO/OLE DB erst bei der nächsten Interaktion mit dem Server, dass dies eingetreten ist. Gehen Sie umsichtig vor, bevor Sie eine Trennung erzwingen.

Der Parameter, der das Verbindungspooling steuert, ist `DBPROPVAL_OS_RESOURCEPOOLING` (1). Diese Option kann mit einem Verbindungsparameter in der Zeichenfolge für die Verbindung ausgeschaltet werden.

Wenn Sie **OLE DB Services=-2** in Ihrer Verbindungszeichenfolge festlegen, dann wird das Verbindungspooling deaktiviert. Nachstehend finden Sie ein Beispiel für eine Verbindungszeichenfolge:

```
Provider=SAOLEDB;OLE DB Services=-2;...
```

Wenn Sie **OLE DB Services=-4** in Ihrer Verbindungszeichenfolge festlegen, dann werden das Verbindungspooling und die Transaktionseintragung deaktiviert. Nachstehend finden Sie ein Beispiel für eine Verbindungszeichenfolge:

```
Provider=SAOLEDB;OLE DB Services=-4;...
```

Falls Sie das Verbindungspooling deaktivieren, kommt es zu einer Performance-Verschlechterung, wenn Ihre Anwendung häufig Verbindungen öffnet/schließt und dabei die gleiche Verbindungszeichenfolge verwendet.

Siehe auch

- [SQL Server Connection Pooling \(ADO.NET\)](#)
- [Überschreiben der Provider-Service-Standardwerte](#)
- [OLE DB, ODBC und Oracle-Verbindungspooling \(ADO.NET\)](#)

Microsoft-Verbindungsserver

Sie können einen Microsoft-Verbindungsserver einrichten, der den SQL Anywhere OLE DB-Provider verwendet, um Zugriff auf eine SQL Anywhere-Datenbank zu erhalten. SQL-Abfragen können entweder mithilfe der vierteiligen Tabellenreferenzsyntax von Microsoft oder mit der `OPENQUERY` SQL-Funktion von Microsoft durchgeführt werden. Im Folgenden finden Sie ein Beispiel der vierteiligen Syntax:

```
SELECT * FROM SADATABASE.demo.GROUPO.Customers
```

In diesem Beispiel ist **SADATABASE** der Name des Verbindungsservers, **demo** der Name des Katalogs bzw. der Datenbank, **GROUPO** der Tabelleneigentümer in der SQL Anywhere-Datenbank und **Customers** der Tabellenname in der SQL Anywhere-Datenbank.

Die andere Form verwendet die `OPENQUERY`-Funktion von Microsoft.

```
SELECT * FROM OPENQUERY( SADATABASE, 'SELECT * FROM Customers' )
```

In der `OPENQUERY`-Syntax wird die zweite `SELECT`-Anweisung ('`SELECT * FROM Customers`') an den SQL Anywhere-Server zur Ausführung übergeben.

Bei komplexen Abfragen ist `OPENQUERY` möglicherweise die bessere Wahl, da die gesamte Abfrage auf dem SQL Anywhere-Server ausgewertet wird. Mit der vierteiligen Syntax ruft SQL Server möglicherweise den Inhalt aller von der Abfrage referenzierten Tabellen ab, bevor es sie auswerten kann (z.B. Abfragen mit `WHERE` oder `JOIN`, verschachtelte Abfragen usw.). Bei Abfragen mit sehr großen Tabellen kann die Verarbeitungszeit sehr lang sein, wenn Sie vierteilige Syntax verwenden. Im folgenden

Beispiel für eine vierteilige Abfrage übergibt SQL Server eine einfache SELECT-Anweisung für die komplette Tabelle (keine WHERE-Klausel) über den OLE DB-Provider an den SQL Anywhere-Datenbankserver und wertet anschließend die WHERE-Bedingung selbst aus.

```
SELECT ID, Surname, GivenName FROM [SADATABASE].[demo].[GROUP0].[Customers]
WHERE Surname = 'Elkins'
```

Statt eine Zeile in der Ergebnismenge an SQL Server zurückzugeben, werden alle Zeilen zurückgegeben und anschließend wird diese Ergebnismenge von SQL Server auf eine Zeile reduziert. Im folgenden Beispiel wird ein identisches Ergebnis erzeugt, jedoch wird nur eine Zeile an SQL Server zurückgegeben.

```
SELECT * FROM OPENQUERY( SADATABASE,
'SELECT ID, Surname, GivenName FROM [GROUP0].[Customers]
WHERE Surname = ''Elkins'' )
```

Sie können einen Verbindungsserver einrichten, der den SQL Anywhere OLE DB-Provider mit einer interaktiven Microsoft SQL Server-Anwendung oder einem SQL Server-Skript verwendet.

Hinweis

Bevor Sie einen Verbindungsserver einrichten, sollten Sie unter Windows Vista oder späteren Versionen von Windows ein paar Dinge beachten. SQL Server läuft auf Ihrem System als Dienst. Abhängig davon, wie der Dienst unter Windows Vista oder späteren Versionen von Windows eingerichtet ist, kann ein Dienst möglicherweise keine Verbindungen über den gemeinsam genutzten Speicher nutzen, keine Server starten und nicht auf Definitionen von Benutzerdatenquellen zugreifen. Beispiel: Ein als **Netzwerkdienst** angemeldeter Dienst kann weder Server starten noch eine Verbindung über Shared Memory herstellen oder auf Benutzerdatenquellen zugreifen. In solchen Fällen muss der SQL Anywhere-Server vorab gestartet und das TCP/IP-Kommunikationsprotokoll verwendet werden. Außerdem gilt: Wenn eine Datenquelle verwendet wird, muss es sich um eine Systemdatenquelle handeln.

Verbindungsserver mit einer interaktiven Anwendung einrichten

Verwenden Sie eine interaktive Microsoft SQL Server-Anwendung, um einen Microsoft-Verbindungsserver einzurichten, der den SQL Anywhere OLE DB-Provider verwendet, um Zugriff auf eine SQL Anywhere-Datenbank zu erhalten.

Voraussetzungen

SQL Server 2000 oder höher.

Aufgabe

1. Bei Microsoft SQL Server 2005/2008 starten Sie das SQL Server Management Studio. Bei anderen Versionen von SQL Server können der Name dieser Anwendung und die Schritte zum Einrichten des Verbindungsservers variieren.

Erweitern Sie im Fensterausschnitt **Objekt-Explorer** den Eintrag **Serverobjekte » Serverobjekte**. Rechtsklicken Sie auf **Verbindungsserver** und klicken Sie auf **Neuer Verbindungsserver**.

2. Füllen Sie die Seite **Allgemein** aus.

Das Feld **Verbindungsserver** auf der Seite **Allgemein** muss den Namen eines **Verbindungservers** enthalten (im obigen Beispiel SADATABASE).

Die Option **Andere Datenquelle** muss aktiviert und **SQL Anywhere OLE DB-Provider 16** aus der Liste **Anbieter** ausgewählt werden.

Das Feld **Produktname** kann einen beliebigen Wert haben (z.B. "SQL Anywhere" oder den Namen Ihrer Anwendung).

Das Feld **Datenquelle** kann einen ODBC-Datenquellennamen (DSN) enthalten. Dies ist lediglich eine Vereinfachung und ein Datenquellename ist nicht erforderlich. Wenn Sie einen System-DSN verwenden, muss es sich bei 32-Bit-Versionen von SQL Server um einen 32-Bit-DSN handeln und bei 64-Bit-Versionen von SQL Server um einen 64-Bit-DSN.

```
Data Source: SQL Anywhere 16 Demo
```

Das Feld **Anbieterzeichenfolge** kann zusätzliche Verbindungsparameter enthalten, z.B. UserID (UID), ServerName (Server) und DatabaseFile (DBF).

```
Provider string: Server=myserver;DBF=sample.db
```

Das Feld **Speicherort** kann das Äquivalent des SQL Anywhere-Verbindungsparameters "Host" enthalten (z.B. localhost:4444 oder 10.25.99.253:2638).

```
Location: AppServer-pc:2639
```

Das Feld **Anfangskatalog** kann den Namen der Datenbank enthalten, mit der eine Verbindung hergestellt werden soll (z.B. "demo"). Die Datenbank muss zuvor gestartet worden sein.

```
Initial Catalog: demo
```

Die Kombination dieser letzten vier Felder sowie die Benutzer-ID und das Kennwort auf der Seite **Sicherheit** müssen genug Informationen enthalten, um eine Verbindung mit einem Datenbankserver herzustellen.

3. Statt im Feld **Anbieterzeichenfolge**, die Benutzer-ID und das Kennwort für die Datenbank als Verbindungsparameter anzugeben, können Sie die Seite **Sicherheit** ausfüllen. Damit vermeiden Sie, dass das Kennwort als Text angezeigt wird.

Klicken Sie in SQL Server 2005/2008 auf die Option **In folgendem Sicherheitskontext verwendet** und füllen Sie die Felder **Remoteanmeldung** und **Mit Kennwort** aus. (Das Kennwort wird mit Sternchen angezeigt).

4. Gehen Sie zur Seite **Serveroptionen** (Serveroptionen).

Aktivieren Sie die Optionen **RPC** und **RPC-Ausgabe**.

Das zu verwendende Verfahren variiert abhängig von den verschiedenen Versionen von Microsoft SQL Server. In SQL Server 2000 gibt es zwei Kontrollkästchen, die für diese beiden Optionen aktiviert werden müssen. In SQL Server 2005/2008 handelt es sich bei diesen Optionen um TRUE/

FALSE-Einstellungen. Setzen Sie die beiden Werte jeweils auf TRUE. Die RPC-Optionen (Remote Procedure Call – entfernter Prozeduraufruf) müssen festgelegt sein, damit gespeicherte Prozeduren bzw. Funktionsaufrufe in einer SQL Anywhere-Datenbank ausgeführt sowie Parameter übergeben und empfangen werden können.

5. Wählen Sie die Anbieteroption InProcess zulassen aus.

Das zu verwendende Verfahren variiert abhängig von den verschiedenen Versionen von Microsoft SQL Server. In SQL Server 2000 bringt Sie die Schaltfläche **Anbieteroptionen** auf die Seite, auf der Sie diese Option auswählen können. Klicken Sie bei SQL Server 2005/2008 mit der rechten Maustaste auf den SAOLEDB.16-Providernamen unter **Verbindungsserver » Provider** und klicken Sie auf **Eigenschaften**. Vergewissern Sie sich, dass das Kontrollkästchen **InProcess zulassen** aktiviert ist. Wenn die Option **InProcess** nicht ausgewählt wurde, schlagen die Abfragen fehl.

6. Andere Provideroptionen können ignoriert werden. Einige dieser Optionen sind für die Abwärtskompatibilität von SQL Server relevant und haben keine Auswirkungen auf die Art, wie SQL Server mit dem SQL Anywhere OLE DB-Provider interagiert. Beispiele dafür sind verschachtelte Abfragen und die Unterstützung des LIKE-Operators. Wenn andere Optionen ausgewählt werden, kann dies zu Syntaxfehlern oder schlechterer Performance führen.

Ergebnisse

Der Microsoft-Verbindungsserver wird konfiguriert.

Verbindungsserver mit einem Skript einrichten

Sie können eine Verbindungsserverdefinition mit einem SQL Server-Skript einrichten.

Voraussetzungen

SQL Server 2005 oder später.

Kontext und Bemerkungen

Nehmen Sie anhand der nachstehenden Schritte die entsprechenden Änderungen im folgenden Skript vor, bevor Sie es in SQL Server ausführen.

```
USE [master]
GO
EXEC master.dbo.sp_addlinkedserver @server=N'SADATABASE',
    @srvproduct=N'SQL Anywhere', @provider=N'SAOLEDB.16',
    @datasrc=N'SQL Anywhere 16 Demo',
    @provstr=N'host=localhost:4444;server=myserver;dbn=demo'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
    @optname=N'rpc', @optvalue=N'true'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
    @optname=N'rpc out', @optvalue=N'true'
GO
-- Set remote login
EXEC master.dbo.sp_addlinkedsrvlogin @rmtsrvname = N'SADATABASE',
    @locallogin = NULL , @useself = N'False',
```

```
@rmtuser = N'DBA', @rmtpassword = N'sql'
GO
-- Set global provider "allow in process" flag
EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.16', N'AllowInProcess', 1
```

Aufgabe

- 1. Wählen Sie einen neuen Namen für den Verbindungsserver. (Im Beispiel wird SADATABASE verwendet.)
- 2. Wählen Sie einen optionalen Datenquellennamen. (Im Beispiel wird SQL Anywhere 16 Demo verwendet.)
- 3. Wählen Sie eine optionale Providerzeichenfolge. (Im Beispiel wird N'host=localhost:4444;server=myserver;dbn=demo' verwendet.)
- 4. Wählen Sie eine ID und ein Kennwort für den entfernten Benutzer. (Im Beispiel werden N'DBA' und N'sql' verwendet.)

Ergebnisse

Ihr geändertes Skript kann in Microsoft SQL Server ausgeführt werden, um einen neuen Verbindungsserver zu erstellen.

Unterstützte OLE DB-Schnittstellen

Die OLE DB-API besteht aus einer Reihe von Schnittstellen. In der folgenden Tabelle wird die Unterstützung für jede Schnittstelle im OLE DB-Treiber von SQL Anywhere beschrieben.

Schnittstelle	Verwendung	Einschränkungen
IAccessor	Bindungen zwischen Clientspeicher- und Datenspeicherwerten definieren	DBACCESSOR_PASSBYREF nicht unterstützt. DBACCESSOR_OPTIMIZED nicht unterstützt.
IAlterIndex IAlterTable	Tabellen, Indizes und Spalten ändern	Nicht unterstützt.
IChapteredRowset	Mit einer segmentierten Zeilengruppe kann auf Zeilen einer Zeilengruppe in getrennten Kapiteln zugegriffen werden.	Nicht unterstützt. SQL Anywhere unterstützt keine segmentierten Zeilengruppen.
IColumnsInfo	Einfache Informationen zu Spalten in einer Zeilengruppe erhalten.	Unterstützt.

Schnittstelle	Verwendung	Einschränkungen
IColumnsRowset	Informationen zu optionalen Metadaten in einer Zeilengruppe und eine Zeilengruppe von Spalten-Metadaten erhalten	Unterstützt.
ICommand	SQL-Anweisungen ausführen.	Unterstützt keinen Aufruf. ICommandProperties: GetProperties mit DBPROPSET_PROPERTYESINEROR um Eigenschaften zu finden, die nicht gesetzt werden konnten.
ICommandPersist	Zustand eines Befehlsobjekts (jedoch nicht jede aktive Zeilengruppe) aufrechterhalten. Diese beständigen Befehlsobjekte können nachfolgend mit der Zeilengruppe PROCEDURES oder VIEWS nummeriert werden.	Unterstützt.
ICommandPrepare	Befehle vorbereiten	Unterstützt.
ICommandProperties	Zeilengruppeneigenschaften für von einem Befehl erstellte Zeilengruppen festlegen. Häufig verwendet, um die Schnittstellen anzugeben, welche die Zeilengruppe unterstützen sollen	Unterstützt.
ICommandText	SQL-Anweisungstext für ICommand festlegen	Nur der SQL-Dialekt DBGUID_DEFAULT wird unterstützt.
ICommandWithParameters	Parameterinformationen für einen Befehl festlegen und beziehen	Nicht für Parameter unterstützt, die als Vektoren skalarer Werte gespeichert werden Keine Unterstützung für BLOB-Parameter
IConvertType		Unterstützt.

Schnittstelle	Verwendung	Einschränkungen
IDBAsynchNotify IDBAsynchStatus	Asynchrone Verarbeitung Client über Ereignisse in der asynchronen Verarbeitung bei der Initialisierung von Datenquellen, dem Anfüllen von Zeilengruppen, usw. informieren	Nicht unterstützt.
IDBCreateCommand	Befehle aus einer Sitzung erstellen	Unterstützt.
IDBCreateSession	Eine Sitzung aus einem Datenquellenobjekt erstellen	Unterstützt.
IDBDataSourceAdmin	Datenquellenobjekte, bei denen es sich um COM-Objekte handelt, die von Clients unterstützt werden, erstellen/zerstören/ändern. Diese Schnittstelle wird nicht für die Verwaltung von Datenspeichern (Datenbanken) verwendet.	Nicht unterstützt.
IDBInfo	Informationen über Schlüsselwörter suchen, die für diesen Provider eindeutig sind (d.h. Schlüsselwörter die nicht dem Standard-SQL entsprechen) Außerdem Informationen zu Literalen, d.h. Sonderzeichen, die in textlich übereinstimmenden Abfragen verwendet werden, und andere Literal-Informationen suchen	Unterstützt.
IDBInitialize	Datenquellenobjekte und Aufzähler initialisieren	Unterstützt.
IDBProperties	Eigenschaften für ein Datenquellenobjekt oder einen Aufzähler verwalten	Unterstützt.
IDBSchemaRowset	Informationen zu Systemtabellen in einer Standardmaske (einer Zeilengruppe) beziehen.	Unterstützt.

Schnittstelle	Verwendung	Einschränkungen
IErrorInfo IErrorLookup IErrorRecords	Unterstützung von ActiveX-Fehler-Objekten	Unterstützt.
IGetDataSource	Gibt einen Schnittstellenzeiger auf das Datenquellenobjekt einer Sitzung zurück	Unterstützt.
IIndexDefinition	Indizes im Datenspeicher erstellen oder löschen	Nicht unterstützt.
IMultipleResults	Mehrere Ergebnisse (Zeilengruppen oder Zeilenanzahlen) aus einem Befehl abrufen	Unterstützt.
IOpenRowset	Auf eine Datenbanktabelle mit ihrem Namen, nicht mit SQL zugreifen	Unterstützt. Öffnen einer Tabelle mit ihrem Namen, nicht über einen GUID, wird unterstützt
IParentRowset	Auf segmentierte/hierarchische Zeilengruppen zugreifen	Nicht unterstützt.
IRowset	Auf Zeilengruppen zugreifen	Unterstützt.
IRowsetChange	Änderungen an Zeilengruppendaten zulassen, die im Datenspeicher reflektiert werden InsertRow/SetData für BLOBs sind noch nicht implementiert.	Unterstützt.
IRowsetChapterMember	Auf segmentierte/hierarchische Zeilengruppen zugreifen	Nicht unterstützt.
IRowsetCurrentIndex	Index für eine Zeilengruppe dynamisch ändern	Nicht unterstützt.
IRowsetFind	Eine Zeile innerhalb einer Zeilengruppe, die mit einem spezifischen Wert übereinstimmt, finden	Nicht unterstützt.
IRowsetIdentity	Zeilen-Handles vergleichen	Nicht unterstützt.

Schnittstelle	Verwendung	Einschränkungen
IRowsetIndex	Auf Datenbankindizes zugreifen	Nicht unterstützt.
IRowsetInfo	Informationen zu Zeilengruppeneigenschaften suchen oder das Objekt suchen, das die Zeilengruppe erstellt hat	Unterstützt.
IRowsetLocate	Position in Zeilen einer Zeilengruppe, unter Verwendung von Lesezeichen.	Unterstützt.
IRowsetNotify	Bietet eine COM-Callback-Schnittstelle für Zeilengruppen-Ereignisse	Unterstützt.
IRowsetRefresh	Den neuesten Wert für Daten beziehen, der für eine Transaktion sichtbar ist	Nicht unterstützt.
IRowsetResynch	Alte OLEDB 1.x-Schnittstelle, von IRowsetRefresh abgelöst.	Nicht unterstützt.
IRowsetScroll	Durch Zeilengruppe blättern, um Zeilendaten abzurufen	Nicht unterstützt.
IRowsetUpdate	Änderungen an Zeilengruppendaten verzögern, bis Update aufgerufen wurde	Unterstützt.
IRowsetView	Ansichten für eine vorhandene Zeilengruppe verwenden	Nicht unterstützt.
ISequentialStream	Eine Blob-Spalte abrufen	Nur für Lesen unterstützt Keine Unterstützung von Set-Data mit dieser Schnittstelle
ISessionProperties	Informationen zu Sitzungseigenschaften beziehen	Unterstützt.
ISourcesRowset	Eine Zeilengruppe von Datenquellensubjekten und Aufzählern beziehen	Unterstützt.
ISQLErrorInfo ISupportErrorInfo	Unterstützung von ActiveX-Fehler-Objekten	Unterstützt.

Schnittstelle	Verwendung	Einschränkungen
ITableDefinition ItableDefinitionWithConstraints	Tabellen mit Integritätsregeln erstellen, löschen und ändern	Unterstützt.
ITransaction	Transaktionen festschreiben oder abbrechen	Nicht alle Kennzeichnungen werden unterstützt.
ITransactionJoin	Verteilte Transaktionen werden unterstützt	Nicht alle Kennzeichnungen werden unterstützt.
ITransactionLocal	Transaktionen für eine Sitzung werden abgewickelt Nicht alle Kennzeichnungen werden unterstützt.	Unterstützt.
ITransactionOptions	Optionen für eine Transaktion beziehen oder einstellen	Unterstützt.
IviewChapter	Mit Ansichten für eine vorhandene Zeilengruppe arbeiten, insbesondere um Nachbearbeitungsfilter/-sortierungen auf Zeilen anzuwenden	Nicht unterstützt.
IviewFilter	Inhalte einer Zeilengruppe werden auf Zeilen, welche eine Reihe von Bedingungen erfüllen, beschränkt	Nicht unterstützt.
IviewRowset	Inhalte einer Zeilengruppe werden beim Öffnen einer Zeilengruppe auf Zeilen, welche eine Reihe von Bedingungen erfüllen, beschränkt	Nicht unterstützt.
IviewSort	Sortierreihenfolge auf eine Ansicht anwenden	Nicht unterstützt.

OLE DB-Provider-Registrierung

Wenn der SAOLEDB-Provider mit dem SQL Anywhere-Installationsprogramm installiert wird, registriert sich der Provider selbst. Bei diesem Registrierungsprozess werden Registrierungseinträge im COM-Abschnitt der Registrierung vorgenommen, damit ADO die DLL finden kann, wenn der SAOLEDB-Provider aufgerufen wird. Wenn Sie den Speicherort Ihrer DLL ändern, müssen Sie sie neu registrieren.

Beispiel

Die folgenden Befehle registrieren den SQL Anywhere OLE DB-Provider, wenn sie in dem Verzeichnis ausgeführt werden, in dem der Provider installiert ist:

```
regsvr32 dboledb16.dll  
regsvr32 dboledba16.dll
```

ODBC-Unterstützung

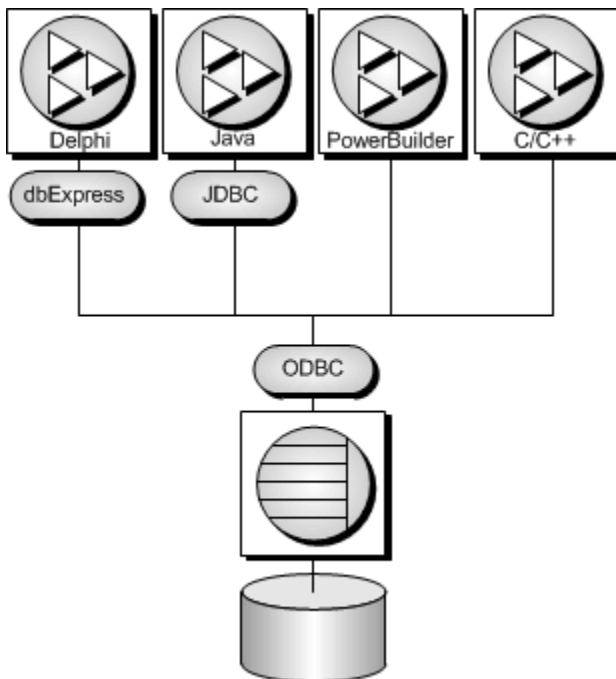
ODBC (Open Database Connectivity) ist eine Standardschnittstelle auf Aufrufebene (Call Level Interface, CLI), die von der Microsoft Corporation entwickelt wurde. Sie basiert auf der SQL Access Group CLI Spezifikation. ODBC-Anwendungen können mit jeder Datenquelle ausgeführt werden, die einen ODBC-Treiber bereitstellt. ODBC bietet eine gute Programmierschnittstelle, wenn Ihre Anwendung auf andere Datenquellen übertragbar sein soll, die mit ODBC-Treibern funktionieren.

ODBC ist eine Schnittstelle auf niedriger Ebene. Fast alle Funktionen von SQL Anywhere sind mit dieser Schnittstelle verfügbar. ODBC ist unter Windows-Betriebssystemen als DLL verfügbar. Unter Unix steht es als Shared Object-Bibliothek zur Verfügung.

Die Kerndokumentation zur ODBC-API finden Sie auf der Microsoft Developer Network Website unter <http://msdn.microsoft.com/de-de/library/ms710252.aspx>.

Voraussetzungen für die Entwicklung von ODBC-Anwendungen

Sie können Anwendungen, wie in der folgenden Abbildung dargestellt, mit einer Reihe verschiedener Entwicklungstools und Programmiersprachen entwickeln und mit der ODBC-API auf den SQL Anywhere-Datenbankserver zugreifen.



Um ODBC-Anwendungen für SQL Anywhere zu entwickeln, brauchen Sie:

- SQL Anywhere.
- Einen C/C++-Compiler, mit dem Sie Programme in Ihrer Betriebssystemumgebung erstellen können.
- Das Microsoft ODBC Software Development Kit. Sie finden es im Microsoft Developer Network. Es umfasst Dokumentation und Tools, um ODBC-Anwendungen zu testen.

Hinweis

Einige Tools zur Entwicklung von Anwendungen, die bereits ODBC unterstützen, haben eine eigene Programmierschnittstelle, die die ODBC-Schnittstelle verbirgt. Die SQL Anywhere-Dokumentation beschreibt nicht, wie Sie diese Tools verwenden.

ODBC-Treibermanager

Microsoft Windows umfasst einen ODBC-Treibermanager. Es gibt keinen Treibermanager für Windows Mobile. Für Unix wird ein Treibermanager mit SQL Anywhere bereitgestellt.

ODBC-Übereinstimmung

SQL Anywhere bietet Unterstützung für ODBC 3.5, das mit dem Microsoft Data Access Kit 2.7 mitgeliefert wird.

Stufen der ODBC-Unterstützung

ODBC-Funktionen werden nach Kompatibilitätsstufen eingeordnet. Eine Funktion ist entweder **Kern**, **Stufe 1** oder **Stufe 2**, wobei Stufe 2 den Standard am besten unterstützt. Diese Funktionen finden Sie in der Microsoft-Dokumentation *ODBC Programmer's Reference* unter <http://msdn.microsoft.com/de-de/library/ms714177.aspx>.

Von SQL Anywhere unterstützte Funktionen

SQL Anywhere unterstützt die ODBC 3.5-Spezifikation folgendermaßen:

- **Kernübereinstimmung** SQL Anywhere unterstützt alle Kernfunktionen.
- **Übereinstimmung mit Stufe 1** SQL Anywhere unterstützt alle Funktionen der Stufe 1, abgesehen von der asynchronen Ausführung von ODBC-Funktionen.

SQL Anywhere unterstützt die gemeinsame Nutzung einer Verbindung durch mehrfache Threads. Die Anforderungen von verschiedenen Threads werden von SQL Anywhere serialisiert.

- **Übereinstimmung mit Stufe 2** SQL Anywhere unterstützt alle Funktionen der Stufe 2, mit Ausnahme der folgenden:
 - Dreiteilige Namen von Tabellen und Ansichten. Dies ist für SQL Anywhere nicht verwendbar.
 - Asynchrone Ausführung von ODBC-Funktionen für bestimmte einzelne Anweisungen
 - Die Fähigkeit, Login-Anforderungen und SQL-Abfragen nach einer gegebenen Zeit abzubrechen (time out)

Entwicklung von ODBC-Anwendungen

Jede C/C++-Quelldatei, die ODBC-Funktionen aufruft, muss eine plattformspezifische ODBC-Header-Datei einbeziehen. Jede plattformspezifische Header-Datei schließt die wichtigste ODBC-Header-Datei *odbc.h* ein, die alle Funktionen, Datentypen und Konstanten definiert, die erforderlich sind, um ein ODBC-Programm zu schreiben.

Führen Sie die folgenden Aufgaben durch, um die ODBC-Headerdatei in eine C/C++-Quelldatei einzubeziehen:

1. Fügen Sie eine Include-Zeile hinzu, die die betreffende plattformspezifische Header-Datei referenziert. Folgende Zeilen müssen benutzt werden:

Betriebssystem	Include-Zeile
Windows	#include "ntodbc.h"
Unix	#include "unixodbc.h"
Windows Mobile	#include "ntodbc.h"

2. Fügen Sie das Verzeichnis, das die Header-Datei enthält, dem Include-Pfad Ihres Compilers hinzu.

Die plattformspezifischen Header-Dateien und *odbc.h* werden im Unterverzeichnis *SDK\Include* Ihres SQL Anywhere-Installationsverzeichnisses installiert.

3. Wenn Sie ODBC-Anwendungen für Unix erstellen, müssen Sie möglicherweise den Makro "UNIX" für 32-Bit-Anwendungen bzw. "UNIX64" für 64-Bit-Anwendungen definieren, damit die Daten korrekt ausgerichtet und dimensioniert werden. Dieser Schritt ist nicht erforderlich, wenn Sie einen der folgenden unterstützten Compiler verwenden:

- GNU C/C++-Compiler auf einer unserer unterstützten Plattformen
- Intel C/C++ Compiler für Linux (icc)
- SunPro C/C++ Compiler für Linux oder Solaris
- VisualAge C/C++ Compiler für AIX
- C/C++ Compiler (cc/aCC) für HP-UX

Sobald Ihr Quellcode geschrieben ist, können Sie die Anwendung kompilieren und verknüpfen. In den folgenden Abschnitten wird beschrieben, wie ausführbare Anwendungen erstellt werden.

ODBC-Anwendungen unter Windows

Dieser Abschnitt betrifft nicht Windows Mobile.

Wenn Sie Ihre Anwendung linken, müssen Sie diese mit der geeigneten Importbibliothekdatei verbinden. Damit ist der Zugriff auf die ODBC-Funktionen gewährleistet: Mit der Importbibliothek werden Eintrittspunkte für den ODBC-Treibermanager *odbc32.dll* definiert. Der Treibermanager lädt seinerseits den SQL Anywhere-ODBC-Treiber *dbodbc16.dll*.

Üblicherweise ist die Importbibliothek unter der *Lib*-Verzeichnisstruktur der Microsoft-Plattform SDK gespeichert:

Betriebssystem	Importbibliothek
Windows (32-Bit)	<i>Lib\odbc32.lib</i>
Windows (64-Bit)	<i>Lib\x64\odbc32.lib</i>

Beispiel

Der folgende Befehl veranschaulicht, wie Sie das Verzeichnis, das die plattformspezifische Importbibliothek enthält, zur Liste der Bibliotheksverzeichnisse in Ihrer LIB-Umgebungsvariablen hinzufügen:

```
set LIB=%LIB%;c:\mssdk\v7.0\lib
```

Der folgende Befehl veranschaulicht, wie Sie die in *odbc.c* gespeicherte Anwendung mit dem entsprechenden Microsoft-Tool kompilieren und verknüpfen:

```
cl odbc.c /Ic:\sa16\SDK\Include odbc32.lib
```

Siehe auch

- „ODBC-Anwendungen unter Windows Mobile“ auf Seite 362

ODBC-Anwendungen unter Windows Mobile

In Windows Mobile gibt es keinen ODBC-Treibermanager. Mit der Importbibliothek (*dbodbc16.lib*) werden die Eintrittspunkte direkt in den SQL Anywhere-ODBC-Treiber *dbodbc16.dll* eingelesen. Diese Datei befindet sich im *SDK\Lib\CE\Arm.50*-Unterverzeichnis Ihrer SQL Anywhere-Installation.

Da es keinen ODBC-Treibermanager für Windows Mobile gibt, müssen Sie den Speicherort der SQL Anywhere ODBC-Treiber-DLL mit dem Parameter DRIVER= in der Verbindungszeichenfolge angeben, die der Funktion SQLDriverConnect übergeben wird. Im Folgenden finden Sie ein Beispiel:

```
szConnStrIn = "driver=\\Windows\\dbodbc16.dll;dbf=\\My Documents\\demo.db"
```

Wenn Sie eine Verknüpfung mit einer ODBC-Anwendung für Windows Mobile erstellen möchten, fügen Sie das Verzeichnis mit der plattformspezifischen Importbibliothek zur Liste der Bibliothekenverzeichnisse hinzu.

Eine Liste der unterstützten Versionen von Windows Mobile finden Sie in der Tabelle "SQL Anywhere für PC-Plattformen" unter <http://www.sybase.com/detail?id=1002288>.

Das Beispielprogramm (*odbc_sample.cpp*) verwendet eine Dateidatenquelle (FileDSN-Verbindungsparameter) mit dem Namen *SQL Anywhere 16 Demo.dsn*. Diese Datei wird im Stammverzeichnis Ihres Windows Mobile-Geräts abgelegt, wenn Sie SQL Anywhere für Windows Mobile auf Ihrem Gerät installieren. Sie können Dateidatenquellen auf Ihrem PC-System mit dem ODBC-Datenquellenadministrator erstellen. Diese müssen jedoch für das PC-System erstellt und dann bearbeitet

werden, um es für die Windows Mobile-Umgebung anzupassen. Nach der erforderlichen Bearbeitung können Sie sie auf das Windows Mobile-Gerät kopieren.

Windows Mobile und Unicode

SQL Anywhere verwendet die Kodierung UTF-8. Hierbei handelt es sich um eine Mehrbyte-Zeichenkodierung, die für die Kodierung von Unicode verwendet werden kann.

Der SQL Anywhere ODBC-Treiber unterstützt entweder ASCII (8-Bit)-Zeichenfolgen oder Unicode (Wide Character)-Zeichenfolgen. Das UNICODE-Makro steuert, ob ODBC-Funktionen ASCII- oder Unicode-Zeichenfolgen erwarten. Wenn Ihre Anwendung mit dem definierten UNICODE-Makro erstellt werden muss, Sie aber die ASCII ODBC-Funktionen nutzen möchten, müssen Sie ebenfalls das SQL_NOUNICODEMAP-Makro definieren.

Die Beispieldatei `%SQLANYSAMPI6%\SQLAnywhere\C\odbc.c` veranschaulicht, wie Sie die Unicode-ODBC-Funktionen verwenden.

ODBC-Anwendungen unter Unix

Im Lieferumfang von SQL Anywhere ist ein ODBC-Treibermanager für Unix enthalten. Außerdem sind Treibermanager von Drittherstellern erhältlich. In diesem Abschnitt wird beschrieben, wie ODBC-Anwendungen aufgebaut werden, die den ODBC-Treibermanager nicht verwenden.

ODBC-Treiber

Der ODBC-Treiber ist ein Shared Object oder eine gemeinsam genutzte Bibliothek. Separate Versionen des SQL Anywhere-ODBC-Treibers werden für Anwendungen mit einem einzelnen sowie mit mehreren Threads angeboten. Es wird ein generischer SQL Anywhere-ODBC-Treiber bereitgestellt, der das verwendete Threading-Modell und direkte Aufrufe der zutreffenden Bibliothek mit einem einzelnen oder mit mehreren Threads erkennt.

Die ODBC-Treiber sind folgende Dateien:

Betriebssystem	Tread-Modell	ODBC-Treiber
(alle Unix-Betriebssysteme außer Mac OS X und HP-UX)	Generisch	<i>libdbodbc16.so (libdbodbc16.so.1)</i>
(alle Unix-Betriebssysteme außer Mac OS X und HP-UX)	Ein Thread	<i>libdbodbc16_n.so (libdbodbc16_n.so.1)</i>
(alle Unix-Betriebssysteme außer Mac OS X und HP-UX)	Mehrere Threads	<i>libdbodbc16_r.so (libdbodbc16_r.so.1)</i>
HP-UX	Generisch	<i>libdbodbc16.sl (libdbodbc16.sl.1)</i>

Betriebssystem	Tread-Modell	ODBC-Treiber
HP-UX	Ein Thread	<i>libdbodbc16_n.sl (libdbodbc16_n.sl.1)</i>
HP-UX	Mehrere Threads	<i>libdbodbc16_r.sl (libdbodbc16_r.sl.1)</i>
Mac OS X	Generisch	<i>libdbodbc16.dylib</i>
Mac OS X	Ein Thread	<i>libdbodbc16_n.dylib</i>
Mac OS X	Mehrere Threads	<i>libdbodbc16_r.dylib</i>

Die Bibliotheken sind mit einer Versionsnummer (in Klammern) als symbolische Verknüpfungen mit der gemeinsam benutzten Bibliothek installiert.

Außerdem werden die folgenden Pakete für Mac OS X bereitgestellt:

Betriebssystem	Tread-Modell	ODBC-Treiber
Mac OS X	Ein Thread	<i>dbodbc16.bundle</i>
Mac OS X	Mehrere Threads	<i>dbodbc16_r.bundle</i>

Wenn Sie eine ODBC-Anwendung unter Unix verknüpfen möchten, erstellen Sie die Verknüpfung für Ihre Anwendung mit dem generischen ODBC-Treiber *libdbodbc16*. Stellen Sie beim Deployment Ihrer Anwendung sicher, dass die geeigneten (oder alle) ODBC-Treiberversionen (ohne oder mit Threading) im Bibliothekspfad des Benutzers verfügbar sind.

Datenquelleninformation

Wenn SQL Anywhere keinen ODBC-Treibermanager erkennt, wird die Systeminformationsdatei für die Datenquelleninformationen verwendet.

Siehe auch

- „ODBC-Datenquellen unter Unix“ [[SQL Anywhere Server - Datenbankadministration](#)]

Verwenden des SQL Anywhere ODBC-Treibermanagers unter Unix

SQL Anywhere umfasst einen ODBC-Treibermanager für Unix. Das Shared Object *libdbodm16* kann auf allen unterstützten Unix-Plattformen als ODBC-Treibermanager verwendet werden. Mit dem Treibermanager können Sie jeden beliebigen ODBC-Treiber ab Version 3.0 laden. Der Treibermanager führt keine Zuordnungen zwischen ODBC 1.0/2.0-Aufrufen und ODBC 3.x-Aufrufen aus. Deshalb müssen Anwendungen, die den Treibermanager verwenden, den Einsatz der ODBC-Funktionsgruppe auf Treiber ab Version 3.0 beschränken. Außerdem kann der Treibermanager von Anwendungen mit und ohne Threading verwendet werden.

Der Treibermanager kann ODBC-Aufrufe für eine angegebene Verbindung protokollieren. Verwenden Sie die Direktiven `TraceLevel` und `TraceLog`, um die Protokollierung zu aktivieren. Diese Direktiven können Teil einer Verbindungszeichenfolge (falls `SQLDriverConnect` verwendet wird) oder eines DSN-Eintrags sein. Die `TraceLog`-Direktive identifiziert die Logdatei, die die Trace-Ausgabe für die Verbindung enthalten soll. Die `TraceLevel`-Direktive steuert den Umfang der gewünschten Protokollierungsinformationen. Es gibt folgende Protokollierungsstufen:

- **NONE** Es werden keine Protokollinformationen ausgegeben.
- **MINIMAL** Name und Parameter der Routine werden in die Ausgabe einbezogen.
- **LOW** Außer den oben genannten Informationen werden die Rückgabewerte in die Ausgabe einbezogen.
- **MEDIUM** Außer den oben genannten Informationen werden das Datum und die Uhrzeit in die Ausgabe einbezogen.
- **HIGH** Außer den oben genannten Informationen werden die Parametertypen in die Ausgabe einbezogen.

ODBC-Treibermanager von Drittherstellern für Unix sind ebenfalls verfügbar. Informationen zu ihrer Verwendung finden Sie in der Dokumentation zu den betreffenden Treibermanagern.

Siehe auch

- „Verwenden des unixODBC-Treibermanagers“ auf Seite 365
- „UTF-32-ODBC-Treibermanager für Unix“ auf Seite 366

Verwenden des unixODBC-Treibermanagers

unixODBC-Versionen vor der Version 2.2.14 haben einige Aspekte der von Microsoft festgelegten 64-Bit-ODBC-Spezifikation falsch implementiert. Dadurch kommt es zu Problemen bei der Verwendung des unixODBC-Treibermanagers mit dem SQL Anywhere 64-Bit ODBC-Treiber.

Um diese Probleme zu vermeiden, sollten Sie sich der Unterschiede bewusst sein. Einer davon ist die Definition von `SQLLEN` und `SQLULEN`. Dies sind 64-Bit-Typen in der Microsoft 64-Bit ODBC-Spezifikation und werden vom SQL Anywhere 64-Bit ODBC-Treiber als 64-Bit-Quantitäten erwartet. Manche Implementierungen von unixODBC definieren diese zwei Typen als 32-Bit-Quantitäten und dies führt zu Problemen, wenn eine Schnittstelle zum SQL Anywhere 64-Bit ODBC-Treiber hergestellt wird.

Es gibt drei Dinge, die Sie tun müssen, um Probleme auf 64-Bit-Plattformen zu vermeiden.

1. Statt die unixODBC-Header-Dateien wie *sql.h* und *sqlext.h* aufzunehmen, sollten Sie die SQL Anywhere ODBC-Header-Datei *unixodbc.h* verwenden. Dies gewährleistet, dass Sie die korrekten Definitionen von `SQLLEN` und `SQLULEN` erhalten. Die Header-Dateien in unixODBC 2.2.14 oder späteren Versionen beheben das Problem.
2. Sie müssen sicherstellen, dass Sie die korrekten Typen bei allen Parametern verwenden. Die Verwendung der korrekten Header-Datei und die strenge Typprüfung (Strong Type Checking) Ihres C/C++-Compilers sollte Sie in diesem Bereich unterstützen. Sie müssen auch sicherstellen, dass Sie

die korrekten Typen für alle Variablen verwendet haben, die vom SQL Anywhere-Treiber indirekt durch Zeiger gesetzt werden.

3. Verwenden Sie nicht die Versionen von des unixODBC-Treibermanagers vor Version 2.2.14. Linken Sie stattdessen direkt mit dem SQL Anywhere-ODBC-Treiber. Stellen Sie z.B. sicher, dass das Shared Object libodbc mit dem SQL Anywhere-Treiber verknüpft ist.

```
libodbc.so.1 -> libdbodbc16_r.so.1
```

Als Alternative können Sie auch den SQL Anywhere-Treibermanager auf Plattformen verwenden, auf denen er verfügbar ist.

Siehe auch

- [„Verwenden des SQL Anywhere ODBC-Treibermanagers unter Unix“ auf Seite 364](#)
- [„ODBC-Anwendungen unter Unix“ auf Seite 363](#)
- [„Hinweise zu 64-Bit-ODBC“ auf Seite 382](#)

UTF-32-ODBC-Treibermanager für Unix

Die Versionen von ODBC-Treibermanagern für, die SQLWCHAR als 32-Bit-Mengen (UTF-32) definieren, können nicht mit dem SQL Anywhere-ODBC-Treiber verwendet werden, der weite Aufrufe unterstützt, da dieser Treiber für 16-Bit-SQLWCHAR erstellt wurde. Für solche Fälle wird eine reine ANSI-Version des SQL Anywhere-ODBC-Treibers zur Verfügung gestellt. Diese Version des ODBC-Treibers unterstützt keine Schnittstellen für weite Aufrufe (z.B. SQLConnectW).

Der Shared Object-Name des Treibers lautet *libdbodbcansi16_r*. Von dem Treiber wird nur eine Variante mit Threading bereitgestellt. Unter Mac OS X ist der Treiber zusätzlich zur dylib-Datei auch in Bundle-Form verfügbar (*dbodbcansi16_r.bundle*). Bestimmte Frameworks, z.B. Real Basic, funktionieren nicht mit der dylib-Datei und erfordern das Bundle.

Der normale ODBC-Treiber behandelt SQLWCHAR-Zeichenfolgen als UTF-16-Zeichenfolgen. Dieser Treiber kann mit einigen ODBC-Treibermanagern nicht verwendet werden, die SQLWCHAR-Zeichenfolgen als UTF-32-Zeichenfolgen behandeln, z.B. iODBC. Bei Verwendung von Unicode-fähigen Treibern konvertieren diese Treibermanager naive Aufrufe von der Anwendung in weite Aufrufe für den Treiber. Ein reiner ANSI-Treiber umgeht dieses Verhalten, sodass der Treiber mit diesen Treibermanagern verwendet werden kann, sofern die Anwendung keine weiten Aufrufe ausführt. Weite Aufrufe über iODBC oder über einen anderen Treibermanager mit ähnlicher Semantik werden weiterhin nicht unterstützt.

ODBC-Beispiele

SQL Anywhere umfasst mehrere ODBC-Beispiele. Die Beispiele befinden sich in den Unterverzeichnissen von *%SQLANYSAMPI6%\SQLAnywhere*.

Die Beispiele in Verzeichnissen, die mit den vier Buchstaben ODBC beginnen, veranschaulichen separate und einfache ODBC-Aufgaben wie das Herstellen einer Verbindung mit einer Datenbank oder das Ausführen von Anweisungen. Ein vollständiges ODBC-Beispielprogramm finden Sie in der Datei

`%SQLANYSAMPI6%\SQLAnywhere\C\odbc.c`. Dieses Programm führt dieselben Aktionen aus wie das Beispielprogramm für dynamische Cursor in Embedded SQL, das sich im gleichen Verzeichnis befindet.

Siehe auch

- „Beispielprogramme mit Embedded SQL“ auf Seite 477

ODBC-Beispielprogramm für Windows aufbauen

Nach dem Aufbauen des ODBC-Beispielprogramms können Sie das Programm ausführen und feststellen, wie es ODBC-Aufgaben ausführt, z.B. das Herstellen einer Verbindung mit einer Datenbank oder das Ausführen von Anweisungen.

Voraussetzungen

Eine neue Version von Microsoft Visual Studio ist erforderlich.

Bei x86/x64-Plattform-Builds mit Visual Studio müssen Sie die richtige Umgebung zum Kompilieren und Verknüpfen einrichten. Dies erfolgt in der Regel in Visual Studio mit `vcvars32.bat` oder `vcvars64.bat` (in älteren Versionen von Visual Studio `vcvarsamd64.bat`).

Kontext und Bemerkungen

Eine Batchdatei im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\C` kann verwendet werden, um alle Beispielanwendungen zu kompilieren und zu verknüpfen.

Aufgabe

1. Öffnen Sie eine Eingabeaufforderung und ändern Sie das Verzeichnis in `%SQLANYSAMPI6%\SQLAnywhere\C`.
2. Führen Sie die Batchdatei `build.bat` aus.

Wenn Aufbaufehler ausgegeben werden, versuchen Sie, die Zielplattform (x86 oder x64) als Argument für `build.bat` anzugeben. Hier sehen Sie ein Beispiel.

```
build x64
```

Ergebnisse

Das ODBC-Beispielprogramm wird aufgebaut.

ODBC-Beispielprogramm für Unix aufbauen

Nach dem Aufbauen des ODBC-Beispielprogramms können Sie das Programm ausführen und feststellen, wie es ODBC-Aufgaben ausführt, z.B. das Herstellen einer Verbindung mit einer Datenbank oder das Ausführen von Anweisungen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Ein Shell-Skript im Verzeichnis *samples-dir/sqlanywhere/c* kann verwendet werden, um alle Beispielanwendungen zu kompilieren und zu verknüpfen.

Aufgabe

1. Öffnen Sie eine Shell und ändern Sie das Verzeichnis in *samples-dir/sqlanywhere/c*.
2. Führen Sie das Shell-Skript *build.sh* aus.

Ergebnisse

Das ODBC-Beispielprogramm wird aufgebaut.

ODBC Beispielprogramme

Sie können das ODBC-Beispielprogramm laden, indem Sie die Datei auf der entsprechenden Plattform ausführen.

- Führen Sie unter 32-Bit-Windows die Datei *%SQLANYSAMPI6%\SQLAnywhere\C\odbcwin.exe* aus.
- Führen Sie unter 64-Bit-Windows die Datei *%SQLANYSAMPI6%\SQLAnywhere\C\odbcx64n.exe* aus.
- Führen Sie unter Unix die Datei *samples-dir/sqlanywhere/c/odbc* aus.

Wählen Sie nach dem Ausführen der Datei eine der Tabellen in der Beispieldatenbank. Sie können z.B. **Customers** oder **Employees** eingeben.

ODBC-Handles

ODBC-Anwendungen verwenden eine kleine Anzahl von **Handles**, um grundlegende Funktionen wie Datenbankverbindungen und SQL-Anweisungen zu definieren. Ein Handle ist ein 32-Bit-Wert.

Die folgenden Handles werden in nahezu allen ODBC-Anwendungen eingesetzt:

- **Umgebung** Das Umgebungs-Handle liefert einen globalen Kontext, in dem auf Daten zugegriffen wird. Jede ODBC-Anwendung muss beim Start genau einen Umgebungs-Handle zuweisen und es bei Beendigung wieder freigeben.

Der folgende Code veranschaulicht, wie ein Umgebungs-Handle zugewiesen wird:

```
SQLRETURN rc;  
SQLHENV env;  
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

- **Verbindung** Eine Verbindung wird spezifiziert durch einen ODBC-Treiber und eine Datenquelle. Eine Anwendung kann über mehrere Verbindungen verfügen, die ihrer Umgebung zugeordnet sind. Ein Verbindungs-Handle zuzuweisen heißt nicht, eine Verbindung herzustellen. Ein Verbindungs-Handle muss zunächst zugewiesen und dann verwendet werden, wenn die Verbindung hergestellt ist.

Der folgende Code veranschaulicht, wie ein Verbindungs-Handle zugewiesen wird:

```
SQLRETURN rc;
SQLHDBC dbc;
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

- **Anweisung** Ein Anweisungs-Handle bietet Zugriff auf eine SQL-Anweisung und damit verbundene Daten, wie etwa Ergebnisgruppen und Parameter. Jede Verbindung kann mehrere Anweisungen haben. Anweisungen werden sowohl für Cursorvorgänge (Daten abrufen) als auch für die Ausführung einfacher Anweisungen benutzt (z.B. INSERT, UPDATE und DELETE).

Der folgende Code veranschaulicht, wie ein Anweisungs-Handle zugewiesen wird:

```
SQLRETURN rc;
SQLHSTMT stmt;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

ODBC-Handles zuweisen

Die für ODBC-Programme erforderlichen Handle-Typen sind:

Element	Handle-Typ
Umgebung	SQLHENV
Verbindung	SQLHDBC
Anweisung	SQLHSTMT
Deskriptor	SQLHDESC

Um ein ODBC-Handle zu verwenden, müssen Sie die folgenden Aufgaben ausführen:

1. Rufen Sie die SQLAllocHandle-Funktion auf.
2. Das Handle in nachfolgenden Funktionsaufrufen verwenden.
3. Geben Sie das Objekt mit SQLFreeHandle frei.

SQLAllocHandle verwendet die folgenden Parameter:

- Einen Bezeichner für den Typ des zuzuweisenden Elements
- Das Handle des übergeordneten Elements
- Einen Zeiger auf den Ort, an dem das Handle zugewiesen wird

Weitere Hinweise finden Sie unter `SQLAllocHandle` in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms712455.aspx>.

`SQLFreeHandle` verwendet die folgenden Parameter:

- Einen Bezeichner für den Typ des freizugebenden Elements
- Das Handle des freizugebenden Elements

Weitere Hinweise finden Sie unter `SQLFreeHandle` in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms710123.aspx>.

Beispiel

Mit dem folgenden Codefragment wird ein Umgebungs-Handle zugewiesen und freigegeben:

```
SQLRETURN rc;
SQLHENV env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
if( rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO )
{
    .
    .
    .
}
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

Siehe auch

- „Fehlerbehandlung in ODBC“ auf Seite 397

Beispiel für ODBC

Ein einfaches ODBC-Programm, das eine Verbindung mit der SQL Anywhere-Beispieldatenbank herstellt und diese sofort wieder trennt, finden sie in `%SQLANYWHERE%\SQLAnywhere\ODBCConnect\odbccconnect.cpp`. Dieses Beispiel zeigt die erforderlichen Schritte zum Einrichten der Umgebung, um eine Verbindung mit einem Datenbankserver aufzunehmen, sowie die notwendigen Schritte für die Trennung vom Server und Freigeben der Ressourcen.

ODBC-Verbindungsfunktionen

ODBC bietet eine Reihe von Verbindungsfunktionen. Welche Sie verwenden, hängt davon ab, wie Ihre Anwendung bereitgestellt und benutzt werden soll:

- **SQLConnect** Die einfachste Verbindungsfunktion.

`SQLConnect` verwendet einen Datenquellennamen und eine optionale Benutzer-ID sowie ein optionales Benutzerkennwort. Verwenden Sie `SQLConnect` zur Hartkodierung eines Datenquellennamens in Ihrer Anwendung.

Weitere Hinweise finden Sie unter "`SQLConnect`" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms711810.aspx>.

- **SQLDriverConnect** Stellt mithilfe einer Verbindungszeichenfolge eine Verbindung zu einer Datenquelle her.

Mit SQLDriverConnect kann die Anwendung SQL Anywhere-spezifische Verbindungsdaten verwenden, die unabhängig von der Datenquelle sind. Darüber hinaus können Sie mit SQLDriverConnect anfordern, dass der SQL Anywhere-Treiber zur Eingabe von Verbindungsdaten auffordert.

Des Weiteren lassen sich mit SQLDriverConnect Verbindungen ohne Angabe einer Datenquelle herstellen. Stattdessen wird der SQL Anywhere-ODBC-Treibername angegeben. Im folgenden Beispiel wird eine Verbindung mit einem Server und einer Datenbank hergestellt, die bereits ausgeführt werden.

```
SQLSMALLINT cso;
SQLCHAR      scso[2048];

SQLDriverConnect( hdbc, NULL,
    "Driver=SQL Anywhere 16;UID=DBA;PWD=sql", SQL_NTS,
    scso, sizeof(scso)-1,
    &cso, SQL_DRIVER_NOPROMPT );
```

Weitere Hinweise finden Sie unter "SQLDriverConnect" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms715433.aspx>.

- **SQLBrowseConnect** Stellt wie SQLDriverConnect mithilfe einer Verbindungszeichenfolge eine Verbindung zu einer Datenquelle her.

Mit SQLBrowseConnect kann Ihre Anwendung eigene Fenster erstellen, die zur Eingabe von Verbindungsdaten auffordern und nach Datenquellen suchen, die von einem bestimmten Treiber verwendet werden (in diesem Fall der SQL Anywhere-Treiber).

Weitere Hinweise finden Sie unter "SQLBrowseConnect" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms714565.aspx>.

Siehe auch

- „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)]

ODBC-Verbindung einrichten

Richten Sie in Ihrer Anwendung eine ODBC-Verbindung ein, um Datenbankvorgänge auszuführen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Ein vollständiges Beispiel finden Sie in `%SQLANYSAMPI6%\SQLAnywhere\ODBCConnect\odbccconnect.cpp`.

Aufgabe

1. ODBC-Umgebung zuweisen.

Beispiel:

```
SQLRETURN rc;  
SQLHENV env;  
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

2. ODBC-Version deklarieren.

Wenn Sie deklarieren, dass die Anwendung ODBC Version 3 befolgen soll, werden SQLSTATE-Werte und einige andere versionsabhängige Funktionen auf das entsprechende Verhalten eingestellt.
Beispiel:

```
rc = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0 );
```

3. Ein ODBC-Verbindungselement zuweisen.

Beispiel:

```
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

4. Diejenigen Verbindungsattribute festlegen, die vor dem Verbinden eingerichtet sein müssen.

Einige Verbindungsattribute können nur vor bzw. nach dem Herstellen einer Verbindung festgelegt werden, während andere davor ebenso wie danach festgelegt werden können. Das Attribut SQL_AUTOCOMMIT kann beispielsweise vor oder nach dem Herstellen der Verbindung festgelegt werden:

```
rc = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,  
(SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
```

In der Standardeinstellung arbeitet ODBC im Autocommit-Modus. Dieser Modus wird abgeschaltet, indem SQL_AUTOCOMMIT auf FALSE gesetzt wird.

5. Falls erforderlich, die Datenquelle oder Verbindungszeichenfolge zusammenstellen.

Je nach Ihrer Anweisung können Sie eine Datenquelle oder Verbindungszeichenfolge hartcodieren oder sie für erhöhte Flexibilität extern speichern.

6. ODBC-Verbindungsfunktion aufrufen.

Beispiel:

```
if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)  
{  
    printf( "dbc allocated\n" );  
    rc = SQLConnect( dbc,  
        (SQLCHAR *) "SQL Anywhere 16 Demo", SQL_NTS,  
        (SQLCHAR *) "DBA", SQL_NTS,  
        (SQLCHAR *) "sql", SQL_NTS );  
    if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)  
    {  
        // Successfully connected.    }  
}
```

Jede Zeichenfolge, die an ODBC übergeben wird, hat eine entsprechende Länge. Ist die Länge unbekannt, können Sie `SQL_NTS` als Argument verwenden, um anzuzeigen, dass es sich um eine **nullterminierte Zeichenfolge** handelt, deren Ende durch das Nullzeichen gekennzeichnet ist (`\0`).

Ergebnisse

Wenn die Anwendung aufgebaut wurde und ausgeführt wird, stellt sie eine ODBC-Verbindung her.

Siehe auch

- „Verbindungsattribute festlegen“ auf Seite 373

Verbindungsattribute festlegen

Die Funktion `SQLSetConnectAttr` wird eingesetzt, um Verbindungsdetails zu steuern. Zum Beispiel deaktiviert die folgende Anweisung das ODBC-Autocommit-Verhalten.

```
rc = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
0 );
```

Weitere Hinweise finden Sie unter "SQLSetConnectAttr" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms713605.aspx>.

Viele Aspekte der Verbindung können über die Verbindungsparameter gesteuert werden.

Siehe auch

- „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)]

Verbindungsattribute abrufen

Die Funktion `SQLGetConnectAttr` wird eingesetzt, um Verbindungsdetails abzurufen. Die folgende Anweisung gibt beispielsweise den Verbindungsstatus zurück.

```
rc = SQLGetConnectAttr( dbc, SQL_ATTR_CONNECTION_DEAD,
(SQLPOINTER)&closed, SQL_IS_INTEGER, 0 );
```

Bei der Verwendung der ODBC-Funktion `SQLGetConnectAttr` zum Abrufen des Attributs `SQL_ATTR_CONNECTION_DEAD` wird der Wert `SQL_CD_TRUE` zurückgegeben, wenn die Verbindung unterbrochen wurde, auch wenn seit der Unterbrechung der Verbindung keine Anforderung an den Server gesendet wurde. Es wird ermittelt, ob die Verbindung unterbrochen wurde, ohne eine Anforderung an den Server zu senden. Die Unterbrechung der Verbindung wird innerhalb weniger Sekunden erkannt. Die Verbindung kann aus mehreren Gründen unterbrochen werden, z.B. aufgrund eines Inaktivitäts-Timeouts.

Weitere Hinweise sowie eine Liste mit Verbindungsattributen finden Sie unter "SQLSetConnectAttr" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms710297.aspx>.

Threads und Verbindungen in ODBC-Anwendungen

Sie können für SQL Anywhere ODBC-Anwendungen mit mehreren Threads entwickeln. Es wird empfohlen, dass Sie für jeden Thread eine eigene Verbindung benutzen.

Sie können für mehrfache Threads eine einfache Verbindung benutzen. Der Datenbankserver erlaubt allerdings nicht mehr als eine aktive Anforderung für eine Verbindung zur gleichen Zeit. Falls ein Thread eine Anweisung ausführt, die lange braucht, müssen alle anderen Threads warten, bis die Anforderung abgeschlossen ist.

Von ODBC geänderte Serveroptionen

Der SQL Anywhere ODBC-Treiber setzt beim Herstellen der Verbindung zu einer SQL Anywhere-Datenbank einige temporäre Serveroptionen. Die folgenden Optionen werden so eingestellt wie angegeben.

- **date_format** `jjjj-mm-tt`
- **date_order** `jmt`
- **isolation_level** basierend auf der Attributeinstellung `SQL_ATTR_TXN_ISOLATION/SA_SQL_ATTR_TXN_ISOLATION` von `SQLSetConnectAttr`. Die folgenden Optionen stehen zur Verfügung.

```
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_READ_COMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE
SA_SQL_TXN_SNAPSHOT
SA_SQL_TXN_STATEMENT_SNAPSHOT
SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
```

- **time_format** `hh:nn:ss`
- **timestamp_format** `jjjj-mm-tt hh:nn:ss.sssss`
- **timestamp_with_time_zone_format** `jjjj-mm-tt hh:nn:ss.sssss +hh:nn`

Um die Standard-Optionseinstellung wiederherzustellen, führen Sie eine SET-Anweisung aus. Die Anweisung im folgenden Beispiel setzt die Option `timestamp_format` zurück.

```
set temporary option timestamp_format =
```

Siehe auch

- „[ODBC-Isolationsstufen](#)“ auf Seite 388

Erweiterte Verbindungsattribute für SQLSetConnectAttr

Der SQL Anywhere ODBC-Treiber unterstützt einige erweiterte Verbindungsattribute.

- **SA_REGISTER_MESSAGE_CALLBACK** Nachrichten können unter Verwendung der SQL MESSAGE-Anweisung vom Datenbankserver an die Clientanwendung gesendet werden. Außerdem können Nachrichten aufgrund von lang laufenden Datenbankserver-Anweisungen generiert werden.

Es kann eine Message-Handler-Routine erstellt werden, um diese Nachrichten abzufangen. Der Message-Handler Callback-Prototyp sieht wie folgt aus:

```
void SQL_CALLBACK message_handler(
SQLHDBC sqlany_dbc,
unsigned char msg_type,
long code,
unsigned short length,
char * message
);
```

Die folgenden zulässigen Werte für Nachrichtentyp *msg_type* sind in *sqldef.h* festgelegt.

- **MESSAGE_TYPE_INFO** Der Nachrichtentyp war INFO.
- **MESSAGE_TYPE_WARNING** Der Nachrichtentyp war WARNING.
- **MESSAGE_TYPE_ACTION** Der Nachrichtentyp war ACTION.
- **MESSAGE_TYPE_STATUS** Der Nachrichtentyp war STATUS.
- **MESSAGE_TYPE_PROGRESS** Der Nachrichtentyp war PROGRESS. Dieser Nachrichtentyp wird aufgrund von lang laufenden Datenbankserver-Anweisungen wie BACKUP DATABASE und LOAD TABLE generiert.

Ein SQLCODE, der der Nachricht zugeordnet ist, kann in *code* übermittelt werden. Wenn keiner verfügbar ist, ist der *code*-Parameterwert 0.

Die Länge der Nachricht ist in *length* enthalten.

Ein Zeiger auf die Nachricht ist in *message* enthalten. Die Nachrichtenzeichenfolge hat kein NULL-Abschlusszeichen. Ihre Anwendung muss in der Lage sein, dies zu verarbeiten. Im Folgenden finden Sie ein Beispiel.

```
memcpy( mybuff, msg, len );
mybuff[ len ] = '\0';
```

Um den Message-Handler in ODBC zu registrieren, rufen Sie die Funktion SQLSetConnectAttr folgendermaßen auf:

```
rc = SQLSetConnectAttr(
hdbc,
SA_REGISTER_MESSAGE_CALLBACK,
(SQLPOINTER) &message_handler, SQL_IS_POINTER );
```

Um die Registrierung des Message-Handlers in ODBC aufzuheben, rufen Sie die Funktion `SQLSetConnectAttr` folgendermaßen auf:

```
rc = SQLSetConnectAttr(  
    hdbc,  
    SA_REGISTER_MESSAGE_CALLBACK,  
    NULL, SQL_IS_POINTER );
```

- **SA_GET_MESSAGE_CALLBACK_PARM** Um den Wert des SQLHDBC-Verbindungshandles abzufragen, der an die Message-Handler Callback-Routine übergeben wird, verwenden Sie `SQLGetConnectAttr` mit dem Parameter `SA_GET_MESSAGE_CALLBACK_PARM`.

```
SQLHDBC callback_hdbc = NULL;  
rc = SQLGetConnectAttr(  
    hdbc,  
    SA_GET_MESSAGE_CALLBACK_PARM,  
    (SQLPOINTER) &callback_hdbc, 0, 0 );
```

Der Rückgabewert ist derselbe wie der Parameterwert, der an die Message-Handler Callback-Routine übergeben wird.

- **SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK** Dies wird verwendet, um eine Callback-Funktion zur Dateiübertragungsvalidierung zu registrieren. Bevor er eine Übertragung zulässt, ruft der Treiber die Validierungs-Callback-Funktion auf, falls vorhanden. Falls die Client-Datenübertragung während der Ausführung von indirekten Anweisungen angefordert wird, lässt der ODBC-Treiber die Übertragung nur zu, wenn die Clientanwendung einen Validierungs-Callback registriert hat. Die Bedingungen, unter denen ein Validierungsaufruf durchgeführt wird, werden unten ausführlich beschrieben.

Der Callback-Prototyp sieht wie folgt aus:

```
int SQL_CALLBACK file_transfer_callback(  
    void * sqlca,  
    char * file_name,  
    int is_write  
);
```

Der Parameter *file_name* ist der Name der zu lesenden oder zu schreibenden Datei. Der Parameter *is_write* ist 0, wenn ein Lesevorgang (Übertragung vom Client zum Datenbankserver) angefordert wird, und Nicht-Null bei einem Schreibvorgang. Die Callback-Funktion sollte 0 zurückgeben, wenn die Übertragung nicht zulässig ist, ansonsten Nicht-Null.

Aus Gründen der Datensicherheit protokolliert der Datenbankserver den Ursprung von Anweisungen, die eine Dateiübertragung anfordern. Der Datenbankserver ermittelt, ob die Anweisung direkt von der Clientanwendung empfangen wurde. Wenn er die Datenübertragung vom Client initiiert, sendet der Datenbankserver die Informationen über den Ursprung der Anweisung an die Clientsoftware. Ihrerseits erlaubt der ODBC-Treiber nur dann eine bedingungslose Datenübertragung, wenn diese aufgrund der Ausführung einer Anweisung angefordert wird, die direkt von der Clientanwendung gesendet wurde. Ansonsten muss die Anwendung den oben beschriebenen Validierungs-Callback registriert haben. Wenn dieser fehlt, wird die Übertragung verweigert und die Anweisung schlägt mit einem Fehler fehl. Wenn die Clientanweisung eine bereits in der Datenbank vorhandene gespeicherte Prozedur aufruft, wird die Ausführung der gespeicherten Prozedur selbst nicht als vom Client initiierte Anweisung angesehen. Wenn die Clientanwendung allerdings explizit eine temporäre gespeicherte

Prozedur erstellt, führt die Ausführung der gespeicherten Prozedur dazu, dass der Datenbankserver die Prozedur als eine vom Client initiierte behandelt. Wenn die Clientanwendung eine Batchanweisung ausführt, wird auf gleiche Weise die Ausführung der Batchanweisung als eine angesehen, die direkt von der Clientanwendung durchgeführt wird.

- **SA_SQL_ATTR_TXN_ISOLATION** Dies wird verwendet, um eine erweiterte Transaktionsisolationsstufe zu setzen. Im folgenden Beispiel wird eine Snapshot-Isolationsstufe gesetzt:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SA_SQL_ATTR_TXN_ISOLATION,
                  SA_SQL_TXN_SNAPSHOT, SQL_IS_INTEGER );
```

Siehe auch

- „MESSAGE-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „progress_messages-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „ODBC-Isolationsstufen“ auf Seite 388

Hinweise zur Windows-Funktion DllMain

ODBC-Funktionen dürfen in einer Windows-DLL nicht direkt oder indirekt von der DllMain-Funktion aufgerufen werden. Die DllMain-Eintrittspunktfunktion soll nur einfache Initialisierungs- und Beendigungsaufgaben ausführen. Der Aufruf von ODBC-Funktionen wie SQLFreeHandle, SQLFreeConnect und SQLFreeEnv können Deadlocks und eine Zirkeldefinition bewirken.

Das folgende Codebeispiel zeigt eine fehlerhafte Programmierung. Wenn der Microsoft ODBC-Treibermanager erkennt, dass der letzte Zugriff auf den SQL Anywhere ODBC-Treiber abgeschlossen wurde, wird der Treiber entladen. Wenn der SQL Anywhere ODBC-Treiber heruntergefahren wird, stoppt er alle aktiven Threads. Die Threadbeendigung führt zu einem rekursiven Thread-Ablösungsaufruf in DllMain. Da der Aufruf in DllMain serialisiert ist und ein Aufruf läuft, wird der neue Thread-Ablösungsaufruf nie gestartet. Der SQL Anywhere ODBC-Treiber wartet unbegrenzt, dass diese Threads beendet werden und Ihre Anwendung hängt.

```
BOOL WINAPI DllMain( HMODULE hinstDLL,
                    DWORD fdwReason,
                    LPVOID lpvReserved
                    )
{
    HANDLE *handles;
    switch( fdwReason ) {
        .
        .
        .
        case DLL_THREAD_DETACH:
            /* do thread cleanup */
            handles = (HANDLE *) TlsGetValue( TlsIndex );
            if( handles != NULL )
            {
                SQLHENV    tls_henv;
                SQLHDBC     tls_hdbc;

                tls_henv = (SQLHENV) handles[0];
                tls_hdbc = (SQLHDBC) handles[1];
```

```
        if( tls_hdbc != NULL )
            SQLFreeHandle( SQL_HANDLE_DBC, tls_hdbc );
        if( tls_henv != NULL )
            SQLFreeHandle( SQL_HANDLE_ENV, tls_henv );
        handles[0] = NULL;
        handles[1] = NULL;
    }
    break;
    .
    .
    .
    }
    return TRUE;          /* indicate success */
}
```

Weitere Informationen finden Sie im Microsoft-Whitepaper *Best Practices for Creating DLLs* unter <http://msdn.microsoft.com/de-de/windows/hardware/gg487379.aspx>.

Möglichkeiten zum Ausführen von SQL-Anweisungen

ODBC umfasst mehrere Funktionen zum Ausführen von SQL-Anweisungen:

- **Direkte Ausführung** SQL Anywhere führt eine syntaktische Analyse der SQL-Anweisung durch, bereitet einen Zugriffsplan vor und führt die Anweisung aus. Das syntaktische Analysieren und das Vorbereiten des Zugriffsplans wird **Vorbereitung** der Anweisung genannt.
- **Vorbereitete Ausführung** Die Vorbereitung der Anweisung erfolgt getrennt von der Ausführung. Bei Anweisungen, die wiederholt ausgeführt werden sollen wird damit vermieden, dass wiederholt vorbereitet werden muss. Damit wird die Performance verbessert.

Siehe auch

- „Vorbereitete Anweisungen ausführen“ auf Seite 380

Direkte Ausführung von Anweisungen

Wenn Sie eine SQL-Anweisung in einer ODBC-Anwendung ausführen möchten, weisen Sie der Anweisung mit **SQLAllocHandle** ein Handle zu und rufen Sie anschließend die **SQLExecDirect**-Funktion auf, um die Anweisung auszuführen. Alle Parameter (z.B. die Argumente einer WHERE-Klausel) müssen in der Anweisung enthalten sein. Eine flexiblere Methode zum Konstruieren von Anweisungen finden Sie unter „Anweisungen mit gebundenen Parametern ausführen“ auf Seite 379.

Die **SQLExecDirect**-Funktion verwendet ein Statement Handle, eine SQL-Zeichenfolge und einen Längen- oder Abschlussindikator (in diesem Fall einen Indikator für Zeichenfolgen mit NULL-Abschlusszeichen). Die Anweisung kann Parameter enthalten.

Weitere Hinweise zu **SQLExecDirect** finden Sie in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms713611.aspx>.

Ein vollständiges Beispiel mit Fehlerprüfung finden Sie in %SQLANYSAMPI6%\SQLAnywhere\ODBCExecute\odbcexecute.cpp.

Beispiel

Das folgende Beispiel zeigt, wie ein SQL_HANDLE_STMT-Handle namens stmt auf einer Verbindung mit dem Handle "dbc" zugewiesen wird:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

Das folgende Beispiel zeigt, wie Sie eine Anweisung deklarieren und ausführen:

```
SQLCHAR deletestmt[ STMT_LEN ] =  
    "DELETE FROM Departments WHERE DepartmentID = 201";  
SQLExecDirect( stmt, deletestmt, SQL_NTS );
```

Die deletestmt-Deklaration sollte in der Regel am Anfang der Funktion erfolgen.

Anweisungen mit gebundenen Parametern ausführen

Konstruieren Sie SQL-Anweisungen, in denen Werte für Anweisungsparameter zur Laufzeit mit gebundenen Parametern festgelegt werden, und führen Sie sie aus. Gebundene Parameter werden auch mit vorbereiteten Anweisungen verwendet, um die Performance von Anweisungen zu steigern, die mehr als einmal ausgeführt werden.

Voraussetzungen

Um das Beispiel erfolgreich ausführen zu können, benötigen Sie die folgenden Systemprivilegien.

- INSERT für die Tabelle "Departments"

Aufgabe

1. Weisen Sie der Anweisung mit SQLAllocHandle ein Handle zu.

Mit der folgenden Anweisung wird z.B. auf einer Verbindung mit dem Handle SQL_HANDLE_STMT ein Handle vom Typ stmt mit dem Namen dbc zugewiesen:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

2. Binden Sie Parameter für die Anweisung mit SQLBindParameter.

Zum Beispiel werden mit den folgenden Zeilen Variable deklariert, die Werte für Abteilungs-ID, Abteilungsname und Manager-ID sowie die Anweisungszeichenfolge enthalten sollen. Anschließend werden Parameter an den ersten, zweiten und dritten Parameter einer Anweisung gebunden, die mit dem Anweisungs-Handle stmt ausgeführt wird.

```
#defined DEPT_NAME_LEN 40  
SQLLEN cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;  
SQLSMALLINT deptID, managerID;  
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];  
  
SQLCHAR insertstmt[ STMT_LEN ] =
```

```
"INSERT INTO Departments "  
" ( DepartmentID, DepartmentName, DepartmentHeadID ) "  
"VALUES ( ?, ?, ?)";  
  
SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,  
    SQL_C_SSHORT, SQL_INTEGER, 0, 0,  
    &deptID, 0, &cbDeptID );  
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,  
    SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,  
    deptName, 0, &cbDeptName );  
SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,  
    SQL_C_SSHORT, SQL_INTEGER, 0, 0,  
    &managerID, 0, &cbManagerID );
```

3. Den Parametern Werte zuweisen.

Mit den folgenden Zeilen werden z.B. den Parametern im Fragment von Schritt 2 Werte zugewiesen.

```
deptID = 201;  
strcpy( (char * ) deptName, "Sales East" );  
managerID = 902;
```

Im Allgemeinen werden diese Variablen als Reaktion auf eine Benutzerhandlung festgelegt.

4. Führen Sie die Anweisung mit SQLExecDirect aus.

Mit der folgenden Zeile wird z.B. eine Anweisungszeichenfolge ausgeführt, die sich in `insertstmt` im Anweisungs-Handle `stmt` befindet.

```
SQLExecDirect( stmt, insertstmt, SQL_NTS );
```

Ergebnisse

Wenn die Anwendung aufgebaut und ausgeführt wird, führt sie die SQL-Anweisungen aus.

Nächste Schritte

Die obigen Codefragmente enthalten keine Fehlerprüfung. Ein vollständiges Beispiel mit Fehlerprüfung finden Sie in `%SQLANYSAMPI6%\SQLAnywhere\ODBCExecute\odbcexecute.cpp`.

Weitere Hinweise zu `SQLExecDirect` finden Sie in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms713611.aspx>.

Siehe auch

- „Vorbereitete Anweisungen ausführen“ auf Seite 380

Vorbereitete Anweisungen ausführen

Vorbereitete Anweisungen bieten Performance-Vorteile bei wiederholt verwendeten Anweisungen.

Voraussetzungen

Um das Beispiel erfolgreich ausführen zu können, benötigen Sie die folgenden Systemprivilegien.

- INSERT für die Tabelle "Departments"

Aufgabe

1. Bereiten Sie die Anweisung mit SQLPrepare vor.

Das folgende Codefragment veranschaulicht z.B., wie eine INSERT-Anweisung vorbereitet wird:

```
rc = SQLPrepare( stmt,
    "INSERT INTO Departments( DepartmentID, DepartmentName,
    DepartmentHeadID ) "
    "VALUES ( ?, ?, ? )",
    SQL_NTS );
```

In diesem Beispiel gilt:

- **rc** Empfängt einen Rückgabecode, der auf Erfolg oder Fehlschlag des Vorgangs getestet werden sollte.
- **stmt** Liefert ein Handle zu der Anweisung, sodass sie später referenziert werden kann.
- **?** Die Fragezeichen sind Platzhalter für Anweisungsparameter. Ein Platzhalter wird in die Anweisung eingefügt, um anzuzeigen, wo auf Hostvariablen zugegriffen werden soll. Ein Platzhalter ist entweder ein Fragezeichen (?) oder eine Referenz auf eine Hostvariable wie in statischen Anweisungen (ein Hostvariablenname mit einem vorangestellten Doppelpunkt). Im letzteren Fall dient der Name der Hostvariable, der im Text der Anweisung benutzt wird, nur als Platzhalter für eine Referenz auf den SQL-Deskriptorbereich. Er muss nicht mit dem tatsächlichen Parameternamen übereinstimmen.

2. Binden Sie die Werte der Anweisungsparameter mit SQLBindParameter.

Der folgende Funktionsaufruf bindet beispielsweise den Wert der DepartmentID-Variablen:

```
rc = SQLBindParameter( stmt,
    1,
    SQL_PARAM_INPUT,
    SQL_C_SSHORT,
    SQL_INTEGER,
    0,
    0,
    &sDeptID,
    0,
    &cbDeptID );
```

In diesem Beispiel gilt:

- **rc** Enthält einen Rückgabecode, der auf Erfolg oder Fehlschlag des Vorgangs überprüft werden sollte
- **stmt** ist das Anweisungs-Handle.
- **1** zeigt an, dass dieser Aufruf den ersten Platzhalter mit einem Wert belegt.
- **SQL_PARAM_INPUT** zeigt an, dass der Parameter eine Eingabeanweisung ist.
- **SQL_C_SHORT** zeigt den C-Datentyp an, der in der Anwendung verwendet wird.
- **SQL_INTEGER** zeigt den SQL-Datentyp an, der in der Datenbank verwendet wird.

Die nächsten beiden Parameter zeigen die Spaltengröße und die Anzahl der Dezimalstellen an; beide sind mit 0 (Null) für Ganzzahlen belegt.

- **&sDeptID** ist ein Zeiger auf einen Puffer für den Parameterwert.
- **0** gibt die Länge des Puffers in Byte an.
- **&cbDeptID** ist ein Zeiger auf einen Puffer für die Länge des Parameterwerts.

3. Binden Sie die beiden anderen Parameter, und weisen Sie sDeptId Werte zu.
4. Führen Sie die Anweisung aus:

```
rc = SQLExecute( stmt );
```

Die Schritte 2 bis 4 können mehrfach ausgeführt werden.

5. Löschen Sie die Anweisung.

Das Löschen der Anweisung setzt die der Anweisung zugewiesenen Ressourcen frei. Anweisungen werden mit SQLFreeHandle gelöscht.

Ergebnisse

Wenn die Anwendung aufgebaut und ausgeführt wird, führt sie die vorbereiteten Anweisungen aus.

Nächste Schritte

Die obigen Codefragmente enthalten keine Fehlerprüfung. Ein vollständiges Beispiel mit Fehlerprüfung finden Sie in %SQLANYSAMPI6%\SQLAnywhere\ODBCPrepare\odbcprepare.cpp.

Weitere Hinweise zu SQLPrepare finden Sie unter "SQLPrepare" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms710926.aspx>.

Siehe auch

- „Vorbereitete Anweisungen“ auf Seite 2

Hinweise zu 64-Bit-ODBC

Wenn Sie eine ODBC-Funktion wie SQLBindCol, SQLBindParameter oder SQLGetData verwenden, werden einige Parameter als SQLLEN oder SQLULEN im Funktionsprototyp typisiert. Abhängig vom Erscheinungsdatum Ihrer Microsoft-ODBC-API-Referenzdokumentation kann derselbe Parameter als SQLINTEGER oder SQLUINTEGER beschrieben sein.

SQLLEN- und SQLULEN-Datenelemente sind 64 Bit in einer 64-Bit-ODBC-Anwendung und 32 Bit in einer 32-Bit-ODBC-Anwendung. SQLINTEGER- und SQLUINTEGER-Datenelemente sind 32 Bit auf allen Plattformen.

Um das Problem zu illustrieren, wurde der folgende ODBC-Funktionsprototyp einer älteren Ausgabe der Microsoft-ODBC-API-Referenz entnommen.

```
SQLRETURN SQLGetData(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLSMALLINT    TargetType,
    SQLPOINTER     TargetValuePtr,
    SQLINTEGER     BufferLength,
    SQLINTEGER     *StrLen_or_IndPtr);
```

Vergleichen Sie dies mit dem Funktionsprototyp, wie er in *sql.h* in Microsoft Visual Studio Version 8 zu finden ist.

```
SQLRETURN SQL_API SQLGetData(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLSMALLINT    TargetType,
    SQLPOINTER     TargetValue,
    SQLLEN        BufferLength,
    SQLLEN        *StrLen_or_Ind);
```

Wie Sie sehen, werden die Parameter *BufferLength* und *StrLen_or_Ind* nun als *SQLLEN* und nicht als *SQLINTEGER* typisiert. Bei der 64-Bit-Plattform sind sie 64-Bit-Quantitäten und keine 32-Bit-Quantitäten, wie dies in der Microsoft-Dokumentation angegeben ist.

Um Probleme bei plattformübergreifenden Kompilierungen zu vermeiden, stellt SQL Anywhere eigene ODBC-Header-Dateien zur Verfügung. Bei Windows-Plattformen sollten Sie die *ntodbc.h*-Header-Datei aufnehmen. Bei Unix-Plattformen wie Linux sollten Sie die *unixodbc.h*-Header-Datei aufnehmen. Die Verwendung dieser Header-Dateien gewährleistet Kompatibilität mit dem entsprechenden SQL Anywhere ODBC-Treiber für die Zielplattform.

Die folgende Tabelle listet einige gebräuchliche ODBC-Typen auf, die dieselben bzw. verschiedene Speichergrößen auf 64-Bit- und 32-Bit-Plattformen haben.

ODBC-API	64-Bit-Plattform	32-Bit-Plattform
SQLINTEGER	32 Bit	32 Bit
SQLUIINTEGER	32 Bit	32 Bit
SQLLEN	64 Bit	32 Bit
SQLULEN	64 Bit	32 Bit
SQLSETPOSIROW	64 Bit	16 Bit
SQL_C_BOOKMARK	64 Bit	32 Bit
BOOKMARK	64 Bit	32 Bit

Wenn Sie Datenvariablen und Parameter nicht korrekt deklarieren, kann das zu fehlerhaftem Software-Verhalten führen.

Die folgende Tabelle fasst die Funktionsprototypen der ODBC-API zusammen, die sich mit Einführung der 64-Bit-Unterstützung geändert haben. Die betroffenen Parameter werden aufgelistet. Der in

Microsoft-Dokumentationen verwendete Parametername wird in Klammern angeführt, wenn er sich vom Parameternamen im Funktionsprototyp unterscheidet. Die Parameternamen sind die, die in den Microsoft Visual Studio Version 8-Header-Dateien verwendet werden.

ODBC-API	Parameter (dokumentierter Parametername)
SQLBindCol	SQLLEN BufferLength SQLLEN *Strlen_or_Ind
SQLBindParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind
SQLBindParameter	SQLULEN cbColDef (ColumnSize) SQLLEN cbValueMax (BufferLength) SQLLEN *pcbValue (Strlen_or_IndPtr)
SQLColAttribute	SQLLEN *NumericAttribute
SQLColAttributes	SQLLEN *pfDesc
SQLDescribeCol	SQLULEN *ColumnSize (ColumnSizePtr)
SQLDescribeParam	SQLULEN *pcbParamDef (ParameterSizePtr)
SQLExtendedFetch	SQLLEN irow (FetchOffset) SQLULEN *pcrow (RowCountPtr)
SQLFetchScroll	SQLLEN FetchOffset
SQLGetData	SQLLEN BufferLength SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLGetDescRec	SQLLEN *Length (LengthPtr)
SQLParamOptions	SQLULEN crow, SQLULEN *pirow
SQLPutData	SQLLEN Strlen_or_Ind
SQLRowCount	SQLLEN *RowCount (RowCountPtr)
SQLSetConnectOption	SQLULEN Value

ODBC-API	Parameter (dokumentierter Parametername)
SQLSetDescRec	SQLLEN Length SQLLEN *StringLength (StringLengthPtr) SQLLEN *Indicator (IndicatorPtr)
SQLSetParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLSetPos	SQLSETPOSIROW irow (RowNumber)
SQLSetScrollOptions	SQLLEN crowKeyset
SQLSetStmtOption	SQLULEN Value

Manche Werte, die von ODBC-API-Aufrufen übergeben und über Zeiger zurückgegeben werden, haben sich geändert, um 64-Bit-Anwendungen zu entsprechen. Beispiel: Die folgenden Werte für die SQLSetStmtAttr- und SQLSetDescField-Funktionen sind nicht mehr SQLINTEGER/SQULINTEGER. Die gleiche Regel gilt auch für die entsprechenden Parameter für die SQLGetStmtAttr- und SQLGetDescField-Funktionen.

ODBC-API	Typ für Value/ValuePtr-Variable
SQLSetStmtAttr(SQL_ATTR_FETCH_BOOKMARK_PTR)	SQLLEN * value
SQLSetStmtAttr(SQL_ATTR_KEYSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_LENGTH)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_ROWS)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_PARAM_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMS_PROCESSED_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_ARRAY_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_ROW_NUMBER)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROWS_FETCHED_PTR)	SQLULEN * value

ODBC-API	Typ für Value/ValuePtr-Variable
SQLSetDescField(SQL_DESC_ARRAY_SIZE)	SQLULEN value
SQLSetDescField(SQL_DESC_BIND_OFFSET_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_ROWS_PROCESSED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_DISPLAY_SIZE)	SQLLEN value
SQLSetDescField(SQL_DESC_INDICATOR_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH_PTR)	SQLLEN * value

Weitere Hinweise finden Sie im Microsoft-Artikel "ODBC 64-Bit API Changes in MDAC 2.7" unter <http://support.microsoft.com/kb/298678>.

Datenausrichtungsanforderungen

Wenn Sie SQLBindCol, SQLBindParameter oder SQLGetData verwenden, wird ein "C"-Datentyp für die Spalte oder den Parameter festgelegt. Auf bestimmten Plattformen muss der für die einzelnen Spalten bereitgestellte Speicher richtig ausgerichtet sein, um einen Wert des angegebenen Typs abrufen oder speichern zu können. Der ODBC-Treiber prüft auf richtige Datenausrichtung. Wenn ein Objekt nicht richtig ausgerichtet ist, gibt der ODBC-Treiber die Meldung **Ungültige Zeichenfolgen- oder Pufferlänge** aus (SQLSTATE HY090 oder S1090).

Die folgende Tabelle listet Ausrichtungsanforderungen für Prozessoren wie Sun Sparc, Itanium-IA64 und ARM-basierte Geräte auf. Die Speicheradresse des Datenwerts muss ein Mehrfaches des angegebenen Werts sein.

C-Datentyp	Erforderliche Ausrichtung
SQL_C_CHAR	keine
SQL_C_BINARY	keine
SQL_C_GUID	keine
SQL_C_BIT	keine
SQL_C_STINYINT	keine
SQL_C_UTINYINT	keine

C-Datentyp	Erforderliche Ausrichtung
SQL_C_TINYINT	keine
SQL_C_NUMERIC	keine
SQL_C_DEFAULT	keine
SQL_C_SSHORT	2
SQL_C_USHORT	2
SQL_C_SHORT	2
SQL_C_DATE	2
SQL_C_TIME	2
SQL_C_TIMESTAMP	2
SQL_C_TYPE_DATE	2
SQL_C_TYPE_TIME	2
SQL_C_TYPE_TIMESTAMP	2
SQL_C_WCHAR	2 (Puffergröße muss auf allen Plattformen ein Mehrfaches von 2 sein)
SQL_C_SLONG	4
SQL_C_ULONG	4
SQL_C_LONG	4
SQL_C_FLOAT	4
SQL_C_DOUBLE	8 (4 für ARM)
SQL_C_SBIGINT	8
SQL_C_UBIGINT	8

Die x86-, x64- und PowerPC-Plattformen erfordern keine Speicherausrichtung. Die x64-Plattform umfasst AMD64-Prozessoren (Advanced Micro Devices) und EM64T-Prozessoren (Intel Extended Memory 64 Technology).

Ergebnismengen in ODBC-Anwendungen

ODBC-Anwendungen verwenden Cursor zum Bearbeiten und Aktualisieren von Ergebnismengen. SQL Anywhere bietet umfassende Unterstützung für verschiedene Arten von Cursor und Cursorvorgängen.

Siehe auch

- „Cursor-Grundsätze“ auf Seite 8

ODBC-Isolationsstufen

Mit `SQLSetConnectAttr` können Sie die Isolationsstufe der Transaktion für eine Verbindung festlegen. Folgende Merkmale bestimmen die von SQL Anywhere bereitgestellte Transaktions-Isolationsstufe:

- **SQL_TXN_READ_UNCOMMITTED** Die Isolationsstufe wird auf "0" gesetzt. Wenn dieser Attributwert festgelegt wird, werden alle Daten, die aus Änderungen durch andere Benutzern stammen, isoliert, und Änderungen anderer Benutzer sind nicht sichtbar. Das erneute Ausführen der Leseanweisung wird von anderen beeinflusst. Wiederholbare Lesevorgänge werden nicht unterstützt. Es handelt sich hierbei um den Standardwert für die Isolationsstufe.
- **SQL_TXN_READ_COMMITTED** Die Isolationsstufe wird auf 1 gesetzt. Wenn dieser Attributwert festgelegt wird, werden die Daten, die aus Änderungen durch andere Benutzern stammen, nicht isoliert, und Änderungen anderer Benutzer sind sichtbar. Das erneute Ausführen der Leseanweisung wird von anderen beeinflusst. Wiederholbare Lesevorgänge werden nicht unterstützt.
- **SQL_TXN_REPEATABLE_READ** Die Isolationsstufe wird auf 2 gesetzt. Wenn dieser Attributwert festgelegt wird, werden alle Daten, die aus Änderungen durch andere Benutzern stammen, isoliert, und Änderungen anderer Benutzer sind nicht sichtbar. Das erneute Ausführen der Leseanweisung wird von anderen beeinflusst. Wiederholbare Lesevorgänge werden unterstützt.
- **SQL_TXN_SERIALIZABLE** Die Isolationsstufe wird auf 3 gesetzt. Wenn dieser Attributwert festgelegt wird, werden alle Daten, die aus Änderungen durch andere Benutzern stammen, isoliert, und Änderungen anderer Benutzer sind nicht sichtbar. Das erneute Ausführen der Leseanweisung wird von anderen nicht beeinflusst. Wiederholbare Lesevorgänge werden unterstützt.
- **SA_SQL_TXN_SNAPSHOT** Die Isolationsstufe wird auf "snapshot" gesetzt. Wenn dieser Attributwert gesetzt ist, wird für die gesamte Transaktion eine einzelne Ansicht der Datenbank bereitgestellt.
- **SA_SQL_TXN_STATEMENT_SNAPSHOT** Die Isolationsstufe wird auf "statement-snapshot" gesetzt. Wenn dieser Attributwert gesetzt ist, wird eine geringere Konsistenz bereitgestellt als bei der Snapshot-Isolation. Er kann jedoch in Fällen nützlich sein, in denen lang laufende Transaktionen dazu führen, dass durch das Speichern der Versionen zu viel Speicherplatz in der temporären Datei belegt wird.
- **SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT** Die Isolationsstufe wird auf "readonly-statement-snapshot" gesetzt. Wenn dieser Attributwert gesetzt ist, wird eine geringere Konsistenz bereitgestellt als bei einer Anweisungs-Snapshot-Isolation, aber die Möglichkeit von Aktualisierungskonflikten wird vermieden. Daher ist diese Isolationsstufe am besten für die

Portierung von Anwendungen geeignet, die ursprünglich dazu gedacht waren, unter anderen Isolationsstufen zu laufen.

Die `allow_snapshot_isolation`-Datenbankoption muss auf "On" gesetzt sein, wenn Sie die Einstellung Snapshot, Statement-snapshot oder Readonly-statement-snapshot verwenden möchten.

Weitere Hinweise finden Sie unter "SQLSetConnectAttr" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms713605.aspx>.

Beispiel

Das folgende Codefragment setzt die Isolationsstufe auf "snapshot":

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION,
    SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

ODBC-Cursormerkmale

ODBC-Funktionen, die Anweisungen ausführen und Ergebnismengen verarbeiten, verwenden für diese Aufgaben Cursor. Anwendungen öffnen implizit einen Cursor, wenn sie die Funktion `SQLExecute` oder `SQLExecDirect` ausführen.

Bei Anwendungen, die sich nur vorwärts durch eine Ergebnismenge bewegen und sie nicht aktualisieren, ist das Cursor-Verhalten recht einfach. Standardmäßig erwarten ODBC-Anwendungen dieses Verhalten. ODBC definiert einen reinen Lesecursor, der nur vorwärts liest, und SQL Anywhere liefert einen Cursor, der für diesen Fall auf Performance optimiert ist.

Ein einfaches Beispiel für einen Vorwärtscursor finden Sie unter „Datenabfrage“ auf Seite 390.

Bei Anwendungen, die sowohl vorwärts als auch rückwärts durch die Ergebnismenge blättern müssen, wie etwa Anwendungen mit grafischer Benutzeroberfläche, ist das Cursor-Verhalten komplexer. Wie reagiert die Anwendung, wenn sie eine Zeile zurückgibt, die von einer anderen Anwendung aktualisiert wurde? ODBC definiert eine Reihe von **scrollfähigen Cursors**, damit Sie das gewünschte Verhalten in Ihre Anwendung einbauen können. SQL Anywhere liefert eine vollständige Gruppe von Cursors, die mit den scrollfähigen ODBC-Cursortypen übereinstimmen.

Die erforderlichen Cursor-Merkmale werden mit der Funktion `SQLSetStmtAttr` gesetzt, die die Anweisungsattribute definiert. Sie müssen `SQLSetStmtAttr` aufrufen, bevor Sie eine Anweisung ausführen, die eine Ergebnismenge erzeugt.

Viele Cursor-Merkmale können mit `SQLSetStmtAttr` festgelegt werden. Folgende Merkmale bestimmen den von SQL Anywhere gelieferten Cursortyp:

- **SQL_ATTR_CURSOR_SCROLLABLE** Setzen Sie dies für einen scrollfähigen Cursor auf `SQL_SCROLLABLE` und für einen Vorwärtscursor auf `SQL_NONSCROLLABLE`. `SQL_NONSCROLLABLE` ist der Standardwert.
- **SQL_ATTR_CONCURRENCY** Auf einen der folgenden Werte setzen:

- **SQL_CONCUR_READ_ONLY** Aktualisierungen deaktivieren. SQL_CONCUR_READ_ONLY ist der Standardwert.
- **SQL_CONCUR_LOCK** Verwendet die niedrigste Sperrstufe, die ausreicht, um eine Zeile zu aktualisieren.
- **SQL_CONCUR_ROWVER** Verwendet optimistische Parallelitätssteuerung, bei der Zeilenversionen verglichen werden, wie z.B. SQLBase ROWID oder Sybase TIMESTAMP.
- **SQL_CONCUR_VALUES** Verwendet optimistische Parallelitätssteuerung, bei der Werte verglichen werden.

Weitere Hinweise finden Sie unter "SQLSetStmtAttr" in der Microsoft-Dokumentation *ODBC API Reference* unter <http://msdn.microsoft.com/de-de/library/ms712631.aspx>.

Beispiel

Für das folgende Fragment ist ein schreibgeschützter scrollfähiger Cursor erforderlich:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE,
                SQL_SCROLLABLE, SQL_IS_UINTEGER );
```

Datenabfrage

Um Zeilen aus einer Datenbank abzurufen, führen Sie mit `SQLExecute` oder `SQLExecDirect` eine `SELECT`-Anweisung aus. Damit wird ein Cursor für die Anweisung geöffnet.

Anschließend rufen Sie mit `SQLFetch` oder `SQLFetchScroll` Zeilen über den Cursor ab. Diese Funktionen rufen die nächste Zeilengruppe aus der Ergebnismenge ab und geben Daten für alle gebundenen Spalten zurück. Mit `SQLFetchScroll` können Zeilengruppen an einer absoluten oder relativen Position oder anhand eines Lesezeichens festgelegt werden. `SQLFetchScroll` ersetzt das ältere `SQLExtendedFetch` aus der ODBC 2.0-Spezifikation.

Wenn eine Anwendung die Anweisung mithilfe von `SQLFreeHandle` freigibt, wird der Cursor geschlossen.

Um Werte von einem Cursor abzurufen, kann die Anwendung `SQLBindCol` oder `SQLGetData` verwenden. Wenn Sie `SQLBindCol` verwenden, werden die Werte automatisch mit jedem Abruf (`fetch`) abgerufen. Wenn Sie `SQLGetData` verwenden, müssen Sie sie für jede Spalte nach jedem Abruf (`fetch`) aufrufen.

Mit `SQLGetData` werden Werte in Teilen für Spalten wie `LONG VARCHAR` oder `LONG BINARY` eingelesen. Alternativ können Sie das `SQL_ATTR_MAX_LENGTH`-Anweisungsattribut auf einen Wert setzen, der groß genug ist, um den Wert für die gesamte Spalte aufzunehmen. Der Standardwert für `SQL_ATTR_MAX_LENGTH` ist 256 kB.

Der SQL Anywhere ODBC-Treiber implementiert `SQL_ATTR_MAX_LENGTH` anders, als von der ODBC-Spezifikation vorgesehen. `SQL_ATTR_MAX_LENGTH` ist als Methode zum Kürzen langer Abrufe vorgesehen. Dies ist z.B. in einem "Vorschau"-Modus möglich, wenn nur der erste Teil der Daten

angezeigt werden soll. Anstatt beispielsweise einen 4-MB-Blob vom Server an die Clientanwendung zu übertragen, könnten z.B. nur die ersten 500 Byte übertragen werden (indem SQL_ATTR_MAX_LENGTH auf 500 gesetzt wird). Der SQL Anywhere ODBC-Treiber unterstützt diese Implementierung nicht.

Mit dem folgenden Codefragment wird ein Cursor für eine Abfrage geöffnet und Daten über den Cursor abgerufen. Auf eine Fehlerprüfung wurde verzichtet, um das Beispiel leichter lesbar zu halten. Das Codefragment stammt aus einem vollständigen Beispiel, das sich unter `%SQLANYSAMPI6%\SQLAnywhere\ODBCSelect\odbcselect.cpp` befindet.

```
SQLINTEGER cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];
SQLSMALLINT deptID, managerID;
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN rc;

SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
SQLSetEnvAttr( env,
               SQL_ATTR_ODBC_VERSION,
               (void *)SQL_OV_ODBC3, 0 );
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLConnect( dbc,
            (SQLCHAR *) "SQL Anywhere 16 Demo", SQL_NTS,
            (SQLCHAR *) "DBA", SQL_NTS,
            (SQLCHAR *) "sql", SQL_NTS );
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLBindCol( stmt, 1,
            SQL_C_SSHORT, &deptID, 0, &cbDeptID );
SQLBindCol( stmt, 2,
            SQL_C_CHAR, deptName,
            sizeof(deptName), &cbDeptName );
SQLBindCol( stmt, 3,
            SQL_C_SSHORT, &managerID, 0, &cbManagerID );
SQLExecDirect( stmt, (SQLCHAR * )
               "SELECT DepartmentID, DepartmentName, DepartmentHeadID "
               "FROM Departments "
               "ORDER BY DepartmentID", SQL_NTS );
while( ( rc = SQLFetch( stmt ) ) != SQL_NO_DATA )
{
    printf( "%d %20s %d\n", deptID, deptName, managerID );
}
SQLFreeHandle( SQL_HANDLE_STMT, stmt );
SQLDisconnect( dbc );
SQLFreeHandle( SQL_HANDLE_DBC, dbc );
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

Die Anzahl der Zeilenpositionen, die Sie mit einem Fetch-Vorgang abrufen können, wird durch die Größe einer Ganzzahl bestimmt. Mit einem Fetch-Vorgang können Sie Zeilen bis zu Nummer 2147483646 abrufen, wobei es sich um die größtmögliche Ganzzahl, die in einem 32-Bit-Integer-Datentyp gespeichert werden kann, minus 1 handelt. Wenn Sie negative Werte verwenden (Zeilen in Bezug auf das Ende), können Sie Fetch-Vorgänge nach unten bis zum kleinsten negativen Wert, der in einer Ganzzahl möglich ist, plus 1 ausführen.

Zeilenaktualisierungen und -löschungen durch einen Cursor

In der Microsoft-Dokumentation *ODBC Programmer's Reference* wird empfohlen, dass Sie SELECT... FOR UPDATE benutzen, um anzuzeigen, dass eine Abfrage mit positionierten Vorgängen aktualisierbar ist. Es ist nicht erforderlich, die FOR UPDATE-Klausel in SQL Anywhere zu verwenden. SELECT-Anweisungen können automatisch aktualisiert werden, sofern die folgenden Bedingungen erfüllt sind:

- Die zu Grunde liegende Abfrage unterstützt Aktualisierungen.

Unter der Voraussetzung, dass eine Datenmanipulationsanweisung für die Spalten im Ergebnis sinnvoll ist, kann die Datenmanipulationsanweisung im Cursor ausgeführt werden.

Die Datenbankoption `ansi_update_constraints` schränkt den Typ aktualisierbarer Abfragen ein.

- Der Cursortyp unterstützt Aktualisierungen.

Wenn Sie einen schreibgeschützten Cursor verwenden, können Sie die Ergebnismenge nicht aktualisieren.

Es gibt in ODBC zwei Alternativen, um positionsbasierte Updates und positionsbasiertes Löschen durchzuführen.

- Die Funktion `SQLSetPos` wird verwendet.

Abhängig von den angegebenen Parametern (`SQL_POSITION`, `SQL_REFRESH`, `SQL_UPDATE`, `SQL_DELETE`) legt `SQLSetPos` die Cursorposition fest und ermöglicht der Anwendung, die Darstellung der Daten in der Ergebnismenge oder die Daten selbst zu aktualisieren oder zu löschen.

Diese Methode muss bei SQL Anywhere angewendet werden.

- Mithilfe von `SQLExecute` positionsbasierte UPDATE- und DELETE-Anweisungen senden. Diese Methode darf nicht bei SQL Anywhere angewendet werden.

Siehe auch

- „`ansi_update_constraints`-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]

Lesezeichen

ODBC bietet **Lesezeichen**, die verwendet werden, um Zeilen in einem Cursor zu identifizieren. SQL Anywhere unterstützt Lesezeichen für wertsensitive und insensitive Cursor. Die ODBC-Cursortypen `SQL_CURSOR_STATIC` und `SQL_CURSOR_KEYSET_DRIVEN` unterstützen beispielsweise Lesezeichen, während die Cursortypen `SQL_CURSOR_DYNAMIC` und `SQL_CURSOR_FORWARD_ONLY` keine Lesezeichen unterstützen.

Vor der ODBC-Version 3.0 konnte eine Datenbank nur angeben, ob sie Lesezeichen unterstützt oder nicht: Es gab keine Schnittstelle, die diese Informationen für jeden Cursortyp bereitstellte. Der Datenbankserver hatte keine Möglichkeit, anzugeben, welche Typen von Cursor-Lesezeichen unterstützt werden. Für ODBC 2-Anwendungen gibt SQL Anywhere nur zurück, dass Lesezeichen unterstützt

werden. ODBC hindert Sie also nicht an dem Versuch, Lesezeichen mit dynamischen Cursor zu verwenden. Sie sollten diese Kombination allerdings vermeiden.

Hinweise zu gespeicherten Prozeduren

In diesem Abschnitt wird beschrieben, wie gespeicherte Prozeduren aufgerufen und die Ergebnisse in einer ODBC-Anwendung bearbeitet werden.

Prozeduren und Ergebnismengen

Es gibt zwei Arten von Prozeduren: Prozeduren, die Ergebnismengen zurückgeben und solche, die keine zurückgeben. Mit `SQLNumResultCols` können Sie den Unterschied feststellen: die Anzahl der Ergebnisspalten ist 0 (Null), falls die Prozedur keine Ergebnismenge zurückgibt. Wenn eine Ergebnismenge zurückgegeben wird, können Sie die Werte, wie bei jedem anderen Cursor, mit `SQLFetch` oder `SQLExtendedFetch` abrufen.

Parameter müssen an Prozeduren mit Parametermarkierungen (Fragezeichen) übergeben werden. Verwenden Sie `SQLBindParameter`, um jeder Parametermarkierung einen Speicherbereich zuzuweisen: `INPUT`, `OUTPUT` oder `INOUT`.

Um mehrfache Ergebnismengen verarbeiten zu können, muss ODBC den aktuell ausgeführten Cursor beschreiben, und nicht die von der Prozedur definierte Ergebnismenge. Deshalb bezeichnet ODBC die Spaltennamen nicht immer so, wie sie in der `RESULT`-Klausel der gespeicherten Prozedur definiert sind. Um dieses Problem zu vermeiden, können Sie im Ergebnismengencursor Ihrer Prozedur einen Spaltenalias verwenden.

Beispiel 1

Mit diesem Beispiel wird eine Prozedur erstellt und aufgerufen, die keine Ergebnismenge zurückgibt. Die Prozedur übernimmt einen `INOUT`-Parameter und erhöht seinen Wert. Im Beispiel hat die Variable `num_columns` den Wert 0 (Null), da die Prozedur keine Ergebnismenge zurückgibt. Auf eine Fehlerprüfung wurde verzichtet, um das Beispiel leichter lesbar zu halten.

```
HDBC dbc;
SQLHSTMT stmt;
SQLINTEGER I;
SQLSMALLINT num_columns;

SQLAllocStmt( dbc, &stmt );
SQLExecDirect( stmt,
    "CREATE PROCEDURE Increment( INOUT a INT )"
    "BEGIN "
    "    SET a = a + 1 "
    "END", SQL_NTS );

/* Call the procedure to increment 'I' */
I = 1;
SQLBindParameter( stmt, 1, SQL_C_LONG, SQL_INTEGER, 0, 0, &I, NULL );
SQLExecDirect( stmt, "CALL Increment( ? )", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
```

Beispiel 2

Mit dem folgenden Beispiel wird eine Prozedur aufgerufen, die eine Ergebnismenge zurückgibt. Im Beispiel hat die Variable `num_columns` den Wert 2, denn die Prozedur gibt eine Ergebnismenge mit zwei Spalten zurück. Auf eine Fehlerprüfung wurde auch hier verzichtet, um das Beispiel leichter lesbar zu halten.

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT num_columns;

SQLCHAR ID[ 10 ];
SQLCHAR Surname[ 20 ];

SQLExecDirect( stmt,
    "CREATE PROCEDURE EmployeeList() "
    "RESULT( ID CHAR(10), Surname CHAR(20) ) "
    "BEGIN "
    "    SELECT EmployeeID, Surname FROM Employees "
    "END", SQL_NTS );

/* Call the procedure - print the results */
SQLExecDirect( stmt, "CALL EmployeeList()", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
SQLBindCol( stmt, 1, SQL_C_CHAR, &ID, sizeof(ID), NULL );
SQLBindCol( stmt, 2, SQL_C_CHAR, &Surname, sizeof(Surname), NULL );

for( ;; )
{
    rc = SQLFetch( stmt );
    if( rc == SQL_NO_DATA_FOUND )
    {
        rc = SQLMoreResults( stmt );
        if( rc == SQL_NO_DATA_FOUND ) break;
    }
    else
    {
        do_something( ID, Surname );
    }
}
```

Siehe auch

- „Gespeicherte Prozeduren, Trigger, Batches und benutzerdefinierte Funktionen“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)]

ODBC Escape-Syntax

Sie können ODBC JDBC-Escape-Syntax aus jeder ODBC-Anwendung verwenden. Diese Escape-Syntax ermöglicht es Ihnen, einige gängige Funktionen unabhängig vom verwendeten Datenbank-Managementsystem aufzurufen. Das allgemeine Syntaxformat lautet wie folgt:

{ **keyword** *Parameter* }

Die Gruppe der Schlüsselwörter umfasst:

- **{d date-string}** Die Zeichenfolge date ist ein Datumswert, der von SQL Anywhere akzeptiert wird.
- **{t time-string}** Die Zeichenfolge time ist ein Zeitwert, der von SQL Anywhere akzeptiert wird.
- **{ts date-string time-string}** Die Zeichenfolge date/time ist ein Zeitstempelwert, der von SQL Anywhere akzeptiert wird.
- **{guid uuid-string}** Die Zeichenfolge uuid ist eine beliebige gültige GUID-Zeichenfolge, z.B. 41dfe9ef-db91-11d2-8c43-006008d26a6f.
- **{oj outer-join-expr}** Die outer-join-expr ist ein gültiger OUTER JOIN-Ausdruck, der von SQL Anywhere akzeptiert wird.
- **{? = call func(p1,...)}** Die Funktion ist ein beliebiger gültiger Funktionsaufruf, der von SQL Anywhere akzeptiert wird.
- **{call proc(p1,...)}** Die Prozedur ist ein beliebiger gültiger Aufruf einer gespeicherten Prozedur, der von SQL Anywhere akzeptiert wird.
- **{fn func(p1,...)}** Die Funktion ist eine der Funktionen der Bibliothek, die unten aufgelistet werden.

Sie können die Escape-Syntax aber benutzen, um auf eine Bibliothek von Funktionen zuzugreifen, die im ODBC-Treiber implementiert sind und Zahlen-, Zeichenfolgen-, Zeit-, Datums- und Systemfunktionen umfassen.

Wenn Sie beispielsweise das aktuelle Datum in einer vom jeweiligen Datenbank-Managementsystem unabhängigen Weise abrufen möchten, führen Sie folgenden Befehl aus:

```
SELECT { FN CURDATE() }
```

Die folgenden Tabellen listen die Funktionen, die vom SQL Anywhere-JDBC-Treiber unterstützt werden.

Vom SQL Anywhere-ODBC-Treiber unterstützte Funktionen

Nummerische Funktionen	Zeichenfolgenfunktionen	Systemfunktionen	Datum/Zeit-Funktionen
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH	CONVERT	CURRENT_TIME-STAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME

Nummerische Funktionen	Zeichenfolgenfunktionen	Systemfunktionen	Datum/Zeit-Funktionen
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOURL
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

Die ODBC-Escape-Syntax ist identisch mit der JDBC-Escape-Syntax. In Interactive SQL, das JDBC verwendet, *müssen* die geschweiften Klammern verdoppelt werden. Zwischen aufeinanderfolgenden geschweiften Klammern darf kein Leerzeichen stehen. "{{" ist korrekt, "{" nicht. Außerdem dürfen Sie kein Zeilenschaltungszeichen in der Anweisung verwenden. Die Escape-Syntax kann in gespeicherten Prozeduren nicht verwendet werden, weil diese nicht von Interactive SQL syntaktisch analysiert werden.

Um z.B. Datenbankeigenschaften mit der Prozedur sa_db_info unter Verwendung der SQL-Escape-Syntax zu erhalten, müssen Sie folgenden Aufruf in Interactive SQL ausführen:

```
{{CALL sa_db_info( 0 ) }}
```

Fehlerbehandlung in ODBC

Fehler in ODBC werden mit dem Rückgabewert jedes einzelnen ODBC-Funktionsaufrufs und entweder mit der Funktion `SQLError` oder der Funktion `SQLGetDiagRec` gemeldet. Die Funktion `SQLError` wurde in ODBC-Versionen bis Version 3 (ausschließlich Version 3) verwendet. Ab Version 3 wird die Funktion `SQLError` nicht mehr weiter entwickelt und durch die Funktion `SQLGetDiagRec` ersetzt.

Jede ODBC-Funktion gibt ein `SQLRETURN` zurück, das einen der folgenden Statuscodes annehmen kann:

Statuscode	Beschreibung
<code>SQL_SUCCESS</code>	Kein Fehler.
<code>SQL_SUCCESS_WITH_INFO</code>	Die Funktion wurde vollständig ausgeführt, aber ein Aufruf von <code>SQLError</code> zeigt eine Warnung an. Der häufigste Grund für diesen Status ist, dass ein Wert zurückgegeben wurde, der für den von der Anwendung zur Verfügung gestellten Puffer zu lang ist.
<code>SQL_ERROR</code>	Die Funktion wurde wegen eines Fehlers nicht vollständig ausgeführt. Sie können <code>SQLError</code> ausführen, um mehr Informationen über das Problem zu erhalten.
<code>SQL_INVALID_HANDLE</code>	Ein ungültiger Umgebungs-, Verbindungs- oder Statement-Handle wurden als Argument übergeben. Dies passiert häufig, wenn ein Handle benutzt wird, nachdem es freigegeben wurde, oder falls das Handle ein Null-Zeiger ist.
<code>SQL_NO_DATA_FOUND</code>	Keine Hinweise verfügbar. Der häufigste Grund für diesen Status ist, dass - beim Abrufen von einem Cursor - keine weiteren Zeilen mehr in dem Cursor waren.
<code>SQL_NEED_DATA</code>	Für einen Parameter werden Daten benötigt. Dies ist eine erweiterte Funktion, die in der Hilfedatei unter <code>SQLParamData</code> und <code>SQLPutData</code> beschrieben wird.

Mit jeder Umgebung, jeder Verbindung und jedem Statement-Handle können ein oder mehrere Fehler oder Warnungen verbunden sein. Jeder Aufruf von `SQLError` oder `SQLGetDiagRec` gibt Hinweise für einen Fehler zurück und entfernt die Hinweise über diesen Fehler. Falls Sie `SQLError` oder `SQLGetDiagRec` nicht aufrufen, um alle Fehlermeldungen zu entfernen, werden die Fehlermeldungen beim nächsten Funktionsaufruf entfernt, der das gleiche Handle als Parameter übergibt.

Jeder Aufruf von `SQLError` kann drei Handles für Umgebung, Verbindung und Anweisung übergeben. Der erste Aufruf benutzt `SQL_NULL_HSTMT`, um den mit einer Verbindung zusammenhängenden Fehler zu erhalten. Auf die gleiche Weise gibt ein Aufruf mit `SQL_NULL_DBC` und `SQL_NULL_HSTMT` alle mit dem Umgebungshandle zusammenhängenden Fehler zurück.

Jeder Aufruf von `SQLGetDiagRec` kann entweder ein Umgebungs-, Verbindungs- oder Anweisungshandle übergeben. Der erste Aufruf benutzt `SQL_HANDLE_DBC`, um den mit einer Verbindung zusammenhängenden Fehler zurückzugeben. Der zweite Aufruf benutzt `SQL_HANDLE_STMT`, um den mit der soeben ausgeführten Anweisung zusammenhängenden Fehler zurückzugeben.

`SQLError` und `SQLGetDiagRec` geben `SQL_SUCCESS` zurück, falls ein Fehler zu melden ist (*nicht* `SQL_ERROR`), und `SQL_NO_DATA_FOUND`, falls keine weiteren Fehler zu melden sind.

Beispiel 1

Der folgende Auszug aus einem Beispielcode benutzt `SQLError` und Rückgabecodes:

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
UCHAR errmsg[100];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
    SQLError( env, dbc, SQL_NULL_HSTMT, NULL, NULL,
             errmsg, sizeof(errmsg), NULL );
    print_error( "Allocation failed", errmsg );
    return;
}

/* Delete items for order 2015 */
rc = SQLExecDirect( stmt,
                    "DELETE FROM SalesOrderItems WHERE ID=2015",
                    SQL_NTS );
if( rc == SQL_ERROR )
{
    SQLError( env, dbc, stmt, NULL, NULL,
             errmsg, sizeof(errmsg), NULL );
    print_error( "Failed to delete items", errmsg );
    return;
}
```

Beispiel 2

Der folgende Auszug aus einem Beispielcode benutzt `SQLGetDiagRec` und Rückgabecodes:

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT errmsglen;
SQLINTEGER errnative;
SQLCHAR errmsg[255];
SQLCHAR errstate[5];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
```

```
    SQLGetDiagRec( SQL_HANDLE_DBC, dbc, 1, errstate,
                  &errnative, errmsg, sizeof(errmsg), &errmsglen );
    print_error( "Allocation failed", errstate, errnative, errmsg );
    return;
}

rc = SQLExecDirect( stmt,
                   "DELETE FROM SalesOrderItems WHERE ID=2015",
                   SQL_NTS );
if( rc == SQL_ERROR )
{
    SQLGetDiagRec( SQL_HANDLE_STMT, stmt, 1, errstate,
                  &errnative, errmsg, sizeof(errmsg), &errmsglen );
    print_error( "Failed to delete items", errstate, errnative, errmsg );
    return;
}
```

Java in der Datenbank

SQL Anywhere stellt ein Verfahren zum Ausführen von Java-Klassen innerhalb der Umgebung des Datenbankservers bereit. Die Java-Methoden auf dem Datenbankserver bieten leistungsfähige Möglichkeiten, einer Datenbank Programmierlogik hinzuzufügen.

Die Java-Unterstützung in der Datenbank bietet folgende Möglichkeiten:

- Verwenden Sie Java-Komponenten in den verschiedenen Schichten Ihrer Anwendung (Client, mittlere Schicht oder Server) und setzen Sie es überall ein, wo es für Sie am sinnvollsten ist. SQL Anywhere wird damit zu einer Plattform für verteilte Datenverarbeitung.
- Java ist eine leistungsfähigere Sprache für die Erstellung von Logik in der Datenbank als die SQL-Sprache für gespeicherte Prozeduren.
- Java kann auf dem Datenbankserver verwendet werden, ohne dass die Integrität, Sicherheit oder Robustheit der Datenbank und des Servers verletzt werden.

Der SQLJ-Standard

Java in der Datenbank basiert auf dem vorgeschlagenen Standard SQLJ Part 1 (ANSI/INCITS 331.1-1999). SQLJ Part 1 bietet Spezifikationen für den Aufruf von statischen Java-Methoden in Form von gespeicherten SQL-Prozeduren und Funktionen.

Informationen zu Java in der Datenbank

Die folgende Tabelle ist ein Wegweiser für die Java-Dokumentationen, die Sie je nach Ihren Interessen und Vorkenntnissen benutzen können.

Gesuchtes Thema	Empfohlene Lektüre
Sie sind ein Java-Entwickler, der erste Schritte erlernen möchte.	„Praktische Einführung: Java in der Datenbank verwenden“
Sie möchten die Hauptmerkmale von Java in der Datenbank kennen lernen.	„Häufig gestellte Fragen zu Java in der Datenbank“
Sie möchten herausfinden, wie der Zugriff auf Daten aus Java erfolgt.	„JDBC-Unterstützung“

Häufig gestellte Fragen zu Java in der Datenbank

In diesem Abschnitt werden die Hauptmerkmale von Java in der Datenbank beschrieben.

Die wichtigsten Funktionen von Java in der Datenbank

Detaillierte Erläuterungen aller nachstehenden Punkte sind in den folgenden Abschnitten zu finden.

- **Sie können Java in der Datenbank ausführen** Eine externe Java VM führt den Java-Code für den Datenbankserver aus.
- **Sie können aus Java auf Daten zugreifen** SQL Anywhere ermöglicht den Zugriff auf Daten aus Java.
- **SQL wird bewahrt** Der Einsatz von Java ändert das Verhalten bestehender SQL-Anweisungen oder andere Aspekte des nicht mit Java verbundenen Verhaltens der relationalen Datenbank nicht.

Vorteile von Java

Java bietet eine Reihe von Merkmalen, die für den Einsatz in Datenbanken besonders gut geeignet sind:

- Präzise Fehlerprüfung bei der Kompilierung
- Integrierte Fehlerbehandlung mit einer gut definierten Methode für die Fehlerbehandlung
- Integrierte Sammlung von Abfalldaten (Speicher wird freigesetzt)
- Eliminierung vieler fehleranfälliger Programmiertechniken
- Leistungsfähige Sicherheitsfunktionen
- Plattformunabhängige Ausführung von Java-Code.

Eigene Java-Klassen in Datenbanken verwenden

Die Sprache Java ist leistungsfähiger als SQL. Java ist eine objektorientierte Sprache, ihre Anweisungen (Quellcode) werden daher in Form von Klassen geliefert. Um Java in einer Datenbank auszuführen, schreiben Sie die Java-Instruktionen außerhalb der Datenbank in kompilierte Klassen (**Bytecode**), bei denen es sich um Binärdateien mit Java-Instruktionen handelt.

Kompilierte Klassen können aus Clientanwendungen genauso leicht und auf dieselbe Weise aufgerufen werden wie gespeicherte Prozeduren. Java-Klassen können Informationen über den Gegenstand der Klasse und Computerlogik enthalten. Sie können beispielsweise Java-Code planen, schreiben und kompilieren, um eine Employees-Klasse mit verschiedenen Methoden zu erstellen, die Vorgänge in einer Employees-Tabelle ausführen. Sie installieren Ihre Java-Klassen als Objekte in einer Datenbank und schreiben SQL-Mantelfunktionen oder -Prozeduren, um die Methoden in den Java-Klassen aufzurufen.

Nach der Installation können Sie diese Klassen mithilfe von gespeicherten Prozeduren über den Datenbankserver ausführen. Beispiel: Die folgende Anweisung erstellt die Schnittstelle für eine Java-Prozedur:

```
CREATE PROCEDURE MyMethod()  
EXTERNAL NAME 'JDBCExample.MyMethod()' V'  
LANGUAGE JAVA;
```

SQL Anywhere ermöglicht eine Laufzeit-Umgebung für Java-Klassen und ist keine Java-Entwicklungsumgebung. Sie benötigen eine Java-Entwicklungsumgebung, z.B. das Java Development Kit

(JDK), um Java zu schreiben und zu kompilieren. Außerdem benötigen Sie eine Java-Laufzeitumgebung, um Java-Klassen auszuführen.

Sie können viele der Klassen verwenden, die Teil der Java-API sind, wie sie mit dem Java Development Kit geliefert wird. Sie können auch Klassen verwenden, die von Java-Entwicklern erstellt und kompiliert wurden.

Siehe auch

- „CREATE PROCEDURE-Anweisung [externer Aufruf]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Java-Klassen in Datenbanken installieren“ auf Seite 410

Java in einer Datenbank ausführen

SQL Anywhere startet eine Java VM. Die Java VM interpretiert kompilierte Java-Anweisungen und führt sie für den Datenbankserver aus. Der Datenbankserver startet die Java VM automatisch, wenn sie benötigt wird: Sie brauchen keine ausdrücklichen Bedienungsmaßnahmen auszuführen, um die Java VM zu starten oder zu stoppen.

Der SQL-Abfrageprozessor auf dem Datenbankserver wurde erweitert, sodass er die Java VM für die Ausführung von Java-Anweisungen aufrufen kann. Er kann auch Abfragen von der Java VM verarbeiten, um den Datenzugriff aus Java zu aktivieren.

Fehlerbehandlung in Java

Fehler in Java-Anwendungen generieren ein Ausnahmebedingungsobjekt, das den Fehler darstellt (dies wird als **Ausnahmefehler ausgeben** bezeichnet). Eine ausgelöste Ausnahmebedingung beendet ein Java-Programm, wenn sie nicht in der Anwendung abgefangen und entsprechend verarbeitet wird.

Sowohl Java-API-Klassen, als auch benutzerdefinierte Klassen können einen Ausnahmefehler auswerfen. Benutzer können sogar eigene Ausnahmebedingungsklassen erstellen, die ihre eigenen benutzerdefinierten Fehlerklassen ausgeben können.

Wenn im Hauptteil der Methode, in der der Ausnahmefehler auftrat, keine Ausnahmeroutine vorgesehen ist, wird die Suche nach einer Ausnahmeroutine über den kompletten Aufrufstack fortgesetzt. Wenn die Spitze des Aufrufstacks erreicht ist und noch immer keine Ausnahmeroutine gefunden wurde, wird die Standard-Ausnahmeroutine des Java-Interpreters aufgerufen und das Programm beendet.

In SQL Anywhere gilt: Wenn eine SQL-Anweisung eine Java-Methode aufruft, wird ein SQL-Fehler generiert. Der vollständige Text der Java-Ausnahmebedingung sowie die Java-Stack Trace wird im Meldungsfenster des Servers angezeigt.

Praktische Einführung: Java in der Datenbank verwenden

Die folgenden Abschnitte beschreiben die erforderlichen Schritte, um Java-Methoden zu erstellen und sie aus SQL-Anweisungen aufzurufen. Es wird gezeigt, wie Sie eine Java-Klasse kompilieren und sie in der Datenbank installieren, um sie für die Verwendung in SQL Anywhere verfügbar zu machen. Darüber hinaus wird gezeigt, wie auf die Klasse und ihre Mitglieder und Methoden von SQL-Anweisungen aus zugegriffen wird.

Diese folgenden Abschnitte setzen voraus, dass Sie ein Java Development Kit (JDK) installiert haben, einschließlich dem Java Compiler (javac) und der Java VM.

Der Quellcode und die Batchdateien für das Beispiel befinden sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\JavaInvoice`.

Privilegien

Sie müssen die folgenden Systemprivilegien haben.

- MANAGE ANY EXTERNAL ENVIRONMENT
- SET ANY SYSTEM OPTION
- SERVER OPERATOR
- MANAGE ANY EXTERNAL OBJECT
- CREATE PROCEDURE

Lektion 1: Java-Programme kompilieren

Der erste Schritt bei der Verwendung von Java in der Datenbank besteht darin, den Java-Code zu schreiben und zu kompilieren.

Voraussetzungen

Installieren Sie ein Java Development Kit (JDK), einschließlich des Java-Compilers (javac) und einer Java-Laufzeitumgebung (Java Runtime Environment, JRE).

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Java in der Datenbank verwenden](#)“.

Kontext und Bemerkungen

Der Datenbankserver verwendet die CLASSPATH-Umgebungsvariable, um eine Datei während der Installation von Klassen zu finden.

Aufgabe

1. Öffnen Sie eine Eingabeaufforderung und navigieren Sie zum Ordner %SQLANYAMP16%\SQLAnywhere\JavaInvoice.

```
cd %SQLANYAMP16%\SQLAnywhere\JavaInvoice
```

2. Kompilieren Sie das Java-Quellcodebeispiel mithilfe des folgenden Befehls:

```
javac Invoice.java
```

3. Dieser Schritt ist optional. Bevor Sie den Datenbankserver starten, vergewissern Sie sich, dass der Speicherort Ihrer kompilierten Klassendatei in der CLASSPATH-Umgebungsvariablen enthalten ist. Dies ist der CLASSPATH des Datenbankservers, der verwendet wird, nicht der CLASSPATH des Clients, auf dem Interactive SQL läuft. Hier sehen Sie ein Beispiel:

```
SET CLASSPATH=%SQLANYAMP16%\SQLAnywhere\JavaInvoice
```

Ergebnisse

Der Befehl `javac` erstellt eine Klassendatei, die in der Datenbank installiert werden kann.

Nächste Schritte

In der nächsten Lektion wählen Sie eine Java VM aus, die verwendet wird, um Ihren Java-Code auszuführen. Gehen Sie weiter zu [„Lektion 2: Auswählen einer Java VM“ auf Seite 405](#).

Lektion 2: Auswählen einer Java VM

Der Datenbankserver muss so eingerichtet sein, dass er eine Java Virtual Machine (VM) finden kann. Da Sie für jede Datenbank eine andere Java VM angeben können, kann die `ALTER EXTERNAL ENVIRONMENT`-Anweisung verwendet werden, um den Speicherort (Pfad) der Java VM anzugeben.

Voraussetzungen

Schließen Sie die Schritte in Lektion 1 ab, bevor Sie Lektion 2 versuchen.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: [„Praktische Einführung: Java in der Datenbank verwenden“](#).

Kontext und Bemerkungen

Wenn die Java-Laufzeitumgebung (JRE) nicht installiert ist, können Sie eine beliebige Java-JRE installieren und ausführen, sofern Sie mindestens Version 1.6 verwenden. Die meisten Java-Installationsprogramme richten die Umgebungsvariable `JAVA_HOME` oder `JAVAHOME` ein. Wenn keine dieser beiden Umgebungsvariablen vorhanden ist, können Sie manuell eine erstellen und mit ihr auf das Stammverzeichnis Ihrer Java VM zeigen. Diese Konfiguration ist jedoch nicht erforderlich, wenn Sie die `ALTER EXTERNAL ENVIRONMENT`-Anweisung verwenden.

Aufgabe

1. Verwenden Sie Interactive SQL, um den Personal Datenbankserver zu starten und eine Verbindung mit der Beispieldatenbank herzustellen.

```
dbisql -c "DSN=SQL Anywhere 16 Demo"
```

2. Dieser Schritt ist optional. Führen Sie eine Anweisung ähnlich der folgenden aus.

```
ALTER EXTERNAL ENVIRONMENT JAVA  
LOCATION 'c:\\jdk1.7.0\\jre\\bin\\java.exe';
```

Wenn der Speicherort der Java VM nicht mit der LOCATION-Klausel der ALTER EXTERNAL ENVIRONMENT JAVA-Anweisung angegeben wurde oder der angegebene Speicherort falsch ist, sucht der Datenbankserver folgendermaßen nach dem Speicherort der Java VM:

- Umgebungsvariable JAVA_HOME überprüfen
 - Umgebungsvariable JAVAHOME überprüfen
 - System-PATH überprüfen
 - Wenn die VM nicht gefunden werden kann, wird ein Fehler zurückgegeben.
3. Dieser Schritt ist optional. Verwenden Sie die java_vm_options-Datenbankoption, um zusätzliche Befehlszeilenoptionen anzugeben, die zum Starten der Java VM erforderlich sind.

```
SET OPTION PUBLIC.java_vm_options='java-options';
```

4. Verwenden Sie die START JAVA-Anweisung, um die Java VM zu starten.

```
START JAVA;
```

Diese Anweisung versucht, die Java VM vorab zu laden. Wenn der Datenbankserver die Java VM nicht finden und starten kann, wird eine Fehlermeldung ausgegeben. Diese Anweisung ist optional, da der Datenbankserver automatisch die Java VM lädt, wenn dies erforderlich ist.

Ergebnisse

Die LOCATION-Klausel der ALTER EXTERNAL ENVIRONMENT JAVA-Anweisung gibt den Speicherort der Java VM an. Die START JAVA-Anweisung lädt die Java VM.

Nächste Schritte

In der nächsten Lektion installieren Sie die Java-Klasse in der Datenbank. Gehen Sie weiter zu [„Lektion 3: Java-Klassen installieren“ auf Seite 407](#).

Siehe auch

- „ALTER EXTERNAL ENVIRONMENT-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „START JAVA-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „java_vm_options-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]

Lektion 3: Java-Klassen installieren

Java-Klassen können erst verwendet werden, nachdem sie in einer Datenbank installiert wurden. Sie können Java-Klassen aus Sybase Central oder aus Interactive SQL installieren.

Voraussetzungen

Schließen Sie die Schritte in Lektion 1 und 2 ab, bevor Sie Lektion 3 versuchen.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: [„Praktische Einführung: Java in der Datenbank verwenden“](#).

Kontext und Bemerkungen

Der Datenbankserver verwendet die CLASSPATH-Umgebungsvariable, um eine Datei während der Installation von Klassen zu finden. Wenn die in der INSTALL JAVA-Anweisung aufgelistete Datei sich in einem Verzeichnis oder in einer ZIP-Datei befindet, die in der CLASSPATH-Umgebungsvariable des Datenbankservers angegeben ist, findet der Server die Datei und installiert die Klasse.

Aufgabe

1. Prüfen Sie, ob die CLASSPATH-Umgebungsvariable aus der ersten Lektion richtig gesetzt ist. Sie sollte das Verzeichnis enthalten, in dem sich die Datei *Invoice.class* befindet.

```
CALL sa_split_list(CAST(xp_getenv('CLASSPATH') AS LONG VARCHAR), ';');
```

2. Verwenden Sie Interactive SQL, um eine Anweisung ähnlich der folgenden auszuführen. Der *Pfad* zum Speicherort Ihrer kompilierten Klassendatei ist nicht erforderlich, wenn diese über den CLASSPATH des Datenbankservers gefunden werden kann. Wenn der *Pfad* angegeben wird, muss er für den Datenbankserver zugänglich sein.

```
INSTALL JAVA NEW  
FROM FILE 'path\\Invoice.class';
```

Ergebnisse

Die Klasse wird nun in der Beispieldatenbank installiert.

Nachfolgende Änderungen an der Klassendatei werden *nicht* automatisch in die Kopie der Klassendatei in der Datenbank übernommen. Sobald die Klassendatei neu kompiliert wird, können Sie mithilfe der INSTALL JAVA UPDATE-Anweisung die Klassendatei in die Datenbank laden.

Nächste Schritte

In der nächsten Lektion rufen Sie die Methoden in der Java-Klasse über SQL auf. Gehen Sie weiter zu [„Lektion 4: Methoden in Java-Klassen aufrufen“ auf Seite 408](#).

Siehe auch

- „INSTALL JAVA-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Java-Klassen in Datenbanken installieren“ auf Seite 410

Lektion 4: Methoden in Java-Klassen aufrufen

Für den Zugriff auf die Java-Methoden in der Klasse müssen Sie gespeicherte Prozeduren oder Funktionen erstellen, die als Wrapper für die Methoden in der Klasse fungieren.

Voraussetzungen

Schließen Sie die Schritte in Lektion 1, 2 und 3 ab, bevor Sie Lektion 4 versuchen.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Java in der Datenbank verwenden](#)“.

Kontext und Bemerkungen

Die Java-Klassendatei, die die kompilierten Methoden aus dem Invoice-Beispiel enthält, wurde in die Datenbank geladen.

Aufgabe

1. Erstellen Sie die folgende gespeicherte SQL-Prozedur, um die Methode Invoice.main in der Beispielsklasse aufzurufen:

```
CREATE PROCEDURE InvoiceMain( IN arg1 CHAR(50) )  
EXTERNAL NAME 'Invoice.main([Ljava/lang/String; )V'  
LANGUAGE JAVA;
```

Diese gespeicherte Prozedur agiert als Wrapper für die Java-Methode.

2. Führen Sie die gespeicherte Prozedur aus, um die Java-Methode aufzurufen:

```
CALL InvoiceMain('to you');
```

In der Datenbankserver-Konsole bzw. im Fenster "Datenbankservermeldungen" wird die Meldung "Hello to you" angezeigt. Der Datenbankserver hat die Ausgabe von System.out dorthin umgeleitet

3. Die folgenden gespeicherten Prozeduren veranschaulichen, wie Argumente an die Java-Methoden in der Invoice-Klasse übergeben und Rückgabewerte daraus abgerufen werden: Wenn Sie den Java-Quellcode überprüfen, sehen Sie, dass die init-Methode der Invoice-Klasse sowohl STRING-Argumente als auch DOUBLE-Argumente annimmt. STRING-Argumente werden mit `Ljava/lang/String;` angegeben. DOUBLE-Argumente werden mit `D` angegeben. Die Methode gibt "void" zurück und dies wird mit `V` nach der rechten Klammer angegeben.

```
CREATE PROCEDURE init( IN arg1 CHAR(50),  
                      IN arg2 DOUBLE,  
                      IN arg3 CHAR(50),  
                      IN arg4 DOUBLE)
```

```
EXTERNAL NAME 'Invoice.init(Ljava/lang/String;DLjava/lang/String;D)V'
LANGUAGE JAVA;
```

4. Die folgenden Funktionen rufen Java-Methoden auf, die keine Argumente übernehmen und einen DOUBLE-Wert (D) oder einen STRING-Wert (Ljava/lang/String;) zurückgeben.

```
CREATE FUNCTION rateOfTaxation()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.rateOfTaxation()D'
LANGUAGE JAVA;

CREATE FUNCTION totalSum()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.totalSum()D'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem1Description()
RETURNS CHAR(50)
EXTERNAL NAME 'Invoice.getLineItem1Description()Ljava/lang/String;'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem1Cost()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.getLineItem1Cost()D'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem2Description()
RETURNS CHAR(50)
EXTERNAL NAME 'Invoice.getLineItem2Description()Ljava/lang/String;'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem2Cost()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.getLineItem2Cost()D'
LANGUAGE JAVA;
```

5. Das folgende Beispiel veranschaulicht einen einfachen Aufruf an die gespeicherte Prozedur, die als Wrapper für die init-Methode der Invoice-Klasse fungiert:

```
CALL init('Shirt',10.00,'Jacket',25.00);
```

6. Die folgende SELECT-Anweisung ruft mehrere andere Methoden in der Invoice-Klasse auf:

```
SELECT getLineItem1Description() as Item1,
       getLineItem1Cost() as Item1Cost,
       getLineItem2Description() as Item2,
       getLineItem2Cost() as Item2Cost,
       rateOfTaxation() as TaxRate,
       totalSum() as Cost;
```

Die SELECT-Anweisung gibt sechs Spalten zurück.

Item1	Item1Cost	Item2	Item2Cost	TaxRate	Cost
Shirt	10	Jacket	25	0.15	40.25

Ergebnisse

Sie haben gespeicherte Prozeduren oder Funktionen erstellt, die als Wrapper für die Methoden in der Java-Klasse fungieren. In diesen Lektionen wurden Sie durch die Schritte geführt, die beim Schreiben von Java-Methoden und beim Aufrufen dieser Methoden aus SQL erforderlich sind.

Siehe auch

- „CREATE PROCEDURE-Anweisung [externer Aufruf]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „CREATE FUNCTION-Anweisung [externer Aufruf]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Java-Klassen in Datenbanken installieren

Sie können Java-Klassen in einer Datenbank als:

- **Einzelne Klasse** Sie können eine einzelne Klasse in einer Datenbank aus einer kompilierten Klassendatei installieren. Klassendateien haben normalerweise die Erweiterung *.class*.
- **JAR-Datei** Sie können eine Serie von Klassen auf einmal installieren, wenn sie in einer komprimierten oder unkomprimierten JAR-Datei zusammengefasst sind. JAR-Dateien haben normalerweise die Erweiterung *.jar* oder *.zip*. SQL Anywhere unterstützt alle komprimierten JAR-Dateien, die mit dem JAR-Dienstprogramm oder mit anderen JAR-Komprimierungsschemata erstellt werden.

Erstellung von Klassendateien

Die Details der einzelnen Schritte können zwar unterschiedlich sein, wenn ein Java-Entwicklungstool verwendet wird, aber in der Regel umfasst das Erstellen eigener Klassen die folgenden Schritte:

1. Definieren Sie Ihre Klasse.

Schreiben Sie den Java-Code, der Ihre Klasse definiert.

2. Benennen und speichern Sie Ihre Klasse.

Speichern Sie Ihre Klassendeklaration (Java-Code) in einer Datei mit der Erweiterung *.java*. Achten Sie darauf, dass der Name der Datei mit dem Namen der Klasse übereinstimmt, und dass die Groß- und Kleinschreibung beider Namen identisch ist.

Beispiel: Eine Klasse namens "Dienstprogramm" muss in einer Datei namens *Dienstprogramm.java* gespeichert werden.

3. Kompilieren Sie Ihre Klasse.

Mit diesem Schritt wird die Klassendeklaration mit dem Java-Code in eine neue, eigene Datei verwandelt, die Byte-Code enthält. Der Name der neuen Datei ist mit dem Namen der Java-Codedatei

identisch, hat jedoch die Erweiterung *.class*. Eine kompilierte Java-Klasse kann in einer Java-Laufzeitumgebung unabhängig von der Plattform ausgeführt werden, auf der sie kompiliert wurde. Auch das Betriebssystem der Java-Laufzeitumgebung spielt keine Rolle.

Klassendateien installieren

Machen Sie Ihre Java-Klasse innerhalb der Datenbank verfügbar, indem Sie die Klasse in der Datenbank installieren.

Voraussetzungen

Um eine Klasse zu installieren, benötigen Sie das **MANAGE ANY EXTERNAL OBJECT**-Systemprivileg.

Sie müssen den Pfad und den Dateinamen der Klassendatei kennen, die Sie installieren möchten.

Aufgabe

1. Stellen Sie in Sybase Central eine Verbindung zur Datenbank mithilfe des **SQL Anywhere 16**-Plug-Ins her.
2. Öffnen Sie den Ordner **Externe Umgebungen**.
3. In diesem Ordner öffnen Sie den Ordner **Java**.
4. Rechtsklicken Sie auf den rechten Fensterausschnitt und klicken Sie auf **Neu » Java-Klasse**.
5. Befolgen Sie die Anweisungen des Assistenten.

Ergebnisse

Die Klasse wird in der Datenbank installiert und kann verwendet werden.

Siehe auch

- „INSTALL JAVA-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

JAR-Dateien installieren

Installieren Sie die JAR-Datei in der Datenbank, um sie innerhalb der Datenbank verfügbar zu machen.

Voraussetzungen

Um eine JAR-Datei zu installieren, benötigen Sie das **MANAGE ANY EXTERNAL OBJECT**-Systemprivileg.

Sie müssen den Pfad und den Dateinamen der JAR-Datei kennen, die Sie installieren möchten. Eine JAR-Datei kann die Erweiterung **JAR** oder **ZIP** haben. Jede JAR-Datei muss einen Namen in der Datenbank haben. Normalerweise benutzen Sie denselben Namen wie die JAR-Datei, ohne die Erweiterung.

Beispiel: Wenn Sie eine JAR-Datei namens *meinejar.zip* installieren, sollten Sie ihr im Allgemeinen den JAR-Namen *meinejar* geben.

Kontext und Bemerkungen

Es ist sinnvoll und allgemein üblich, Gruppen von miteinander verwandten Klassen in Paketen zu sammeln und diese Pakete in einer **JAR-Datei** zusammenzufassen.

Aufgabe

1. Stellen Sie in Sybase Central eine Verbindung zur Datenbank mithilfe des **SQL Anywhere 16**-Plug-Ins her.
2. Öffnen Sie den Ordner **Externe Umgebungen**.
3. In diesem Ordner öffnen Sie den Ordner **Java**.
4. Rechtsklicken Sie auf den rechten Fensterausschnitt und klicken Sie auf **Neu » JAR-Datei**.
5. Befolgen Sie die Anweisungen des Assistenten.

Ergebnisse

Eine JAR-Datei wurde in der Datenbank installiert und kann verwendet werden.

Siehe auch

- „INSTALL JAVA-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Klassen und JAR-Dateien aktualisieren

Ersetzen Sie Klassen und JAR-Dateien mithilfe von Sybase Central durch aktualisierte Kopien.

Voraussetzungen

Um eine Klasse oder JAR-Datei zu aktualisieren, benötigen Sie das **MANAGE ANY EXTERNAL OBJECT**-Systemprivileg.

Sie benötigen eine neuere Version der kompilierten Klassendatei oder JAR-Datei.

Kontext und Bemerkungen

Die neue Definition wird nur von Verbindungen verwendet, die erst nach der Installation der Klasse eingerichtet wurden oder die Klasse zum ersten Mal nach der Installation der Klasse verwenden. Sobald die Java VM eine Klassendefinition lädt, bleibt diese im Speicher, bis die Verbindung geschlossen wird. Wenn Sie eine Java-Klasse oder auf einer Klasse basierende Objekte in der aktuellen Verbindung verwendet haben, trennen Sie die Verbindung und stellen Sie sie wieder her, damit die neue Klassendefinition wirksam wird.

Aufgabe

1. Stellen Sie in Sybase Central eine Verbindung zur Datenbank mithilfe des **SQL Anywhere 16**-Plug-Ins her.
2. Öffnen Sie den Ordner **Externe Umgebungen**.
3. In diesem Ordner öffnen Sie den Ordner **Java**.
4. Suchen Sie den Unterordner, der die Klasse oder JAR-Datei enthält, die aktualisiert werden soll.
5. Klicken Sie auf die Klasse oder die JAR-Datei und klicken Sie auf **Datei » Aktualisieren**.
6. Im Fenster **Aktualisieren** geben Sie den Standort und den Namen der Klasse oder JAR-Datei ein, die Sie aktualisieren wollen. Sie können mit **Durchsuchen** danach suchen.

Ergebnisse

Die neue Definition wird nur von Verbindungen verwendet, die erst nach der Installation der Klasse eingerichtet wurden oder die Klasse zum ersten Mal nach der Installation der Klasse verwenden. Sobald die Java VM eine Klassendefinition lädt, bleibt diese im Speicher, bis die Verbindung geschlossen wird. Wenn Sie eine Java-Klasse oder auf einer Klasse basierende Objekte in der aktuellen Verbindung verwendet haben, trennen Sie die Verbindung und stellen Sie sie wieder her, damit die neue Klassendefinition wirksam wird.

Nächste Schritte

Sie können eine Java- Klasse oder JAR-Datei auch aktualisieren, indem Sie auf die Klasse oder den JAR-Dateinamen rechtsklicken und **Aktualisieren** wählen.

Sie können eine Java-Klasse oder JAR-Datei auch aktualisieren, indem Sie auf der Registerkarte **Allgemein** ihres Fensters **Eigenschaften** auf **Jetzt aktualisieren** klicken.

Siehe auch

- „INSTALL JAVA-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Besondere Funktionen von Java-Klassen in der Datenbank

In diesem Abschnitt werden besondere Funktionen von Java-Klassen beschrieben, die in einer Datenbank eingesetzt werden.

main-Methode aufrufen

Sie starten Java-Anwendungen (außerhalb der Datenbank), indem Sie die Java VM mit einer Klasse ausführen, die eine main-Methode hat.

Die Invoice-Klasse in der Datei %SQLANYSAMPI6%\SQLAnywhere\JavaInvoice\Invoice.java hat beispielsweise eine main-Methode. Wenn Sie die Klasse von der Befehlszeile aus mit einem Befehl wie dem folgenden ausführen, wird die main-Methode ausgeführt.

```
java Invoice
```

main-Methode einer Klasse über SQL aufrufen

Führen Sie die folgenden Schritte aus, um die main-Methode einer in Java geschriebenen Klasse aufzurufen:

1. Deklarieren Sie die main-Methode in Java mit einem Array von Zeichenfolgen als Argument:

```
public static void main( java.lang.String args[] )
{
    ...
}
```

2. Erstellen Sie eine gespeicherte Prozedur als Wrapper für diese Methode.

```
CREATE PROCEDURE JavaMain( IN arg CHAR(50) )
EXTERNAL NAME 'JavaClass.main([Ljava/lang/String; )V'
LANGUAGE JAVA;
```

3. Rufen Sie die main-Methode mit der SQL CALL-Anweisung auf.

```
CALL JavaMain( 'Hello world' );
```

Aufgrund der Einschränkungen der SQL-Sprache kann nur eine einzelne Zeichenfolge übergeben werden.

Siehe auch

- „CREATE PROCEDURE-Anweisung [externer Aufruf]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Threads in Java-Anwendungen

Sie können mehrere Threads in einer Java-Anwendung benutzen, indem Sie Funktionen des Pakets `java.lang.Thread` einsetzen.

Sie können Threads in Java-Anwendungen synchronisieren, vorläufig aussetzen, wieder aufnehmen, unterbrechen oder stoppen.

Keine solche Ausnahmebedingung

Wenn Sie beim Aufruf einer Java-Methode eine falsche Anzahl von Argumenten angeben oder einen falschen Datentyp verwenden, antwortet die Java VM mit einem `java.lang.NoSuchMethodException`-Fehler. Überprüfen Sie Anzahl und Typ der Argumente.

Siehe auch

- „Lektion 4: Methoden in Java-Klassen aufrufen“ auf Seite 408

Ergebnismengen aus Java-Methoden zurückgeben

Schreiben Sie eine Java-Methode, die eine Ergebnismenge an die aufrufende Umgebung zurückgibt, und binden Sie diese Methode in eine in SQL geschriebene gespeicherte Prozedur ein, die als EXTERNAL NAME of LANGUAGE JAVA deklariert ist.

Führen Sie die folgenden Aufgaben aus, um Ergebnismengen aus einer Java-Methode zurückzugeben:

1. Achten Sie darauf, dass die Java-Methode als öffentlich und statisch in einer öffentlichen Klasse deklariert wird.
2. Für jede Ergebnismenge, die die Methode zurückgeben soll, muss die Methode einen Parameter vom Typ `java.sql.ResultSet[]` haben. Diese Ergebnismengen-Parameter müssen am Ende der Parameterliste stehen.
3. In der Methode erstellen Sie erst eine Instanz von `java.sql.ResultSet` und ordnen sie dann einem der `ResultSet[]`-Parameter zu.
4. Erstellen Sie eine in SQL geschriebene gespeicherte Prozedur des Typs EXTERNAL NAME LANGUAGE JAVA. Dieser Prozedurtyp ist ein Wrapper für eine Java-Methode. Sie können einen Cursor für die Ergebnismenge der SQL-Prozedur auf dieselbe Weise verwenden wie jede andere Prozedur, die Ergebnismengen zurückgibt.

Hinweise zur Syntax für gespeicherte Prozeduren, die als Wrapper für Java-Methoden verwendet werden, finden Sie unter „CREATE PROCEDURE-Anweisung [externer Aufruf]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*].

Beispiel

Die folgende Beispielklasse hat eine einzige Methode, die eine Abfrage ausführt und die Ergebnismenge an die aufrufende Umgebung zurückgibt.

```
import java.sql.*;

public class MyResultSet
{
    public static void return_rset( ResultSet[] rset1 )
        throws SQLException
    {
        Connection conn = DriverManager.getConnection(
            "jdbc:default:connection" );
        Statement stmt = conn.createStatement();
        ResultSet rset =
            stmt.executeQuery (
                "SELECT Surname " +
                "FROM Customers" );
        rset1[0] = rset;
    }
}
```

Die Ergebnismenge wird in SQL mit der Anweisung CREATE PROCEDURE exponiert, womit die Anzahl der von der Prozedur zurückgegebenen Ergebnismengen und die Signatur der Java-Methode angezeigt werden.

Eine Anweisung vom Typ CREATE PROCEDURE, die eine Ergebnismenge anzeigt, könnte wie folgt definiert werden:

```
CREATE PROCEDURE result_set()  
  RESULT (SurName person_name_t)  
  DYNAMIC RESULT SETS 1  
  EXTERNAL NAME  
    'MyResultSet.return_rset([Ljava/sql/ResultSet;)V'  
  LANGUAGE JAVA;
```

Ein Cursor kann für diese Prozedur genauso wie für andere SQL Anywhere-Prozeduren geöffnet werden, die Ergebnismengen zurückgeben.

Die Zeichenfolge ([Ljava/sql/ResultSet;)V ist eine Java-Methodensignatur, also eine kompakte Zeichendarstellung der Anzahl und der Typen der Parameter und Rückgabewerte.

Siehe auch

- „CREATE PROCEDURE-Anweisung [externer Aufruf]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Ergebnismengen aus Java zurückgeben“ auf Seite 443

Aus Java über gespeicherte Prozeduren zurückgegebene Werte

Sie können gespeicherte Prozeduren, die mit EXTERNAL NAME LANGUAGE JAVA erstellt wurden, als Wrapper für Java-Methoden verwenden. In diesem Abschnitt wird beschrieben, wie Sie Ihre Java-Methode so schreiben, dass OUT- oder INOUT-Parameter in der gespeicherten Prozedur benutzt werden können.

Java hat keine explizite Unterstützung für INOUT- oder OUT-Parameter. Anstelle dessen können Sie ein Array des Parameters verwenden. Beispiel: Um einen Ganzzahl-OUT-Parameter zu benutzen, erstellen Sie ein Array von genau einer Ganzzahl:

```
public class Invoice  
{  
  public static boolean testOut( int[] param )  
  {  
    param[0] = 123;  
    return true;  
  }  
}
```

Die folgende Prozedur benutzt die testOut-Methode:

```
CREATE PROCEDURE testOut( OUT p INTEGER )  
  EXTERNAL NAME 'Invoice.testOut([I)Z'  
  LANGUAGE JAVA;
```

Die Zeichenfolge ([I] Z ist eine Java-Methodensignatur, die anzeigt, dass die Methode einen einzelnen Parameter hat, der ein Array von Ganzzahlen darstellt und einen Booleschen Wert zurückgibt. Definieren Sie die Methode so, dass der Methodenparameter, den Sie als OUT- oder INOUT-Parameter verwenden möchten, ein Array eines Java-Datentyps ist, der dem SQL-Datentyp des OUT- oder INOUT-Parameters entspricht.

Um dies zu testen, rufen Sie die gespeicherte Prozedur mit einer nicht initialisierten Variablen auf.

```
CREATE VARIABLE zap INTEGER;  
CALL testOut( zap );  
SELECT zap;
```

Die Ergebnismenge ist 123.

Hinweise zur Syntax, einschließlich der Methodensignatur, finden Sie unter „[CREATE PROCEDURE-Anweisung \[externer Aufruf\]](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*].

Sicherheitsmanagement für Java

Java bietet Sicherheitsmanager, die Sie zur Kontrolle des Benutzerzugriffs auf vertrauliche Funktionen Ihrer Anwendungen verwenden können, z.B. Dateizugriff und Netzwerkzugriff. Sie sollten die von Ihrer Java VM unterstützten Sicherheitsverwaltungsfunktionen nutzen.

Java VM starten und stoppen

Die Java VM wird automatisch geladen, wenn der erste Java-Vorgang ausgeführt wird. Wenn Sie die VM explizit laden möchten, damit sie für Java-Vorgänge bereit ist, führen Sie die folgende Anweisung aus:

```
START JAVA;
```

Sie können die Java VM mit der STOP JAVA-Anweisung entladen, wenn Java nicht verwendet wird. Hierbei gilt die folgende Syntax:

```
STOP JAVA;
```

Shutdown-Hooks in der Java VM

Der Java VM ClassLoader von SQL Anywhere, der zur Bereitstellung von Unterstützung für JAVA in der Datenbank dient, ermöglicht es Anwendungen, Shutdown-Hooks zu installieren. Diese Shutdown-Hooks sind denjenigen ähnlich, die Anwendungen zusammen mit der JVM Runtime installieren. Wenn eine Verbindung, die die Unterstützung für Java in der Datenbank nutzt, eine STOP JAVA-Anweisung ausführt oder getrennt wird, führt der ClassLoader für die Verbindung vor dem Entladen alle Shutdown-Hooks aus, die für die betreffende Verbindung installiert wurden. Bei regulären JAVA-Daten in den Datenbankanwendungen, die alle Java-Klassen in der Datenbank installieren, sollte die Installation der Shutdown-Hooks nicht erforderlich sein. Die ClassLoader-Shutdown-Hooks dürfen nur mit großer Vorsicht verwendet werden und nur dazu, systemweite Ressourcen zu bereinigen, die für die jeweilige Verbindung zugewiesen wurden, die Java stoppt. Außerdem sind jdbc:default-JDBC-Anforderungen in

Shutdown-Hooks nicht zulässig, da die jdbc:default-Verbindung bereits geschlossen ist, bevor der ClassLoader-Shutdown-Hook aufgerufen wird.

Damit Sie einen Shutdown-Hook mit dem Java VM ClassLoader von SQL Anywhere installieren können, muss sich die Datei *sajvm.jar* im Classpath des Java-Compilers befinden und sie muss Code ähnlich dem folgenden ausführen:

```
SDHookThread hook = new SDHookThread( ... );  
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();  
((iAnywhere.sa.jvm.SAClassLoader)classLoader).addShutdownHook( hook );
```

Die SDHookThread-Klasse erweitert die Standard-Thread-Klasse und der obige Code muss von einer Klasse ausgeführt werden, die vom ClassLoader für die aktuelle Verbindung geladen wurde. Jede Klasse, die innerhalb der Datenbank installiert ist und später über einen externen Umgebungsaufwurf aufgerufen wird, wird automatisch von der richtigen Java VM ClassLoader-Instanz für SQL Anywhere ausgeführt.

Um einen shutdown-Hook aus der Liste von Java VM ClassLoader für SQL Anywhere zu entfernen, muss eine Anwendung Code ausführen, der folgendem Beispiel ähnelt:

```
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();  
((iAnywhere.sa.jvm.SAClassLoader)classLoader).removeShutdownHook( hook );
```

Der obige Code muss von einer Klasse ausgeführt werden, die von ClassLoader für die aktuelle Verbindung geladen wurde.

JDBC-Unterstützung

JDBC ist eine Schnittstelle auf Aufrufebene für Java-Anwendungen. JDBC bietet Java-Programmierern eine einheitliche Schnittstelle zu einer Vielzahl von relationalen Datenbanken sowie eine gemeinsame Basis, auf der Tools und Schnittstellen auf höherer Ebene aufgebaut werden können. JDBC ist jetzt ein Standardbestandteil von Java und wurde in das JDK aufgenommen.

SQL Anywhere verfügt über einen 4.0-Treiber, bei dem es sich um einen Type 2-Treiber handelt.

SQL Anywhere unterstützt außerdem einen reinen Java JDBC-Treiber namens jConnect, der bei Sybase erhältlich ist.

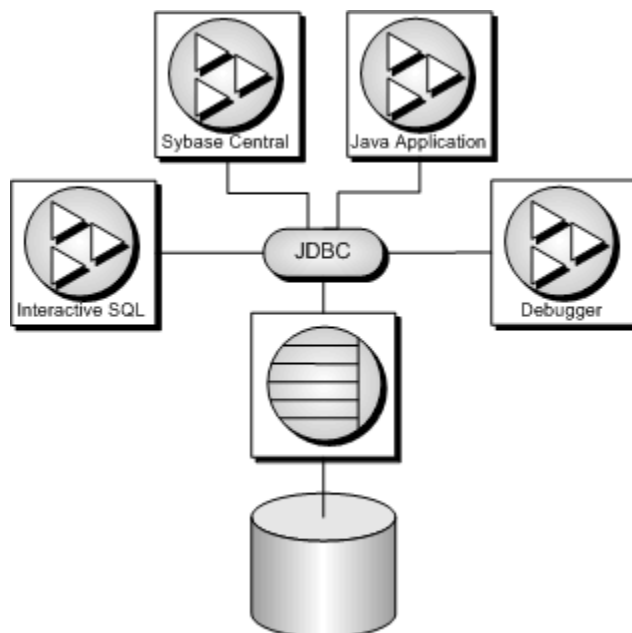
Zusätzlich zur Verwendung von JDBC als clientseitige Anwendungsprogrammierschnittstelle können Sie JDBC innerhalb des Datenbankservers verwenden, um mit Java in der Datenbank auf Daten zuzugreifen.

Siehe auch

- „JDBC-Treiber“ auf Seite 420

JDBC-Anwendungen

Sie können Java-Anwendungen entwickeln, die die JDBC-API verwenden, um eine Verbindung mit SQL Anywhere herzustellen. Einige der im Lieferumfang von SQL Anywhere enthaltenen Anwendungen verwenden JDBC, z.B. der Debugger, Sybase Central und Interactive SQL.



Java und JDBC sind ebenfalls wichtige Entwicklungswerkzeuge für UltraLite-Anwendungen.

JDBC kann sowohl aus Clientanwendungen als auch innerhalb einer Datenbank eingesetzt werden. Java-Klassen, die JDBC verwenden, bieten eine leistungsstärkere Alternative zu gespeicherten Prozeduren in SQL für die Einbeziehung von Programmierlogik in die Datenbank.

JDBC bietet eine SQL-Schnittstelle für Java-Anwendungen: Wenn Sie auf relationale Daten von Java zugreifen möchten, tun Sie dies über JDBC-Aufrufe.

Der Ausdruck **Clientanwendung** bezieht sich sowohl auf Anwendungen, die auf dem Computer des Benutzers laufen, als auch auf Mittelschicht-Anwendungsserver.

Die Beispiele veranschaulichen die unterschiedlichen Funktionen beim Einsatz von JDBC in SQL Anywhere. Weitere Hinweise zum Programmieren von JDBC finden Sie in jedem beliebigen JDBC-Programmierhandbuch.

Sie können JDBC auf folgende Weise mit SQL Anywhere verwenden:

- **JDBC auf dem Client** Java-Clientanwendungen können JDBC-Aufrufe in SQL Anywhere ausführen. Die Verbindung erfolgt über einen JDBC-Treiber.

SQL Anywhere verfügt über einen JDBC 4.0-Treiber, bei dem es sich um einen Type 2-Treiber handelt, und unterstützt außerdem den jConnect-Treiber für reine Java-Anwendungen, der ein Type 4-JDBC-Treiber ist.
- **JDBC in der Datenbank** In einer Datenbank installierte Java-Klassen können JDBC-Aufrufe ausführen, um mithilfe eines internen JDBC-Treibers auf Daten in der Datenbank zuzugreifen und diese zu ändern.

JDBC-Ressourcen

- **Beispiel-Quellcode** Quellcode für die Beispiele in diesem Abschnitt finden Sie im Verzeichnis `%SQLANYAMP16%\SQLAnywhere\JDBC`.
- **JDBC-Spezifikation** Weitere Hinweise über die JDBC Data Access API finden Sie unter <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html>.
- **Erforderliche Software** Sie benötigen TCP/IP für den jConnect-Treiber.

Der jConnect-Treiber steht unter <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect> zur Verfügung.

Siehe auch

- „jConnect-JDBC-Treiber“ auf Seite 424

JDBC-Treiber

SQL Anywhere unterstützt die folgenden zwei JDBC-Treiber:

- **SQL Anywhere JDBC 4.0-Treiber** Dieser Treiber kommuniziert mit SQL Anywhere über das Command Sequence-Client/Server-Protokoll. Sein Verhalten ist mit ODBC-, Embedded SQL- und

OLE DB-Anwendungen konsistent. Der SQL Anywhere JDBC 4.0-Treiber ist der empfohlene JDBC-Treiber für die Verbindung mit SQL Anywhere-Datenbanken. Der JDBC 4.0-Treiber kann nur mit JRE 1.6 oder später verwendet werden.

Der JDBC 4.0-Treiber nutzt die Vorteile der neuen automatischen JDBC-Treiberregistrierung. Wenn eine Anwendung den JDBC 4.0-Treiber verwenden soll, ist es daher nicht mehr notwendig, einen `Class.forName`-Aufruf zum Laden des JDBC-Treibers auszuführen. Stattdessen reicht es aus, die Datei *sajdbc4.jar* im Klassendatei-Pfad zu speichern und einfach `DriverManager.getConnection()` aufzurufen, wobei die URL mit `jdbc:sqlanywhere` beginnen muss.

Der JDBC 4.0-Treiber enthält nun die Ladelisten-Informationen, mit denen er als OSGi-Bundle (Open Services Gateway-Initiative) geladen werden kann.

Wenn Sie den JDBC 4.0-Treiber verwenden, wird in den Metadaten für NCHAR-Daten jetzt der Spaltentyp als `java.sql.Types.NCHAR`, `NVARCHAR` oder `LONGNVARCHAR` zurückgegeben. Zusätzlich können Anwendungen jetzt NCHAR-Daten unter Verwendung der Methode `Get/SetNString` oder `Get/SetNClob` statt `Get/SetString` bzw. `Get/SetClob` abrufen.

- **jConnect** Dieser Treiber ist eine 100% Pure Java-Implementierung des Treibers. Er kommuniziert mit SQL Anywhere über das TDS-Client/Server-Protokoll.

jConnect und die jConnect-Dokumentation stehen unter <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect> zur Verfügung.

Bei der Auswahl des geeigneten Treibers sollten die folgenden Faktoren beachtet werden:

- **Funktionen** Der SQL Anywhere JDBC 4.0-Treiber und jConnect sind JDBC 4.0-konform. Der SQL Anywhere JDBC-Treiber stellt vollständig scrollfähige Cursor bereit, wenn er mit einer SQL Anywhere-Datenbank verbunden ist. Der jConnect JDBC-Treiber stellt scrollfähige Cursor bereit, wenn er mit einem SQL Anywhere-Datenbankserver verbunden ist, aber die Ergebnismenge wird im Cache auf der Clientseite zwischengespeichert. Der jConnect-JDBC-Treiber stellt vollständig scrollfähige Cursor bereit, wenn er mit einer Adaptive Server Enterprise-Datenbank verbunden ist.

Die JDBC 4.0-API-Dokumentation ist verfügbar unter <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html>. Eine Zusammenfassung der vom SQL Anywhere-JDBC-Treiber unterstützten JDBC-API-Methoden finden Sie unter „JDBC 4.0-API-Unterstützung“ auf Seite 453.

- **"Pure Java"** Der jConnect-Treiber ist eine Pure Java-Lösung. Die SQL Anywhere-JDBC-Treiber basieren auf den SQL Anywhere-ODBC-Treibern und sind keine reinen Java-Lösungen.
- **Performance** Die SQL Anywhere-JDBC-Treiber bieten für die meisten Einsatzbereiche eine bessere Performance als der jConnect-Treiber.
- **Kompatibilität** Das vom jConnect-Treiber verwendete TDS-Protokoll wird mit Adaptive Server Enterprise gemeinsam genutzt. Einige Aspekte des Verhaltens dieses Treibers werden durch das Protokoll bestimmt und sind so konfiguriert, dass die Kompatibilität mit Adaptive Server Enterprise gewährleistet bleibt.

Hinweise zur Plattformverfügbarkeit für die SQL Anywhere-JDBC-Treiber und jConnect finden Sie unter <http://www.sybase.com/detail?id=1061806>.

Siehe auch

- „jConnect unter Windows Mobile“ [[SQL Anywhere Server - Datenbankadministration](#)]

JDBC-Programmstruktur

In JDBC-Anwendungen sind folgende Abläufe typisch:

1. **Verbindungsobjekt erstellen** Mit dem Aufruf der getConnection-Klassenmethode der Klasse DriverManager wird ein Connection-Objekt erstellt, das eine Verbindung mit einer Datenbank einrichtet.
2. **Statement-Objekt erstellen** Das Connection-Objekt generiert ein Statement-Objekt.
3. **SQL-Anweisung übergeben** Eine SQL-Anweisung, die in der Datenbankumgebung ausgeführt wird, wird an das Statement-Objekt übergeben. Wenn die Anweisung eine Abfrage ist, wird durch diese Aktion ein ResultSet-Objekt zurückgegeben.

Das ResultSet-Objekt enthält die von der SQL-Anweisung zurückgegebenen Daten, gibt jedoch jeweils nur eine Zeile aus (ähnlich wie die Arbeitsweise des Cursors).

4. **Schleife über die Zeilen der Ergebnismenge** Die next-Methode des ResultSet-Objekts führt zwei Aktionen aus:
 - Die aktuelle Zeile (die Zeile in der durch das ResultSet-Objekt ausgegebenen Ergebnismenge), wird eine Zeile weitergeschoben.
 - Ein boolescher Wert wird zurückgegeben, der angibt, ob eine Zeile vorhanden ist, zu der weitergeschoben werden kann.
5. **Für jede Zeile Werte abrufen** Für jede Zeile im ResultSet-Objekt werden Werte entweder mit dem Namen oder der Position der Spalte abgerufen. Mit der getData-Methode können Sie den Wert einer Spalte der aktuellen Zeile abrufen.

Java-Objekte können JDBC-Objekte verwenden, um mit einer Datenbank zu interagieren und Daten für ihre eigene Verwendung abrufen.

Unterschiede zwischen client- und serverseitigen JDBC-Verbindungen

Der Unterschied zwischen dem JDBC-Treiber auf dem Client bzw. auf dem Datenbankserver liegt darin, dass eine Verbindung mit der Datenbankumgebung hergestellt wird.

- **Clientseitig** Auf Clientseite erfordert die Herstellung einer Verbindung den SQL Anywhere JDBC-Treiber oder den jConnect JDBC-Treiber. Durch die Übergabe von Argumenten an DriverManager.getConnection wird die Verbindung hergestellt. Die Datenbankumgebung ist von der Perspektive der Clientanwendung aus eine externe Anwendung.

- **Serverseitig** Wenn JDBC innerhalb des Datenbankservers eingesetzt wird, ist bereits eine Verbindung vorhanden. Die Zeichenfolge "jdbc:default:connection" wird an DriverManager.getConnection übergeben, wodurch die JDBC-Anwendung mit der aktuellen Benutzerverbindung arbeiten kann. Dies ist ein schneller, effizienter und sicherer Vorgang, weil die Clientanwendung die Sicherheitsprüfung der Datenbank zur Herstellung der Verbindung bereits bestanden hat. Benutzer-ID und Kennwort wurden angegeben und brauchen nicht noch einmal angegeben zu werden. Der serverseitige JDBC-Treiber kann nur mit der Datenbank der aktuellen Verbindung eine Verbindung herstellen.

JDBC-Klassen können so geschrieben werden, dass sie auf der Client- und auf der Serverseite ausgeführt werden, indem Sie eine einzige bedingte Anweisung für die Angabe des URL verwenden. Eine externe Verbindung erfordert den Hostnamen und die Portnummer, während die interne Verbindung "jdbc:default:connection" benötigt.

SQL Anywhere JDBC-Treiber

Der SQL Anywhere JDBC 4.0-Treiber bietet im Vergleich zum jConnect-JDBC-Treiber in reinem Java zwar einige Vorteile im Hinblick auf Performance und Funktionen, stellt aber keine reine Java-Lösung bereit. In der Regel wird der SQL Anywhere JDBC 4.0-Treiber empfohlen.

Siehe auch

- „JDBC-Treiber“ auf Seite 420

So laden Sie den SQL Anywhere JDBC 4.0-Treiber

Vergewissern Sie sich, dass sich der SQL Anywhere JDBC 4.0-Treiber in Ihrem Klassendatei-Pfad befindet.

```
set classpath=%SQLANY16%\java\sajdbc4.jar;%classpath%
```

Der JDBC 4.0-Treiber nutzt die Vorteile der neuen automatischen JDBC-Treiberregistrierung. Der Treiber wird beim Start der Ausführung automatisch geladen, wenn er sich im Klassendatei-Pfad befindet.

Erforderliche Dateien

Die Java-Komponente des SQL Anywhere JDBC 4.0-Treibers ist in der Datei *sajdbc4.jar* enthalten, die Sie im Unterverzeichnis *Java* der SQL Anywhere-Installation finden. Unter Windows ist die native Komponente die Datei *dbjdbc16.dll* im Unterverzeichnis *bin32* oder *bin64* der SQL Anywhere-Installation. Für Unix und Linux ist die native Komponente die Datei *libdbjdbc16.so*. Diese Komponente muss sich im Systempfad befinden.

Verbindungszeichenfolgen für den SQL Anywhere JDBC-Treiber

Um über den SQL Anywhere-JDBC-Treiber eine Verbindung zu einer Datenbank herzustellen, müssen Sie eine URL für die Datenbank angeben. Beispiel:

```
Connection con = DriverManager.getConnection(
    "jdbc:sqlanywhere:DSN=SQL Anywhere 16 Demo" );
```

Die URL enthält **jdbc:sqlanywhere:**, gefolgt von einer Verbindungszeichenfolge. Wenn sich die Datei *sajdbc4.jar* in Ihrem Klassendatei-Pfad befindet, wurde der JDBC 4.0-Treiber automatisch geladen und verarbeitet die URL. Wie im Beispiel gezeigt, kann eine ODBC-Datenquelle (**DSN**) aus Gründen des Bedienkomforts angegeben werden, aber Sie können auch explizite Verbindungsparameter, die durch Semikola getrennt sind, zusätzlich zu oder an Stelle der Datenquellen-Verbindungsparameter verwenden.

Falls Sie keine Datenquelle verwenden, müssen Sie in der Verbindungszeichenfolge alle erforderlichen Verbindungsparameter angeben:

```
Connection con = DriverManager.getConnection(
    "jdbc:sqlanywhere:UserID=DBA;Password=sql;Start=..." );
```

Der **Driver**-Verbindungsparameter ist nicht erforderlich, da weder der ODBC-Treiber noch der ODBC-Treibermanager verwendet wird. Wenn er vorhanden ist, wird er ignoriert.

Siehe auch

- „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)]

jConnect-JDBC-Treiber

Der jConnect-Treiber steht unter <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect> als getrennter Download zur Verfügung. Die Dokumentation für jConnect ist auf derselben Seite zu finden.

Wenn Sie JDBC von einem Applet aus verwenden möchten, müssen Sie Verbindungen mit SQL Anywhere-Datenbanken mithilfe des jConnect-JDBC-Treibers herstellen.

Die Dateien des jConnect-Treibers

jConnect wird als JAR-Datei namens *jconn4.jar* geliefert. Diese Datei befindet sich in Ihrem jConnect-Installationsspeicherort.

Klassendatei-Pfad für jConnect festlegen

Damit Ihre Anwendung jConnect verwenden kann, müssen sich die jConnect-Klassen beim Kompilieren und Ausführen in Ihrem Klassendatei-Pfad befinden, sodass der Java-Compiler und die Java-Machine die notwendigen Dateien finden können.

Der folgende Befehl fügt den jConnect -Treiber in eine vorhandene CLASSPATH-Umgebungsvariable ein, wobei *jConnect-Pfad* Ihr jConnect-Installationsverzeichnis ist.

```
set classpath=jconnect-path\classes\jconn4.jar;%classpath%
```

jConnect-Klassen importieren

Die Klassen in jConnect sind alle in `com.sybase.jdbc4.jdbc`. Sie müssen diese Klassen am Anfang einer jeden Quelldatei importieren:

```
import com.sybase.jdbc4.jdbc.*
```

Kennwörter verschlüsseln

SQL Anywhere unterstützt Kennwortverschlüsselung bei jConnect-Verbindungen.

jConnect-Systemobjekte in einer Datenbank installieren

Um mit jConnect auf Systemtabellendaten (Datenbank-Metadaten) zugreifen zu können, müssen Sie die jConnect-Systemobjekte zu Ihrer Datenbank hinzufügen.

Voraussetzungen

Sie müssen die Systemprivilegien ALTER DATABASE, BACKUP DATABASE und SERVER OPERATOR haben und als einziger Benutzer mit der Datenbank verbunden sein.

Sichern Sie Ihre Datenbankdateien vor dem Upgrade. Wenn Sie ein Upgrade einer Datenbank versuchen und es fehlschlägt, wird die Datenbank unbenutzbar.

Kontext und Bemerkungen

jConnect-Systemobjekte werden standardmäßig in einer SQL Anywhere-Datenbank installiert, wenn Sie das Dienstprogramm dbinit ausführen. Außerdem sind jConnect-Systemobjekte nicht für Windows Mobile-Datenbanken erforderlich. Sie können die jConnect-Systemobjekte der Datenbank hinzufügen, wenn Sie die Datenbank erstellen, oder zu einem späteren Zeitpunkt, indem Sie ein Upgrade der Datenbank durchführen. Sie können das Upgrade der Datenbank aus Sybase Central oder mit dem dbupgrad-Dienstprogramm durchführen.

Aufgabe

1. Stellen Sie in Sybase Central eine Verbindung zur Datenbank mithilfe des **SQL Anywhere 16**-Plug-Ins her.
2. Klicken Sie auf **Extras » SQL Anywhere 16** und dann auf **Upgrade einer Datenbank**.
3. Befolgen Sie die Anweisungen im **Assistenten zum Upgrade einer Datenbank**.

Ergebnisse

Die jConnect-Systemobjekte werden zur Datenbank hinzugefügt.

Siehe auch

- „Dienstprogramm zum Upgrade (dbupgrad)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „jConnect unter Windows Mobile“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Sicherung und Datenwiederherstellung“ [[SQL Anywhere Server - Datenbankadministration](#)]

jConnect-Treiber laden

Vergewissern Sie sich, dass sich der jConnect-Treiber in Ihrem Klassendatei-Pfad befindet. Die Treiberdatei *jconn4.jar* befindet sich im Unterverzeichnis *classes* Ihrer jConnect-Installation.

```
set classpath=.;c:\jConnect-7_0\classes\jconn4.jar;%classpath%
```

Der jConnect-Treiber nutzt die Vorteile der neuen automatischen JDBC-Treiberregistrierung. Der Treiber wird beim Start der Ausführung automatisch geladen, wenn er sich im Klassendatei-Pfad befindet.

jConnect-Treiber-Verbindungszeichenfolgen

Um per jConnect eine Verbindung zu einer Datenbank herzustellen, müssen Sie eine URL (Uniform Resource Locator) für die Datenbank angeben. Beispiel:

```
Connection con = DriverManager.getConnection(  
    "jdbc:sybase:Tds:localhost:2638", "DBA", "sql");
```

Die URL setzt sich wie folgt zusammen:

```
jdbc:sybase:Tds:host:port
```

Die einzelnen Komponenten sind:

- **jdbc:sybase:Tds** Der jConnect JDBC Treiber unter Verwendung des TDS-Anwendungsprotokolls
- **host** Die IP-Adresse oder der Name des Computers, auf dem der Server läuft. Wenn Sie eine Verbindung mit demselben Host herstellen, können Sie localhost verwenden, d.h. das Computersystem, bei dem Sie angemeldet sind.
- **port** Die Portnummer, an der der Datenbankserver auf Verbindungsanforderungen wartet. Die SQL Anywhere zugewiesene Portnummer ist 2638. Sie sollten diese Nummer verwenden, es sei denn, es gibt besondere Gründe, dies nicht zu tun.

Die Verbindungszeichenfolge darf nicht länger als 252 Zeichen sein.

Wenn Sie SQL Anywhere Personal Server verwenden, achten Sie darauf, in der Befehlszeile die Option für die TCP/IP-Unterstützung einzuschließen, wenn der Server gestartet wird.

Datenbanken mit jConnect-Verbindungszeichenfolgen angeben

Jeder SQL Anywhere-Datenbankserver kann eine oder mehrere Datenbank laden. Wenn der angegebene URL bei der Verbindungserstellung über jConnect einen Server, aber keine Datenbank angibt, wird versucht, eine Verbindung zur Standarddatenbank auf dem Server herzustellen.

Sie können eine bestimmte Datenbank angeben, indem Sie die URL-Angabe wie folgt erweitern:

Mit dem Parameter ServiceName

```
jdbc:sybase:Tds:host:port?ServiceName=database
```

Das Fragezeichen, gefolgt von einer Reihe von Zuordnungen, ist ein Standardverfahren, um einer URL Argumente zu liefern. Die Groß- und Kleinschreibung von ServiceName wird nicht berücksichtigt, und vor bzw. nach dem Gleichheitszeichen (=) dürfen keine Leerstellen stehen. Der Parameter *Datenbank* ist der Datenbankname, nicht der Servername. Der Datenbankname darf weder den Pfad noch das Dateisuffix enthalten. Beispiel:

```
Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638?ServiceName=demo", "DBA", "sql");
```

Mit dem Parameter RemotePWD

Für die Übergabe zusätzlicher Parameter an den Server gibt es eine Behelfslösung.

Dieses Verfahren gestattet es Ihnen unter Verwendung des Felds RemotePWD, zusätzliche Parameter bereitzustellen, wie etwa den Datenbanknamen oder eine Datenbankdatei. Mit der put-Methode legen Sie RemotePWD als ein Properties-Feld fest.

Der folgende Code veranschaulicht, wie das Feld verwendet wird.

```
import java.util.Properties;
.
.
.
Properties props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "sql" );
props.put( "RemotePWD", ",DatabaseFile=mydb.db" );

Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", props );
```

Wie in diesem Beispiel zu sehen, muss dem Verbindungsparameter DatabaseFile ein Komma vorangestellt werden. Mit dem DatabaseFile-Parameter für die Datenbankdatei können Sie mit jConnect eine Datenbank auf einem Server starten. Standardmäßig wird die Datenbank mit autostop=YES gestartet. Wenn Sie utility_db mit dem Verbindungsparameter DatabaseFile (DBF) oder DatabaseName (DBN) angeben (z.B. DBN=utility_db), wird die Dienstprogramme-Datenbank automatisch gestartet.

Siehe auch

- „Die Dienstprogrammdatenbank“ [[SQL Anywhere Server - Datenbankadministration](#)]

Für jConnect-Verbindungen eingestellte Datenbankoptionen

Wenn sich eine Anwendung mit der Datenbank unter Verwendung des jConnect-Treibers verbindet, wird die gespeicherte Prozedur sp_tsql_environment aufgerufen. Die sp_tsql_environment-Prozedur legt einige Datenbankoptionen für die Kompatibilität mit Adaptive Server Enterprise fest.

Siehe auch

- „Eigenschaften von Sybase Open Client- und jConnect-Verbindungen“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „sp_tsql_environment-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Verbindungen aus einer JDBC-Clientanwendung

Datenbank-Metadaten sind bei der Verwendung eines SQL Anywhere-JDBC-Treibers immer verfügbar.

Wenn Sie von einer JDBC-Anwendung, die jConnect verwendet, auf Datenbank-Systemtabellen zugreifen (Datenbank-Metadaten), müssen Sie eine Reihe von jConnect-Systemobjekten in Ihre

Datenbank einfügen. Diese Prozeduren werden in allen Datenbanken standardmäßig installiert. Die dbinit-Option -i verhindert diese Installation.

Weitere Hinweise zum Hinzufügen von jConnect-Systemobjekten in eine Datenbank finden Sie unter [„jConnect-JDBC-Treiber“ auf Seite 424](#).

Die folgende, vollständige Java-Anwendung ist ein Befehlszeilenprogramm, das eine Verbindung mit einer laufenden Datenbank herstellt, eine Reihe von Daten in der Befehlszeile ausgibt und beendet wird.

Der erste Schritt einer beliebigen JDBC-Anwendung ist die Herstellung der Verbindung mit der Datenbank, wenn sie mit dieser arbeiten will.

Dieses Beispiel veranschaulicht eine externe Verbindung, bei der es sich um eine reguläre Client/Server-Verbindung handelt. Hinweise zum Erstellen einer internen Verbindung von Java-Klassen, die im Datenbankserver laufen, finden Sie unter [„Verbindung von einer serverseitigen JDBC-Klasse aus herstellen“ auf Seite 431](#).

Code eines Verbindungsbeispiels

Im folgenden Beispiel wird standardmäßig die JDBC 4.0-Version des SQL Anywhere-JDBC-Treibers verwendet, um eine Verbindung mit der Datenbank herzustellen. Wenn Sie einen anderen Treiber verwenden möchten, können Sie den Treibernamen (jdbc4, jConnect) in der Befehlszeile übergeben. Beispiele für die Verwendung des JDBC 4.0-Treibers und von jConnect sind im Code enthalten. In diesem Beispiel wird vorausgesetzt, dass bereits ein Datenbankserver unter Verwendung der Beispieldatenbank gestartet wurde. Der Quellcode befindet sich in der Datei *JDBCConnect.java* im Verzeichnis `%SQLANYSAAMP16%\SQLAnywhere\JDBC`.

```
import java.io.*;
import java.sql.*;

public class JDBCConnect
{
    public static void main( String args[] )
    {
        try
        {
            String arg;
            Connection con;

            // Select the JDBC driver and create a connection.
            // May throw a SQLException.
            // Choices are:
            // 1. jConnect driver
            // 2. SQL Anywhere JDBC 4.0 driver
            arg = "jdbc4";
            if( args.length > 0 ) arg = args[0];
            if( arg.compareToIgnoreCase( "jconnect" ) == 0 )
            {
                con = DriverManager.getConnection(
                    "jdbc:sybase:Tds:localhost:2638", "DBA", "sql");
            }
            else
            {
                con = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
            }
        }
    }
}
```

```

        System.out.println("Using "+arg+" driver");

        // Create a statement object, the container for the SQL
        // statement. May throw a SQLException.
        Statement stmt = con.createStatement();

        // Create a result set object by executing the query.
        // May throw a SQLException.
        ResultSet rs = stmt.executeQuery(
            "SELECT ID, GivenName, Surname FROM Customers");

        // Process the result set.
        while (rs.next())
        {
            int value = rs.getInt(1);
            String FirstName = rs.getString(2);
            String LastName = rs.getString(3);
            System.out.println(value+" "+FirstName+" "+LastName);
        }
        rs.close();
        stmt.close();
        con.close();
    }
    catch (SQLException sqe)
    {
        System.out.println("Unexpected exception : " +
            sqe.toString() + ", sqlstate = " +
            sqe.getSQLState());

        System.exit(1);
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.exit(1);
    }

    System.exit(0);
}
}

```

Funktionsweise des Verbindungsbeispiels

Das Beispiel für eine externe Verbindung ist ein Java-Befehlszeilenprogramm.

Pakete importieren

Die Anwendung erfordert eine Reihe von Paketen, die in der ersten Zeile von *JDBCCConnect.java* importiert werden:

- Das Paket `java.io` enthält die Java-Eingabe/Ausgabe-Klassen, die für die Ausgabe im Konsolenfenster erforderlich sind.
- Das Paket `java.sql` enthält die JDBC-Klassen von `java.sql`, die für alle JDBC-Anwendungen erforderlich sind.

Die Methode `main`

Jede Java-Anwendung erfordert eine Klasse mit einer Methode mit dem Namen `main`, die beim Programmstart aufgerufen wird. In diesem einfachen Beispiel ist `JDBCCConnect.main` die einzige Methode in der Anwendung.

Die Methode `JDBCConnect.main` führt die folgenden Aufgaben aus:

1. Bestimmt anhand der Befehlszeilenoption, welcher Treiber zu laden ist. Die SQL Anywhere-Treiber für JDBC 4.0 und jConnect 7.0 werden beim Start automatisch geladen, wenn sie sich im Klassendatei-Pfad befinden.
2. Stellt unter Verwendung der ausgewählten JDBC-Treiber-URL eine Verbindung mit der laufenden Standarddatenbank her. `DriverManager.getConnection` stellt unter Verwendung der angegebenen URL eine Verbindung her.
3. Erstellt ein Anweisungsobjekt, das der Wrapper für die SQL-Anweisung ist.
4. Erstellt eine Ergebnismenge, indem eine SQL-Anfrage ausgeführt wird.
5. Durchläuft die Ergebnismenge und gibt die Spalteninformationen aus.
6. Schließt die Ergebnismengen-, das Anweisungs- und das Verbindungsobjekt.

Verbindungsbeispiel ausführen

Die Schritte zum Erstellen und Ausführen einer JDBC-Anwendung werden anhand eines Beispiels gezeigt.

Voraussetzungen

Ein Java Development Kit (JDK) muss installiert sein.

Kontext und Bemerkungen

Zwei verschiedene Typen von Verbindungen mit JDBC können hergestellt werden. Eine ist die clientseitige Verbindung und die andere ist die serverseitige Verbindung. Im folgenden Beispiel wird eine clientseitige Verbindung verwendet.

Aufgabe

1. Wechseln Sie an einer Eingabeaufforderung in das Verzeichnis `%SQLANY16%\SQLAnywhere\JDBC`.
2. Starten Sie einen Datenbankserver mit der Beispieldatenbank auf Ihrem lokalen Computer mit dem folgenden Befehl:

```
dbsrv16 "%SQLANY16%\demo.db"
```
3. Setzen Sie die `CLASSPATH`-Umgebungsvariable. In diesem Beispiel wird der in der Datei `sajdbc4.jar` enthaltene SQL Anywhere JDBC 4.0-Treiber verwendet.

```
set classpath=.;%SQLANY16%\java\sajdbc4.jar
```

Wenn Sie stattdessen den jConnect-Treiber verwenden, geben Sie Folgendes an (wobei *Pfad* Ihr jConnect-Installationsverzeichnis ist):

```
set classpath=.;jconnect-path\classes\jconn4.jar
```

4. Führen Sie den folgenden Befehl aus, um das Beispiel zu kompilieren:

```
javac JDBCCConnect.java
```

5. Führen Sie den folgenden Befehl aus, um das Beispiel auszuführen:

```
java JDBCCConnect
```

Fügen Sie ein Befehlszeilenargument ein, z.B. **jconnect**, um einen anderen JDBC-Treiber zu laden.

```
java JDBCCConnect jconnect
```

6. Überprüfen Sie, ob an der Eingabeaufforderung eine Liste von Identifizierungsnummern mit Kundennamen angezeigt wird.

Wenn der Verbindungsversuch fehlschlägt, erscheint stattdessen eine Fehlermeldung. Prüfen Sie, ob Sie alle erforderlichen Schritte ausgeführt haben. Überprüfen Sie, ob Ihr Klassendatei-Pfad richtig ist. Eine falsche CLASSPATH-Einstellung führt dazu, dass eine Klasse nicht gefunden werden kann.

Ergebnisse

Eine Liste von Identifizierungsnummern mit Kundennamen wird angezeigt.

Verbindung von einer serverseitigen JDBC-Klasse aus herstellen

SQL-Anweisungen werden in JDBC mithilfe der Methode `createStatement` des `Connection`-Objekts aufgebaut. Auch Klassen, die innerhalb des Servers laufen, müssen eine Verbindung herstellen, damit ein `Connection`-Objekt erstellt wird.

Eine Verbindung von einer serverseitigen JDBC-Klasse herzustellen ist einfacher, als eine externe Verbindung einzurichten. Da der Benutzer bereits mit der Datenbank verbunden ist, verwendet die Klasse einfach die aktuelle Verbindung.

Beispielcode für eine serverseitige Verbindung

Der folgende Code ist der Quellcode für das serverseitige Verbindungsbeispiel. Dies ist eine modifizierte Version des `JDBCCConnect.java`-Beispiels und befindet sich in `%SQLANYWHERE%\JDBC\JDBCCConnect2.java`.

```
import java.io.*;
import java.sql.*;

public class JDBCCConnect2
{
    public static void main( String args[] )
    {
        try
        {
```

```
// Open the connection. May throw a SQLException.
Connection con = DriverManager.getConnection(
    "jdbc:default:connection" );

// Create a statement object, the container for the SQL
// statement. May throw a SQLException.
Statement stmt = con.createStatement();
// Create a result set object by executing the query.
// May throw a SQLException.
ResultSet rs = stmt.executeQuery(
    "SELECT ID, GivenName, Surname FROM Customers");

// Process the result set.
while (rs.next())
{
    int value = rs.getInt(1);
    String FirstName = rs.getString(2);
    String LastName = rs.getString(3);
    System.out.println(value+" "+FirstName+" "+LastName);
}
rs.close();
stmt.close();
con.close();
}
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
}
catch (Exception e)
{
    e.printStackTrace();
}
}
```

Unterschiede des Beispiels für eine serverseitige Verbindung

Das serverseitige Verbindungsbeispiel ist nahezu identisch mit dem clientseitigen Verbindungsbeispiel, mit den folgenden Unterschieden:

1. Der JDBC-Treiber muss nicht vorher geladen werden.
2. Sie stellt über die aktuelle Verbindung eine Verbindung mit der laufenden Standarddatenbank her:
Die URL im getConnection-Aufruf wurde wie folgt geändert:

```
Connection con = DriverManager.getConnection(
    "jdbc:default:connection" );
```

3. Die System.exit()-Anweisungen wurden entfernt.

Beispiel für eine serverseitige Verbindung ausführen

Die Schritte zum Erstellen und Ausführen einer serverseitigen JDBC-Anwendung werden anhand eines Beispiels gezeigt.

Voraussetzungen

Ein Java Development Kit (JDK) muss installiert sein.

Sie müssen die folgenden Systemprivilegien haben.

- MANAGE ANY EXTERNAL OBJECT zum Installieren von Java-Klassen
- CREATE PROCEDURE und CREATE EXTERNAL REFERENCE zum Erstellen von externen Prozeduren

Kontext und Bemerkungen

Zwei verschiedene Typen von Verbindungen mit JDBC können hergestellt werden. Eine ist die clientseitige Verbindung und die andere ist die serverseitige Verbindung. Im folgenden Beispiel wird eine serverseitige Verbindung verwendet.

Aufgabe

1. Wechseln Sie an einer Eingabeaufforderung in das Verzeichnis `%SQLANYSAMPl6%\SQLAnywhere\JDBC`.

```
cd %SQLANYSAMPl6%\SQLAnywhere\JDBC
```

2. Beim serverseitigen JDBC ist es nicht erforderlich, die CLASSPATH-Umgebungsvariable zu setzen, außer wenn der Server aus einem anderen aktuellen Arbeitsverzeichnis gestartet wird.

```
set classpath=.;%SQLANYSAMPl6%\SQLAnywhere\JDBC
```

3. Starten Sie einen Datenbankserver mit der Beispieldatenbank auf Ihrem lokalen Computer, indem Sie den folgenden Befehl ausführen:

```
dbsrvl6 "%SQLANYSAMPl6%\demo.db"
```

4. Geben Sie folgenden Befehl ein, um das Beispiel zu kompilieren:

```
javac JDBCConnect2.java
```

5. Installieren Sie die Klasse mithilfe von Interactive SQL in derselben Datenbank. Führen Sie die folgende Anweisung aus (wobei möglicherweise ein Pfad zur Klassendatei erforderlich ist):

```
INSTALL JAVA NEW  
FROM FILE 'JDBCConnect2.class';
```

Es besteht außerdem die Möglichkeit, die Klasse mit Sybase Central zu installieren. Während Sie mit der Beispieldatenbank verbunden sind, öffnen Sie den Unterordner **Java** unter **Externe Umgebungen** und klicken auf **Datei » Neu » Java-Klasse**. Befolgen Sie dann die Anweisungen im Assistenten.

- Definieren Sie eine gespeicherte Prozedur namens JDBCConnect, die als Wrapper für die Methode JDBCConnect2.main in der Klasse agiert:

```
CREATE PROCEDURE JDBCConnect(OUT args LONG VARCHAR)
  EXTERNAL NAME 'JDBCConnect2.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
```

- Rufen Sie die Methode JDBCConnect2.main wie folgt auf:

```
CALL JDBCConnect();
```

Beim ersten Aufruf einer Java-Klasse in einer Sitzung muss die Java VM gestartet werden. Dies kann einige Sekunden dauern.

- Überprüfen Sie, ob im Meldungsfenster des Datenbankservers eine Liste mit Identifizierungsnummern und Kundennamen angezeigt wird.

Wenn der Verbindungsversuch fehlschlägt, erscheint stattdessen eine Fehlermeldung. Prüfen Sie, ob Sie alle erforderlichen Schritte ausgeführt haben.

Ergebnisse

Eine Liste von Identifizierungsnummern mit Kundennamen wird im Meldungsfenster des Datenbankservers angezeigt.

Hinweise zu JDBC-Verbindungen

- Autocommit-Modus (automatischer Festschreibemodus)** Die JDBC-Spezifikation erfordert, dass standardmäßig nach jeder Datenmanipulationsanweisung ein COMMIT ausgeführt wird. Derzeit ist das clientseitige JDBC-Verhalten, die Transaktion festzuschreiben (Autocommit ist TRUE), und das serverseitige Verhalten ist, sie nicht festzuschreiben (Autocommit ist FALSE). Um in clientseitigen und serverseitigen Anwendungen dasselbe Verhalten zu erhalten, können Sie eine Anweisung, wie die folgende, verwenden:

```
con.setAutoCommit( false );
```

In dieser Anweisung ist con das aktuelle Verbindungsobjekt. Sie könnten auch Autocommit auf TRUE setzen.

- Isolationsstufe der Transaktion einstellen** Um die Isolationsstufe der Transaktion einzustellen, muss die Anwendung die Connection.setTransactionIsolation-Methode mit einen der folgenden Werte aufrufen.

Verwenden Sie für den SQL Anywhere JDBC 4.0-Treiber Folgendes:

- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE
- sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
- sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_SNAPSHOT
- sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_READONLY_SNAPSHOT

Im folgenden Beispiel wird mithilfe des JDBC 4.0-Treibers die Isolationsstufe der Transaktion auf SNAPSHOT gesetzt.

```
try
{
    con.setTransactionIsolation(
        sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
    );
}
catch( Exception e )
{
    System.err.println( "Error! Could not set isolation level" );
    System.err.println( e.getMessage() );
    printExceptions( (SQLException)e );
}
```

Weitere Hinweise zu `getTransactionIsolation` und `setTransactionIsolation` finden Sie in der `java.sql.Connection`-Schnittstelle unter <http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/>.

- **Standardwerte für die Verbindung** Bei der serverseitigen JDBC erstellt nur der erste Aufruf von `getConnection("jdbc:default:connection")` eine neue Verbindung mit den Standardwerten. Nachfolgende Aufrufe geben einen Wrapper der aktuellen Verbindung mit allen unveränderten Verbindungseigenschaften zurück. Wenn Sie bei der ersten Verbindungsaufnahme Autocommit auf `FALSE` setzen, geben alle nachfolgenden `getConnection`-Aufrufe innerhalb desselben Java-Codes eine Verbindung mit Autocommit mit dem Wert `FALSE` zurück.

Es kann sinnvoll sein, die Verbindungseigenschaften beim Schließen der Verbindung auf die Standardwerte zurücksetzen zu lassen, damit nachfolgende Verbindungen mit Standard-JDBC-Werten eingerichtet werden. Der folgende Code legt dies fest:

```
Connection con =
    DriverManager.getConnection("jdbc:default:connection");

boolean oldAutoCommit = con.getAutoCommit();
try
{
    // main body of code here
}
finally
{
    con.setAutoCommit( oldAutoCommit );
}
```

Diese Vorgänge beziehen sich nicht nur auf Autocommit, sondern auch auf andere Verbindungseigenschaften, wie etwa die Isolationsstufe von Transaktionen und den reinen Lesemodus.

Weitere Hinweise zu `getTransactionIsolation`, `setTransactionIsolation` und `isReadOnly` finden Sie in der Dokumentation zur `java.sql.Connection`-Schnittstelle unter <http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/>.

Datenzugriff mit JDBC

Java-Anwendungen, die einige oder alle Klassen in der Datenbank enthalten, bieten einen erheblichen Vorteil gegenüber traditionellen in SQL geschriebenen gespeicherten Prozeduren. Als Einführung kann es jedoch hilfreich sein, Parallelen zu mit SQL geschriebenen gespeicherten Prozeduren zu ziehen, um die Fähigkeiten von JDBC zu demonstrieren. In den folgenden Beispielen werden Java-Klassen geschrieben, die eine Zeile in die Tabelle `Departments` einfügen.

Wie auch bei anderen Schnittstellen können SQL-Anweisungen in JDBC entweder **statisch** oder **dynamisch** sein. Static SQL-Anweisungen werden in der Java-Anwendung aufgebaut und zur Datenbank gesandt. Der Datenbankserver analysiert die Anweisung syntaktisch, wählt einen Ausführungsplan und führt die Anweisung aus. Syntaktische Analyse und Auswahl eines Ausführungsplans werden in der Folge als **Vorbereitung** der Anweisung bezeichnet.

Wenn eine ähnliche Anweisung häufig ausgeführt werden soll (z.B. viele Einfügungen in eine Tabelle), kann dies zu erheblichem Overhead in der statischen SQL führen, weil jedesmal der Vorbereitungsschritt ausgeführt werden muss.

Im Gegensatz dazu enthält eine Dynamic SQL-Anweisung Platzhalter. Die Anweisung wird einmal mithilfe dieser Platzhalter vorbereitet und kann anschließend ohne zusätzlichen Vorbereitungsaufwand vielfach ausgeführt werden. Weitere Hinweise zu Dynamic SQL finden Sie unter „[Vorbereitete Anweisungen für effizienteren Zugriff verwenden](#)“ auf Seite 440.

Vorbereitung für die JDBC-Beispiele

Die Codefragmente im folgenden Abschnitt stammen aus der vollständigen Klasse in `%SQLANYSAMPI6%\SQLAnywhere\JDBC\JDBCExample.java`. In Vorbereitung auf diese Abschnitte wird die Java-Beispielanwendung kompiliert und in der Datenbank installiert.

Voraussetzungen

Ein Java Development Kit (JDK) muss installiert sein.

Um eine Klasse zu installieren, benötigen Sie das `MANAGE ANY EXTERNAL OBJECT`-Systemprivileg.

Aufgabe

1. Kompilieren Sie den Quellcode `JDBCExample.java`.

2. Stellen Sie über Interactive SQL eine Verbindung mit der Datenbank her.
3. Installieren Sie die Datei *JDBCExample.class* in der Beispieldatenbank, indem Sie die folgende Anweisung in Interactive SQL ausführen:

```
INSTALL JAVA NEW
FROM FILE 'JDBCExample.class';
```

Wenn der Datenbankserver nicht aus demselben Verzeichnis gestartet wurde wie die Klassendatei und der Pfad zur Klassendatei nicht im CLASSPATH des Datenbankservers aufgeführt ist, müssen Sie den Pfad zur Klassendatei in der INSTALL-Anweisung angeben.

Es besteht außerdem die Möglichkeit, die Klasse mit Sybase Central zu installieren. Während Sie mit der Beispieldatenbank verbunden sind, öffnen Sie den Unterordner **Java** unter **Externe Umgebungen** und klicken auf **Datei » Neu » Java-Klasse**. Befolgen Sie die Anweisungen des Assistenten.

Ergebnisse

Die Klassendatei JDBCExample wird in der Datenbank installiert und ist bereit für die Demonstration.

Einfügen, Aktualisieren und Löschen mit JDBC

SQL-Anweisungen wie INSERT, UPDATE und DELETE, die keine Ergebnismengen zurückgeben, werden mithilfe der Methode `executeUpdate` der Klasse `Statement` ausgeführt. Anweisungen wie CREATE TABLE und andere Anweisungen zur Datendefinition können auch mit `executeUpdate` ausgeführt werden.

Die Methoden `addBatch`, `clearBatch`, und `executeBatch` der Klasse `Statement` können ebenfalls verwendet werden. Da die JDBC-Spezifikation im Hinblick auf das Verhalten der `executeBatch`-Methode der Klasse `Statement` unklar ist, sollten die folgenden Hinweise in Betracht gezogen werden, wenn diese Methode mit dem SQL Anywhere-JDBC-Treiber verwendet wird:

- Die Verarbeitung des Batches wird sofort angehalten, wenn eine SQL-Ausnahmebedingung oder eine Ergebnismenge erkannt wird. Beim Anhalten der Batchverarbeitung wird von der `executeBatch`-Methode ein `BatchUpdateException`-Objekt ausgegeben. Mit dem Aufruf der `getUpdateCounts`-Methode für das `BatchUpdateException`-Objekt wird ein Ganzzahl-Array von Zeilenzahlangaben zurückgegeben, in dem die Menge der Anzahlangaben vor dem Batchfehler eine gültige nicht negative Aktualisierungsanzahl enthält, während alle Anzahlangaben ab dem Batchfehler den Wert -1 enthalten. Das Casting des `BatchUpdateException`-Objekts in ein `SQLException`-Objekt liefert zusätzliche Einzelheiten dazu, warum die Batchverarbeitung angehalten wurde.
- Der Batchinhalt wird nur gelöscht, wenn die `clearBatch`-Methode explizit aufgerufen wurde. Daher wird bei wiederholtem Aufrufen der `executeBatch`-Methode der Batch immer wieder neu ausgeführt. Außerdem wird beim Aufrufen von `execute(sql_query)` oder `executeQuery(sql_query)` zwar die angegebene SQL Abfrage korrekt ausgeführt, aber der zugrundeliegende Batch wird nicht gelöscht. Wenn Sie also die `executeBatch`-Methode aufrufen, gefolgt von `execute(sql_query)` und gefolgt von einem erneuten Aufruf der `executeBatch`-Methode, werden die im Batch zusammengefassten Anweisungen ausgeführt, anschließend wird die angegebene SQL-Abfrage ausgeführt und schließlich werden die im Batch zusammengefassten Anweisungen erneut ausgeführt.

Das folgende Codefragment veranschaulicht, wie eine INSERT-Anweisung ausgeführt wird. Es verwendet ein Statement-Objekt, das an die InsertStatic-Methode als Argument übergeben wurde.

```
public static void InsertStatic( Statement stmt )
{
    try
    {
        int iRows = stmt.executeUpdate(
            "INSERT INTO Departments (DepartmentID, DepartmentName)"
            + " VALUES (201, 'Eastern Sales')" );
        // Print the number of rows inserted
        System.out.println(iRows + " rows inserted");
    }
    catch (SQLException sqe)
    {
        System.out.println("Unexpected exception : " +
            sqe.toString() + ", sqlstate = " +
            sqe.getSQLState());
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

Hinweise

- Dieses Codefragment ist Teil der Datei *JDBCExample.java*, die im Verzeichnis *%SQLANYSAMPI6%\SQLAnywhere\JDBC* enthalten ist.
- Die Methode `executeUpdate` gibt eine Ganzzahl zurück, die die Anzahl der Zeilen ausdrückt, die von dem Vorgang betroffen waren. In diesem Fall würde ein erfolgreicher INSERT-Vorgang den Wert Eins (1) zurückgeben.
- Bei der Ausführung als serverseitige Klasse erfolgt die Ausgabe von `System.out.println` im Meldungsfenster des Datenbankservers.

Statische INSERT- und DELETE-Anweisungen über JDBC verwenden

Eine JDBC-Beispielanwendung wird vom Datenbankserver aus aufgerufen, um Zeilen in der Tabelle "Departments" mithilfe von statischen SQL-Anweisungen einzufügen oder zu löschen.

Voraussetzungen

Ein Java Development Kit (JDK) muss installiert sein.

Um eine externe Prozedur zu erstellen, benötigen Sie die Systemprivilegien CREATE PROCEDURE und CREATE EXTERNAL REFERENCE. Außerdem müssen Sie die Privilegien SELECT, DELETE und INSERT für das Datenbankobjekt haben, das Sie ändern möchten.

Aufgabe

1. Stellen Sie über Interactive SQL eine Verbindung mit der Datenbank her.
2. Die Klasse JDBCExample muss installiert sein.

Weitere Hinweise, wie Sie die Java-Beispiele installieren können, finden Sie unter [„Vorbereitung für die JDBC-Beispiele“ auf Seite 436](#).

3. Definieren Sie eine gespeicherte Prozedur namens JDBCExample, die als Wrapper für die Methode JDBCExample.main in der Klasse agiert:

```
CREATE PROCEDURE JDBCExample(IN arg CHAR(50))
  EXTERNAL NAME 'JDBCExample.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
```

4. Rufen Sie die Methode JDBCExample.main wie folgt auf:

```
CALL JDBCExample( 'insert' );
```

Die Argumentzeichenfolge 'insert' bewirkt, dass die Methode InsertStatic aufgerufen wird.

5. Vergewissern Sie sich, dass in die Tabelle Departments eine zusätzliche Zeile eingefügt wurde.

```
SELECT * FROM Departments;
```

Das Beispielprogramm zeigt den aktualisierten Inhalt der Tabelle "Departments" im Meldungsfenster des Datenbankservers an.

6. In der Beispielklasse DeleteStatic gibt es eine ähnliche Methode, die zeigt, wie die soeben hinzugefügte Zeile gelöscht wird. Rufen Sie die Methode JDBCExample.main wie folgt auf:

```
CALL JDBCExample( 'delete' );
```

Die Argumentzeichenfolge 'delete' bewirkt, dass die Methode DeleteStatic aufgerufen wird.

7. Überprüfen Sie, ob die Zeile aus der Tabelle Departments gelöscht wurde.

```
SELECT * FROM Departments;
```

Das Beispielprogramm zeigt den aktualisierten Inhalt der Tabelle "Departments" im Meldungsfenster des Datenbankservers an.

Ergebnisse

Zeilen können mithilfe von statischen SQL-Anweisungen in einer serverseitigen JDBC-Anwendung in einer Tabelle eingefügt und gelöscht werden.

Vorbereitete Anweisungen für effizienteren Zugriff verwenden

Wenn Sie eine Statement-Schnittstelle verwenden, muss jede Anweisung, die Sie an die Datenbank senden, syntaktisch analysiert werden. Außerdem ist ein Zugriffsplan zu generieren und die Anweisung auszuführen. Die Schritte vor der Ausführung werden als **Vorbereitung** der Anweisung bezeichnet.

Sie können Performance-Vorteile erzielen, wenn Sie die PreparedStatement-Schnittstelle verwenden. Damit haben Sie die Möglichkeit, eine Anweisung mit Platzhaltern vorzubereiten, und dann beim Ausführen der Anweisung diesen Platzhaltern Werte zuzuordnen.

Der Einsatz von vorbereiteten Anweisungen ist besonders dann nützlich, wenn viele ähnliche Aktionen ausgeführt werden, wie etwa das Einfügen von vielen Zeilen.

Beispiel

Das folgende Beispiel zeigt, wie die Schnittstelle PreparedStatement benutzt werden kann, obwohl das Einfügen einer einzelnen Zeile in der Praxis nicht mit vorbereiteten Anweisungen erfolgen sollte.

Die folgende InsertDynamic-Methode der Klasse JDBCExample führt eine vorbereitete Anweisung aus:

```
public static void InsertDynamic( Connection con,
                                String ID, String name )
{
    try
    {
        // Build the INSERT statement
        // ? is a placeholder character
        String sqlStr = "INSERT INTO Departments " +
            "( DepartmentID, DepartmentName ) " +
            "VALUES ( ? , ? )";

        // Prepare the statement
        PreparedStatement stmt =
            con.prepareStatement( sqlStr );

        // Set some values
        int idValue = Integer.valueOf( ID );
        stmt.setInt( 1, idValue );
        stmt.setString( 2, name );

        // Execute the statement
        int iRows = stmt.executeUpdate();

        // Print the number of rows inserted
        System.out.println(iRows + " rows inserted");
    }
    catch (SQLException sqe)
    {
        System.out.println("Unexpected exception : " +
            sqe.toString() + ", sqlstate = " +
            sqe.getSQLState());
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

Hinweise

- Dieses Codefragment ist Teil der Datei *JDBCExample.java*, die im Verzeichnis *%SQLANYWHERE%\SQLAnywhere\JDBC* enthalten ist.
- Die Methode `executeUpdate` gibt eine Ganzzahl zurück, die die Anzahl der Zeilen ausdrückt, die von dem Vorgang betroffen waren. In diesem Fall würde ein erfolgreicher INSERT-Vorgang den Wert Eins (1) zurückgeben.
- Bei der Ausführung als serverseitige Klasse erfolgt die Ausgabe von `System.out.println` im Meldungsfenster des Datenbankservers.

Vorbereitete INSERT- und DELETE-Anweisungen über JDBC verwenden

Eine JDBC-Beispielanwendung wird vom Datenbankserver aus aufgerufen, um Zeilen in der Tabelle "Departments" mithilfe von vorbereiteten SQL-Anweisungen einzufügen oder zu löschen.

Voraussetzungen

Ein Java Development Kit (JDK) muss installiert sein.

Um eine externe Prozedur zu erstellen, benötigen Sie die Systemprivilegien CREATE PROCEDURE und CREATE EXTERNAL REFERENCE. Außerdem müssen Sie die Privilegien SELECT, DELETE und INSERT für das Datenbankobjekt haben, das Sie ändern möchten.

Aufgabe

1. Stellen Sie über Interactive SQL eine Verbindung mit der Datenbank her.
2. Die Klasse *JDBCExample* muss installiert sein.

Weitere Hinweise, wie Sie die Java-Beispiele installieren können, finden Sie unter [„Vorbereitung für die JDBC-Beispiele“ auf Seite 436](#).

3. Definieren Sie eine gespeicherte Prozedur namens *JDBCInsert*, die als Wrapper für die Methode *JDBCExample.Insert* in der Klasse agiert:

```
CREATE PROCEDURE JDBCInsert(IN arg1 INTEGER, IN arg2 CHAR(50))
  EXTERNAL NAME 'JDBCExample.Insert(ILjava/lang/String;)'
  LANGUAGE JAVA;
```

4. Rufen Sie die Methode *JDBCExample.Insert* wie folgt auf:

```
CALL JDBCInsert( 202, 'Southeastern Sales' );
```

Die Insert-Methode veranlasst, dass die Methode *InsertDynamic* aufgerufen wird.

5. Vergewissern Sie sich, dass in die Tabelle *Departments* eine zusätzliche Zeile eingefügt wurde.

```
SELECT * FROM Departments;
```

Das Beispielprogramm zeigt den aktualisierten Inhalt der Tabelle "Departments" im Meldungsfenster des Datenbankservers an.

6. In der Beispielklasse DeleteDynamic gibt es eine ähnliche Methode, die zeigt, wie die soeben hinzugefügte Zeile gelöscht wird.

Definieren Sie eine gespeicherte Prozedur namens JDBCDelete, die als Wrapper für die Methode JDBCExample.Delete in der Klasse agiert:

```
CREATE PROCEDURE JDBCDelete(IN arg1 INTEGER)
EXTERNAL NAME 'JDBCExample.Delete(I)V'
LANGUAGE JAVA;
```

7. Rufen Sie die Methode JDBCExample.Delete wie folgt auf:

```
CALL JDBCDelete( 202 );
```

Die Delete-Methode veranlasst, dass die Methode DeleteDynamic aufgerufen wird.

8. Überprüfen Sie, ob die Zeile aus der Tabelle Departments gelöscht wurde.

```
SELECT * FROM Departments;
```

Das Beispielprogramm zeigt den aktualisierten Inhalt der Tabelle "Departments" im Meldungsfenster des Datenbankservers an.

Ergebnisse

Zeilen können mithilfe von vorbereiteten SQL-Anweisungen in einer serverseitigen JDBC-Anwendung in einer Tabelle eingefügt und gelöscht werden.

Siehe auch

- „Vorbereitete Anweisungen“ auf Seite 2

JDBC-Batch-Methoden

Die addBatch-Methode der Klasse PreparedStatement wird für die Ausführung von Batch-Einfügungen (oder breiten Einfügungen) verwendet. Es folgen einige Richtlinien zur Verwendung dieser Methode.

1. Eine INSERT-Anweisung sollte unter Verwendung einer der preparedStatement-Methoden für die Klasse Connection vorbereitet werden.

```
// Build the INSERT statement
String sqlStr = "INSERT INTO Departments " +
                "( DepartmentID, DepartmentName ) " +
                "VALUES ( ? , ? )";
// Prepare the statement
PreparedStatement stmt =
    con.prepareStatement( sqlStr );
```

2. Die Parameter für die vorbereitete INSERT-Anweisung sollten folgendermaßen festgelegt und in Stapeln gesetzt werden.

```
// loop to batch "n" sets of parameters
for( i=0; i < n; i++ )
{
    // "stmt" is the original prepared insert statement from step 1.
    stmt.setSomeType( 1, param_1 );
    stmt.setSomeType( 2, param_2 );
    .
    .
    .
    // There are "m" parameters in the statement.
    stmt.setSomeType( m , param_m );

    // Add the set of parameters to the batch and
    // move to the next row of parameters.
    stmt.addBatch();
}

```

Beispiel:

```
for( i=0; i < 5; i++ )
{
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );
    stmt.addBatch();
}

```

3. Der Batch muss unter Verwendung der executeBatch-Methode der Klasse PreparedStatement ausgeführt werden.

BLOB-Parameter werden in Batches nicht unterstützt.

Wenn Sie Batch-Einfügungen mit dem SQL Anywhere-JDBC-Treiber durchführen, wird empfohlen, eine kleine Spaltengröße zu verwenden. Die Verwendung von Batch-Einfügungen, zum Einfügen von großen Binär- oder Zeichendaten in long binary- oder long varchar-Spalten wird nicht empfohlen und kann zu einer Verminderung der Performance führen. Der Grund dafür ist, dass der SQL Anywhere-JDBC-Treiber viel Speicherplatz zuweisen muss, um die einzelnen Batch-Einfügezeilen aufzunehmen. In allen anderen Fällen sollte die Verwendung von Batch-Einfügungen eine bessere Performance liefern als die Verwendung von einzelnen Einfügungen.

Ergebnismengen aus Java zurückgeben

Dieser Abschnitt beschreibt, wie eine oder mehrere Ergebnismengen aus Java-Methoden verfügbar gemacht werden.

Sie müssen eine Java-Methode erstellen, die mindestens eine Ergebnismenge an die aufrufende Umgebung zurückgibt, und diese Methode in eine gespeicherte SQL-Prozedur einfügen. Das folgende Codefragment veranschaulicht, wie mehrere Ergebnismengen an den Aufrufer dieser Java-Prozedur zurückgegeben werden können. Es verwendet drei executeQuery-Anweisungen, um drei verschiedene Ergebnismengen zu erhalten.

```
public static void Results( ResultSet[] rset )
    throws SQLException
{
    // Demonstrate returning multiple result sets
}

```

```
Connection con = DriverManager.getConnection(
    "jdbc:default:connection" );
rset[0] = con.createStatement().executeQuery(
    "SELECT * FROM Employees" +
    " ORDER BY EmployeeID" );
rset[1] = con.createStatement().executeQuery(
    "SELECT * FROM Departments" +
    " ORDER BY DepartmentID" );
rset[2] = con.createStatement().executeQuery(
    "SELECT i.ID,i.LineID,i.ProductID,i.Quantity," +
    "        s.OrderDate,i.ShipDate," +
    "        s.Region,e.GivenName||' '||e.Surname" +
    " FROM SalesOrderItems AS i" +
    " JOIN SalesOrders AS s" +
    " JOIN Employees AS e" +
    " WHERE s.ID=i.ID" +
    "        AND s.SalesRepresentative=e.EmployeeID" );
con.close();
}
```

Hinweise

- Dieses serverseitige JDBC-Beispiel ist Teil der Datei *JDBCExample.java*, die im Verzeichnis %SQLANYSP16%\SQLAnywhere\JDBC enthalten ist.
- Damit wird eine Verbindung mit der laufenden Standarddatenbank über getConnection eingerichtet.
- Die executeQuery-Methoden geben Ergebnismengen zurück.

Ergebnismengen aus JDBC zurückgeben

Eine JDBC-Beispielanwendung wird vom Datenbankserver aus aufgerufen, um mehrere Ergebnismengen zurückzugeben.

Voraussetzungen

Ein Java Development Kit (JDK) muss installiert sein.

Um eine externe Prozedur zu erstellen, benötigen Sie die Systemprivilegien CREATE PROCEDURE und CREATE EXTERNAL REFERENCE. Außerdem müssen Sie die Privilegien SELECT, DELETE und INSERT für das Datenbankobjekt haben, das Sie ändern möchten.

Aufgabe

1. Stellen Sie über Interactive SQL eine Verbindung mit der Datenbank her.
2. Die Klasse JDBCExample muss installiert sein.

Weitere Hinweise, wie Sie die Java-Beispiele installieren können, finden Sie unter „[Vorbereitung für die JDBC-Beispiele](#)“ auf Seite 436.

3. Definieren Sie eine gespeicherte Prozedur namens JDBCResults, die als Wrapper für die JDBCExample.Results-Methode in der Klasse fungiert.

Beispiel:

```
CREATE PROCEDURE JDBCResults(OUT args LONG VARCHAR)
  DYNAMIC RESULT SETS 3
  EXTERNAL NAME 'JDBCExample.Results([Ljava/sql/ResultSet;)V'
  LANGUAGE JAVA;
```

In diesem Beispiel werden 3 Ergebnismengen zurückgegeben.

4. Setzen Sie die folgenden Interactive SQL-Optionen, sodass Sie alle Ergebnisse der Abfrage sehen können:
 - a. Klicken Sie auf **Extras » Optionen**.
 - b. Klicken Sie auf **SQL Anywhere**.
 - c. Klicken Sie auf die Registerkarte **Ergebnisse**.
 - d. Setzen Sie den Wert für **Maximale Anzahl von anzuzeigenden Zeilen** auf **5000**.
 - e. Klicken Sie auf **Alle Ergebnismengen anzeigen**.
 - f. Klicken Sie auf **OK**.
5. Rufen Sie die JDBCExample.Results-Methode auf.

```
CALL JDBCResults();
```

6. Überprüfen Sie die drei Ergebnisregisterkarten **Ergebnismenge 1**, **Ergebnismenge 2** und **Ergebnismenge 3**.

Ergebnisse

Drei verschiedene Ergebnismengen werden von einer serverseitigen JDBC-Anwendung zurückgegeben.

JDBC-Hinweise

- **Zugriffsprivilegien** Wie bei allen Java-Klassen in der Datenbank können alle Benutzer auf Klassen mit JDBC-Anweisungen zugreifen, vorausgesetzt, Ihnen wurde mit der GRANT EXECUTE-Anweisung das Privileg zum Ausführen der gespeicherten Prozedur erteilt, die als Wrapper für die Java-Methode fungiert.
- **Ausführungsprivilegien** Java-Klassen werden mit den Privilegien der gespeicherten Prozedur ausgeführt, die als Wrapper für die Java-Methode fungiert. (Standardmäßig ist dies SQL SECURITY DEFINER.)

JDBC-Callbacks

Der SQL Anywhere-JDBC-Treiber unterstützt zwei asynchrone Callbacks, einen für die Verarbeitung der SQL MESSAGE-Anweisung und den anderen für die Validierung von Dateiübertragungsanforderungen. Diese ähneln den Callbacks, die vom SQL Anywhere-ODBC-Treiber unterstützt werden.

Nachrichten können unter Verwendung der SQL MESSAGE-Anweisung vom Datenbankserver an die Clientanwendung gesendet werden. Außerdem können Nachrichten aufgrund von lang laufenden Datenbankserver-Anweisungen generiert werden.

Es kann eine Message-Handler-Routine erstellt werden, um diese Nachrichten abzufangen. Nachstehend finden Sie ein Beispiel für die Message-Handler-Callback-Routine.

```

class T_message_handler implements sybase.jdbc4.sqlanywhere.ASAMessageHandler
{
    private final int MSG_INFO      = 0x80 | 0;
    private final int MSG_WARNING   = 0x80 | 1;
    private final int MSG_ACTION    = 0x80 | 2;
    private final int MSG_STATUS    = 0x80 | 3;
    T_message_handler()
    {
    }

    public SQLException messageHandler(SQLException sqe)
    {
        String msg_type = "unknown";

        switch( sqe.getErrorCode() ) {
            case MSG_INFO:      msg_type = "INFO      "; break;
            case MSG_WARNING:   msg_type = "WARNING   "; break;
            case MSG_ACTION:    msg_type = "ACTION    "; break;
            case MSG_STATUS:    msg_type = "STATUS    "; break;
        }

        System.out.println( msg_type + ": " + sqe.getMessage() );
        return sqe;
    }
}

```

Eine Client-Dateiübertragungsanforderung kann validiert werden. Bevor der JDBC-Treiber eine Übertragung zulässt, ruft er die Validierungs-Callback-Funktion auf, falls vorhanden. Falls die Clientdatenübertragung während der Ausführung von indirekten Anweisungen angefordert wird, lässt der JDBC-Treiber die Übertragung nur zu, wenn die Clientanwendung einen Validierungs-Callback registriert hat. Die Bedingungen, unter denen ein Validierungsaufruf durchgeführt wird, werden unten ausführlicher beschrieben. Das Folgende ist ein Beispiel für eine Callback-Routine zur Dateiübertragungsvalidierung .

```

class T_filetrans_callback implements
sybase.jdbc4.sqlanywhere.SAValidateFileTransferCallback
{
    T_filetrans_callback()
    {
    }

    public int callback(String filename, int is_write)
    {
        System.out.println( "File transfer granted for file " + filename
+
                                " with an is_write value of " +
is_write );
        return( 1 ); // 0 to disallow, non-zero to allow
    }
}

```

Das Argument filename ist der Name der zu lesenden bzw. zu schreibenden Datei. Der Parameter is_write ist 0, wenn ein Lesevorgang (Übertragung vom Client zum Datenbankserver) angefordert wird, und Nicht-Null bei einem Schreibvorgang. Die Callback-Funktion sollte 0 zurückgeben, wenn die Übertragung nicht zulässig ist, ansonsten Nicht-Null.

Aus Gründen der Datensicherheit protokolliert der Datenbankserver den Ursprung von Anweisungen, die eine Dateiübertragung anfordern. Der Datenbankserver ermittelt, ob die Anweisung direkt von der

Clientanwendung empfangen wurde. Wenn er die Datenübertragung vom Client initiiert, sendet der Datenbankserver die Informationen über den Ursprung der Anweisung an die Clientsoftware. Seinerseits erlaubt der JDBC-Treiber nur dann eine bedingungslose Datenübertragung, wenn diese aufgrund der Ausführung einer Anweisung angefordert wird, die direkt von der Clientanwendung gesendet wurde. Ansonsten muss die Anwendung den oben beschriebenen Validierungs-Callback registriert haben. Wenn dieser fehlt, wird die Übertragung verweigert und die Anweisung schlägt mit einem Fehler fehl. Wenn die Clientanweisung eine bereits in der Datenbank vorhandene gespeicherte Prozedur aufruft, wird die Ausführung der gespeicherten Prozedur selbst nicht als vom Client initiierte Anweisung angesehen. Wenn die Clientanwendung allerdings explizit eine temporäre gespeicherte Prozedur erstellt, führt die Ausführung der gespeicherten Prozedur dazu, dass der Datenbankserver die Prozedur als eine vom Client initiierte behandelt. Wenn die Clientanwendung eine Batchanweisung ausführt, wird auf gleiche Weise die Ausführung der Batchanweisung als eine angesehen, die direkt von der Clientanwendung durchgeführt wird.

Das folgende Beispiel einer Java-Anwendung veranschaulicht die Verwendung der vom SQL Anywhere JDBC 4.0-Treiber unterstützten Callbacks. Sie müssen die Datei `%SQLANY16%\java\sajdbc4.jar` in den Classpath einfügen.

```
import java.io.*;
import java.sql.*;
import java.util.*;

public class callback
{
    public static void main (String args[]) throws IOException
    {
        Connection          con = null;
        Statement            stmt;

        System.out.println ( "Starting... " );
        con = connect();
        if( con == null )
        {
            return; // exception should already have been reported
        }
        System.out.println ( "Connected... " );
        try
        {
            // create and register message handler callback
            T_message_handler message_worker = new T_message_handler();

            ((sybase.jdbc4.sqlanywhere.IConnection)con).setASAMessageHandler( message_worker );

            // create and register validate file transfer callback
            T_filetrans_callback filetran_worker = new
            T_filetrans_callback();

            ((sybase.jdbc4.sqlanywhere.IConnection)con).setSAValidateFileTransferCallback( filetran_worker );

            stmt = con.createStatement();

            // execute message statements to force message handler to be
            called
            stmt.execute( "MESSAGE 'this is an info    message' TYPE INFO TO
            CLIENT" );
            stmt.execute( "MESSAGE 'this is an action message' TYPE ACTION
```

```
TO CLIENT" );
    stmt.execute( "MESSAGE 'this is a warning message' TYPE WARNING
TO CLIENT" );
    stmt.execute( "MESSAGE 'this is a status message' TYPE STATUS
TO CLIENT" );

    System.out.println( "\n=====\\n" );

    stmt.execute( "set temporary option
allow_read_client_file='on'" );
    try
    {
        stmt.execute( "drop procedure read_client_file_test" );
    }
    catch( SQLException dummy )
    {
        // ignore exception if procedure does not exist
    }
    // create procedure that will force file transfer callback to be
called
    stmt.execute( "create procedure read_client_file_test()" +
        "begin" +
        "    declare v long binary;" +
        "    set v = read_client_file('sample.txt');" +
        "end" );

    // call procedure to force validate file transfer callback to be
called
    try
    {
        stmt.execute( "call read_client_file_test()" );
    }
    catch( SQLException filetrans_exception )
    {
        // Note: Since the file transfer callback returns 1,
        // do not expect a SQL exception to be thrown
        System.out.println( "SQLException: " +
            filetrans_exception.getMessage() );
    }
    stmt.close();
    con.close();
    System.out.println( "Disconnected" );
}
catch( SQLException sqe )
{
    printExceptions(sqe);
}
}

private static Connection connect()
{
    Connection connection;

    System.out.println( "Using jdbc4 driver" );
    try
    {
        connection = DriverManager.getConnection(
            "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
    }
    catch( Exception e )
    {
        System.err.println( "Error! Could not connect" );
        System.err.println( e.getMessage() );
        printExceptions( (SQLException)e );
    }
}
```

```

        connection = null;
    }
    return connection;
}

static private void printExceptions(SQLException sqe)
{
    while (sqe != null)
    {
        System.out.println("Unexpected exception : " +
            "SqlState: " + sqe.getSQLState() +
            " " + sqe.toString() +
            ", ErrorCode: " + sqe.getErrorCode());
        System.out.println( "=====\n" );
        sqe = sqe.getNextException();
    }
}
}

```

Siehe auch

- „MESSAGE-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

JDBC-Escape-Syntax

Sie können JDBC-Escape-Syntax aus jeder JDBC-Anwendung verwenden, auch aus Interactive SQL. Diese Escape-Syntax ermöglicht es Ihnen, gespeicherte Prozeduren unabhängig vom verwendeten Datenbank-Managementsystem aufzurufen. Das allgemeine Syntaxformat lautet wie folgt:

{ **keyword** *Parameter* }

Die Gruppe der Schlüsselwörter umfasst:

- **{d date-string}** Die Zeichenfolge date ist ein Datumswert, der von SQL Anywhere akzeptiert wird.
- **{t time-string}** Die Zeichenfolge time ist ein Zeitwert, der von SQL Anywhere akzeptiert wird.
- **{ts date-string time-string}** Die Zeichenfolge date/time ist ein Zeitstempelwert, der von SQL Anywhere akzeptiert wird.
- **{guid uuid-string}** Die Zeichenfolge uuid ist eine beliebige gültige GUID-Zeichenfolge, z.B. 41dfe9ef-db91-11d2-8c43-006008d26a6f.
- **{oj outer-join-expr}** Die outer-join-expr ist ein gültiger OUTER JOIN-Ausdruck, der von SQL Anywhere akzeptiert wird.
- **{? = call func(p1,...)}** Die Funktion ist ein beliebiger gültiger Funktionsaufruf, der von SQL Anywhere akzeptiert wird.
- **{call proc(p1,...)}** Die Prozedur ist ein beliebiger gültiger Aufruf einer gespeicherten Prozedur, der von SQL Anywhere akzeptiert wird.
- **{fn func(p1,...)}** Die Funktion ist eine der Funktionen der Bibliothek, die unten aufgelistet werden.

Sie können die Escapesyntax benutzen, um auf eine Bibliothek von Funktionen zuzugreifen, die im JDBC-Treiber implementiert sind und Zahlen-, Zeichenketten-, Zeit-, Datums- und Systemfunktionen umfassen.

Wenn Sie beispielsweise das aktuelle Datum in einer vom jeweiligen Datenbank-Managementsystem unabhängigen Weise abrufen möchten, führen Sie folgenden Befehl aus:

```
SELECT { FN CURDATE( ) }
```

Die verfügbaren Funktionen richten sich nach dem von Ihnen benutzten JDBC-Treiber. Die folgenden Tabellen zeigen die Funktionen, die vom SQL Anywhere-JDBC-Treiber und vom jConnect-Treiber unterstützt werden.

Vom SQL Anywhere-JDBC-Treiber unterstützte Funktionen

Nummerische Funktionen	Zeichenfolgenfunktionen	Systemfunktionen	Datum/Zeit-Funktionen
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH		CURRENT_TIME-STAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOURL
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW

Nummerische Funktionen	Zeichenfolgenfunktionen	Systemfunktionen	Datum/Zeit-Funktionen
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

Von jConnect unterstützte Funktionen

Nummerische Funktionen	Zeichenfolgenfunktionen	Systemfunktionen	Datum/Zeit-Funktionen
ABS	ASCII	DATABASE	CURDATE
ACOS	CHAR	IFNULL	CURTIME
ASIN	CONCAT	USER	DAYNAME
ATAN	DIFFERENCE	CONVERT	DAYOFMONTH
ATAN2	LCASE		DAYOFWEEK
CEILING	LENGTH		HOURL
COS	REPEAT		MINUTE
COT	RIGHT		MONTH
DEGREES	SOUNDEX		MONTHNAME
EXP	SPACE		NOW
FLOOR	SUBSTRING		QUARTER
LOG	UCASE		SECOND

Nummerische Funktionen	Zeichenfolgenfunktionen	Systemfunktionen	Datum/Zeit-Funktionen
LOG10			TIMESTAMPADD
PI			TIMESTAMPDIFF
POWER			YEAR
RADIANS			
RAND			
ROUND			
SIGN			
SIN			
SQRT			
TAN			

Eine Anweisung, die die Escape-Syntax verwendet, funktioniert in SQL Anywhere, Adaptive Server Enterprise, Oracle, SQL Server oder anderen Datenbank-Managementsystemen, mit denen Sie verbunden sind.

In Interactive SQL *müssen* die Klammern doppelt sein. Zwischen aufeinanderfolgenden geschweiften Klammern darf kein Leerzeichen stehen. "{{" ist korrekt, "{ {" nicht. Außerdem dürfen Sie kein Zeilenschaltungszeichen in der Anweisung verwenden. Die Escape-Syntax kann in gespeicherten Prozeduren nicht verwendet werden, weil diese nicht von Interactive SQL syntaktisch analysiert werden.

Um z.B. Datenbankeigenschaften mit der Prozedur sa_db_info unter Verwendung der SQL-Escape-Syntax zu erhalten, müssen Sie folgenden Aufruf in Interactive SQL ausführen:

```
{{CALL sa_db_info( 0 ) }}
```

JDBC 4.0-API-Unterstützung

Alle obligatorischen Klassen und Methoden der JDBC 4.0-Spezifikation werden vom SQL Anywhere JDBC-Treiber unterstützt. Einige optionale Methoden der java.sql.Blob-Schnittstelle werden nicht unterstützt. Diese optionalen Methoden sind folgende:

- long position(Blob pattern, long start);
- long position(byte[] pattern, long start);
- OutputStream setBinaryStream(long pos)
- int setBytes(long pos, byte[] bytes)
- int setBytes(long pos, byte[] bytes, int offset, int len);
- void truncate(long len);

OData-Unterstützung

OData (Open Data Protocol) ermöglicht Datendienste über RESTful HTTP. Sie können damit Vorgänge durch URIs (Universal Resource Identifier) für den Zugriff auf und das Ändern von Daten durchführen.

In diesem Abschnitt wird die OData-Unterstützung für SQL Anywhere – der OData Server – beschrieben. Dabei wird davon ausgegangen, dass Sie bereits mit Konzepten des OData-Protokolls vertraut sind.

Siehe auch

- <http://www.odata.org/documentation/IndexV2>

OData Server-Architektur

Der OData Server besteht aus den folgenden Komponenten:

- **OData Producer** Der **OData Producer** ist ein Java-Servlet, das die JDBC-API verwendet, um eine Verbindung mit einem SQL Anywhere-Datenbankserver herzustellen. Der OData Producer verarbeitet OData-Anforderungen und -Antworten und bildet eine Schnittstelle zur Datenbank.

Die folgende Tabelle veranschaulicht, wie der OData Producer OData-Konzepte relationalen Datenbankkonzepten zuordnet:

OData-Konzept	Datenbankäquivalent
Entitätstyp	Tabelle oder Ansicht
Entitätstypinstanz	Zeile
Schlüssel	Primärschlüssel
Verknüpfung	Fremdschlüssel
Eigenschaft	Spalte

- **HTTP-Server** Ein HTTP-Server verarbeitet OData-Anforderungen von Webclients.

Der OData Server verwendet den Jetty WebServer als HTTP-Server. Dieser **eingebettete HTTP-Server** fungiert auch als Java-Servlet-Behälter, der erforderlich ist, um den OData Producer zu hosten.

Sie können zum Verarbeiten von OData-Anforderungen als Alternative zum eingebetteten HTTP-Server auch Ihren eigenen HTTP-Server verwenden, sofern Ihre Lösung auch Java-Servlets hosten kann. Sie können beispielsweise einen IIS- oder Apache-Server zum Weiterleiten von Anforderungen an einen Tomcat-Server einrichten.

OData-Clientanforderungen werden über URIs an einen HTTP-Server gesendet und vom OData Producer verarbeitet, der dann eine Schnittstelle zum Datenbankserver bildet, um Datenbankanforderungen abzusetzen und Inhalte für die OData-Antworten abzurufen.

Das OData-Schema für jeden Client basiert auf den Berechtigungen des Clients für Datenbankverbindungen. Clients können keine Datenbankobjekte anzeigen oder ändern, für die sie nicht die entsprechende Berechtigung haben.

Sie können Clients Zugriff auf die Datenbank erteilen, indem Sie entweder eine vorkonfigurierte Verbindungszeichenfolge oder die HTTP-Basic-Authentifizierung verwenden.

Siehe auch

- „OData Server ausführen und Clients einrichten“ auf Seite 464
- <http://www.odata.org/documentation/IndexV2>

Einschränkungen für den OData Server

Der OData Server ist mit den Spezifikationen der OData-Protokollversion 2 kompatibel, wobei folgende Einschränkungen gelten, die in den OData-Protokollspezifikationen nicht explizit definiert sind:

- **Schemaänderungen** Sie müssen das OData Server-Dienstprogramm neu starten, wenn Sie Änderungen am Datenbankschema vornehmen, damit die Änderungen in Kraft treten können und für OData-Clients sichtbar werden.
- **\$orderby-Abfragen** Sie dürfen nur nach Entitätseigenschaften sortieren. Die Sortierung nach Richtung wird unterstützt, aber nicht die Sortierung nach Ausdrücken.

Nicht unterstützte OData-Protokollfunktionen

Die folgenden OData-Protokollfunktionen werden vom OData Producer nicht unterstützt:

- Tiefe Einfügungen und Aktualisierungen
- Dienstvorgänge
- Dynamische Eigenschaften
- Komplexe Typen
- Datenträgertypen
- HTTP-Etags

Siehe auch

- „Sicherheitshinweise zum OData Server“ auf Seite 456
- <http://www.odata.org/documentation/IndexV2>

Sicherheitshinweise zum OData Server

Vor dem Einrichten eines OData Servers sind Sicherheitsmaßnahmen zu bedenken.

HTTPS-Zertifizierung

In der OData Server-Konfigurationsdatei angegebene HTTPS-Zertifikatdetails gelten nur für den eingebetteten HTTP-Server. Weitere Hinweise zur Behandlung der HTTPS-Zertifizierung durch einen alternativen HTTP-Server finden Sie in der HTTP-Serverdokumentation.

Datenverkehr ohne Verwendung des SSL-Protokolls

Es wird empfohlen, in Produktionsdeployments immer SSL zu verwenden.

Der gesamte Datenverkehr zwischen dem OData Producer und den Clients wird als reiner Text übertragen, einschließlich Benutzer-IDs und Kennwörter, wenn in der OData Server-Konfigurationsdatei nicht die SSLKeyStore-Option angegeben ist. Diese Option ist in den Standardkonfigurationen nicht angegeben.

SQL Anywhere-Datenbanken bis Version 12

Wenn Sie den OData Producer verwenden, um eine SQL Anywhere-Datenbank bis Version 12 abzufragen, werden alle Datenbankobjekte bereitgestellt. Sie werden nicht auf der Basis der Anmeldeinformationen von Benutzern gefiltert.

Der Datenbankserver gibt einen Fehler zurück, wenn Sie versuchen, auf ein Objekt zuzugreifen, ohne das erforderliche Datenbankprivileg zu haben.

Siehe auch

- „OData Server konfigurieren“ auf Seite 457
- „OData Server-Dienstprogramm (dbosrv16.exe)“ auf Seite 464
- <http://www.odata.org/documentation/IndexV2>
- <http://wiki.eclipse.org/Jetty>
- <http://docs.codehaus.org/display/JETTY/Jetty+Wiki>

OData Server konfigurieren

Datenbankverbindungsparameter, Optionen für den eingebetteten HTTP-Server und Optionen für den OData Producer müssen in einer Konfigurationsdatei konfiguriert werden. Geben Sie diese Datei beim Starten des OData Servers in der Befehlszeile an.

Optionen für den eingebetteten HTTP-Server

Die folgenden Optionen für den eingebetteten HTTP-Server können in der Konfigurationsdatei angegeben werden:

Option	Beschreibung
LogFile = <i>Pfad-und-Dateiname</i>	<p>Gibt den Pfad und den Namen der Datei an, in der der eingebettete HTTP-Server die OData Producer-Ausgabe protokolliert.</p> <p>Das Standardverhalten ist, die Protokollierung zu deaktivieren.</p> <p>Der Pfad ist relativ zum Speicherort der Programmdatei des Servers.</p>
LogVerbosity = { 1 2 3 4 }	<p>Legt die Ausführlichkeitsstufe der Protokollierung fest. Höhere Ausführlichkeitsstufen protokollieren zusätzliche Informationen und enthalten die Informationen aller niedrigeren Stufen.</p> <p>Ausführlichkeitsstufe 1 gibt Informationen zu unerwarteten Fehlern aus.</p> <p>Ausführlichkeitsstufe 2 gibt allgemeine Informationen und Konfigurationsmeldungen aus.</p> <p>Ausführlichkeitsstufe 3 gibt detaillierte Informationen zu HTTP-Anforderungen aus.</p> <p>Ausführlichkeitsstufe 4 gibt Debugging-Meldungen aus.</p>
ServerPort = <i>Portnummer</i>	<p>Gibt die Portnummer an, die der eingebettete HTTP-Server abhören soll.</p> <p>Die Standardeinstellung ist 80.</p>
ShutdownListenerPort = <i>Portnummer</i>	<p>Legt die Portnummer fest, an der der eingebettete HTTP-Server auf Anforderungen zum Herunterfahren warten soll.</p> <p>Die Standardeinstellung ist 2449.</p>
SSLKeyStore = <i>Pfad-und-Dateiname</i>	<p>Gibt den Pfad und den Dateinamen für einen Java Key Store an, der ein SSL-Zertifikat enthält, das der eingebettete HTTP-Server zum Verschlüssen des Datenverkehrs verwendet.</p> <p>SSL wird aktiviert und unverschlüsselter HTTP-Datenverkehr wird blockiert, wenn diese Option angegeben wurde.</p> <p>Der Pfad ist relativ zum Speicherort der Programmdatei des Servers.</p>
SSLKeyStorePassword = <i>SSLKeyStore-Kennwort</i>	<p>Gibt das Kennwort an, das der eingebettete HTTP Server verwendet, um die Authentifizierung mit dem durch die SSLKeyStore-Option festgelegten Java Key Store durchzuführen.</p>

OData Producer-Optionen und Einstellungen für Datenbankverbindungsparameter

Die folgenden OData Producer-Optionen und Verbindungsparametereinstellungen können in der Konfigurationsdatei festgelegt werden:

Option	Beschreibung
Authentication = { none database }	<p>Gibt die Anmeldeinformationen an, die für Verbindungen mit der Datenbank verwendet werden.</p> <p>Die Standardeinstellung database gibt an, dass jeder Benutzer eine Verbindung mit personalisierten Anmeldeinformationen herstellt, die an die DbConnectionString-Option angehängt werden und so eine eigene vollständige Verbindungszeichenfolge für die Datenbank bilden. Diese Anmeldeinformationen werden mit der HTTP-Basic-Authentifizierung angefordert.</p> <p>Die Einstellung none zeigt an, dass alle Benutzer die Verbindung mit derselben, durch die DbConnectionString-Option festgelegten Verbindungszeichenfolge herstellen.</p>
ConnectionPoolMaximum = <i>max_Anz_Verbindungen</i>	<p>Gibt die maximale Anzahl der gleichzeitigen Verbindungen an, die der OData Producer für den Verbindungspool offen hält.</p> <p>Je nach Serverlast nutzt der Verbindungspool möglicherweise weniger Verbindungen.</p> <p>Standardmäßig ist die Verbindungspoolgröße durch die maximale Anzahl von gleichzeitigen Verbindungen begrenzt, die vom Datenbankserver zugelassen werden.</p>
DbConnectionString = <i>Verbindungszeichenfolge</i>	<p>Gibt die Verbindungszeichenfolge an, die für Verbindungen mit der Datenbank verwendet wird.</p> <p>Die Verbindungszeichenfolge sollte die Parameter UID und PWD ausschließen, wenn die Authentication-Option auf die Datenbank gesetzt ist.</p>
DbProduct = sqlanywhere	<p>Zeigt den Typ des Datenbankservers an, mit dem sich der OData Producer verbindet.</p>
PageSize = <i>max_Anz_Entitäten</i>	<p>Gibt die maximale Anzahl der Entitäten an, die in eine Antwort auf eine Entitätsgruppenabfrage eingefügt werden, bevor der nächste Link ausgegeben wird.</p> <p>Die Standardeinstellung ist 100.</p>

Option	Beschreibung
Model = <i>Pfad-und-Dateiname</i>	<p>Gibt den Pfad und den Namen einer OSDL-Datei (OData Service Definition Language) an, die das OData Producer-Dienstmodell enthält, in dem festgelegt wird, welche Tabellen und Ansichten in den OData-Metadaten bereitgestellt werden.</p> <p>Das Standardverhalten besteht darin, Tabellen und Ansichten auf Basis von Benutzerprivilegien bereitzustellen. Tabellen und Ansichten ohne Primärschlüssel werden nicht bereitgestellt.</p> <p>Der Pfad ist relativ zum Speicherort der Programmdatei des Servers.</p>
ModelConnectionString = <i>Verbindungszeichenfolge</i>	<p>Gibt eine Verbindungszeichenfolge an, die der OData Producer verwendet, um die OSDL-Datei während des Starts zu validieren.</p> <p>Die OSDL-Validierung stellt sicher, dass die aufgelisteten Tabellen und Spalten vorhanden sind, dass die Schlüssellisten entsprechend verwendet werden und dass die Datei semantisch korrekt ist.</p> <p>Die Verbindungszeichenfolge sollte die Parameter UID und PWD enthalten.</p> <p>Als Standardverhalten wird angenommen, dass die OSDL-Datei gültig ist.</p>
ReadOnly = { true false }	<p>Gibt an, ob Änderungsanforderungen ignoriert werden sollen.</p> <p>Die Standardeinstellung ist FALSE.</p>
ServiceRoot = <i>/ Pfadpräfix</i>	<p>Gibt das Stammverzeichnis des OData-Diensts auf dem OData Server an.</p> <p>Die Standardeinstellung ist /odata.</p> <p>Der Zugriff auf alle Ressourcen für diesen OData Producer erfolgt über URIs im folgenden Format:</p> <p><i>scheme:host:port/path-prefix/resource-path[query-options]</i></p> <p>Beispiel: <code>http://services.odata.org/OData/OData.svc/Category(1)/Products?\$top=2&\$orderby=name</code></p>

Siehe auch

- „OData Server-Dienstprogramm (dbosrv16.exe)“ auf Seite 464
- „OData Producer-Dienstmodelle erstellen“ auf Seite 463

Beispiel

Das folgende Beispiel veranschaulicht, wie eine Konfigurationsdatei für die Verwendung mit dem OData Server formatiert wird:

```
# Embedded HTTP server options
# -----
LogFile = ../../odata.log
LogVerbosity = 1
ServerPort = 8000
ShutdownListenerPort = 8083
SSLKeyStore = ../../samplekeystore.jks
SSLKeyStorePassword = pwd

# OData Producer options
# -----
Authentication = none
ConnectionPoolMaximum = 10
DbProduct = sqlanywhere
Model = ../../model.osdl
ModelConnectionString = uid=dba;pwd=sql;eng=orderentry;dbf=orderentry.db
PageSize = 100
ReadOnly = false
ServiceRoot = /odata/MyProducer

# Database connection parameters
# -----
DbConnectionString = uid=dba;pwd=sql;eng=orderentry;dbf=orderentry.db
```

Speichern Sie die Datei als *server.properties* und führen Sie anschließend den folgenden Befehl aus, um den OData Server mit den in dieser Konfigurationsdatei festgelegten Optionen zu starten:

```
dbosrv16 server.properties
```

HTTP-Server für OData-Vorgänge einrichten

Sie können entweder den eingebetteten HTTP-Server oder Ihren eigenen HTTP-Server einrichten.

Hinweis

Beim Einrichten eines HTTP-Servers sind Sicherheitsmaßnahmen zu bedenken. Weitere Hinweise finden Sie unter „[Sicherheitshinweise zum OData Server](#)“ auf Seite 456.

Eingebetteten HTTP-Server einrichten

Das OData Server-Dienstprogramm initiiert eine Instanz des eingebetteten HTTP-Servers und lädt automatisch den OData Producer als Java-Servlet.

Der eingebettete HTTP-Server kann nicht manuell konfiguriert oder verwendet werden, um anderen Nicht-OData-Inhalt bereitzustellen, beispielsweise HTML-Dateien. Einige HTTP-Serveroptionen können jedoch in der OData Server-Konfigurationsdatei angegeben werden.

Die folgenden Schritte sind erforderlich, um den eingebetteten HTTP-Server einzurichten und zu starten:

1. Erstellen Sie eine OData Server-Konfigurationsdatei, die die Einstellungen sowohl für den OData Producer als auch für den eingebetteten HTTP-Server enthält.

- Speichern Sie die Konfigurationsdatei auf dem Computer, der als HTTP-Server fungieren soll, und laden Sie anschließend die Konfigurationsdatei, wenn Sie das OData Server-Dienstprogramm an einer Eingabeaufforderung ausführen.

Hinweis

Der OData-Endpunkt befindet sich im **odata**-Kontextpfad, wenn Sie den eingebetteten HTTP-Server verwenden.

Um beispielsweise eine OData-Anforderung an einen lokal gehosteten HTTP-Server auszuführen, der an Port 8080 läuft, müssen Sie die Adresse <http://localhost:8080/odata> verwenden.

Alternativen HTTP-Server einrichten

Um den OData Producer mit einem alternativen HTTP-Server verwenden zu können, müssen Sie ein OData Producer-Deployment auf dem Server durchführen. Der OData Producer muss als Java-Servlet ausgeführt werden, das in einen HTTP-Server geladen werden kann. Sie können beispielsweise Tomcat als Java-Servlet-Behälter verwenden und mit einem Apache-HTTP-Server koppeln.

Hinweis

IIS kann Java-Servlets nicht ausführen. Es ist jedoch möglich, einen Konnektor zu konfigurieren, der Servlet-Anforderungen von einem IIS-Server auf einen anderen Server umleitet, der in der Lage ist, sie auszuführen.

Der Prozess des OData Producer-Deployments unterscheidet sich je nach HTTP-Server. Weitere Hinweise finden Sie in der Dokumentation Ihres HTTP-Servers.

Die folgenden Schritte sind für ein erfolgreiches OData Producer-Deployment erforderlich:

- Erstellen Sie eine OData Server-Konfigurationsdatei.

Konfigurationsoptionen, die für den eingebetteten HTTP-Server spezifisch sind, einschließlich Sicherheitsmaßnahmen, werden nicht auf den alternativen HTTP-Server angewendet.

Der OData Producer beachtet die Protokollierungskonfiguration des HTTP-Servers. Weitere Hinweise zur Protokollierung von HTTP-Servervorgängen finden Sie in Ihrer HTTP-Serverdokumentation.

- Stellen Sie die OData Server-Konfigurationsdatei zusammen mit der folgenden Datei bereit, die sich im Verzeichnis `%SQLANY16%\Java\` befindet:

- `dbodata.jar`

- Führen Sie das Deployment des JDBC-Clients durch. Dies umfasst den SQL Anywhere JDBC 4.0-Treiber und die unterstützenden Dateien.
- Laden Sie die `SQLAnywhereODataServlet`-Klasse, die die Java EE `HttpServlet`-Schnittstelle implementiert, in den HTTP-Server.

Siehe auch

- „Deployment von JDBC-Clients“ auf Seite 982
- „OData Server-Dienstprogramm (dbosrv16.exe)“ auf Seite 464
- „OData Server konfigurieren“ auf Seite 457
- <http://docs.oracle.com/javaee/1.3/api/javax/servlet/http/package-summary.html>

OData Producer-Dienstmodelle erstellen

Sie können ein OData Producer-Dienstmodell erstellen, um bestimmte Tabellen und Ansichten bereitzustellen, indem Sie eine OSDL-Datei (OData Service Definition Language) angeben.

OSDL-Syntax

```
service [ namespace "namespace-name" ] {
    "owner"."{table-name | view-name}" [ keys( "column-name", ... ) ];
    ...
}
```

Parameter

- **Namespace-Name** Der zu definierende Namespace. Wenn diese Option angegeben ist, wird **_Container** an *Namespace-Name* angehängt, um einen Behälternamen zu generieren.
- **Tabellenname** Der Name der Tabelle, die in den OData-Metadaten bereitgestellt werden soll.
- **Ansichtsname** Der Name der Ansicht, die in den OData-Metadaten bereitgestellt werden soll.
- **Spaltenname** Ein Spaltenname, der als Primärschlüssel verwendet werden soll, wenn in der durch *Tabellenname* oder *Ansichtsname* definierten Tabelle keiner definiert ist.

Mehrere Spaltennamen können referenziert werden.

Bemerkungen

Sie können einen Dienst definieren, der Referenzen auf mehrere Tabellen oder Ansichten enthält.

Die folgenden Einschränkungen gelten für die **keys**-Klausel:

- Sie müssen eine Schlüsselliste angeben, wenn Sie eine Tabelle oder Ansicht referenzieren, die keinen Primärschlüssel enthält.
- Sie müssen keine Schlüsselliste angeben, wenn Sie eine Tabelle referenzieren, die einen Primärschlüssel enthält.

Das Modell wird auf den OData Producer angewendet, wenn Sie die Textdatei mit der **Model**-Option in Ihrer OData Server-Konfigurationsdatei referenzieren.

Siehe auch

- „OData Server konfigurieren“ auf Seite 457
- „OData Server-Dienstprogramm (dbosrv16.exe)“ auf Seite 464
- „OData Server ausführen und Clients einrichten“ auf Seite 464

Beispiel

Im Folgenden finden Sie ein Beispiel für ein Dienstmodell, das eine Tabelle und eine Ansicht bereitstellt:

```
service namespace "DBServer" {  
    "dba"."TableWithPrimaryKey";  
    "dba"."ViewWithoutPrimaryKey" keys("primary_key1", "primary_key2");  
}
```

OData Server ausführen und Clients einrichten

OData Server-Beispiele sind im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\` verfügbar. Die Beispiele veranschaulichen, wie der OData Server ausgeführt wird, wie ein OData Client eingerichtet wird und wie Anforderungen und Antworten zwischen beiden ausgetauscht werden.

Erstellen Sie Sicherungskopien der ursprünglichen Beispiele, bevor Sie Änderungen am Dateiinhalt vornehmen.

Verbindung mit dem OData Server über einen .NET-Client herstellen

Das OData-Beispiel SalesOrder zeigt, wie ein Microsoft .NET-OData-Client verwendet wird, um OData-Anforderungen an einen OData Server zu senden, der sich mit der SQL Anywhere-Beispieldatenbank verbindet.

Weitere Hinweise finden Sie hier: `%SQLANYSAMPI6%\SQLAnywhere\ODataSalesOrders\readme.txt`.

Verbindung mit dem OData Server über einen Java-Client herstellen

Das OData-Beispiel Security zeigt, wie ein OData4J-Java-Client verwendet wird, um OData-Anforderungen über HTTPS an einen OData Server zu senden, der sich mit der SQL Anywhere-Beispieldatenbank verbindet.

Dieses Beispiel zeigt, wie ein OData Producer-Dienstmodell und die Datenbankauthentifizierung verwendet werden.

Weitere Informationen finden Sie hier: `%SQLANYSAMPI6%\SQLAnywhere\ODataSecurity\readme.txt`

Siehe auch

- „OData Server-Dienstprogramm (dbosrv16.exe)“ auf Seite 464
- „Dienstprogramm für Dienste (dbsvc) für Windows“ [*SQL Anywhere Server - Datenbankadministration*]

OData Server-Dienstprogramm (*dbosrv16.exe*)

Startet den eingebetteten HTTP-Server und hostet den OData Producer.

Syntax

dbosrv16 *properties-filename*

Option	Beschreibung
<i>Eigenschaften-Dateiname</i>	Verwendet die Optionen in der Konfigurationsdatei zum Einrichten der Datenbankverbindung.

Bemerkungen

Das OData Server-Dienstprogramm enthält sowohl die Servlet-Implementierung als auch die Komponenten, die zum Starten des eigenen HTTP-Servers erforderlich sind.

Wenn Sie Änderungen am Datenbankschema vornehmen, müssen Sie das OData-Dienstprogramm Serverstopp verwenden und anschließend den OData Server neu starten, damit die Änderungen in Kraft treten und für OData-Clients sichtbar werden.

Sie können das OData Server-Dienstprogramm als Dienst starten, indem Sie das Dienstprogramm dbsvc mit der **-t OData**-Option ausführen.

Siehe auch

- „HTTP-Server für OData-Vorgänge einrichten“ auf Seite 461
- „OData Server konfigurieren“ auf Seite 457
- „OData-Dienstprogramm Serverstopp (dbostop.exe)“ auf Seite 465
- „Dienstprogramm für Dienste (dbsvc) für Windows“ [*SQL Anywhere Server - Datenbankadministration*]

OData-Dienstprogramm Serverstopp (*dbostop.exe*)

Stoppt den eingebetteten HTTP-Server und den OData Producer.

Syntax

dbostop [**-q**] { **-f** *properties-filename* | **-p** *port-number* }

Option	Beschreibung
-f <i>Eigenschaften-Dateiname</i>	Verwendet die durch die ShutdownListenerPort-Option angegebene Portnummer, um eine Anforderung zum Herunterfahren zu senden.
-p <i>Portnummer</i>	Gibt die Portnummer an, an die eine Anforderung zum Herunterfahren gesendet werden soll. Der Standardwert ist 2449.
-q	Versucht, einen Server sanft herunterzufahren. Es werden keine Meldungen angezeigt.

Bemerkungen

Wenn die Optionen **-p** oder **-f** nicht bereitgestellt sind, versucht dieses Dienstprogramm, den eingebetteten HTTP-Server und den OData Producer auf Port 2449 herunterzufahren.

Die Portnummer für das Herunterfahren muss mit der ShutdownListenerPort-Option in der OData Server-Konfigurationsdatei identisch sein, die zum Starten des OData Servers verwendet wurde.

Wenn Sie Änderungen am Datenbankschema vornehmen, müssen Sie das OData-Dienstprogramm Serverstopp verwenden und anschließend den OData Server neu starten, damit die Änderungen in Kraft treten und für OData-Clients sichtbar werden.

Siehe auch

- [„OData Server konfigurieren“ auf Seite 457](#)
- [„OData Server-Dienstprogramm \(dbosrv16.exe\)“ auf Seite 464](#)

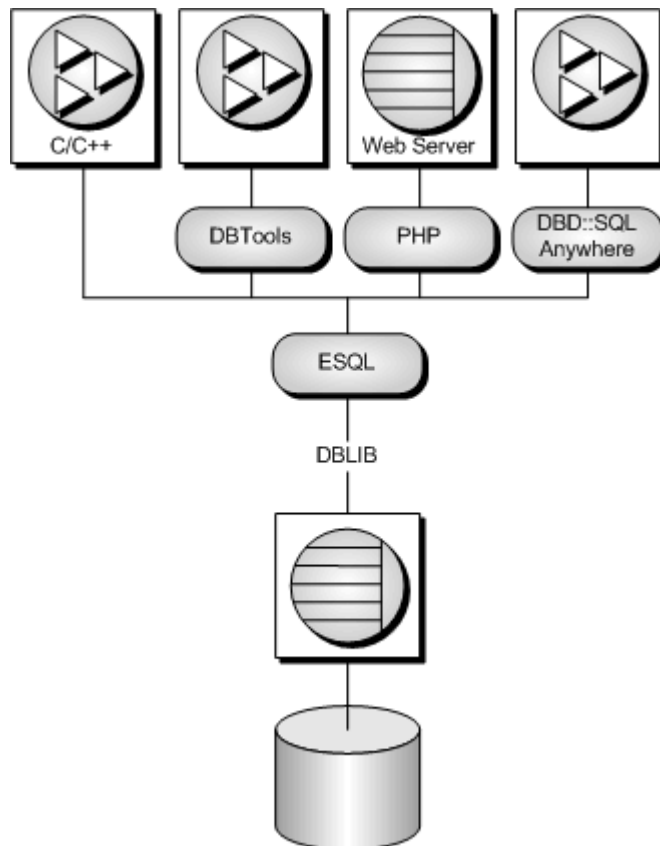
Embedded SQL

SQL-Anweisungen, die in einer C- oder C++-Quellendatei eingebettet (embedded) sind, werden als Embedded SQL referenziert. Ein Präprozessor übersetzt diese Anweisungen in Aufrufe einer Laufzeitbibliothek. Embedded SQL ist ein ISO/ANSI- und IBM-Standard.

Embedded SQL kann auf andere Datenbanken und andere Umgebungen portiert werden und ist in allen Betriebsumgebungen funktional äquivalent. Dabei handelt es sich um eine umfassende Schnittstelle auf niedriger Ebene, die alle für das Produkt erforderlichen Funktionen beinhaltet. Für Embedded SQL sind Kenntnisse in C oder C++ erforderlich.

Embedded SQL-Anwendungen

Sie können C- oder C++-Anwendungen entwickeln, die mithilfe der SQL Anywhere Embedded SQL-Schnittstelle auf den SQL Anywhere-Server zugreifen. Bei den Befehlszeilen-Datenbanktools handelt es sich um Anwendungen, die auf diese Weise programmiert wurden.



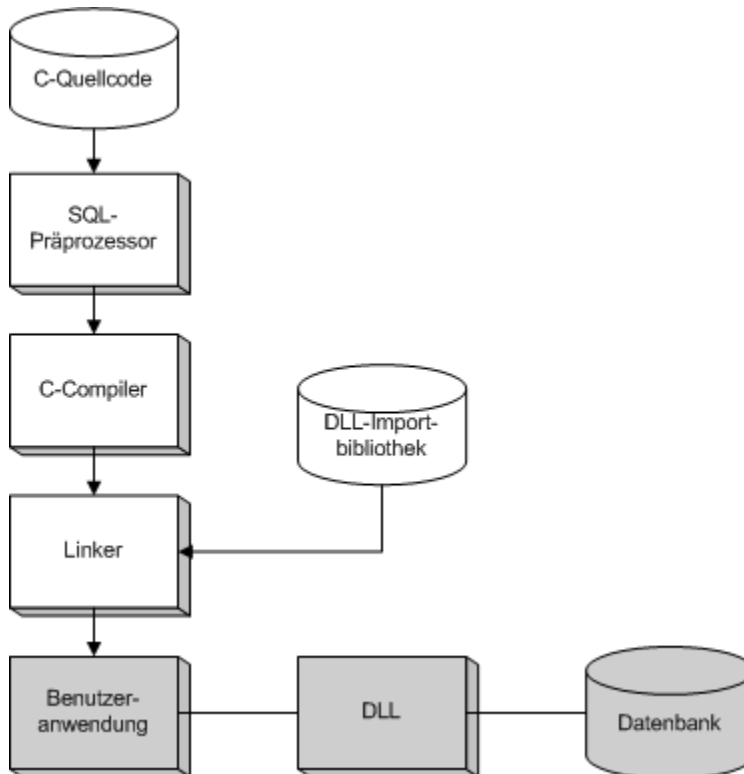
Embedded SQL ist eine Datenbank-Programmierschnittstelle für die Programmiersprachen C und C++. Es besteht aus SQL-Anweisungen, die in C- oder C++-Quellcode eingebettet sind. Die SQL-Anweisungen werden von einem **SQL-Präprozessor** in C- oder C++-Quellcode konvertiert, die Sie anschließend kompilieren.

Zur Laufzeit verwenden Embedded SQL-Anwendungen eine SQL Anywhere-**Schnittstellenbibliothek** namens DBLIB, um mit einem Datenbankserver zu kommunizieren. DBLIB ist auf den meisten Plattformen eine **DLL** (Dynamic Link Library --dynamische Verknüpfungsbibliothek) oder ein Shared Object.

- Unter Windows-Betriebssystemen heißt die Schnittstellenbibliothek *dblib16.dll*.
- Bei Unix-Betriebssystemen heißt die Schnittstellenbibliothek je nach Betriebssystem *libdblib16.so*, *libdblib16.sl*, oder *libdblib16.a*.
- Unter Mac OS X heißt die Schnittstellenbibliothek *libdblib16.dylib.1*.

SQL Anywhere stellt zwei Arten von Embedded SQL bereit. Static Embedded SQL ist einfacher zu verwenden, jedoch weniger flexibel als Dynamic Embedded SQL.

Überblick über den Entwicklungsprozess



Sobald das Anwendungsprogramm erfolgreich vom SQL-Präprozessor konvertiert und anschließend kompiliert ist, wird es mit der **Importbibliothek** für die DBLIB verknüpft, um eine ausführbare Programmdatei zu erzeugen. Wenn der Datenbankserver läuft, verwendet diese Programmdatei DBLIB, um mit dem Datenbankserver zu interagieren. Solange das Programm durch den Präprozessor läuft, braucht der Datenbankserver nicht zu laufen.

Für Windows gibt es 32-Bit- und 64-Bit-Importbibliotheken für Microsoft Visual C++. Die Verwendung von Importbibliotheken ist eine der Methoden für die Entwicklung von Anwendungen, die Funktionen in DLLs aufrufen. Es wird jedoch empfohlen, dass die Bibliothek dynamisch geladen wird, wodurch sich die Verwendung von Importbibliotheken erübrigt.

Siehe auch

- „DBLIB unter Windows dynamisch laden“ auf Seite 476

Der SQL-Präprozessor

Der SQL-Präprozessor ist eine Programmdatei namens *sqlpp*.

Die Präprozessor-Befehlszeile sieht wie folgt aus:

sqlpp [*options*] *sql-filename* [*output-filename*]

Der Präprozessor übersetzt die Embedded SQL-Anweisungen in einer C- oder C++-Quelldatei in C-Code und schreibt das Ergebnis in eine Ausgabedatei. Ein C- oder C++-Compiler wird für die Verarbeitung der Ausgabedatei eingesetzt. Die normale Dateinamenerweiterung für Quelldateien mit Embedded SQL ist *.sqlc*. Der voreingestellte Name für die Ausgabedatei ist *sql-filename* mit der Erweiterung *.c*. Wenn *sql-filename* die Erweiterung *.c* hat, wird die Standarderweiterung für die Ausgabedatei auf *.cc* geändert.

Hinweis

Wenn eine Anwendung für die Verwendung mit einer neuen Version der Datenbank-Interfacebibliothek neu aufgebaut wird, müssen die Embedded SQL-Dateien mit einem SQL-Präprozessor derselben Version vorverarbeitet werden.

Die folgende Tabelle beschreibt die Präprozessoroptionen.

Option	Beschreibung
-d	Code erzeugen, der die Größe des Datenbereichs reduziert. Datenstrukturen werden vor der Verwendung zur Ausführungszeit wieder benutzt und initialisiert. Dies erhöht die Code-Größe.

Option	Beschreibung
-e <i>Stufe</i>	<p>Kennzeichnet jedes Static Embedded SQL-Element als Fehler, das nicht Teil eines angegebenen Standards ist. Der Wert <i>Stufe</i> gibt den zu verwendenden Standard an. Beispiel: <code>sqlpp -e c03</code> ... kennzeichnet alle Syntaxelemente, die nicht Teil des SQL/2008-Kernstandards sind. Die folgenden Werte werden für <i>Stufe</i> unterstützt:</p> <ul style="list-style-type: none"> • c08 Kennzeichnet Syntax, die nicht SQL/2008-Kernsyntax ist • p08 Kennzeichnet Syntax, die nicht Full-SQL/2008-Syntax ist • c03 Kennzeichnet Syntax, die nicht SQL/2003-Kernsyntax ist • p03 Kennzeichnet Syntax, die nicht Full-SQL/2003-Syntax ist • c99 Kennzeichnet Syntax, die nicht SQL/1999-Kernsyntax ist • p99 Kennzeichnet Syntax, die nicht Full-SQL/1999-Syntax ist • e92 Kennzeichnet Syntax, die nicht Entry-Level der SQL/1992 Syntax ist • i92 Kennzeichnet Syntax, die nicht Intermediate-Level der SQL/1992 Syntax ist • f92 Kennzeichnet Syntax, die nicht Full-SQL/1992 Syntax ist • t Kennzeichnet Nicht-Standard-Hostvariablentypen • u Kennzeichnet Syntax, die nicht von UltraLite unterstützt wird <p>Für die Kompatibilität mit früheren Versionen von SQL Anywhere können Sie auch e, i und f angeben, die e92, i92 bzw. f92 entsprechen.</p>
-h <i>Breite</i>	Begrenzt die maximale Länge von Ausgabezeilen durch sqlpp auf <i>Breite</i> . Das Fortsetzungszeichen ist ein Backslash (\) und der Mindestwert von <i>Breite</i> ist 10.
-k	Teilt dem Präprozessor mit, dass das zu kompilierende Programm eine Benutzerdeklaration von SQLCODE enthält. Die Definition muss vom Typ "long" sein, doch sie muss sich nicht in einem Deklarationsabschnitt befinden.

Option	Beschreibung
-m <i>Modus</i>	<p>Geben Sie den Cursor-Aktualisierbarkeitsmodus an, falls dieser nicht explizit in der Embedded SQL-Anwendung angegeben ist. Der <i>Modus</i> kann einer der folgenden sein:</p> <ul style="list-style-type: none"> • HISTORICAL In früheren Versionen nahmen Embedded SQL-Cursor standardmäßig den Wert FOR UPDATE oder READ ONLY an (abhängig von der Abfrage und dem ansi_update_constraints-Optionswert). Explizite Cursor-Aktualisierbarkeit wurde in DECLARE CURSOR angegeben. Verwenden Sie diese Option, um dieses Verhalten beizubehalten. • READONLY Embedded SQL-Cursor werden standardmäßig auf READ ONLY gesetzt. Explizite Cursor-Aktualisierbarkeit wird in PREPARE angegeben. Dies ist die Standardeinstellung. READ ONLY-Cursor können zu einer verbesserten Performance führen.
-n	<p>Erzeugt Zeilennummerinformationen in der C-Datei. Diese umfassen <i>#line</i>-Direktiven an den betreffenden Stellen im erzeugten C-Code. Wenn der von Ihnen verwendete Compiler die Anweisung <i>#line</i> unterstützt, lässt diese Option den Compiler Fehler in Zeilennummern in der SQC-Datei protokollieren (der Datei mit Embedded SQL), und zwar im Gegensatz zum Protokollieren von Fehlern in Zeilennummern in der C-Datei, die vom SQL-Präprozessor erzeugt wird. Ebenso werden die <i>#line</i>-Anweisungen indirekt vom Source Level Debugger verwendet, sodass Sie bei der Fehlersuche gleichzeitig die SQC-Quelldatei sehen können.</p>
-o <i>Betriebssystem</i>	<p>Geben Sie das Zielbetriebssystem an. Folgende Betriebssysteme werden unterstützt:</p> <ul style="list-style-type: none"> • WINDOWS Microsoft Windows, einschließlich Windows Mobile. • UNIX Verwenden Sie diese Option, wenn Sie eine 32-Bit-Unix-Anwendung erstellen. • UNIX64 Verwenden Sie diese Option, wenn Sie eine 64-Bit-Unix-Anwendung erstellen.
-q	Dialogfreier Modus - keine Meldungen ausgeben.
-r-	Nicht wiedereintretenden (non-reentrant) Code generieren.
-s <i>Länge</i>	<p>Setzt die maximale Größe für Zeichenfolgen fest, die der Präprozessor in die C-Datei ausgibt. Zeichenfolgen, die länger als dieser Wert sind, werden mithilfe einer Liste von Zeichen initialisiert ('a','b','c' etc.). Die meisten C-Compiler sind in der Größe der Zeichenfolgenlitterale begrenzt, die Sie handhaben können. Mit dieser Option wird die obere Grenze festgesetzt. Der Standardwert ist 500.</p>
-u	Code für UltraLite generieren.

Option	Beschreibung
-w <i>Stufe</i>	<p>Kennzeichnet jedes Static Embedded SQL-Element als Warnung, das nicht Teil eines angegebenen Standards ist. Der Wert <i>Stufe</i> gibt den zu verwendenden Standard an. Beispiel: <code>sqlpp -w c08</code> ... kennzeichnet alle SQL-Syntaxelemente, die nicht Teil der SQL/2008-Kernsyntax sind. Die folgenden Werte werden für <i>Stufe</i> unterstützt:</p> <ul style="list-style-type: none"> • c08 Kennzeichnet Syntax, die nicht SQL/2008-Kernsyntax ist • p08 Kennzeichnet Syntax, die nicht Full-SQL/2008-Syntax ist • c03 Kennzeichnet Syntax, die nicht SQL/2003-Kernsyntax ist • p03 Kennzeichnet Syntax, die nicht Full-SQL/2003-Syntax ist • c99 Kennzeichnet Syntax, die nicht SQL/1999-Kernsyntax ist • p99 Kennzeichnet Syntax, die nicht Full-SQL/1999-Syntax ist • e92 Kennzeichnet Syntax, die nicht Entry-Level der SQL/1992 Syntax ist • i92 Kennzeichnet Syntax, die nicht Intermediate-Level der SQL/1992 Syntax ist • f92 Kennzeichnet Syntax, die nicht Full-SQL/1992 Syntax ist • t Kennzeichnet Nicht-Standard-Hostvariablentypen • u Kennzeichnet Syntax, die nicht von UltraLite unterstützt wird <p>Für die Kompatibilität mit früheren Versionen von SQL Anywhere können Sie auch e, i und f angeben, die e92, i92 bzw. f92 entsprechen.</p>
-x	Ändert Mehrbyte-Zeichenfolgen in Escape-Sequenzen, sodass sie vom Compiler verarbeitet werden können.
-z <i>cs</i>	<p>Geben Sie die Kollationssequenz an. Um eine Liste mit den empfohlenen Kollationssequenzen zu erhalten, führen Sie an der Eingabeaufforderung <code>dbinit -l</code> aus.</p> <p>Die Kollationssequenz hilft dem Präprozessor, die Zeichen zu verstehen, die im Quellcode des Programms verwendet werden, z.B. bei der Identifizierung von alphabetischen Zeichen, die für die Verwendung in Bezeichnungen geeignet sind. Wenn -z nicht angegeben ist, versucht der Präprozessor eine sinnvolle Kollation zu ermitteln, basierend auf dem Betriebssystem sowie den Umgebungsvariablen SALANG und SACHARSET.</p>
<i>sql-filename</i>	Ein C- oder C++-Programm, das zu verarbeitende Embedded SQL-Elemente enthält
<i>output-file-name</i>	Die C-Quelldatei, die vom SQL-Präprozessor erstellt wurde

Siehe auch

- „Embedded SQL“ auf Seite 467
- „sql_flagger_error_level-Option“ [*SQL Anywhere Server - Datenbankadministration*]
- „sql_flagger_warning_level-Option“ [*SQL Anywhere Server - Datenbankadministration*]
- „SQLFLAGGER-Funktion [Verschiedene]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_ansi_standard_packages-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Fehlermeldungen des SQL-Präprozessors“ [*Fehlermeldungen*]
- „SQLCA-Verwaltung für Code mit mehreren Threads oder "reentrant"-Code“ auf Seite 498
- „UltraLite Embedded SQL API-Referenz“ [*UltraLite - C- und C++-Programmierung*]
- „SACHARSET-Umgebungsvariable“ [*SQL Anywhere Server - Datenbankadministration*]
- „SALANG-Umgebungsvariable“ [*SQL Anywhere Server - Datenbankadministration*]

Unterstützte Compiler

Der SQL-Präprozessor für C wird bisher zusammen mit den folgenden Compilern verwendet:

Betriebssystem	Compiler	Version
Windows	Microsoft Visual C++	6.0 oder später
Windows Mobile	Microsoft Visual C++	2005
Unix	GNU oder nativer Compiler	

Header-Dateien für Embedded SQL

Alle Header-Dateien befinden sich nach der Installation von SQL Anywhere im Unterverzeichnis *SDK Include* des Installationsverzeichnis.

Dateiname	Beschreibung
<i>sqlca.h</i>	Wichtigste Header-Datei, wird in alle Embedded SQL-Programme eingefügt. Die Datei fügt die Strukturdefinition für den SQL-Kommunikationsbereich ein (SQL Communication Area - SQLCA) sowie Prototypen für alle Embedded SQL-Datenbankschnittstellenfunktionen.
<i>sqlda.h</i>	SQL-Deskriptorbereich-Strukturdefinition, wird in Embedded SQL-Programme eingefügt, die Dynamic SQL verwenden
<i>sqldef.h</i>	Definition der Datentypen für die Embedded SQL-Schnittstelle. Die Datei enthält auch Strukturdefinitionen und Rückgabewerte, die gebraucht werden, um den Datenbankserver aus einem C-Programm zu starten.
<i>sqlerr.h</i>	Definitionen für Fehlercodes, die im Feld sqlcode des SQLCA-Bereichs zurückgegeben werden

Dateiname	Beschreibung
<i>sqlstate.h</i>	Definitionen für ANSI/ISO SQL-Standardfehlerzustände, die im Feld <i>sqlstate</i> des SQLCA-Bereichs zurückgegeben werden
<i>pshpk1.h</i> , <i>pshpk4.h</i> , <i>poppk.h</i>	Diese Header stellen sicher, dass das Strukturverdichten (structure packing) korrekt gehandhabt wird.

Importbibliotheken

Auf Windows-Plattformen werden alle Importbibliotheken in den Unterverzeichnissen *SDK\Lib* des SQL Anywhere-Installationsverzeichnisses installiert. Windows-Importbibliotheken werden in den Unterverzeichnissen *SDK\Lib\x86* und *SDK\Lib\x64* gespeichert. Windows Mobile-Importbibliotheken werden im Unterverzeichnis *SDK\Lib\CE\Arm.50* installiert. Eine Exportdefinitionsliste wird in *SDK\Lib\Def\dblib.def* gespeichert.

Auf Unix-Plattformen werden alle Importbibliotheken in den Unterverzeichnissen *lib32* und *lib64* des SQL Anywhere-Installationsverzeichnisses installiert.

Auf Mac OS X-Plattformen werden alle Importbibliotheken in den Unterverzeichnissen *System/lib32* und *System/lib64* des SQL Anywhere-Installationsverzeichnisses installiert.

Betriebssystem	Compiler	Importbibliothek
Windows	Microsoft Visual C++	<i>dblibtm.lib</i>
Windows Mobile	Microsoft Visual C++ 2005	<i>dblib16.dll</i>
Unix (Anwendungen ohne Threading)	Alle Compiler	<i>libdblib16.so</i> , <i>libdbtasks16.so</i> , <i>libdblib16.sl</i> , <i>libdbtasks16.sl</i>
Unix (Anwendungen mit Threading)	Alle Compiler	<i>libdblib16_r.so</i> , <i>libdbtasks16_r.so</i> , <i>libdblib16_r.sl</i> , <i>libdbtasks16_r.sl</i>
Mac OS X (Anwendungen mit Threading)	Alle Compiler	<i>libdblib16.dylib</i> , <i>libdbtasks16.dylib</i>
Mac OS X (Anwendungen mit Threading)	Alle Compiler	<i>libdblib16_r.dylib</i> , <i>libdbtasks16_r.dylib</i>

Die *libdbtasks16*-Bibliotheken werden von den *libdblib16*-Bibliotheken aufgerufen. Einige Compiler ermitteln die Position von *libdbtasks16* automatisch. Bei anderen müssen Sie sie explizit angeben.

Beispielprogram mit Embedded SQL

Es folgt ein sehr einfaches Beispiel für ein Embedded SQL-Programm.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main()
{
    db_init( &sqlca );
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    EXEC SQL UPDATE Employees
        SET Surname = 'Plankton'
        WHERE EmployeeID = 195;
    EXEC SQL COMMIT WORK;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( 0 );
error:
    printf( "update unsuccessful -- sqlcode = %ld\n",
        sqlca.sqlcode );
    db_fini( &sqlca );
    return( -1 );
}
```

Dieses Beispiel stellt eine Verbindung mit der Datenbank her, aktualisiert den Nachnamen des Angestellten mit der Nummer 195, schreibt die Änderung fest und beendet das Programm. Es gibt praktisch keine Interaktion zwischen dem Embedded-SQL-Code und dem C-Code. C-Code wird in diesem Beispiel ausschließlich zur Ablaufsteuerung benutzt. Die Anweisung WHENEVER wird zur Fehlerprüfung verwendet. Die Fehleraktion (in diesem Beispiel GOTO) wird nach jeder SQL-Anweisung ausgeführt, die einen Fehler verursacht.

Siehe auch

- „Abruf von Daten mit Embedded SQL“ auf Seite 514

Struktur von Embedded SQL-Programmen

In Embedded-SQL-Programmen sind SQL-Anweisungen in normalen C- oder C++-Code eingebettet. Alle Embedded SQL-Anweisungen beginnen mit den Worten EXEC SQL und enden mit einem Semikolon (;). Normale Kommentare der Sprache C sind innerhalb von Embedded SQL-Anweisungen erlaubt.

Der Quellcode eines C-Programms, das Embedded SQL benutzt, muss vor der ersten Embedded SQL-Anweisung in der Quelldatei folgende Anweisung enthalten:

```
EXEC SQL INCLUDE SQLCA;
```

Jedes C-Programm, das Embedded SQL verwendet, muss zuerst einen SQLCA initialisieren.

```
db_init( &sqlca );
```

Eine der ersten Embedded SQL-Anweisungen, die von dem C-Programm ausgeführt wird, muss eine CONNECT-Anweisung sein. Die CONNECT-Anweisung stellt eine Verbindung mit dem

Datenbankserver her und spezifiziert die Benutzer-ID für die Autorisierung aller Anweisungen, die während der Verbindung ausgeführt werden.

Ausnahme: Einige Embedded SQL-Anweisungen erzeugen keinen C-Code oder benötigen keine Verbindung mit der Datenbank. Diese Anweisungen sind vor der CONNECT-Anweisung erlaubt. Die wichtigsten sind die INCLUDE-Anweisung und die WHENEVER-Anweisung für die Fehlerbehandlung.

Jedes C-Programm, das Embedded SQL verwendet, muss alle SQLCAs finalisieren, die initialisiert wurden.

```
db_fini( &sqlca );
```

DBLIB unter Windows dynamisch laden

Laden Sie DBLIB dynamisch aus Ihrer eingebetteten Anwendung mithilfe des *esqldll.c*-Moduls im Unterverzeichnis *SDK\C* Ihres Installationsverzeichnis, sodass Sie auf das Linken mit der Importbibliothek verzichten können.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Diese Aufgabe ist eine Alternative zur üblichen Methode des Verknüpfens einer Anwendung mit einer statischen Importbibliothek für eine DLL (Dynamic Link Library --dynamische Verknüpfungsbibliothek), die die erforderlichen Funktionsdefinitionen enthält.

Eine ähnliche Aufgabe kann verwendet werden, um DBLIB auf Unix-Plattformen dynamisch zu laden.

Aufgabe

1. Ihre Anwendung muss `db_init_dll` aufrufen, um die DBLIB-DLL zu laden, und `db_fini_dll`, um die DBLIB-DLL wieder freizugeben. Der `db_init_dll`-Aufruf muss vor allen Funktionen in der Datenbankschnittstelle erfolgen, und keine Funktion in der Schnittstelle darf nach `db_fini_dll` aufgerufen werden.

Sie müssen trotzdem die Bibliotheksfunktionen `db_init` und `db_fini` aufrufen.

2. Fügen Sie vor der Anweisung `EXEC SQL INCLUDE SQLCA` mit `#include` die Header-Datei *esqldll.h* ein, oder fügen Sie *sqlca.h* mit `#include` in Ihr Embedded SQL-Programm ein. Die Header-Datei *esqldll.h* enthält *sqlca.h*.
3. Ein SQL OS (Operating System - Betriebssystem)-Makro muss definiert werden. Die Header-Datei *sqlos.h*, die mit *sqlca.h* eingefügt wird, versucht das passende Makro zu bestimmen und zu definieren. Bei bestimmten Kombinationen von Betriebssystemen und Compilern kann dieser Versuch allerdings fehlschlagen. In diesem Fall müssen Sie entweder mit `#define` eine Definition am Beginn Ihrer Datei einfügen, oder die Definition mit einer Compiler-Option liefern. Der Makro, der für Windows definiert werden muss, wird unten dargestellt.

Makro	Unterstützte Plattformen
_SQL_OS_WINDOWS	Alle Windows-Betriebssysteme

- Kompilieren Sie *esqldll.c*.
- Statt mit der Importbibliothek linken Sie Ihre Embedded SQL-Anwendung mit dem Objektmodul *esqldll.obj*.

Ergebnisse

Die DBLIB-Schnittstellen-DLL wird dynamisch geladen, wenn Sie ihre Embedded SQL-Anwendung ausführen.

Beispiel

Ein Beispielprogramm, aus dem ersichtlich ist, wie die Schnittstellenbibliothek dynamisch geladen wird, finden Sie im Verzeichnis *%SQLANY16%\SQLAnywhere\ESQLDynamicLoad*. Der Quellcode befindet sich in *sample.sqc*.

Im folgenden Beispiel wird unter Windows die Datei *sample.sqc* mit dem Code aus *esqldll.c* kompiliert und verknüpft.

```
sqlpp sample.sqc
cl sample.c %SQLANY16%\sdk\c\esqldll.c /I%SQLANY16%\sdk\include Advapi32.lib
```

Beispielprogramme mit Embedded SQL

Die Embedded SQL-Beispiele sind Teil der SQL Anywhere-Installation. Sie werden im Verzeichnis *%SQLANY16%\SQLAnywhere\C* abgelegt. Unter Windows Mobile ist ein weiteres Beispiel im Verzeichnis *\SQLAnywhere\CE\esql_sample* verfügbar.

- Das Beispiel *cur.sqc* für statische Cursor in Embedded SQL zeigt die Verwendung von Static SQL-Anweisungen.
- Das Beispiel *dcur.sqc* für dynamische Cursor in Embedded SQL zeigt die Verwendung von Dynamic SQL-Anweisungen.

Um die Anzahl der Codezeilen zu verringern, die in den Beispielprogrammen mehrmals vorkommen, sind die Mainlines und die Druckfunktionen in einer separaten Datei enthalten. Und zwar in *mainch.c* für zeichenbasierte Systeme und in *mainwin.c* für fensterbasierte Systeme.

Die Beispielprogramme liefern jeweils folgende drei Routinen, die von den Mainlines aufgerufen werden:

- WSQLEX_Init** Stellt eine Verbindung zur Datenbank her und öffnet den Cursor.
- WSQLEX_Process_Command** Verarbeitet Benutzerbefehle und bedient den Cursor.
- WSQLEX_Finish** Schließt den Cursor und trennt die Verbindung zur Datenbank.

Die Funktion der Mainline ist:

1. Sie ruft die `WSQLEX_Init`-Routine auf.
2. Sie ist eine Schleife, die Befehle vom Benutzer abholt, und `WSQL_Process_Command` aufruft, bis der Benutzer das Programm beendet.
3. Sie ruft die `WSQLEX_Finish`-Routine auf.

Die Verbindung mit der Datenbank wird mit der Embedded SQL-Anweisung `CONNECT` hergestellt, die die erforderliche Benutzer-ID und das Kennwort bereitstellt.

Zusätzlich zu diesen Beispielen können Sie als Teil von SQL Anywhere weitere Programme und Quellendateien finden, die Eigenschaften des Servers für bestimmte Betriebssystem-Plattformen veranschaulichen.

Beispiel für statischen Cursor

Dieses Beispiel zeigt die Verwendung von Cursor. Der Cursor, der hier verwendet wurde, ruft bestimmte Informationen aus der Tabelle `Employees` in der Beispieldatenbank ab. Der Cursor wird statisch deklariert, das heißt, die SQL-Anweisung, die die Information abrufen, ist im Quellprogramm enthalten. Dies ist ein gutes Beispiel, um zu lernen, wie Cursor funktionieren. Das Beispiel für dynamische Cursor konvertiert dieses erste Beispiel und verwendet Dynamic SQL-Anweisungen.

Die `open_cursor`-Routine deklariert einen Cursor für die jeweilige SQL-Abfrage und öffnet ebenfalls den Cursor.

Die `print`-Routine druckt eine Seite mit Informationen. Sie führt *pagesize*-mal eine Schleife aus, die eine einzelne Zeile vom Cursor abrufen und druckt. Die Abrufroutine überprüft, ob eine Warnsituation vorliegt (wie Zeile nicht gefunden, `SQLCODE 100`) und gibt gegebenenfalls die erforderlichen Fehlermeldung aus. Außerdem positioniert das Programm den Cursor neu in der Zeile vor derjenigen, die am Anfang der aktuellen Datenseite ausgegeben wurde.

Die Routinen `move`, `top` und `bottom` positionieren den Cursor mit der geeigneten Form der `FETCH`-Anweisung. Diese Form der `FETCH`-Anweisung ruft nicht tatsächlich die Daten ab, sondern positioniert lediglich den Cursor. Außerdem wurde eine allgemeine Routine namens `move` zur relativen Positionierung implementiert, die abhängig vom Vorzeichen des Parameters den Cursor in die eine oder andere Richtung bewegt.

Beendet der Benutzer das Programm, wird der Cursor geschlossen und die Datenbankverbindung getrennt. Der Cursor wird durch eine `ROLLBACK WORK`-Anweisung geschlossen und die Verbindung mit `DISCONNECT` getrennt.

Siehe auch

- „Beispielprogramm für statische Cursor ausführen“ auf Seite 479
- „Beispiel für dynamischen Cursor“ auf Seite 480

Beispielprogramm für statische Cursor ausführen

Führen Sie das Beispielprogramms für statische Cursor aus.

Voraussetzungen

Unter Windows ist eine neue Version von Microsoft Visual Studio erforderlich.

Bei x86/x64-Plattform-Builds mit Visual Studio müssen Sie die richtige Umgebung zum Kompilieren und Verknüpfen einrichten. Dies erfolgt in der Regel in Visual Studio mit *vcvars32.bat* oder *vcvars64.bat* (in älteren Versionen von Visual Studio *vcvarsamd64.bat*).

Kontext und Bemerkungen

Die Programmdateien und der entsprechende Quellcode befinden sich im Verzeichnis *%SQLANYWHERE%\SQLAnywhere\C*. Unter Windows Mobile ist ein weiteres Beispiel im Verzeichnis *\SQLAnywhere\CE\esql_sample* verfügbar.

Aufgabe

1. Starten Sie die SQL Anywhere-Beispieldatenbank *demo.db*.
2. Dateien zum Aufbauen der Beispielprogramme werden mit dem Beispielcode mitgeliefert.

Führen Sie unter Windows die Batchdatei *build.bat* aus.

Wenn Aufbaufehler ausgegeben werden, versuchen Sie, die Zielplattform (x86 oder x64) als Argument für *build.bat* anzugeben. Hier sehen Sie ein Beispiel.

```
build x64
```

Verwenden Sie unter Windows Mobile die Projektdatei *esql_sample.sln* für Microsoft Visual C++.
Diese Datei befindet sich unter *SQLAnywhere\CE\esql_sample*.

Verwenden Sie unter Unix das Shell-Skript *build.sh*, um das Beispiel aufzubauen.

3. Führen Sie unter 32-Bit-Windows die Datei *curwin.exe* aus.

Führen Sie unter 64-Bit-Windows die Datei *curx64.exe* aus.

Führen Sie unter Unix die Datei *cur* aus.

4. Befolgen Sie die Anweisungen auf dem Bildschirm.

Ergebnisse

Die verschiedenen Befehle verändern einen Datenbankcursor und geben Abfrageergebnisse am Bildschirm aus. Geben Sie einfach den Buchstaben des Befehls ein, den Sie ausführen möchten. Bei einigen Systemen müssen Sie anschließend die Eingabetaste drücken, um den Befehl auszuführen.

Beispiel für dynamischen Cursor

Dieses Beispiel veranschaulicht, wie Cursor für eine Dynamic SQL-SELECT-Anweisung verwendet werden.

Das dynamische Cursor-Beispielprogramm (dcur) erlaubt dem Benutzer, eine Tabelle auszuwählen, die er anschauen möchte. Das Programm zeigt dann so viele Informationen aus dieser Tabelle an, wie auf den Bildschirm passen.

Wenn dieses Programm ausgeführt wird, werden Sie zur Eingabe einer Verbindungszeichenfolge aufgefordert. Im Folgenden finden Sie ein Beispiel.

```
UID=DBA;PWD=sql;DBF=demo.db
```

Das C-Programm mit dem Embedded SQL-Code befindet sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\C`. Unter Windows Mobile befindet sich ein dynamisches Cursorbeispiel im Verzeichnis `\SQLAnywhere\CE\esql_sample`.

Das dcur-Programm verwendet die Embedded SQL-Schnittstellenfunktion `db_string_connect`, um eine Verbindung mit der Datenbank herzustellen. Diese Funktion liefert die zusätzliche Funktionalität, um die Zeichenfolge für die Verbindung zu unterstützen, die benutzt wird, um die Verbindung zur Datenbank herzustellen.

Die Routine `open_cursor` baut zuerst die SELECT-Anweisung auf:

```
SELECT * FROM table-name
```

Dabei gilt: *Tabellenname* ist der Parameter, der an die Routine übergeben wird. Dann bereitet sie eine Dynamic SQL-Anweisung mithilfe dieser Zeichenfolge vor.

Die Embedded SQL-Anweisung DESCRIBE wird benutzt, um die Ergebnisse der SELECT-Anweisung in die SQLDA-Struktur zu schreiben.

Hinweis

Zu Beginn wird die Größe der SQLDA geschätzt (3). Ist das nicht groß genug, wird die tatsächliche Größe der Auswahlliste, die vom Datenbankserver zurückgegeben wurde, benutzt, um eine SQLDA der richtigen Größe zuzuweisen.

Die SQLDA-Struktur wird dann mit Puffern gefüllt, die Zeichenfolgen enthalten, die wiederum für das Ergebnis der Abfrage stehen. Die Routine `fill_s_sqlda` konvertiert alle Datentypen in der SQLDA in DT_STRING und weist Puffer der passenden Größe zu.

Dann wird für diese Anweisung ein Cursor deklariert und geöffnet. Die übrigen Routinen zum Bewegen und Schließen des Cursors sind die gleichen wie beim vorhergehenden Beispiel.

Die Routine `fetch` sieht etwas anders aus: Sie setzt die Ergebnisse in die SQLDA-Struktur und nicht in eine Liste von Hostvariablen. Die `print`-Routine ist leicht verändert, um Ergebnisse aus der SQLDA-Struktur bis zur vollen Bildschirmbreite auszugeben. Die `print`-Routine benutzt auch die Namenfelder des SQLDA-Bereichs, um Überschriften für die Spalten auszugeben.

Siehe auch

- „Beispielprogramm für dynamische Cursor ausführen“ auf Seite 481
- „Beispiel für statischen Cursor“ auf Seite 478

Beispielprogramm für dynamische Cursor ausführen

Führen Sie das Beispielprogramms für dynamische Cursor aus.

Voraussetzungen

Unter Windows ist eine neue Version von Microsoft Visual Studio erforderlich.

Bei x86/x64-Plattform-Builds mit Visual Studio müssen Sie die richtige Umgebung zum Kompilieren und Verknüpfen einrichten. Dies erfolgt in der Regel in Visual Studio mit *vcvars32.bat* oder *vcvars64.bat* (in älteren Versionen von Visual Studio *vcvarsamd64.bat*).

Kontext und Bemerkungen

Die Programmdateien und der entsprechende Quellcode befinden sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\C`. Unter Windows Mobile ist ein weiteres Beispiel im Verzeichnis `\SQLAnywhere\CE\esql_sample` verfügbar.

Aufgabe

1. Starten Sie die SQL Anywhere-Beispieldatenbank *demo.db*.
2. Dateien zum Aufbauen der Beispielprogramme werden mit dem Beispielcode mitgeliefert.

Führen Sie unter Windows die Batchdatei *build.bat* aus.

Wenn Aufbaufehler ausgegeben werden, versuchen Sie, die Zielplattform (x86 oder x64) als Argument für *build.bat* anzugeben. Hier sehen Sie ein Beispiel.

```
build x64
```

Verwenden Sie unter Windows Mobile die Projektdatei *esql_sample.sln* für Microsoft Visual C++. Diese Datei befindet sich unter `SQLAnywhere\CE\esql_sample`.

Verwenden Sie unter Unix das Shell-Skript *build.sh*, um das Beispiel aufzubauen.

3. Führen Sie unter 32-Bit-Windows die Datei *dcurwin.exe* aus.

Führen Sie unter 64-Bit-Windows die Datei *dcurx64.exe* aus.

Für das Windows Mobile-Beispiel müssen Sie die Datei *esql_sample.exe* bereitstellen und auf Ihrem Windows Mobile-Gerät ausführen.

Führen Sie unter Unix die Datei *dcur* aus.

4. Alle Beispielprogramme haben eine Konsolen-Benutzeroberfläche mit einer Eingabeaufforderung. Geben Sie die nachstehende Zeichenfolge ein, damit eine Verbindung mit der Beispieldatenbank hergestellt wird:

`DSN=SQL Anywhere 16 Demo`

5. Jedes einzelne Beispielprogramm fordert Sie auf, eine Tabelle anzugeben. Wählen Sie eine der Tabellen in der Beispieldatenbank. Sie können z.B. **Customers** oder **Employees** eingeben.
6. Befolgen Sie die Anweisungen auf dem Bildschirm.

Ergebnisse

Die verschiedenen Befehle verändern einen Datenbankcursor und geben Abfrageergebnisse am Bildschirm aus. Geben Sie einfach den Buchstaben des Befehls ein, den Sie ausführen möchten. Bei einigen Systemen müssen Sie anschließend die Eingabetaste drücken, um den Befehl auszuführen.

Datentypen in Embedded SQL

Um Informationen zwischen einem Programm und dem Datenbankserver austauschen zu können, müssen alle Datenelemente einen Datentyp haben. Die Konstanten für Embedded SQL haben alle das Präfix `DT_` und befinden sich in der Header-Datei *sqldef.h*. Eine Hostvariable kann jeden der unterstützten Datentypen haben. Sie können diese Datentypen auch in einer `SQLDA`-Struktur verwenden, um Daten mit der Datenbank auszutauschen.

Sie können mit den `DECL_`-Makros in *sqlca.h* Variablen dieser Datentypen definieren. Beispielsweise kann mit `DECL_BIGINT` eine Variable mit einem `BIGINT`-Wert deklariert werden.

Folgende Datentypen werden von der Embedded SQL-Programmierschnittstelle unterstützt:

- **DT_BIT** 8-Bit-Ganzzahl mit Vorzeichen
- **DT_SMALLINT** 16-Bit-Ganzzahl mit Vorzeichen
- **DT_UNSSMALLINT** 16-Bit Ganzzahl ohne Vorzeichen
- **DT_TINYINT** 8-Bit-Ganzzahl mit Vorzeichen
- **DT_BIGINT** 64-Bit-Ganzzahl mit Vorzeichen
- **DT_UNSBIGINT** 64-Bit-Ganzzahl ohne Vorzeichen.
- **DT_INT** 32-Bit-Ganzzahl mit Vorzeichen
- **DT_UNSENT** 32-Bit Ganzzahl ohne Vorzeichen
- **DT_FLOAT** 4-Byte-Gleitkommazahl.
- **DT_DOUBLE** 8-Byte-Gleitkommazahl.

- **DT_DECIMAL** Gepackte Dezimalzahl (systemeigenes Format)

```
typedef struct TYPE_DECIMAL {  
    char array[1];  
} TYPE_DECIMAL;
```

- **DT_STRING** Mit NULL abgeschlossene Zeichenfolge im CHAR-Zeichensatz. Die Zeichenfolge wird mit Leerzeichen aufgefüllt, wenn bei der Initialisierung der Datenbank mit Leerzeichen aufgefüllte Zeichenfolgen verwendet werden.
- **DT_NSTRING** Mit NULL abgeschlossene Zeichenfolge im NCHAR-Zeichensatz. Die Zeichenfolge wird mit Leerzeichen aufgefüllt, wenn bei der Initialisierung der Datenbank mit Leerzeichen aufgefüllte Zeichenfolgen verwendet werden.
- **DT_DATE** Mit NULL abgeschlossene Zeichenfolge, die ein gültiges Datum repräsentiert.
- **DT_TIME** Mit NULL abgeschlossene Zeichenfolge, die eine gültige Zeitangabe repräsentiert.
- **DT_TIMESTAMP** Mit NULL abgeschlossene Zeichenfolge, die einen gültigen Zeitstempel darstellt.
- **DT_FIXCHAR** Zeichenfolge mit fester Länge, die mit Leerzeichen aufgefüllt wird. Im CHAR-Zeichensatz. Die maximale Länge ist 32767 (in Byte). Die Daten enden nicht mit NULL.
- **DT_NFIXCHAR** Zeichenfolge mit fester Länge, die mit Leerzeichen aufgefüllt wird. Im NCHAR-Zeichensatz. Die maximale Länge ist 32767 (in Byte). Die Daten enden nicht mit NULL.
- **DT_VARCHAR** Zeichenfolge variabler Länge, im CHAR-Zeichensatz, mit einem 2-Byte-Längenfeld. Die maximale Länge beträgt 32765 Byte. Beim Senden von Daten müssen Sie das Längenfeld festlegen. Beim Abrufen von Daten setzt der Datenbankserver das Längenfeld. Die Daten werden nicht mit NULL abgeschlossen oder mit Leerzeichen aufgefüllt.

```
typedef struct VARCHAR {  
    a_sql_ulen len;  
    char array[1];  
} VARCHAR;
```

- **DT_NVARCHAR** Zeichenfolge variabler Länge, im NCHAR-Zeichensatz, mit einem 2-Byte-Längenfeld. Die maximale Länge beträgt 32765 Byte. Beim Senden von Daten müssen Sie das Längenfeld festlegen. Beim Abrufen von Daten setzt der Datenbankserver das Längenfeld. Die Daten werden nicht mit NULL abgeschlossen oder mit Leerzeichen aufgefüllt.

```
typedef struct NVARCHAR {  
    a_sql_ulen len;  
    char array[1];  
} NVARCHAR;
```

- **DT_LONGVARCHAR** Lange Zeichenfolge von variabler Länge, im CHAR-Zeichensatz.

```
typedef struct LONGVARCHAR {  
    a_sql_uint32 array_len; /* number of allocated bytes in array */  
    a_sql_uint32 stored_len; /* number of bytes stored in array  
                             * (never larger than array_len) */  
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated expression  
                             * (may be larger than array_len) */  
}
```

```
char      array[1];  /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

Die LONGVARCHAR-Struktur kann mit Daten über 32767 Byte verwendet werden. Große Daten können auf einmal oder mithilfe der Anweisung GET DATA in Einzelschritten abgerufen werden. Sie können entweder auf einmal oder in Abschnitten an den Server übermittelt werden, indem sie an eine Datenbankvariable mit der SET-Anweisung angehängt werden. Die Daten werden nicht mit NULL abgeschlossen oder mit Leerzeichen aufgefüllt.

- **DT_LONGNVARCHAR** Lange Zeichenfolge von variabler Länge, im NCHAR-Zeichensatz. Der Makro definiert wie folgt eine Struktur:

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
                             * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated expression
                             * (may be larger than array_len) */
    char      array[1];  /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

Die LONGNVARCHAR-Struktur kann mit Daten über 32767 Byte verwendet werden. Große Daten können auf einmal oder mithilfe der Anweisung GET DATA in Einzelschritten abgerufen werden. Sie können entweder auf einmal oder in Abschnitten an den Server übermittelt werden, indem sie an eine Datenbankvariable mit der SET-Anweisung angehängt werden. Die Daten werden nicht mit NULL abgeschlossen oder mit Leerzeichen aufgefüllt.

- **DT_BINARY** Binärdaten variabler Länge mit einem 2-Byte-Längenfeld. Die maximale Länge beträgt 32765 Byte. Wenn Sie dem Datenbankserver Informationen liefern, müssen Sie das Längenfeld setzen. Wenn Sie Informationen vom Datenbankserver abrufen, setzt der Datenbankserver das Längenfeld.

```
typedef struct BINARY {
    a_sql_ulen len;
    char      array[1];
} BINARY;
```

- **DT_LONGBINARY** Lange Binärdaten. Der Makro definiert wie folgt eine Struktur:

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
                             * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated expression
                             * (may be larger than array_len) */
    char      array[1];  /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

Die LONGBINARY-Struktur kann mit Daten über 32767 Byte verwendet werden. Große Daten können auf einmal oder mithilfe der Anweisung GET DATA in Einzelschritten abgerufen werden. Sie können entweder auf einmal oder in Abschnitten an den Server übermittelt werden, indem sie an eine Datenbankvariable mit der SET-Anweisung angehängt werden.

- **DT_TIMESTAMP_STRUCT** SQLDATETIME-Struktur mit Feldern für jeden Teil eines Zeitstempels

```
typedef struct sqldatetime {
    unsigned short year; /* for example 1999 */
    unsigned char month; /* 0-11 */
    unsigned char day_of_week; /* 0-6 0=Sunday */
    unsigned short day_of_year; /* 0-365 */
    unsigned char day; /* 1-31 */
    unsigned char hour; /* 0-23 */
    unsigned char minute; /* 0-59 */
    unsigned char second; /* 0-59 */
    unsigned long microsecond; /* 0-999999 */
} SQLDATETIME;
```

Die Struktur SQLDATETIME kann verwendet werden, um Felder vom Typ DATE, TIME und TIMESTAMP abzurufen (oder einen beliebigen Datentyp, der in einen dieser Datentypen konvertiert werden kann). Anwendungen haben häufig eigene Formate und einen eigenen Programmcode für Zeit und Datum. Das Abrufen von Daten in diese Struktur vereinfacht die Datenbearbeitung für Sie. Felder vom Typ DATE, TIME und TIMESTAMP können auch mit einem beliebigen Zeichendatentyp abgerufen und aktualisiert werden.

Wenn Sie eine SQLDATETIME-Struktur verwenden, um ein Datum, eine Zeit oder einen Zeitstempel in die Datenbank einzugeben, werden die Felder day_of_year und day_of_week ignoriert.

- **DT_VARIABLE** Mit NULL abgeschlossene Zeichenfolge. Die Zeichenfolge muss der Name einer SQL-Variablen sein, deren Wert vom Datenbankserver benutzt wird. Dieser Datentyp wird ausschließlich benutzt, um Daten an den Datenbankserver zu liefern. Er kann nicht benutzt werden, um Daten vom Datenbankserver abzurufen.

Die Strukturen sind in der Datei *sqlca.h* festgelegt. Die Datentypen VARCHAR, NVARCHAR, BINARY, DECIMAL und LONG sind für die Deklaration von Hostvariablen nicht geeignet, weil sie ein Ein-Zeichen-Array enthalten. Sie sind allerdings nützlich, um Variablen dynamisch zuzuweisen oder andere Variablen umzuwandeln ("Typecasting").

Datenbank-Datentypen DATE und TIME

Die Embedded SQL-Schnittstelle bietet keine Datentypen, die den verschiedenen Datenbank-Datentypen von DATE und TIME entsprechen. Diese Datenbank-Datentypen werden alle entweder mit der Struktur SQLDATETIME oder mit Zeichenfolgen abgerufen und aktualisiert.

Siehe auch

- „GET DATA-Anweisung [ESQL]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Senden und Abrufen langer Werte mit Embedded SQL“ auf Seite 522
- „date_format-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „date_order-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „time_format-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „timestamp_format-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]

Hostvariablen in Embedded SQL

Hostvariablen sind C-Variablen, die dem SQL-Präprozessor bekannt gemacht werden. Hostvariablen können benutzt werden, um Werte an den Datenbankserver zu übergeben oder um Werte vom Datenbankserver abzurufen.

Hostvariablen sind recht einfach zu verwenden, sie unterliegen jedoch einigen Einschränkungen. Dynamic SQL ist eine allgemeinere Art, um Informationen mit dem Datenbankserver auszutauschen, mithilfe einer Struktur namens SQL-Deskriptor-Bereich (SQL Descriptor Area - SQLDA). Der SQL-Präprozessor erzeugt für jede einzelne Anweisung, in der Hostvariablen verwendet werden, automatisch ein SQLDA.

Hostvariablen können nicht in Batches benutzt werden. Hostvariablen können nicht in einer Unterabfrage einer SET-Anweisung verwendet werden.

Siehe auch

- [„Static und Dynamic SQL“ auf Seite 501](#)

Deklaration von Hostvariablen

Hostvariablen werden definiert, indem sie in einen Deklarationsabschnitt eingefügt werden. Wie im ANSI Embedded SQL-Standard beschrieben, werden Hostvariablen definiert, indem die normalen Deklarationen von C-Variablen wie folgt eingeschlossen werden:

```
EXEC SQL BEGIN DECLARE SECTION;
/* C variable declarations */
EXEC SQL END DECLARE SECTION;
```

Diese Hostvariablen können dann anstelle von konstanten Werten in jeder SQL-Anweisung verwendet werden. Wenn der Datenbankserver die Anweisung ausführt, wird der Wert der Hostvariablen benutzt. Hostvariablen können nicht anstelle von Tabellen- oder Spaltennamen verwendet werden: Dafür ist Dynamic SQL erforderlich. Der Variablenname hat in einer SQL-Anweisung einen Doppelpunkt (:) als Präfix, um ihn von anderen Namen zu unterscheiden, die in der Anweisung erlaubt sind.

Im SQL-Präprozessor wird C-Sprachcode nur innerhalb von DECLARE SELECTION durchsucht. Daher sind innerhalb von DECLARE SECTION Initialisierungsroutinen auf Variablen zulässig, TYPEDEF-Typen und -Strukturen jedoch nicht.

Beispiel

Das folgende Beispiel zeigt den Gebrauch von Hostvariablen für eine INSERT-Anweisung. Variablen werden vom Programm belegt und dann in die Datenbank eingefügt:

```
EXEC SQL BEGIN DECLARE SECTION;
long employee_number;
char employee_name[50];
char employee_initials[8];
char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* program fills in variables with appropriate values
*/
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone );
```

Siehe auch

- [„Beispiel für statischen Cursor“ auf Seite 478](#)

C-Hostvariablentypen

Nur eine begrenzte Anzahl von C-Datentypen werden als Hostvariablen unterstützt. Andererseits haben bestimmte Hostvariablen keinen entsprechenden Datentyp in C.

In der Header-Datei *sqlca.h* festgelegte Makros können dazu verwendet werden, Hostvariablen des folgenden Typs zu definieren: NCHAR, VARCHAR, NVARCHAR, LONGVARCHAR, LONGNVARCHAR, BINARY, LONGBINARY, DECIMAL, DT_FIXCHAR, DT_NFIXCHAR, DATETIME (SQLDATETIME), BIT, BIGINT oder UNSIGNED BIGINT. Sie werden wie folgt benutzt:

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_NCHAR          v_nchar[10];
DECL_VARCHAR( 10 )  v_varchar;
DECL_NVARCHAR( 10 ) v_nvarchar;
DECL_LONGVARCHAR( 32768 ) v_longvarchar;
DECL_LONGNVARCHAR( 32768 ) v_longnvarchar;
DECL_BINARY( 4000 ) v_binary;
DECL_LONGBINARY( 128000 ) v_longbinary;
DECL_DECIMAL( 30, 6 ) v_decimal;
DECL_FIXCHAR( 10 )  v_fixchar;
DECL_NFIXCHAR( 10 ) v_nfixchar;
DECL_DATETIME       v_datetime;
DECL_BIT            v_bit;
DECL_BIGINT         v_bigint;
DECL_UNSIGNED_BIGINT v_ubigint;
EXEC SQL END DECLARE SECTION;
```

Der Präprozessor erkennt diese Makros innerhalb eines Embedded SQL-Deklarationsabschnitts und behandelt Variablen ihrem Typ entsprechend. Es wird empfohlen, den Typ DECIMAL (DT_DECIMAL, DECL_DECIMAL) nicht zu verwenden, da das Format von Dezimalzahlen herstellerspezifisch ist.

Die folgende Tabelle zeigt die C-Variablentypen, die als Hostvariablen erlaubt sind und die entsprechenden Datentypen der Embedded SQL-Schnittstelle.

C-Datentyp	Datentyp der Embedded SQL-Schnittstelle	Beschreibung
short si; short int si;	DT_SMALLINT	16-Bit-Ganzzahl mit Vorzeichen
unsigned short int usi;	DT_UNSSMALLINT	16-Bit Ganzzahl ohne Vorzeichen
long l; long int l;	DT_INT	32-Bit-Ganzzahl mit Vorzeichen
unsigned long int ul;	DT_UNSENT	32-Bit Ganzzahl ohne Vorzeichen
DECL_BIGINT ll;	DT_BIGINT	64-Bit-Ganzzahl mit Vorzeichen

C-Datentyp	Datentyp der Embedded SQL-Schnittstelle	Beschreibung
<code>DECL_UNSIGNED_BIGINT ull;</code>	DT_UNSBIGINT	64-Bit-Ganzzahl ohne Vorzeichen.
<code>float f;</code>	DT_FLOAT	einfachgenauer 4-Byte-Gleitkommazahl-Wert.
<code>double d;</code>	DT_DOUBLE	doppeltgenauer 8-Byte-Gleitkommazahl-Wert.
<code>char a[n]; /*n>=1*/</code>	DT_STRING	Mit NULL abgeschlossene Zeichenfolge im CHAR-Zeichensatz. Die Zeichenfolge wird mit Leerzeichen aufgefüllt, wenn bei der Initialisierung der Datenbank mit Leerzeichen aufgefüllte Zeichenfolgen verwendet werden. Diese Variable speichert n-1 Byte und das Nullabschlusszeichen.
<code>char *a;</code>	DT_STRING	Mit NULL abgeschlossene Zeichenfolge im CHAR-Zeichensatz. Diese Variable zeigt auf einen Bereich, der bis zu 32766 Byte plus das Nullabschlusszeichen speichern kann.
<code>DECL_NCHAR a[n]; /*n>=1*/</code>	DT_NSTRING	Mit NULL abgeschlossene Zeichenfolge im NCHAR-Zeichensatz. Die Zeichenfolge wird mit Leerzeichen aufgefüllt, wenn bei der Initialisierung der Datenbank mit Leerzeichen aufgefüllte Zeichenfolgen verwendet werden. Diese Variable speichert n-1 Byte plus das Nullabschlusszeichen.
<code>DECL_NCHAR *a;</code>	DT_NSTRING	Mit NULL abgeschlossene Zeichenfolge im NCHAR-Zeichensatz. Diese Variable zeigt auf einen Bereich, der bis zu 32766 Byte plus das Nullabschlusszeichen speichern kann.

C-Datentyp	Datentyp der Embedded SQL-Schnittstelle	Beschreibung
<code>DECL_VARCHAR(n) a;</code>	DT_VARCHAR	Zeichenfolge variabler Länge im CHAR-Zeichensatz mit einem 2-Byte-Längenfeld. Nicht mit NULL abgeschlossene oder mit Leerzeichen aufgefüllte Zeichenfolge. Der Maximalwert für n ist 32765 (Byte).
<code>DECL_NVARCHAR(n) a;</code>	DT_NVARCHAR	Zeichenfolge variabler Länge im NCHAR-Zeichensatz mit einem 2-Byte-Längenfeld. Nicht mit NULL abgeschlossene oder mit Leerzeichen aufgefüllte Zeichenfolge. Der Maximalwert für n ist 32765 (Byte).
<code>DECL_LONGVARCHAR(n) a;</code>	DT_LONGVARCHAR	Zeichenfolge variabler Länge im CHAR-Zeichensatz mit drei 4-Byte-Längenfeldern. Nicht mit NULL abgeschlossene oder mit Leerzeichen aufgefüllte Zeichenfolge.
<code>DECL_LONGNVARCHAR(n) a;</code>	DT_LONGNVARCHAR	Zeichenfolge variabler Länge im NCHAR-Zeichensatz mit drei 4-Byte-Längenfeldern. Nicht mit NULL abgeschlossene oder mit Leerzeichen aufgefüllte Zeichenfolge.
<code>DECL_BINARY(n) a;</code>	DT_BINARY	Zeichenfolge mit variabler Länge mit 2-Byte-Längenfeld. Der Maximalwert für n ist 32765 (Byte).
<code>DECL_LONGBINARY(n) a;</code>	DT_LONGBINARY	Lange Binärdaten von variabler Länge mit drei 4-Byte-Längenfeldern.

C-Datentyp	Datentyp der Embedded SQL-Schnittstelle	Beschreibung
<code>char a; /*n=1*/ DECL_FIXCHAR(n) a;</code>	DT_FIXCHAR	Zeichenfolge von fester Länge in CHAR-Zeichensatz. Mit Leerzeichen aufgefüllt, aber ohne Nullabschlusszeichen. Der Maximalwert für n ist 32767 (Byte).
<code>DECL_NCHAR a; /*n=1*/ DECL_NFIXCHAR(n) a;</code>	DT_NFIXCHAR	Zeichenfolge von fester Länge in NCHAR-Zeichensatz. Mit Leerzeichen aufgefüllt, aber ohne Nullabschlusszeichen. Der Maximalwert für n ist 32767 (Byte).
<code>DECL_DATETIME a;</code>	DT_TIME-STAMP_STRUCT	SQLDATETIME-Struktur

Zeichensätze

Für DT_FIXCHAR, DT_STRING, DT_VARCHAR und DT_LONGVARCHAR sind die Zeichendaten im CHAR-Zeichensatz der Anwendung. Dies ist gewöhnlich der Zeichensatz der Umgebungsvariablen der Anwendung. Eine Anwendung kann den CHAR-Zeichensatz mithilfe des Verbindungsparameters CHARSET ändern oder indem sie die Funktion db_change_char_charset aufruft.

Für DT_NFIXCHAR, DT_NSTRING, DT_NVARCHAR und DT_LONGNVARCHAR sind die Daten im NCHAR-Zeichensatz der Anwendung. Standardmäßig ist der NCHAR-Zeichensatz der Anwendung identisch mit dem CHAR-Zeichensatz. Eine Anwendung kann den NCHAR-Zeichensatz ändern, indem sie die Funktion db_change_nchar_charset aufruft.

Datenlänge

Unabhängig von den verwendeten CHAR- und NCHAR-Zeichensätzen werden alle Datenlängen in Byte angegeben.

Wenn zwischen dem Server und der Anwendung eine Zeichensatzkonvertierung durchgeführt wird, muss die Anwendung sicherstellen, dass die Puffer ausreichend groß sind, um die konvertierten Daten zu verarbeiten, und zusätzliche GET DATA-Anweisungen ausgeben, wenn Daten gekürzt werden.

Zeiger auf ein Zeichen (pointer to char)

Die Datenbankschnittstelle geht bei einer Hostvariable, die als **pointer to char** (`char *a`) deklariert ist, von einer Länge von 32767 Byte aus. Jede Hostvariable vom Typ "pointer to char", mit der Informationen aus der Datenbank abgerufen werden, muss auf einen Puffer zeigen, der groß genug ist, um jeden Wert enthalten zu können, der möglicherweise von der Datenbank zurückkommt.

Dies kann riskant sein, da jemand die Definition der Spalte in der Datenbank ändern könnte, sodass sie größer wird als zu dem Zeitpunkt, zu dem das Programm geschrieben wurde. Dies wiederum könnte zu

unvorhersehbaren Speicherbelegungen und daraus resultierenden Problemen führen. Es ist besser ein deklariertes Array zu verwenden, auch als Parameter einer Funktion, der als Zeiger auf Zeichen übergeben wird. Diese Vorgehensweise gestattet es der SQL-Anweisung, die Größe des Arrays zu erkennen.

Gültigkeitsbereich von Hostvariablen

Die Deklaration einer Hostvariablen kann im Allgemeinen überall dort stehen, wo auch eine C-Variablendeklaration stehen könnte. Das gilt auch für den Parameterdeklarationsbereich einer C-Funktion. Die dort deklarierten C-Variablen haben ihre normalen Aufgaben (sie stehen innerhalb des Blocks zur Verfügung, in dem sie definiert sind). Da aber der SQL-Präprozessor den C-Code nicht durchsucht, erkennt er keine C-Blöcke.

Für den SQL-Präprozessor sind Hostvariablen globale Variable für die Quelldatei. Zwei Hostvariablen können nicht den gleichen Namen haben.

Siehe auch

- „Sprachumgebungen“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Verbindungsparameter CharSet (CS)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „db_change_char_charset-Funktion“ auf Seite 538
- „db_change_nchar_charset-Funktion“ auf Seite 539
- „GET DATA-Anweisung [ESQL]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Hostvariablen verwenden

Hostvariablen können unter folgenden Umständen benutzt werden:

- In SELECT-, INSERT-, UPDATE- und DELETE-Anweisungen sind sie überall dort erlaubt, wo eine Zahlen- oder Zeichenfolgenkonstante erlaubt ist.
- In der INTO-Klausel einer SELECT oder FETCH-Anweisung.
- Hostvariablen können in Embedded SQL-Anweisungen auch anstelle eines Anweisungsnamens, eines Cursornamens oder eines Optionsnamens benutzt werden.
- Bei CONNECT-, DISCONNECT- und SET CONNECT-Anweisungen kann eine Hostvariable anstelle eines Servernamens, eines Datenbanknamens, eines Verbindungsnamens, einer Benutzer-ID, eines Kennworts oder einer Verbindungszeichenfolge verwendet werden.
- Bei SET OPTION und GET OPTION kann eine Hostvariable anstelle eines Optionswerts benutzt werden.

Hostvariablen können unter folgenden Umständen nicht benutzt werden:

- Hostvariablen können in Anweisungen nicht anstelle eines Tabellennamens oder eines Spaltennamens benutzt werden.
- Hostvariablen können nicht in Batches benutzt werden.

- Hostvariablen können nicht in einer Unterabfrage einer SET-Anweisung verwendet werden.

Hostvariablen SQLCODE und SQLSTATE

Der ISO/ANSI-Standard gestattet es einer Embedded SQL-Quelldatei, die folgenden speziellen Hostvariablen innerhalb eines Embedded SQL-Deklarationsabschnitts zu deklarieren:

```
long SQLCODE;  
char SQLSTATE[6];
```

Diese Variablen werden ggf. nach einer Embedded SQL-Anweisung gesetzt, die eine Datenbankankforderung enthält (EXEC SQL-Anweisungen, außer DECLARE SECTION, INCLUDE, WHENEVER SQLCODE usw.). Die Hostvariablen SQLCODE und SQLSTATE müssen daher im Bereich aller Embedded SQL-Anweisungen sichtbar sein, die Datenbankankforderungen generieren.

Folgendes ist gültiges Embedded SQL:

```
EXEC SQL INCLUDE SQLCA;  
// declare SQLCODE with global scope  
EXEC SQL BEGIN DECLARE SECTION;  
long SQLCODE;  
EXEC SQL END DECLARE SECTION;  
sub1() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    char SQLSTATE[6];  
    EXEC SQL END DECLARE SECTION;  
    exec SQL CREATE TABLE ...  
}  
sub2() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    char SQLSTATE[6];  
    EXEC SQL END DECLARE SECTION;  
    exec SQL DROP TABLE ...  
}
```

Der folgende Code ist kein gültiger Embedded SQL-Code, weil SQLSTATE im Bereich der Funktion sub2 nicht definiert wird:

```
EXEC SQL INCLUDE SQLCA;  
sub1() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    char SQLSTATE[6];  
    EXEC SQL END DECLARE SECTION;  
    exec SQL CREATE TABLE...  
}  
sub2() {  
    exec SQL DROP TABLE...  
}
```

Die Option -k des SQL-Präprozessors lässt zu, dass die SQLCODE-Variable außerhalb des Bereichs eines Embedded SQL-Deklarationsabschnitts deklariert wird.

Siehe auch

- „Der SQL-Präprozessor“ auf Seite 469

Indikatorvariablen

Indikatorvariablen sind C-Variablen, die zusätzliche Informationen liefern, wenn Sie Daten abrufen oder speichern. Indikatorvariablen werden in folgenden Fällen benutzt:

- **NULL** Damit eine Anwendung mit NULL umgehen kann.
- **Gekürzte Zeichenfolgen** Damit eine Anwendung Fälle behandeln kann, in denen ein abgerufener Wert gekürzt werden muss, um in eine Hostvariable zu passen.
- **Konvertierungsfehler** Für Fehlerinformationen

Eine Indikatorvariable ist eine Hostvariable vom Typ `a_sql_len`, die in einer SQL-Anweisung unmittelbar auf eine normale Hostvariable folgt. In der folgenden INSERT-Anweisung zum Beispiel ist `:ind_phone` eine Indikatorvariable:

```
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

Wenn eine Fetch- oder Execute-Anweisung keine Zeilen vom Server erhält (wie z.B. bei einem Fehler oder wenn das Ende der Ergebnismenge erreicht ist), bleiben die Indikatorwerte unverändert.

Hinweis

Um in Zukunft die Verwendung von 32- und 64-Bit-Längen und -Indikatoren zu ermöglichen, wird 'short int' für Embedded SQL-Indikatorvariablen nicht mehr unterstützt. Verwenden Sie stattdessen `a_sql_len`.

Indikatorvariablen: Der SQL NULL-Wert

Das SQL-Konzept von NULL darf nicht mit der C-Sprachkonstanten mit demselben Namen verwechselt werden. In der SQL-Sprache repräsentiert NULL entweder ein unbekanntes Attribut oder eine ungeeignete Information. Die C-Sprachkonstante steht für einen Zeigerwert, der nicht auf eine Speicherposition zeigt.

Wenn NULL in der Dokumentation von SQL Anywhere verwendet wird, ist die oben genannte Bedeutung der SQL-Datenbank gemeint. Die gleichnamige C-Konstante wird dagegen als Null-Zeiger bezeichnet (Kleinschreibung).

NULL ist nicht dasselbe wie ein beliebiger anderer Wert des für die Spalte festgelegten Typs. Daher wird ein zusätzliches Element neben den regulären Hostvariablen benötigt, um NULL-Werte an die Datenbank zu übergeben oder NULL-Ergebnisse zurückzuerhalten. Für diesen Zweck werden **Indikatorvariablen** eingesetzt.

Indikatorvariablen verwenden, um NULL einzufügen

Eine INSERT-Anweisung könnte wie folgt eine Indikatorvariable einfügen:

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
```

```
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/*
This program fills in the employee number,
name, initials, and phone number.
*/
if( /* Phone number is unknown */ ) {
    ind_phone = -1;
} else {
    ind_phone = 0;
}
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

Hat die Indikatorvariable den Wert -1, wird NULL eingefügt. Hat sie den Wert 0, wird der tatsächliche Wert von employee_phone eingefügt.

Indikatorvariablen verwenden, um NULL abzurufen

Indikatorvariablen werden auch benutzt, um Daten von der Datenbank abzurufen. Sie zeigen an, dass der Wert NULL abgerufen wurde (Indikator ist negativ). Falls NULL von der Datenbank abgerufen wird und keine Indikatorvariable zur Verfügung steht, wird ein Fehler erzeugt (SQLE_NO_INDICATOR).

Indikatorvariablen: Gekürzte Werte

Indikatorvariablen zeigen an, ob irgendwelche abgerufenen Werte gekürzt wurden, um in eine Hostvariable zu passen. So können Anwendungen mit dem Kürzen von Werten angemessen umgehen.

Falls ein Wert beim Abrufen gekürzt wurde, wird die Indikatorvariable mit einem positiven Wert belegt. Wenn die tatsächliche Länge des Datenbankwerts größer ist als 32767 Byte, enthält die Indikatorvariable den Wert 32767.

Indikatorvariablen: Konvertierungsfehler

Als Voreinstellung ist die Datenbankoption conversion_error auf On gesetzt, jede fehlgeschlagene Datentypkonvertierung führt zu einem Fehler und es wird keine Zeile zurückgegeben.

Sie können Indikatorvariablen verwenden, um festzustellen, in welcher Spalte eine fehlgeschlagene Datentypkonvertierung auftrat. Wenn Sie die Datenbankoption conversion_error auf Off setzen, erzeugt jede fehlgeschlagene Datentypkonvertierung (statt eines Fehlers) die Warnung CANNOT_CONVERT. Hat die Spalte, in der der Konvertierungsfehler auftrat, eine Indikatorvariable, wird diese Variable mit -2 belegt.

Wenn Sie beim Einfügen von Daten in die Datenbank die Datenbankoption conversion_error auf Off setzen, wird der Wert NULL eingefügt, falls eine fehlgeschlagene Konvertierung auftritt.

Zusammenfassung: Werte für Indikatorvariable

Folgende Tabelle bietet eine Zusammenfassung zum Gebrauch von Indikatorvariablen.

Indikatorwert	Wert an die Datenbank übergeben	Wert von der Datenbank erhalten
> 0	Wert der Hostvariablen	Der abgerufene Wert wurde gekürzt: tatsächliche Länge in der Indikatorvariablen.
0	Wert der Hostvariablen	Entweder war das Abrufen erfolgreich oder conversion_error ist auf "On" gesetzt.
-1	NULL	NULL-Ergebnis.
-2	NULL	Konvertierungsfehler (nur wenn conversion_error auf Off gesetzt ist). SQLCODE gibt die Warnung CAN_NOT_CONVERT aus.
< -2	NULL	NULL-Ergebnis.

Siehe auch

- „GET DATA-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

SQL-Kommunikationsbereich (SQLCA)

Der SQL-Kommunikationsbereich (**SQL Communication Area - SQLCA**) ist ein Speicherbereich, der bei jeder Datenbankanforderung benutzt wird, um Statistiken und Fehlermeldungen von der Anwendung zum Datenbankserver zu übermitteln und umgekehrt. Der SQLCA-Bereich wird als Handle für die Kommunikationsverbindung zwischen Anwendung und Datenbank benutzt. Er wird allen Datenbank-Bibliotheksfunktionen übergeben, die mit dem Datenbankserver kommunizieren müssen. Er wird implizit allen Embedded SQL-Anweisungen übergeben.

Eine globale SQLCA-Variable ist in der Schnittstellenbibliothek definiert. Der Präprozessor erzeugt eine externe Referenz auf die globale SQLCA-Variable und eine externe Referenz auf einen Zeiger auf die Variable. Die externe Referenz heißt `sqlca` und ist vom Typ `SQLCA`. Der Zeiger heißt `sqlcaptr`. Die aktuelle globale Variable wird in der Importbibliothek deklariert.

Der SQLCA-Bereich ist in der Header-Datei `sqlca.h` definiert, die im Unterverzeichnis `SDK\Include` Ihres Installationsverzeichnis enthalten ist.

SQLCA-Bereich bietet Fehlercodes

Sie referenzieren den SQLCA-Bereich, um einen bestimmten Fehlercode zu testen. Die Felder `sqlcode` und `sqlstate` enthalten Fehlercodes, falls eine Anforderung an die Datenbank einen Fehler hervorrief. Einige C-Makros sind definiert, um die Felder `sqlcode` und `sqlstate` und einige weitere Felder anzusprechen.

SQLCA-Felder

Die SQLCA-Felder haben folgende Bedeutung:

- **sqlcaid** Ein 8-Byte-Zeichenfeld, das die Zeichenfolge SQLCA zur Identifizierung der SQLCA-Struktur enthält. Dieses Feld unterstützt die Fehlersuche, wenn Sie Speicherinhalte untersuchen.
- **sqlcabc** Eine 32-Bit-Ganzzahl, die die Länge der SQLCA-Struktur enthält (136 Bytes).
- **sqlcode** Eine 32-Bit-Ganzzahl, die den Fehlercode angibt, wenn die Datenbank einen Fehler bei einer Anforderung feststellt. Definitionen für die Fehlercodes befinden sich in der Header-Datei *sqlerr.h*. Bei einer erfolgreichen Operation ist der Fehlercode 0 (Null), bei einer Warnung ist er positiv, bei einem Fehler ist er negativ.
- **sqlerrml** Die Länge der Daten im Feld sqlerrmc.
- **sqlerrmc** Kann eine oder mehrere Zeichenfolgen enthalten, die in eine Fehlermeldung einzufügen sind. Einige Fehlermeldungen enthalten eine oder mehr Platzhalterzeichenfolgen (%1, %2, ...), die durch die Zeichenfolgen in diesem Feld ersetzt werden.

Wenn zum Beispiel ein Fehler `Tabelle nicht gefunden` erzeugt wird, enthält `sqlerrmc` den Tabellennamen, der dann an passender Stelle in die Fehlermeldung eingefügt wird.

- **sqlerrp** Reserviert
- **sqlerrd** Ein Array von 32-Bit-Ganzzahlen.
- **sqlwarn** Reserviert
- **sqlstate** Der Statuswert SQLSTATE. Der ANSI SQL-Standard definiert diesen Rückgabewerttyp für SQL-Anweisungen zusätzlich zu dem SQLCODE-Wert. Der Wert von SQLSTATE ist immer eine fünf Zeichen lange mit NULL abgeschlossene Zeichenfolge, aufgeteilt in eine zwei Zeichen lange Klasse (die ersten zwei Zeichen) und eine drei Zeichen lange Unterklasse. Jedes Zeichen kann eine Ziffer von 0 bis 9 oder ein Großbuchstabe von A bis Z sein.

Jede Klasse oder Unterklasse, die mit 0 bis 4 oder A bis H beginnt, ist durch den SQL-Standard definiert. Andere Klassen und Unterklassen sind durch die Implementierung definiert. Hat SQLSTATE den Wert '00000', ist kein Fehler und keine Warnung aufgetreten.

sqlerror-Array

Das Array `sqlerror` besteht aus folgenden Elementen:

- **sqlerrd[1] (SQLIOCOUNT)** Die Anzahl der Eingabe-/Ausgabe-Vorgänge, die erforderlich waren, um eine Anweisung auszuführen.

Der Datenbankserver setzt diese Zählung nicht für jede Anweisung auf 0 (Null). Ihr Programm kann diese Variable vor der Ausführung einer Anweisungssequenz auf 0 (Null) setzen. Nach der letzten Anweisung zeigt der Wert der Variablen die Gesamtzahl der Eingabe-/Ausgabe-Vorgänge für die gesamte Anweisungssequenz an.

- **sqlerrd[2] (SQLCOUNT)** Der Wert dieses Felds hängt davon ab, welche Anweisung ausgeführt wurde.
 - **INSERT-, UPDATE-, PUT und DELETE-Anweisungen** Anzahl der Zeilen, die von der Anweisung betroffen waren.

- **OPEN- und RESUME-Anweisungen** Bei Cursor OPEN oder RESUME wird das Feld entweder mit der tatsächlichen Anzahl der Zeilen im Cursor (ein Wert größer als *oder gleich* 0) oder mit einer Schätzung der Anzahl (ein negative Zahl, deren absoluter Wert die Schätzung ist) belegt. Es wird sich um die tatsächliche Anzahl der Zeilen handeln, wenn der Datenbankserver sie errechnen kann, ohne die Zeilen zu zählen. Die Datenbank kann auch so konfiguriert werden, dass sie immer die tatsächliche Anzahl der Zeilen liefert (mit der Option `row_counts`).
- **FETCH Cursor-Anweisung** Das Feld SQLCOUNT wird belegt, falls eine SQLE_NOTFOUND-Warnung zurückgegeben wird. Es enthält die Anzahl der Zeilen, um die eine FETCH RELATIVE- oder eine FETCH ABSOLUTE-Anweisung den Bereich der möglichen Cursorpositionen überschritten hat (ein Cursor kann sich auf einer Zeile, vor der ersten Zeile oder nach der letzten Zeile befinden). Bei weiten Abrufen entspricht SQLCOUNT der Anzahl der tatsächlich abgerufenen Zeilen und ist kleiner oder gleich der Anzahl der angeforderten Zeilen. Während eines weiten Abrufs wird SQLE_NOTFOUND nur gesetzt, wenn keine Zeilen zurückgegeben werden.

Der Wert ist 0, falls die Zeile nicht gefunden wurde, die Position aber gültig ist, z.B. wenn FETCH RELATIVE 1 ausgeführt wird, und die Position die letzte Zeile eines Cursors ist. Der Wert ist positiv, falls das Abrufen jenseits des Cursors versucht wurde, und negativ, falls das Abrufen vor dem Anfang des Cursors versucht wurde.

- **GET DATA-Anweisung** Das Feld SQLCOUNT enthält die tatsächliche Länge des Werts.
- **DESCRIBE-Anweisung** Wenn die Klausel WITH VARIABLE RESULT verwendet wird, um Prozeduren zu beschreiben, die mehr als ein Ergebnis haben können, wird SQLCOUNT auf einen der folgenden Werte gesetzt:
 - **0** Die Ergebnismenge kann sich ändern: Der Prozeduraufruf sollte nach jeder OPEN-Anweisung erneut beschrieben werden.
 - **1** Die Ergebnismenge ist unveränderlich. Eine erneute Beschreibung ist nicht erforderlich.

Beim Syntaxfehler SQLE_SYNTAX_ERROR enthält dieses Feld die ungefähre Zeichenposition innerhalb der Anweisung, in der der Fehler erkannt wurde.

- **sqlerrd[3] (SQLIOESTIMATE)** Die geschätzte Anzahl der für den Abschluss der Anweisung erforderlichen Eingabe-/Ausgabe-Vorgänge. Dieses Feld wird bei einer OPEN- oder EXPLAIN-Anweisung belegt.

Siehe auch

- „SQL Anywhere - Fehlermeldungen“ [[Fehlermeldungen](#)]
- „SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE“ [[Fehlermeldungen](#)]
- „SQL Anywhere-Fehlermeldungen - sortiert nach SQLSTATE“ [[Fehlermeldungen](#)]
- „Mehrzeilige Abrufe oder Array-Abrufe“ auf Seite 518

SQLCA-Verwaltung für Code mit mehreren Threads oder "reentrant"-Code

Sie können Embedded SQL-Anweisungen in Code mit mehreren Threads oder in reentrant-Code verwenden. Wenn Sie allerdings eine einfache Verbindung verwenden, sind Sie auf eine aktive Anforderung pro Verbindung beschränkt. Vermeiden Sie bei einer Anwendung mit mehreren Threads, für alle Threads dieselbe Verbindung zur Datenbank zu verwenden, ausgenommen, Sie verwenden Semaphore für die Zugriffssteuerung.

Sie können ohne Einschränkung für jeden Thread, der die Datenbank nutzen will, eine separate Verbindung öffnen. Der SQLCA-Bereich wird von der Laufzeitbibliothek benutzt, um zwischen den verschiedenen Thread-Kontexten zu unterscheiden. Daher braucht jeder Thread, der eine Verbindung zur Datenbank benötigt, seinen eigenen SQLCA-Bereich. Die Ausnahme ist, dass ein Thread die Funktion `db_cancel_request` verwenden kann, um eine Anweisung abubrechen, die unter Verwendung des SQLCA dieses Threads auf einem anderen Thread ausgeführt wird.

Das folgende Beispiel enthält Embedded SQL-Reentrant-Code mit mehreren Threads.

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <process.h>
#include <windows.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

#define TRUE 1
#define FALSE 0

// multithreading support

typedef struct a_thread_data {
    SQLCA sqlca;
    int num_iters;
    int thread;
    int done;
} a_thread_data;

// each thread's ESQL test

EXEC SQL SET SQLCA "&thread_data->sqlca";

static void PrintSQLError( a_thread_data * thread_data )
/*****
{
    char                buffer[200];

    printf( "%d: SQL error %d -- %s ... aborting\n",
        thread_data->thread,
        SQLCODE,
        sqlerror_message( &thread_data->sqlca,
            buffer, sizeof( buffer ) ) );
    exit( 1 );
}

EXEC SQL WHENEVER SQLERROR { PrintSQLError( thread_data ); };
```

```

static void do_one_iter( void * data )
{
    a_thread_data * thread_data = (a_thread_data *)data;
    int i;
    EXEC SQL BEGIN DECLARE SECTION;
    char user[ 20 ];
    EXEC SQL END DECLARE SECTION;

    if( db_init( &thread_data->sqlca ) != 0 ) {
        for( i = 0; i < thread_data->num_iters; i++ ) {
            EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";
            EXEC SQL SELECT USER INTO :user;
            EXEC SQL DISCONNECT;
        }
        printf( "Thread %d did %d iters successfully\n",
            thread_data->thread, thread_data->num_iters );
        db_fini( &thread_data->sqlca );
    }
    thread_data->done = TRUE;
}

int main()
{
    int num_threads = 4;
    int thread;
    int num_iters = 300;
    int num_done = 0;
    a_thread_data *thread_data;
    thread_data = (a_thread_data *)malloc( sizeof(a_thread_data) *
num_threads );
    for( thread = 0; thread < num_threads; thread++ ) {
        thread_data[ thread ].num_iters = num_iters;
        thread_data[ thread ].thread = thread;
        thread_data[ thread ].done = FALSE;
        if( _beginthread( do_one_iter,
            8096,
            (void *)&thread_data[thread] ) <= 0 ) {
            printf( "FAILED creating thread.\n" );
            return( 1 );
        }
    }
    while( num_done != num_threads ) {
        Sleep( 1000 );
        num_done = 0;
        for( thread = 0; thread < num_threads; thread++ ) {
            if( thread_data[ thread ].done == TRUE ) {
                num_done++;
            }
        }
    }
    return( 0 );
}

```

Siehe auch

- „Anforderungsverwaltung mit Embedded SQL“ auf Seite 531

Mehrere SQLCAs

Sie dürfen die Option (-r-) des SQL-Präprozessors nicht verwenden, die einen Code ohne Wiedereinstieg (Non-Reentrant) erstellt. Der Reentrant-Code ist etwas umfangreicher und etwas langsamer, weil keine statisch initialisierten globalen Variablen benutzt werden können. Diese Auswirkungen sind allerdings minimal.

Jeder SQLCA-Bereich, der in Ihrem Programm benutzt wird, muss mit einem Aufruf von `db_init` initialisiert und am Ende mit einem Aufruf von `db_fini` wieder freigegeben werden.

Die Embedded SQL-Anweisung `SET SQLCA` wird verwendet, um den SQL-Präprozessor anzuweisen, einen anderen SQLCA-Bereich für Datenbankankorderungen zu verwenden. Ein Beispiel ist die folgende Anweisung: `EXEC SQL SET SQLCA 'task_data->sqlca' ;` wird zu Beginn eines Programms oder in einer Header-Datei verwendet, um SQLCA-Referenzen so einzustellen, dass sie auf Task-spezifische Daten zeigen. Die Performance ist nicht betroffen, weil diese Anweisung keinen Code generiert. Sie ändert den Status innerhalb des Präprozessors, sodass jede Referenz auf den SQLCA-Bereich die angegebene Zeichenfolge benutzt.

Jeder Thread braucht seinen eigenen SQLCA-Bereich. Diese Einschränkung gilt auch für Code in einer Shared Library (einer DLL beispielsweise), die Embedded SQL benutzt und in Ihrer Anwendung von mehr als einem Thread aufgerufen wird.

Sie können die Unterstützung mehrfacher SQLCAs in jeder der unterstützten Embedded SQL-Umgebungen verwenden, sie ist jedoch nur für reentrant-Code erforderlich.

Sie müssen nicht mehrere SQLCA-Bereiche verwenden, um Verbindungen zu mehr als einer Datenbank herzustellen oder um mehr als eine Verbindung zu einer einzelnen Datenbank zu halten.

Jeder SQLCA-Bereich kann über eine nicht benannte Verbindung verfügen. Jeder SQLCA-Bereich hat eine aktive oder aktuelle Verbindung.

Alle Vorgänge über eine Datenbankverbindung müssen denselben SQLCA-Bereich verwenden, der beim Einrichten der Verbindung benutzt wurde.

Hinweis

Vorgänge, die sich auf verschiedene Verbindungen beziehen, sind den normalen Datensatz-Sperrmechanismen unterworfen und können sich gegenseitig blockieren oder möglicherweise zu einer Deadlock-Situation führen.

Siehe auch

- „[SET SQLCA-Anweisung \[ESQL\]](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „[SET CONNECTION-Anweisung \[Interactive SQL\] \[ESQL\]](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Static und Dynamic SQL

Es gibt zwei Möglichkeiten, SQL-Anweisungen in ein C-Programm einzubetten:

- Statische Anweisungen
- Dynamische Anweisungen

Static SQL-Anweisungen

Alle Standard-SQL-Anweisungen zur Datenmanipulation und Datendefinition können in ein C-Programm eingebettet werden, indem man sie mit dem Präfix **EXEC SQL** versieht und die Anweisung mit einem Semikolon (;) abschließt. Diese Anweisungen werden als **statische** Anweisungen bezeichnet.

Statische Anweisungen können Referenzen auf Hostvariablen enthalten. Hostvariablen können nur anstelle von Zeichenfolgen- und numerischen Konstanten benutzt werden. Sie können nicht verwendet werden, um Spalten- oder Tabellennamen zu ersetzen; für diese Vorgänge sind dynamische Anweisungen erforderlich.

Siehe auch

- „Hostvariablen in Embedded SQL“ auf Seite 485

Dynamic SQL-Anweisungen

In C werden Zeichenfolgen in Zeichen-Arrays gespeichert. Dynamische Anweisungen werden in C-Zeichenfolgen konstruiert. Diese dynamischen Anweisungen können dann mit der PREPARE-Anweisung und der EXECUTE-Anweisung ausgeführt werden. Solche SQL-Anweisungen können Hostvariablen nicht in der gleichen Weise ansprechen wie statische Anweisungen, denn auf C-Variablen kann nicht über den Namen zugegriffen werden, solange das C-Programm ausgeführt wird.

Um Informationen zwischen dynamischen SQL-Anweisungen und C-Variablen auszutauschen, wird eine Datenstruktur namens SQL-Deskriptorbereich (**SQL Descriptor Area - SQLDA**) verwendet. Diese Struktur wird vom SQL-Präprozessor erzeugt, wenn Sie bei der EXECUTE-Anweisung in der USING-Klausel eine Liste von Hostvariablen spezifizieren. Jeder Variablen entspricht jeweils ein Platzhalter in der vorbereiteten Anweisung, zugeordnet über die Position.

Ein **Platzhalter** wird in die Anweisung eingefügt, um anzuzeigen, wo auf Hostvariablen zugegriffen werden soll. Ein Platzhalter ist entweder ein Fragezeichen (?) oder eine Referenz auf eine Hostvariable wie in statischen Anweisungen (ein Hostvariablenname mit einem vorangestellten Doppelpunkt). Im letzteren Fall dient der Name der Hostvariable, der im Text der Anweisung benutzt wird, nur als Platzhalter für eine Referenz auf den SQL-Deskriptorbereich.

Eine Hostvariable, mit der Informationen an die Datenbank übergeben werden, wird eine **Bindevariable** genannt.

Beispiel

```
EXEC SQL BEGIN DECLARE SECTION;
char      comm[200];
```

```
char      street[30];
char      city[20];
a_sql_len cityind;
long      empnum;
EXEC SQL END DECLARE SECTION;

...
sprintf( comm,
        "UPDATE %s SET Street = :?, City = :?"
        "WHERE EmployeeID = :?",
        tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
EXEC SQL EXECUTE S1 USING :street, :city:cityind, :empnum;
```

Wenn Sie diese Methode verwenden möchten, müssen Sie wissen, wieviele Hostvariablen in der Anweisung vorkommen. Normalerweise ist das nicht der Fall. Daher können Sie Ihre eigene SQLDA-Struktur aufsetzen und diesen SQLDA-Bereich in der USING-Klausel der EXECUTE-Anweisung angeben.

Die Anweisung DESCRIBE BIND VARIABLES gibt die Namen der Hostvariablen zu den Bindevariablen zurück, die in einem Prepared-Statement gefunden wurden. Dies erleichtert es dem C-Programm, die Hostvariablen zu verwalten. Im Folgenden finden Sie die allgemeine Methode:

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
...
sprintf( comm,
        "UPDATE %s SET Street = :street, City = :city"
        " WHERE EmployeeID = :empnum",
        tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
/* Assume that there are no more than 10 host variables.
 * See next example if you cannot put a limit on it. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE BIND VARIABLES FOR S1 INTO sqlda;
/* sqlda->sqld will tell you how many
 * host variables there were. */
/* Fill in SQLDA_VARIABLE fields with
 * values based on name fields in sqlda. */
...
EXEC SQL EXECUTE S1 USING DESCRIPTOR sqlda;
free_sqlda( sqlda );
```

SQLDA-Inhalt

Der SQLDA-Bereich besteht aus einem Array von Variablendeskriptoren. Jeder Deskriptor beschreibt die Attribute der entsprechenden Variablen des C-Programms, oder die Stelle, an der die Datenbank Daten speichert oder von der sie Daten abrufen:

- Datentyp
- Länge, wenn *Typ* ein Zeichenfolgentyp ist
- Speicheradresse
- Indikatorvariable

Indikatorvariablen und NULL

Die Indikatorvariable wird verwendet, um NULL an die Datenbank zu übergeben, oder um NULL von der Datenbank abzurufen. Der Datenbankserver verwendet die Indikatorvariable auch, um anzuzeigen,

unter welchen Bedingungen Werte während einer Datenbankoperation gekürzt wurden. Die Indikatorvariable wird mit einem positiven Wert besetzt, wenn nicht genug Platz zur Verfügung stand, um einen Wert von der Datenbank zu erhalten.

Siehe auch

- „Der SQL-Deskriptor-Bereich (SQLDA)“ auf Seite 504
- „EXECUTE-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Indikatorvariablen“ auf Seite 493

Die dynamische SELECT-Anweisung

Eine SELECT-Anweisung, die nur eine einzige Zeile zurückgibt, kann dynamisch vorbereitet werden, gefolgt von einer EXECUTE-Anweisung mit einer INTO-Klausel, um das einzeilige Ergebnis abzurufen. SELECT-Anweisungen, die mehrere Zeilen zurückgeben, werden dagegen mithilfe von dynamischen Cursor verwaltet.

Mit dynamischen Cursor werden Ergebnisse in eine Hostvariablen-Liste geschrieben, oder in einen SQLDA-Bereich, der mit der FETCH-Anweisung angegeben wird (FETCH INTO und FETCH USING DESCRIPTOR). Da die Anzahl der Elemente in der Auswahlliste normalerweise unbekannt ist, wird in der Regel der SQLDA-Bereich benutzt. Die Anweisung DESCRIBE SELECT LIST richtet einen SQLDA-Bereich ein mit Typen für die Elemente der Auswahlliste. Der Platzbedarf für die Werte wird dann mit der Funktion fill_sqlda oder fill_s_sqlda zugewiesen, und die Informationen werden mit der Anweisung FETCH USING DESCRIPTOR abgerufen.

Ein typisches Szenario sieht wie folgt aus:

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
int actual_size;
SQLDA * sqlda;
...
sprintf( comm, "SELECT * FROM %s", table_name );
EXEC SQL PREPARE S1 FROM :comm;
/* Initial guess of 10 columns in result.
   If it is wrong, it is corrected right
   after the first DESCRIBE by reallocating
   sqlda and doing DESCRIBE again. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE SELECT LIST FOR S1
      INTO sqlda;
if( sqlda->sqld > sqlda->sqln )
{
    actual_size = sqlda->sqld;
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( actual_size );
    EXEC SQL DESCRIBE SELECT LIST FOR S1
          INTO sqlda;
}
fill_sqlda( sqlda );
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
EXEC SQL WHENEVER NOTFOUND {break};
for( ;; )
{
```

```
EXEC SQL FETCH C1 USING DESCRIPTOR sqlda;  
/* do something with data */  
}  
EXEC SQL CLOSE C1;  
EXEC SQL DROP STATEMENT S1;
```

Hinweis

So stellen Sie sicher, dass Anweisungen beendet werden, damit nicht unnötigerweise Ressourcen gebunden bleiben.

Das Beispiel für den dynamischen Cursor zeigt den Einsatz von Cursor für eine dynamische SELECT-Anweisung.

Siehe auch

- „PREPARE-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „DESCRIBE-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Beispiel für dynamischen Cursor“ auf Seite 480
- „Referenz der Bibliotheksfunktionen“ auf Seite 532

Der SQLA-Deskriptor-Bereich (SQLDA)

Der SQLDA-Bereich (SQL Descriptor Area) ist eine Schnittstellenstruktur, die für Dynamic SQL-Anweisungen verwendet wird. Die Struktur wird verwendet, um Informationen über Hostvariablen und Ergebnisse der SELECT-Anweisung mit der Datenbank auszutauschen. Der SQLDA-Bereich ist in der Header-Datei *sqlca.h* definiert.

In der gemeinsamen Schnittstellenbibliothek oder DLL für die Datenbank-Schnittstelle gibt es Funktionen, die Sie zur Verwaltung der SQLDA-Bereiche verwenden können.

Wenn Hostvariablen mit statischen SQL-Anweisungen verwendet werden, erzeugt der Präprozessor einen SQLDA-Bereich für diese Hostvariable. Was dann mit der Datenbank ausgetauscht wird, ist tatsächlich dieser SQLDA-Bereich.

Siehe auch

- „Referenz der Bibliotheksfunktionen“ auf Seite 532

Die SQLDA-Header-Datei

Der Inhalt von *sqlca.h* ist der folgende:

```
#ifndef _SQLDA_H_INCLUDED  
#define _SQLDA_H_INCLUDED  
#define II_SQLDA  
#include "sqlca.h"  
#if defined( _SQL_PACK_STRUCTURES )  
    #if defined( _MSC_VER ) && _MSC_VER > 800  
        #pragma warning(push)  
        #pragma warning(disable:4103)  
    #endif  
#endif
```

```

#include "pshpk1.h"
#endif
#define SQL_MAX_NAME_LEN    30
#define _sqldafar
typedef short int a_sql_type;

struct sqlname {
    short int    length; /* length of char data */
    char        data[ SQL_MAX_NAME_LEN ]; /* data */
};

struct sqlvar {
    /* array of variable descriptors */
    short int    sqltype; /* type of host variable */
    a_sql_len    sqllen; /* length of host variable */
    void         *sqldata; /* address of variable */
    a_sql_len    *sqlind; /* indicator variable pointer */
    struct sqlname sqlname;
};

#if defined( _SQL_PACK_STRUCTURES )
#include "poppk.h"
/* The SQLDA should be 4-byte aligned */
#include "pshpk4.h"
#endif

struct sqlda {
    unsigned char    sqldaaid[8]; /* eye catcher "SQLDA" */
    a_sql_int32      sqldabc; /* length of sqlda structure */
    short int        sqln; /* descriptor size in number of entries */
    short int        sqld; /* number of variables found by DESCRIBE */
    struct sqlvar     sqlvar[1]; /* array of variable descriptors */
};

#define SCALE(sqllen)      ((sqllen)/256)
#define PRECISION(sqllen) ((sqllen)&0xff)
#define SET_PRECISION_SCALE(sqllen,precision,scale) \
    sqllen = (scale)*256 + (precision)
#define DECIMALSTORAGE(sqllen) (PRECISION(sqllen)/2 + 1)
typedef struct sqlda    SQLDA;
typedef struct sqlvar    SQLVAR, SQLDA_VARIABLE;
typedef struct sqlname  SQLNAME, SQLDA_NAME;
#ifndef SQLDASIZE
#define SQLDASIZE(n) ( sizeof( struct sqlda ) + \
    (n-1) * sizeof( struct sqlvar ) )
#endif
#if defined( _SQL_PACK_STRUCTURES )
#include "poppk.h"
#if defined( _MSC_VER ) && _MSC_VER > 800
#pragma warning(pop)
#endif
#endif
#endif

```

SQLDA-Felder

Die SQLDA-Felder haben folgende Bedeutung:

Feld	Beschreibung
sqldaid	Ein 8-Byte-Zeichenfeld, das die Zeichenfolge SQLDA zur Identifizierung der SQLDA-Struktur enthält. Dieses Feld unterstützt die Fehlersuche, wenn Sie Speicherinhalte untersuchen.
sqldabc	Eine 32-Bit-Ganzzahl, die die Länge der SQLDA-Struktur enthält
sqln	Die Anzahl der im Array sqlvar zugeordneten Variablendeskriptoren
sqld	Die Anzahl der gültigen Variablendeskriptoren (die eine Hostvariable beschreiben). Dieses Feld wird von der DESCRIBE-Anweisung festgelegt. Sie können es auch festlegen, wenn Daten an den Datenbankserver geliefert werden.
sqlvar	Ein Array, bestehend aus Deskriptoren vom Typ struct sqlvar, von denen jeder eine Hostvariable beschreibt

SQLDA-Hostvariablen-Beschreibungen

Jede sqlvar-Struktur im SQLDA-Bereich beschreibt eine Hostvariable. Die Felder der sqlvar-Struktur haben folgende Bedeutung:

- **sqltype** Der Typ der Variablen, die von diesem Deskriptor beschrieben wird.

Das "low order Bit" zeigt an, ob NULL erlaubt ist. Gültige Typen- und Konstantendefinitionen befinden sich in der Header-Datei *sqldef.h*.

Dieses Feld wird von der DESCRIBE-Anweisung belegt. Sie können dieses Feld mit jedem Typ belegen, wenn Sie Daten an den Datenbankserver liefern oder Daten vom Datenbankserver abrufen. Erforderliche Typkonvertierungen erfolgen automatisch.

- **sqln** Länge der Variable. Ein sqln-Wert hat den Typ a_sql_len. Was der Wert dieser Variable tatsächlich bedeutet, hängt von den Informationen zum Typ ab und davon, wie der SQLDA-Bereich verwendet wird.

Bei LONG VARCHAR-, LONG NVARCHAR- und LONG BINARY-Datentypen wird das Feld array_len der Datentypstruktur DT_LONGVARCHAR, DT_LONGNVARCHAR oder DT_LONGBINARY anstelle des Felds sqln verwendet.

- **sqldata** Ein Zeiger auf den Speicher, der von der Variablen belegt wird. Dieser Speicherinhalt muss typ- und längenmäßig den Angaben in den Feldern sqltype und sqln entsprechen.

Bei UPDATE- und INSERT-Anweisungen spielt diese Variable keine Rolle für den Vorgang, falls der sqldata-Zeiger ein Null-Zeiger ist. Bei FETCH werden keine Daten zurückgegeben, falls der sqldata-Zeiger ein Null-Zeiger ist. In anderen Worten, die vom Zeiger sqldata zurückgegebene Spalte ist eine **ungebundene Spalte**.

Falls die DESCRIBE-Anweisung LONG NAMES benutzt, enthält dieses Feld den Langnamen der Ergebnismengen-Spalte. Falls die DESCRIBE-Anweisung darüber hinaus eine DESCRIBE USER

TYPES-Anweisung ist, dann enthält dieses Feld den Langnamen des benutzerdefinierten Datentyps statt der Spalte. Ist der Typ ein vordefinierter Datentyp, ist das Feld leer.

- **sqlind** Ein Zeiger auf den Indikatorwert. Ein Indikatorwert hat den Typ `a_sql_len`. Ein negativer Indikatorwert zeigt NULL an. Ein positiver Indikatorwert bedeutet, dass diese Variable von einer FETCH-Anweisung gekürzt wurde. Der Indikatorwert enthält die Länge der Daten, bevor sie gekürzt wurden. Der Wert -2 zeigt einen Konvertierungsfehler an, wenn die Datenbankoption `conversion_error` auf Off gesetzt ist.

Ist der `sqlind`-Zeiger ein Null-Zeiger, gehört keine Indikatorvariable zu dieser Hostvariable.

Das `sqlind`-Feld wird auch in der DESCRIBE-Anweisung verwendet, um Parametertypen anzuzeigen. Ist der Datentyp benutzerdefiniert, wird das Feld auf `DT_HAS_USERTYPE_INFO` gesetzt. In diesem Fall sollten Sie DESCRIBE USER TYPES ausführen, um Informationen über den benutzerdefinierten Datentyp zu erhalten.

- **sqlname** Eine VARCHAR-ähnliche Struktur wie die Folgende:

```
struct sqlname {  
    short int  length;  
    char  data[ SQL_MAX_NAME_LEN ];  
};
```

Sie wird durch eine DESCRIBE-Anweisung gefüllt und wird nur dafür benutzt. Dieses Feld hat verschiedene Bedeutungen für zwei Formate der DESCRIBE-Anweisung:

- **SELECT LIST** Der Namensdatenpuffer ist mit dem Spaltentitel des entsprechenden Elements in der ausgewählten Liste belegt.
- **BIND VARIABLES** Der Namensdatenpuffer ist mit dem Namen der Hostvariablen belegt, die als Bindevariable benutzt wurde, oder mit "?", falls eine namenlose Parametermarkierung verwendet wurde.

Handelt es sich um eine DESCRIBE SELECT LIST-Anweisung, sind alle vorhandenen Indikatorvariablen mit einer Markierung (flag) belegt, die anzeigt, ob das Listenelement aktualisierbar ist. Weitere Hinweise über diese Markierung finden Sie in der Header-Datei *sqldef.h*.

Handelt es sich um eine DESCRIBE USER TYPES-Anweisung, dann enthält dieses Feld den Langnamen des benutzerdefinierten Datentyps anstelle der Spalte. Ist der Typ ein vordefinierter Datentyp, ist das Feld leer.

Siehe auch

- „Datentypen in Embedded SQL“ auf Seite 482
- „SQLDA sqlen-Feldwerte“ auf Seite 508
- „conversion_error-Option“ [*SQL Anywhere Server - Datenbankadministration*]
- „Indikatorvariablen“ auf Seite 493

SQLDA sqllen-Feldwerte

SQLDA sqllen-Feldwerte nach einem DESCRIBE

Mit der DESCRIBE-Anweisung erhalten Sie Informationen zu den Hostvariablen, die zum Speichern von Daten erforderlich sind, die von der Datenbank abgerufen wurden, oder zu Hostvariablen, die zum Übergeben von Daten an die Datenbank erforderlich sind.

Die folgende Tabelle zeigt die Werte der Strukturelemente sqllen und sqltype, die die DESCRIBE-Anweisung für verschiedene Datenbanktypen zurückgibt (sowohl SELECT LIST als auch BIND VARIABLE DESCRIBE-Anweisungen). Bei einem benutzerdefinierten Datenbank-Datentyp wird der zu Grunde liegende Datentyp beschrieben.

Ihr Programm kann entweder die Typen und Längen verwenden, die DESCRIBE zurückgibt, oder es kann andere Typen verwenden. Der Datenbankserver wird jede erforderliche Typkonvertierung durchführen. Der Speicherbereich, auf den das sqldata-Feld zeigt, muss mit den Feldern sqltype und sqllen übereinstimmen. Der Embedded SQL-Typ wird durch eine bitweise AND-Verknüpfung von sqltype und DT_TYPES (sqltype & DT_TYPES) erhalten.

Datenbank-Feldtyp	Zurückgegebener Embedded SQL-Typ	Von DESCRIBE zurückgegebene Länge (in Byte)
BIGINT	DT_BIGINT	8
BINARY(n)	DT_BINARY	n
BIT	DT_BIT	1
CHAR(n)	DT_FIXCHAR ¹	n-mal die maximale Datenerweiterung bei der Konvertierung vom Datenbankzeichensatz in den CHAR-Zeichensatz des Clients. Wenn diese Länge mehr als 32767 Byte beträgt, dann wird der Embedded SQL-Typ DT_LONGVARCHAR mit einer Länge von 32767 Byte zurückgegeben.
CHAR(n CHAR)	DT_FIXCHAR ¹	n-mal die maximale Zeichenlänge im CHAR-Zeichensatz des Clients. Wenn diese Länge mehr als 32767 Byte beträgt, dann wird der Embedded SQL-Typ DT_LONGVARCHAR mit einer Länge von 32767 Byte zurückgegeben.

Datenbank-Feldtyp	Zurückgegebener Embedded SQL-Typ	Von DESCRIBE zurückgegebene Länge (in Byte)
DATE	DT_DATE	Die Länge der längsten formatierten Zeichenfolge
DECIMAL(p,s)	DT_DECIMAL	Das low byte des Längenfelds im SQLDA-Bereich auf p setzen, das high byte auf s. Siehe PRECISION- und SCALE-Makros in sqlda.h.
DOUBLE	DT_DOUBLE	8
FLOAT	DT_FLOAT	4
INT	DT_INT	4
LONG BINARY	DT_LONGBINARY	32767
LONG NVARCHAR	DT_LONGVARCHAR / DT_LONGNVARCHAR ²	32767
LONG VARCHAR	DT_LONGVARCHAR	32767
NCHAR(n)	DT_FIXCHAR / DT_NFIX-CHAR ²	n-mal die maximale Zeichenlänge im NCHAR-Zeichensatz des Clients. Wenn diese Länge mehr als 32767 Byte beträgt, dann wird der Embedded SQL-Typ DT_LONGNVARCHAR mit einer Länge von 32767 Byte zurückgegeben.
NVARCHAR(n)	DT_VARCHAR / DT_NVARCHAR ²	n-mal die maximale Zeichenlänge im NCHAR-Zeichensatz des Clients. Wenn diese Länge mehr als 32767 Byte beträgt, dann wird der Embedded SQL-Typ DT_LONGNVARCHAR mit einer Länge von 32767 Byte zurückgegeben.
REAL	DT_FLOAT	4
SMALLINT	DT_SMALLINT	2

Datenbank-Feldtyp	Zurückgegebener Embedded SQL-Typ	Von DESCRIBE zurückgegebene Länge (in Byte)
TIME	DT_TIME	Die Länge der längsten formatierten Zeichenfolge
TIMESTAMP	DT_TIMESTAMP	Die Länge der längsten formatierten Zeichenfolge
TINYINT	DT_TINYINT	1
UNSIGNED BIGINT	DT_UNSBIGINT	8
UNSIGNED INT	DT_UNSENT	4
UNSIGNED SMALLINT	DT_UNSSMALLINT	2
VARCHAR(n)	DT_VARCHAR ¹	n-mal die maximale Datenerweiterung bei der Konvertierung vom Datenbankzeichensatz in den CHAR-Zeichensatz des Clients. Wenn diese Länge mehr als 32767 Byte beträgt, dann wird der Embedded SQL-Typ DT_LONGVARCHAR mit einer Länge von 32767 Byte zurückgegeben.
VARCHAR(n CHAR)	DT_VARCHAR ¹	n-mal die maximale Zeichenlänge im CHAR-Zeichensatz des Clients. Wenn diese Länge mehr als 32767 beträgt, dann wird der Embedded SQL-Typ DT_LONGVARCHAR mit der Länge 32767 zurückgegeben.

¹ Der zurückgegebene Typ für CHAR und VARCHAR kann DT_LONGVARCHAR sein, wenn die maximale Bytelänge im CHAR-Zeichensatz des Clients größer als 32767 Byte ist.

² Der für NCHAR und NVARCHAR zurückgegebene Typ kann DT_LONGNVARCHAR sein, wenn die maximale Bytelänge im NCHAR-Zeichensatz des Clients größer als 32767 Byte ist. NCHAR, NVARCHAR und LONG NVARCHAR werden standardmäßig als DT_FIXCHAR, DT_VARCHAR oder DT_LONGVARCHAR beschrieben. Wenn die Funktion db_change_nchar_charset aufgerufen wurde, werden die Typen als DT_NFIXCHAR, DT_NVARCHAR bzw. DT_LONGNVARCHAR beschrieben.

Siehe auch

- „Datentypen in Embedded SQL“ auf Seite 482
- „CHAR-Datentyp“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NCHAR-Datentyp“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NVARCHAR-Datentyp“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „VARCHAR-Datentyp“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „db_change_nchar_charset-Funktion“ auf Seite 539

SQLDA sqllen-Feldwerte beim Senden von Werten

Die folgende Tabelle zeigt, wie Sie die Länge eines Werts spezifizieren, wenn Sie im SQLDA-Bereich Daten an den Datenbankserver liefern.

In diesem Fall sind nur die in der Tabelle gezeigten Datentypen erlaubt. Die Datentypen DT_DATE, DT_TIME und DT_TIMESTAMP werden genauso behandelt wie DT_STRING, wenn Informationen an die Datenbank geliefert werden. Der Wert muss eine mit NULL abgeschlossene Zeichenfolge in einem passenden Datumsformat oder Tageszeitformat sein.

Datentyp in Embedded SQL	Programmaktion zur Einstellung der Länge
DT_BIGINT	Keine Aktion erforderlich
DT_BINARY(n)	Die Länge wird einem Feld in der Struktur BINARY entnommen.
DT_BIT	Keine Aktion erforderlich
DT_DATE	Die Länge, die durch das abschließende Nullzeichen festgelegt wird.
DT_DOUBLE	Keine Aktion erforderlich
DT_FIXCHAR(n)	Das Längenfeld im SQLDA-Bereich bestimmt die Länge der Zeichenfolge.
DT_FLOAT	Keine Aktion erforderlich
DT_INT	Keine Aktion erforderlich
DT_LONGBINARY	Längenfeld ignoriert.
DT_LONGNVARCHAR	Längenfeld ignoriert.
DT_LONGVARCHAR	Längenfeld ignoriert.
DT_NFIXCHAR(n)	Das Längenfeld im SQLDA-Bereich bestimmt die Länge der Zeichenfolge.

Datentyp in Embedded SQL	Programmaktion zur Einstellung der Länge
DT_NSTRING	Die Länge wird durch das abschließende \0 bestimmt. Wenn die Option ansi_blanks auf On gesetzt ist und die Datenbank mit Leerzeichen aufgefüllt wird, muss das Längenfeld in der SQLDA auf die Länge des Puffers gesetzt werden, der den Wert enthält (mindestens die Länge des Werts plus Platz für das abschließende Nullzeichen).
DT_NVARCHAR	Die Länge wird einem Feld in der Struktur NVARCHAR entnommen.
DT_SMALLINT	Keine Aktion erforderlich
DT_STRING	Die Länge wird durch das abschließende \0 bestimmt. Wenn die Option ansi_blanks auf On gesetzt ist und die Datenbank mit Leerzeichen aufgefüllt wird, muss das Längenfeld in der SQLDA auf die Länge des Puffers gesetzt werden, der den Wert enthält (mindestens die Länge des Werts plus Platz für das abschließende Nullzeichen).
DT_TIME	Die Länge, die durch das abschließende Nullzeichen festgelegt wird.
DT_TIMESTAMP	Die Länge, die durch das abschließende Nullzeichen festgelegt wird.
DT_TIMESTAMP_STRUCT	Keine Aktion erforderlich
DT_UNSBIGINT	Keine Aktion erforderlich
DT_UNSENT	Keine Aktion erforderlich
DT_UNSSMALLINT	Keine Aktion erforderlich
DT_VARCHAR(n)	Die Länge wird einem Feld in der Struktur VARCHAR entnommen.
DT_VARIABLE	Die Länge wird durch das abschließende \0 bestimmt.

Siehe auch

- „LONG-Daten mit Dynamic SQL senden“ auf Seite 527

SQLDA sqllen-Feldwerte beim Abrufen von Daten

Die folgende Tabelle zeigt die Werte des Längenfelds, wenn Sie Daten von der Datenbank abrufen und den SQLDA-Bereich verwenden. Das Feld sqllen wird beim Abrufen von Daten nie geändert.

In diesem Fall sind nur die in der Tabelle gezeigten Datentypen erlaubt. Die Datentypen DT_DATE, DT_TIME und DT_TIMESTAMP werden genau so behandelt wie DT_STRING, wenn Informationen von der Datenbank abgerufen werden. Der Wert wird als Zeichenfolge im aktuellen Datumsformat formatiert.

Datentyp in Embedded SQL	Auf welchen Wert muss das Programm das Längenfeld setzen, wenn es Daten von der Datenbank abruft?	Wie gibt die Datenbank Längeninformationen zurück, nachdem ein Wert abgerufen wurde?
DT_BIGINT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_BINARY(n)	Maximale Länge der Struktur BINARY (n+2). Der Maximalwert für n ist 32765.	Das len-Feld der Struktur BINARY ist auf die tatsächliche Länge in Byte gesetzt.
DT_BIT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_DATE	Länge des Puffers.	Nullzeichen am Ende der Zeichenfolge .
DT_DOUBLE	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_FIXCHAR(n)	Länge des Puffers in Byte. Der Maximalwert für n ist 32767.	Bis zur Länge des Puffers aufgefüllt mit Leerzeichen.
DT_FLOAT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_INT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_LONGBINARY	Längenfeld ignoriert.	Längenfeld ignoriert.
DT_LONGNVARCHAR	Längenfeld ignoriert.	Längenfeld ignoriert.
DT_LONGVARCHAR	Längenfeld ignoriert.	Längenfeld ignoriert.
DT_NFIXCHAR(n)	Länge des Puffers in Byte. Der Maximalwert für n ist 32767.	Bis zur Länge des Puffers aufgefüllt mit Leerzeichen.
DT_NSTRING	Länge des Puffers.	Nullzeichen am Ende der Zeichenfolge .

Datentyp in Embedded SQL	Auf welchen Wert muss das Programm das Längenfeld setzen, wenn es Daten von der Datenbank abrufen?	Wie gibt die Datenbank Längeninformationen zurück, nachdem ein Wert abgerufen wurde?
DT_NVARCHAR(n)	Maximale Länge der Struktur NVARCHAR (n+2). Der Maximalwert für n ist 32765.	Das len-Feld der NVARCHAR-Struktur wird auf die tatsächliche Länge (in Byte) der Zeichenfolge gesetzt.
DT_SMALLINT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_STRING	Länge des Puffers.	Nullzeichen am Ende der Zeichenfolge .
DT_TIME	Länge des Puffers.	Nullzeichen am Ende der Zeichenfolge .
DT_TIMESTAMP	Länge des Puffers.	Nullzeichen am Ende der Zeichenfolge .
DT_TIMESTAMP_STRUCT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_UNSBIGINT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_UNSENT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_UNSSMALLINT	Keine Aktion erforderlich	Keine Aktion erforderlich
DT_VARCHAR(n)	Maximale Länge der Struktur VARCHAR (n+2). Der Maximalwert für n ist 32765.	Das len-Feld der VARCHAR-Struktur wird auf die tatsächliche Länge (in Byte) der Zeichenfolge gesetzt.

Siehe auch

- „LONG-Daten mit Dynamic SQL abrufen“ auf Seite 524

Abruf von Daten mit Embedded SQL

In Embedded SQL werden Daten mit der Anweisung SELECT abgerufen. Dabei werden zwei Fälle unterschieden:

- **Die SELECT-Anweisung gibt höchstens eine Zeile zurück** Verwenden Sie eine INTO-Klausel, damit die zurückgegebenen Werte direkt Hostvariablen zugeordnet werden.
- **Die SELECT-Anweisung kann mehrere Zeilen zurückgeben** Verwenden Sie Cursor zur Verwaltung der Zeilen in der Ergebnismenge.

Siehe auch

- „FETCH-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

SELECT-Anweisungen, die höchstens eine Zeile zurückgeben

Eine einzeilige Abfrage fragt höchstens eine Zeile von der Datenbank ab. In einer SELECT-Anweisung für eine einzeilige Abfrage befindet sich eine INTO-Klausel nach der Auswahlliste und vor der FROM-Klausel. Die INTO-Klausel enthält eine Liste der Hostvariablen, um den Wert der einzelnen Elemente der Auswahlliste zu empfangen. Die Anzahl der Hostvariablen muss mit der Anzahl der Auswahllisten-Elemente übereinstimmen. Die Hostvariablen können von Indikatorvariablen gefolgt sein, um NULL-Ergebnisse anzuzeigen.

Sobald die SELECT-Anweisung ausgeführt wird, ruft der Datenbankserver die Ergebnisse ab und schreibt sie in die Hostvariablen. Falls die Abfrageergebnisse mehr als eine Zeile enthalten, gibt der Datenbankserver einen Fehler zurück.

Wenn die Abfrage dazu führt, dass keine Zeilen ausgewählt werden, wird ein Fehler zurückgegeben, der darauf hinweist, dass keine Zeilen gefunden werden konnten (SQLCODE 100). Fehler und Warnungen werden in der SQLCA-Struktur zurückgegeben.

Beispiel

Das folgende Codefragment gibt zum Beispiel 1 zurück, falls eine Zeile der Tabelle Employees erfolgreich abgerufen wird, 0, falls die Zeile nicht existiert, und -1, falls ein Fehler auftritt.

```
EXEC SQL BEGIN DECLARE SECTION;
long      id;
char      name[41];
char      sex;
char      birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;

...
int find_employee( long employee_id )
{
    id = employee_id;
    EXEC SQL SELECT GivenName ||
        ' ' || Surname, Sex, BirthDate
        INTO :name, :sex,
            :birthdate:ind_birthdate
        FROM Employees
        WHERE EmployeeID = :id;
    if( SQLCODE == SQLE_NOTFOUND )
    {
        return( 0 ); /* employee not found */
    }
    else if( SQLCODE < 0 )
    {
        return( -1 ); /* error */
    }
    else
    {
        return( 1 ); /* found */
    }
}
```

```
}
}
```

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495

Cursor in Embedded SQL

Ein Cursor wird benutzt, um Zeilen aus einer Abfrage abzurufen, die mehrere Zeilen in ihrer Ergebnismenge hat. Ein Cursor ist ein Handle oder ein Name (identifiziert) für die SQL-Abfrage und eine Position innerhalb der Ergebnismenge.

Die Cursor-Verwaltung in Embedded SQL erfordert folgende Schritte:

1. Deklarieren Sie einen Cursor für eine bestimmte SELECT-Anweisung mit der DECLARE CURSOR-Anweisung.
2. Öffnen Sie den Cursor mit der Anweisung OPEN.
3. Rufen Sie die Ergebnisse Zeile für Zeile mit der FETCH-Anweisung aus dem Cursor ab.
4. Wiederholen Sie das Abrufen der Zeilen, bis die Warnung Zeile nicht gefunden zurückgegeben wird.

Fehler und Warnungen werden in der SQLCA-Struktur zurückgegeben.

5. Schließen Sie den Cursor mit der CLOSE-Anweisung.

Als Voreinstellung werden Cursor automatisch am Ende der Transaktion geschlossen (bei COMMIT oder ROLLBACK). Cursor, die mit einer WITH HOLD-Klausel geöffnet werden, bleiben für folgende Transaktionen geöffnet, bis sie explizit geschlossen werden.

Das folgende einfache Beispiel zeigt den Gebrauch von Cursor:

```
void print_employees( void )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char    name[50];
    char    sex;
    char    birthdate[15];
    a_sql_len ind_birthdate;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL DECLARE C1 CURSOR FOR
        SELECT GivenName || ' ' || Surname, Sex, BirthDate FROM Employees;
    EXEC SQL OPEN C1;
    for( ;; )
    {
        EXEC SQL FETCH C1 INTO :name, :sex, :birthdate:ind_birthdate;
        if( SQLCODE == SQLE_NOTFOUND )
        {
            break;
        }
        else if( SQLCODE < 0 )
        {

```

```

        break;
    }

    if( ind_birthdate < 0 )
    {
        strcpy( birthdate, "UNKNOWN" );
    }
    printf( "Name: %s Sex: %c Birthdate: %s\n", name, sex, birthdate );
}
EXEC SQL CLOSE C1;
}

```

Cursor positionieren

Ein Cursor wird an einer von drei Stellen positioniert:

- Auf einer Zeile
- Vor der ersten Zeile
- Nach der letzten Zeile

Absolute Zeile
ab Start

Absolute Zeile
ab Ende

0	Vor der ersten Zeile	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		-3
$n - 1$		-2
n		-1
$n + 1$	Nach der letzten Zeile	0

Wenn ein Cursor geöffnet wird, wird er vor der ersten Zeile positioniert. Die Cursorposition kann mit der FETCH-Anweisung verschoben werden. Er kann absolut positioniert werden, entweder bezogen auf den Anfang oder auf das Ende des Abfrageergebnisses. Er kann auch relativ zur aktuellen Cursorposition verschoben werden.

Mit speziellen **positionierten** Versionen der Anweisungen UPDATE und DELETE können Sie die Zeile an der aktuellen Cursorposition aktualisieren oder löschen. Wenn der Cursor vor der ersten oder hinter der letzten Zeile positioniert ist, wird ein Fehler zurückgegeben, der angibt, dass es keine entsprechende Zeile im Cursor gibt.

Die Anweisung PUT kann verwendet werden, um eine Zeile in einen Cursor einzufügen.

Probleme mit der Cursorpositionierung

Einfügungen und einige Aktualisierungen zu DYNAMIC SCROLL-Cursoren können Probleme bei der Cursorpositionierung verursachen. Der Datenbankserver platziert eingefügte Zeilen an unvorhersehbaren Positionen innerhalb eines Cursors, falls die SELECT-Anweisung keine ORDER BY-Klausel hat. In einigen Fällen erscheint die eingefügte Zeile erst, wenn der Cursor geschlossen und wieder geöffnet wurde.

Bei SQL Anywhere tritt dies auf, wenn zum Öffnen eines Cursors eine temporäre Tabelle erstellt werden musste.

Die UPDATE-Anweisung kann bewirken, dass sich eine Zeile im Cursor verschiebt. Das passiert, wenn der Cursor eine ORDER BY-Klausel hat, die einen vorhandenen Index benutzt (es wird keine temporäre Tabelle erstellt).

Siehe auch

- „CLOSE-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „DECLARE CURSOR-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „DELETE-Anweisung (positionsbasiert) [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „FETCH-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „OPEN-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „PUT-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „UPDATE-Anweisung (positionsbasiert) [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Cursor-Grundsätze“ auf Seite 8
- „Beispiel für statischen Cursor“ auf Seite 478
- „Beispiel für dynamischen Cursor“ auf Seite 480
- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Tipp: Arbeitstabellen in der Abfrageverarbeitung verwenden ("Alle Zeilen" als Optimierungsziel verwenden)“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*]

Mehrzeilige Abrufe oder Array-Abrufe

Die FETCH-Anweisung kann so modifiziert werden, dass sie mehr als eine Zeile auf einmal abruft. Das kann die Performance verbessern. Man nennt diese Methode **mehrzeiliges Abrufen (wide fetch)** oder auch **Array-Abrufen (array fetch)**.

SQL Anywhere unterstützt auch weites Speichern und weite Einfügungen.

Um mehrzeilige Abrufe in Embedded SQL zu verwenden, fügen Sie die FETCH-Abrufanweisung wie folgt in Ihren Code ein:

```
EXEC SQL FETCH ... ARRAY nnn
```

Dabei ist ARRAY *nnn* das letzte Element der FETCH-Anweisung. Die Abrufanzahl *nnn* kann eine Hostvariable sein. Die Anzahl der Variablen im SQLDA-Bereich muss das Produkt aus *nnn* multipliziert mit der Anzahl der Spalten pro Zeile sein. Die erste Zeile wird in die SQLDA-Variablen von 0 bis (Spalten pro Zeile)-1 geschrieben, usw.

Jede Spalte muss in jeder Zeile des SQLDA-Bereichs vom selben Typ sein, sonst wird der Fehler SQLDA_INCONSISTENT ausgegeben.

Der Server gibt in SQLCOUNT die Anzahl der Datensätze zurück, die abgerufen wurden. Diese Anzahl ist immer größer als 0 (Null), außer wenn ein Fehler oder eine Warnung ausgegeben wird. Bei einem mehrzeiligen Abruf gibt ein SQLCOUNT von 1 ohne Fehlerzustand an, dass eine gültige Zeile abgerufen wurde.

Beispiel

Der folgende Beispielcode zeigt den Gebrauch von mehrzeiligen Abrufen. Dieser Code befindet sich auch in %SQLANYSAMPI6%\SQLAnywhere\esqlwidefetch\widefetch.sqc.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR { PrintSQLError();
                             goto err; };

static void PrintSQLError()
{
    char buffer[200];

    printf( "SQL error %d -- %s\n",
            SQLCODE,
            sqlerror_message( &sqlca,
                             buffer,
                             sizeof( buffer ) ) );
}

static SQLDA * PrepareSQLDA(
    a_sql_statement_number stat0,
    unsigned width,
    unsigned *cols_per_row )

/* Allocate a SQLDA to be used for fetching from
the statement identified by "stat0". "width"
rows are retrieved on each FETCH request.
The number of columns per row is assigned to
"cols_per_row". */
{
    int          num_cols;
    unsigned     row, col, offset;
    SQLDA *      sqlda;
    EXEC SQL BEGIN DECLARE SECTION;
    a_sql_statement_number stat;
    EXEC SQL END DECLARE SECTION;
    stat = stat0;
    sqlda = alloc_sqlda( 100 );
    if( sqlda == NULL ) return( NULL );
    EXEC SQL DESCRIBE :stat INTO sqlda;
```

```

*cols_per_row = num_cols = sqlda->sqld;
if( num_cols * width > sqlda->sqln )
{
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( num_cols * width );
    if( sqlda == NULL ) return( NULL );
    EXEC SQL DESCRIBE :stat INTO sqlda;
}
// copy first row in SQLDA setup by describe
// to following (wide) rows
sqlda->sqld = num_cols * width;
offset = num_cols;
for( row = 1; row < width; row++ )
{
    for( col = 0;
        col < num_cols;
        col++, offset++ )
    {
        sqlda->sqlvar[offset].sqltype =
            sqlda->sqlvar[col].sqltype;
        sqlda->sqlvar[offset].sqln =
            sqlda->sqlvar[col].sqln;
        // optional: copy described column name
        memcpy( &sqlda->sqlvar[offset].sqlname,
                &sqlda->sqlvar[col].sqlname,
                sizeof( sqlda->sqlvar[0].sqlname ) );
    }
}
fill_s_sqlda( sqlda, 40 );
return( sqlda );
err:
return( NULL );
}
static void PrintFetchedRows(
    SQLDA * sqlda,
    unsigned cols_per_row )
{
    /* Print rows already wide fetched in the SQLDA */
    long    rows_fetched;
    int     row, col, offset;

    if( SQLCOUNT == 0 )
    {
        rows_fetched = 1;
    }
    else
    {
        rows_fetched = SQLCOUNT;
    }
    printf( "Fetched %d Rows:\n", rows_fetched );
    for( row = 0; row < rows_fetched; row++ )
    {
        for( col = 0; col < cols_per_row; col++ )
        {
            offset = row * cols_per_row + col;
            printf( "  \"%s\"",
                (char *)sqlda->sqlvar[offset].sqldata );
        }
        printf( "\n" );
    }
}
static int DoQuery(
    char * query_str0,
    unsigned fetch_width0 )

```

```

{
    /* Wide Fetch "query_str0" select statement
     * using a width of "fetch_width0" rows */
    SQLDA *          sqlda;
    unsigned          cols_per_row;
    EXEC SQL BEGIN DECLARE SECTION;
    a_sql_statement_number stat;
    char *            query_str;
    unsigned          fetch_width;
    EXEC SQL END DECLARE SECTION;

    query_str = query_str0;
    fetch_width = fetch_width0;

    EXEC SQL PREPARE :stat FROM :query_str;
    EXEC SQL DECLARE QCURSOR CURSOR FOR :stat
        FOR READ ONLY;
    EXEC SQL OPEN QCURSOR;
    sqlda = PrepareSQLDA( stat,
        fetch_width,
        &cols_per_row );
    if( sqlda == NULL )
    {
        printf( "Error allocating SQLDA\n" );
        return( SQLE_NO_MEMORY );
    }
    for( ;; )
    {
        EXEC SQL FETCH QCURSOR INTO DESCRIPTOR sqlda
            ARRAY :fetch_width;
        if( SQLCODE != SQLE_NOERROR ) break;
        PrintFetchedRows( sqlda, cols_per_row );
    }
    EXEC SQL CLOSE QCURSOR;
    EXEC SQL DROP STATEMENT :stat;
    free_filled_sqllda( sqlda );
err:
    return( SQLCODE );
}

void main( int argc, char *argv[] )
{
    /* Optional first argument is a select statement,
     * optional second argument is the fetch width */
    char *query_str =
        "SELECT GivenName, Surname FROM Employees";
    unsigned fetch_width = 10;

    if( argc > 1 )
    {
        query_str = argv[1];
        if( argc > 2 )
        {
            fetch_width = atoi( argv[2] );
            if( fetch_width < 2 )
            {
                fetch_width = 2;
            }
        }
    }

    db_init( &sqlca );
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";

    DoQuery( query_str, fetch_width );
}

```

```
EXEC SQL DISCONNECT;  
err:  
  db_fini( &sqlca );  
}
```

Hinweise zur Verwendung von mehrzeiligen Abrufen

- Die Funktion PrepareSQLDA weist den SQLDA-Bereich mithilfe der Funktion alloc_sqlda einem Speicherbereich zu. So wird Platz für Indikatorvariablen ermöglicht, anstatt die Funktion alloc_sqlda_noind zu verwenden.
- Falls weniger als die angeforderte Anzahl von Zeilen, jedoch nicht NULL, abgerufen wurden (zum Beispiel am Ende des Cursors), wird für die nicht abgerufenen SQLDA-Elemente jeweils NULL zurückgegeben, indem der entsprechende Indikatorwert gesetzt wird. Falls keine Indikatorvariablen vorhanden ist, wird ein Fehler erzeugt (SQLE_NO_INDICATOR: keine Indikatorvariable für ein NULL-Ergebnis).
- Falls eine Zeile zurückgegeben wird, die aktualisiert wurde, und die eine Warnung SQLE_ROW_UPDATED_WARNING hervorgerufen hat, wird der Abruf in der Zeile angehalten, die die Warnung verursacht hat. Die Werte aller bis zu diesem Zeitpunkt abgearbeiteten Zeilen werden zurückgegeben (einschließlich der Zeile, die die Warnung verursacht hat). SQLCOUNT enthält die Anzahl der Zeilen, die abgerufen wurden, einschließlich der Zeile, die die Warnung verursacht hat. Alle verbleibenden SQLDA-Elemente werden mit NULL gekennzeichnet.
- Falls eine Zeile, die abgerufen werden soll, gelöscht oder gesperrt wurde und so einen Fehler hervorruft (SQLE_NO_CURRENT_ROW oder SQLE_LOCKED), enthält SQLCOUNT die Anzahl der Zeilen, die vor dem Fehler gelesen wurden. Dies schließt nicht die Zeile ein, die den Fehler verursachte. Der SQLDA-Bereich enthält keine Werte für die Zeilen, denn bei Fehlern werden keine SQLDA-Werte zurückgegeben. Der Wert von SQLCOUNT kann verwendet werden, um den Cursor neu zu positionieren und, falls nötig, um die Zeilen zu lesen.

Siehe auch

- „EXECUTE-Anweisung [ESQL]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „FETCH-Anweisung [ESQL] [SP]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „PUT-Anweisung [ESQL]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „alloc_sqlda-Funktion“ auf Seite 532
- „alloc_sqlda_noind-Funktion“ auf Seite 533

Senden und Abrufen langer Werte mit Embedded SQL

Die Methode zum Senden und Abrufen von LONG VARCHAR-, LONG NVARCHAR- und LONG BINARY-Werten in Embedded SQL-Anwendungen ist anders als bei den übrigen Datentypen. Die Standard-SQLDA-Felder sind auf 32767-Byte-Daten beschränkt, da die Felder, die die Längeninformationen enthalten (sqlen, *sqlind), 16-Bit-Werte sind. Eine Änderung dieser Werte auf 32-Bit-Werte würde vorhandene Anwendungen zerstören.

Die Methode zur Beschreibung von LONG VARCHAR-, LONG NVARCHAR- und LONG BINARY-Werten ist dieselbe wie bei anderen Datentypen.

Static SQL-Strukturen

Es werden separate Felder verwendet, um die zugeordneten, gespeicherten und ungekürzten Längen von LONG BINARY-, LONG VARCHAR- und LONG NVARCHAR-Datentypen aufzunehmen. Die statischen SQL-Datentypen werden in *sqlca.h* folgendermaßen definiert:

```
#define DECL_LONGVARCHAR( size )      \
    struct { a_sql_uint32    array_len; \
              a_sql_uint32    stored_len; \
              a_sql_uint32    untrunc_len; \
              char             array[size+1]; \
    }

#define DECL_LONGNVARCHAR( size )      \
    struct { a_sql_uint32    array_len; \
              a_sql_uint32    stored_len; \
              a_sql_uint32    untrunc_len; \
              char             array[size+1]; \
    }

#define DECL_LONGBINARY( size )      \
    struct { a_sql_uint32    array_len; \
              a_sql_uint32    stored_len; \
              a_sql_uint32    untrunc_len; \
              char             array[size]; \
    }
```

Dynamic SQL-Strukturen

Bei dynamischem SQL ist das Einstellen des sqltype-Felds auf DT_LONGVARCHAR, LONG NVARCHAR oder DT_LONGBINARY ausreichend. Die zugeordnete LONGVARCHAR-, LONGNVARCHAR- und LONGBINARY-Struktur sieht folgendermaßen aus:

```
typedef struct LONGVARCHAR {
    a_sql_uint32    array_len;
    a_sql_uint32    stored_len;
    a_sql_uint32    untrunc_len;
    char            array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

Definition von Strukturelementen

Sowohl für Static als auch für Dynamic SQL-Strukturen sind die Strukturelemente wie folgt definiert:

- **array_len** (Senden und Abrufen.) Die Anzahl der Byte, die für den Array-Teil der Struktur zugewiesen sind.
- **stored_len** (Senden und Abrufen.) Die Anzahl der im Array gespeicherten Byte. Immer kleiner oder gleich array_len und untrunc_len.
- **untrunc_len** (Nur Abruf.) Die Anzahl der Byte, die im Array gespeichert würden, wenn der Wert nicht gekürzt würde. Immer größer oder gleich stored_len. Bei Kürzung ist dieser Wert größer als array_len.

LONG-Daten mithilfe von Static SQL abrufen

Rufen Sie einen Wert vom Typ LONG VARCHAR, LONG NVARCHAR oder LONG BINARY mit statischem SQL ab.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Deklarieren Sie eine Hostvariable vom Typ DECL_LONGVARCHAR, DECL_LONGNVARCHAR oder DECL_LONGBINARY, wie erforderlich. Das Array-Element array_len wird automatisch ausgefüllt.
2. Rufen Sie die Daten mit FETCH, GET DATA oder EXECUTE INTO ab. SQL Anywhere legt die folgenden Informationen fest:
 - **Indikatorvariable** Negativ, wenn der Wert NULL ist, 0, wenn es keine Kürzung gibt, andernfalls die ungekürzte Länge in Byte bis zu einem Maximum von 32767.
 - **stored_len** Die Anzahl der im Array gespeicherten Byte. Immer kleiner oder gleich array_len und untrunc_len.
 - **untrunc_len** Die Anzahl der Byte, die im Array gespeichert würden, wenn der Wert nicht gekürzt würde. Immer größer oder gleich stored_len. Bei Kürzung ist dieser Wert größer als array_len.

Ergebnisse

Die LONG-Daten werden mit statischem SQL abgerufen.

Siehe auch

- „Senden und Abrufen langer Werte mit Embedded SQL“ auf Seite 522
- „Indikatorvariablen“ auf Seite 493

LONG-Daten mit Dynamic SQL abrufen

Rufen Sie einen Wert vom Typ LONG VARCHAR, LONG NVARCHAR oder LONG BINARY mit Dynamic SQL ab.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Stellen Sie das sqltype-Feld auf DT_LONGVARCHAR, DT_LONGNVARCHAR oder DT_LONGBINARY ein, wie erforderlich.

2. Setzen Sie das Feld `sqldata` so, dass es auf die Hostvariablenstruktur `LONGVARCHAR`, `LONGNVARCHAR` oder `LONGBINARY` zeigt.

Sie können den Makro `LONGVARCHARSIZE(n)`, `LONGNVARCHARSIZE(n)` oder `LONGBINARYSIZE(n)` verwenden, um die Gesamtanzahl der Byte zu bestimmen, die zugeordnet werden müssen, um n Byte von Daten im Array-Feld aufzunehmen.

3. Stellen Sie das `array_len`-Feld der Hostvariablenstruktur auf die Anzahl der Bytes ein, die dem Array-Feld zugeordnet sind.
4. Rufen Sie die Daten mit `FETCH`, `GET DATA` oder `EXECUTE INTO` ab. SQL Anywhere legt die folgenden Informationen fest:
 - *** `sqlind`** Dieses `sqlda`-Feld ist negativ, wenn der Wert `NULL` ist, oder 0, wenn es keine Kürzung gibt, beziehungsweise die positive ungekürzte Länge in Byte bis zu einem Höchstwert von 32767.
 - **`stored_len`** Die Anzahl der im Array gespeicherten Byte. Immer kleiner oder gleich `array_len` und `untrunc_len`.
 - **`untrunc_len`** Die Anzahl der Byte, die im Array gespeichert würden, wenn der Wert nicht gekürzt würde. Immer größer oder gleich `stored_len`. Bei Kürzung ist dieser Wert größer als `array_len`.

Ergebnisse

Die LONG-Daten werden mit Dynamic SQL abgerufen.

Beispiel

Das folgende Codefragment illustriert den Mechanismus, der zum Abrufen von `LONG VARCHAR`-Daten mittels dynamischem Embedded SQL verwendet wird. Er dient nur zur Illustration und ist nicht für eine tatsächliche Verwendung bestimmt.

```
#define DATA_LEN 128000
void get_test_var()
{
    LONGVARCHAR *longptr;
    SQLDA      *sqlda;
    SQLVAR      *sqlvar;

    sqlda = alloc_sqlda( 1 );
    longptr = (LONGVARCHAR *)malloc(
        LONGVARCHARSIZE( DATA_LEN ) );
    if( sqlda == NULL || longptr == NULL )
    {
        fatal_error( "Allocation failed" );
    }

    // init longptr for receiving data
    longptr->array_len = DATA_LEN;

    // init sqlda for receiving data
    // (sqlen is unused with DT_LONG types)
    sqlda->sqlid = 1; // using 1 sqlvar
    sqlvar = &sqlda->sqlvar[0];
    sqlvar->sqltype = DT_LONGVARCHAR;
```

```
sqlvar->sqldata = longptr;
printf( "fetching test_var\n" );
EXEC SQL PREPARE select_stmt FROM 'SELECT test_var';
EXEC SQL EXECUTE select_stmt INTO DESCRIPTOR sqlda;
EXEC SQL DROP STATEMENT select_stmt;
printf( "stored_len: %d, untrunc_len: %d, "
        "1st char: %c, last char: %c\n",
        longptr->stored_len,
        longptr->untrunc_len,
        longptr->array[0],
        longptr->array[DATA_LEN - 1] );
free_sqlda( sqlda );
free( longptr );
}
```

Siehe auch

- [„Senden und Abrufen langer Werte mit Embedded SQL“ auf Seite 522](#)
- [„Indikatorvariablen“ auf Seite 493](#)

LONG-Daten mit statischem SQL senden

Senden Sie LONG-Werte mit statischem SQL aus einer Embedded SQL-Anwendung an die Datenbank.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Deklarieren Sie eine Hostvariable vom Typ DECL_LONGVARCHAR, DECL_LONGVARCHAR oder DECL_LONGBINARY, wie erforderlich.
2. Wenn Sie NULL senden, setzen Sie die Indikatorvariable auf einen negativen Wert.
3. Stellen Sie das stored_len-Feld der Hostvariablenstruktur auf die Anzahl der Byte der Daten im Array-Feld.
4. Senden Sie die Daten, indem Sie den Cursor öffnen oder die Anweisung ausführen.

Ergebnisse

Die Embedded SQL-Anwendung ist bereit, LONG-Werte an die Datenbank zu senden.

Beispiel

Das folgende Codefragment illustriert den Mechanismus, der zum Senden von LONG VARCHAR-Daten mittels Static Embedded SQL verwendet wird. Er dient nur zur Illustration und ist nicht für eine tatsächliche Verwendung bestimmt.

```
#define DATA_LEN 12800
EXEC SQL BEGIN DECLARE SECTION;
// SQLPP initializes longdata.array_len
DECL_LONGVARCHAR(128000) longdata;
```

```
EXEC SQL END DECLARE SECTION;

void set_test_var()
{
    // init longdata for sending data
    memset( longdata.array, 'a', DATA_LEN );
    longdata.stored_len = DATA_LEN;

    printf( "Setting test_var to %d a's\n", DATA_LEN );
    EXEC SQL SET test_var = :longdata;
}
```

Siehe auch

- „Senden und Abrufen langer Werte mit Embedded SQL“ auf Seite 522
- „Indikatorvariablen“ auf Seite 493

LONG-Daten mit Dynamic SQL senden

Senden Sie LONG-Werte mit Dynamic SQL aus einer Embedded SQL-Anwendung an die Datenbank.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Stellen Sie das `sqltype`-Feld auf `DT_LONGVARCHAR`, `DT_LONGNVARCHAR` oder `DT_LONGBINARY` ein, wie erforderlich.
2. Wenn Sie NULL senden, stellen Sie ***sqlind** auf einen negativen Wert ein.
3. Wenn Sie nicht NULL senden, setzen Sie das Feld `sqldata` so, dass es auf die `LONGVARCHAR`-, `LONGNVARCHAR`-oder `LONGBINARY`-Hostvariablenstruktur zeigt.

Sie können den Makro `LONGVARCHARSIZE(n)`, `LONGNVARCHARSIZE(n)` oder `LONGBINARYSIZE(n)` verwenden, um die Gesamtanzahl der Byte zu bestimmen, die zugeordnet werden müssen, um *n* Byte von Daten im Array-Feld aufzunehmen.

4. Stellen Sie das `array_len`-Feld der Hostvariablenstruktur auf die Anzahl der Bytes ein, die dem Array-Feld zugeordnet sind.
5. Stellen Sie das `stored_len`-Feld der Hostvariablenstruktur auf die Anzahl der Byte der Daten im Array-Feld. Dieser Wert darf nicht größer als `len` sein.
6. Senden Sie die Daten, indem Sie den Cursor öffnen oder die Anweisung ausführen.

Ergebnisse

Die Embedded SQL-Anwendung ist bereit, LONG-Werte an die Datenbank zu senden.

Siehe auch

- „Senden und Abrufen langer Werte mit Embedded SQL“ auf Seite 522
- „Indikatorvariablen“ auf Seite 493

Einfache gespeicherte Prozeduren in Embedded SQL

Sie können gespeicherte Prozeduren in Embedded SQL erstellen und aufrufen.

Eine CREATE PROCEDURE-Anweisung kann wie jede andere Datendefinitions-Anweisung, z.B. CREATE TABLE, eingebettet werden. Sie können auch eine CALL-Anweisung einbetten, um eine gespeicherte Prozedur auszuführen. Das folgende Code-Fragment veranschaulicht das Erstellen und Ausführen einer gespeicherten Prozedur in Embedded SQL

```
EXEC SQL CREATE PROCEDURE pettycash(
    IN Amount DECIMAL(10,2) )
BEGIN
    UPDATE account
    SET balance = balance - Amount
    WHERE name = 'bank';

    UPDATE account
    SET balance = balance + Amount
    WHERE name = 'pettycash expense';
END;
EXEC SQL CALL pettycash( 10.72 );
```

Wenn Sie Werte von Hostvariablen an eine gespeicherte Prozedur übergeben oder die Ausgabevariablen abrufen möchten, bereiten Sie eine CALL-Anweisung vor und führen Sie sie aus. Das folgende Code-Fragment veranschaulicht die Verwendung von Hostvariablen. Sowohl die USING als auch die INTO-Klausel werden in der EXECUTE-Anweisung benutzt.

```
EXEC SQL BEGIN DECLARE SECTION;
double hv_expense;
double hv_balance;
EXEC SQL END DECLARE SECTION;
hv_expense = 20.00;

db_init( &sqlca );

EXEC SQL CONNECT USING 'UID=DBA;PWD=sql';

EXEC SQL CREATE OR REPLACE PROCEDURE pettycash(
    IN expense DECIMAL(10,2),
    OUT endbalance DECIMAL(10,2) )
BEGIN
    UPDATE account
    SET balance = balance - expense
    WHERE name = 'bank';
    UPDATE account
    SET balance = balance + expense
    WHERE name = 'pettycash expense';

    SET endbalance = ( SELECT balance FROM account
                      WHERE name = 'bank' );
END;
```

```
EXEC SQL PREPARE S1 FROM 'CALL pettycash( ?, ? )';  
EXEC SQL EXECUTE S1 USING :hv_expense INTO :hv_balance;
```

Siehe auch

- „CREATE PROCEDURE-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „EXECUTE-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „PREPARE-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Gespeicherte Prozeduren mit Ergebnismengen

Eine Datenbankprozedur kann auch eine SELECT-Anweisung enthalten. Die Prozedur wird mit einer RESULT-Klausel deklariert, um Anzahl, Name und Typen der Spalten in der Ergebnismenge zu spezifizieren. Die Spalten einer Ergebnismenge unterscheiden sich von den Ausgabeparametern. In einer Prozedur mit Ergebnismenge kann die CALL-Anweisung anstelle der SELECT-Anweisung in der Cursordeklaration verwendet werden:

```
EXEC SQL BEGIN DECLARE SECTION;  
    char hv_name[100];  
EXEC SQL END DECLARE SECTION;  
  
EXEC SQL CREATE PROCEDURE female_employees()  
    RESULT( name char(50) )  
BEGIN  
    SELECT GivenName || Surname FROM Employees  
    WHERE Sex = 'f';  
END;  
  
EXEC SQL PREPARE S1 FROM 'CALL female_employees()';  
  
EXEC SQL DECLARE C1 CURSOR FOR S1;  
EXEC SQL OPEN C1;  
for(;;)  
{  
    EXEC SQL FETCH C1 INTO :hv_name;  
    if( SQLCODE != SQLE_NOERROR ) break;  
    printf( "%s\n", hv_name );  
}  
EXEC SQL CLOSE C1;
```

In diesem Beispiel wurde die Prozedur mit einer OPEN-Anweisung statt einer EXECUTE-Anweisung aufgerufen. Nach der OPEN-Anweisung wird die Prozedur ausgeführt, bis sie eine SELECT-Anweisung erreicht. Zu diesem Zeitpunkt ist C1 ein Cursor für die SELECT-Anweisung innerhalb der Datenbankprozedur. Sie können alle Arten der FETCH-Anweisung verwenden (rückwärts und vorwärts scrollen), solange Sie sie benötigen. Die CLOSE-Anweisung stoppt die Ausführung der Prozedur.

Nach der SELECT-Anweisung wird in der Prozedur keine weitere Anweisung ausgeführt. Verwenden Sie zur Ausführung von Anweisungen nach dem SELECT die Anweisung RESUME Cursorname. Die RESUME-Anweisung gibt entweder die Warnung SQLE_PROCEDURE_COMPLETE zurück oder SQLE_NOERROR, und zeigt damit an, dass es noch einen weiteren Cursor gibt. Das folgende Beispiel zeigt eine zweifache Auswahlprozedur:

```
EXEC SQL CREATE PROCEDURE people()  
    RESULT( name char(50) )  
BEGIN
```

```
SELECT GivenName || Surname
FROM Employees;

SELECT GivenName || Surname
FROM Customers;
END;

EXEC SQL PREPARE S1 FROM 'CALL people()';
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
while( SQLCODE == SQLE_NOERROR )
{
    for(;;)
    {
        EXEC SQL FETCH C1 INTO :hv_name;
        if( SQLCODE != SQLE_NOERROR ) break;
        printf( "%s\n", hv_name );
    }
    EXEC SQL RESUME C1;
}
EXEC SQL CLOSE C1;
```

Dynamische Cursor für CALL-Anweisungen

Diese Beispiele haben statische Cursor verwendet. Dynamische Cursor können auch für die CALL-Anweisung verwendet werden.

Die DESCRIBE-Anweisung funktioniert ohne Einschränkung für Prozeduraufrufe. DESCRIBE OUTPUT erzeugt einen SQLDA-Bereich mit einer Beschreibung für jede Spalte der Ergebnismenge.

Hat die Prozedur keine Ergebnismenge, enthält der SQLDA-Bereich eine Beschreibung für jeden INOUT- oder OUT-Parameter der Prozedur. Eine DESCRIBE INPUT-Anweisung erzeugt einen SQLDA-Bereich mit einer Beschreibung für jeden IN- oder INOUT-Parameter der Prozedur.

DESCRIBE ALL

DESCRIBE ALL beschreibt IN-, INOUT-, OUT- und RESULT-Parameter. DESCRIBE ALL benutzt die Indikatorvariablen im SQLDA-Bereich, um zusätzliche Information zu liefern.

Die Bits für DT_PROCEDURE_IN und DT_PROCEDURE_OUT werden in den Indikatorvariablen gesetzt, wenn eine CALL-Anweisung beschrieben wird. DT_PROCEDURE_IN gibt einen IN- oder INOUT-Parameter an, und DT_PROCEDURE_OUT gibt einen INOUT- oder OUT-Parameter an. In RESULT-Spalten von Prozeduren sind beide Bits bereinigt.

Nach einem Beschreibungs-OUTPUT können diese Bits benutzt werden, um zwischen Anweisungen zu unterscheiden, die Ergebnismengen umfassen (müssen OPEN, FETCH, RESUME, CLOSE verwenden), und Anweisungen, die keine Ergebnismengen umfassen (müssen EXECUTE verwenden).

Mehrfache Ergebnismengen

Haben Sie eine Prozedur, die mehrere Ergebnismengen zurückgibt, müssen Sie sie nach jeder RESUME-Anweisung neu beschreiben, falls sich die Form der Ergebnismengen ändert.

Sie müssen den Cursor beschreiben, nicht die Anweisung, um die aktuelle Position des Cursors neu zu beschreiben.

Siehe auch

- „CREATE PROCEDURE-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Die dynamische SELECT-Anweisung“ auf Seite 503
- „DESCRIBE-Anweisung [ESQL]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Anforderungsverwaltung mit Embedded SQL

Da eine typische Embedded SQL-Anwendung warten muss, bis jede Datenbankanforderung abgeschlossen ist, bevor der nächste Schritt unternommen wird, kann eine Anwendung, die mehrere Ausführungs-Threads verwendet, mit anderen Tasks fortfahren.

Wenn Sie einen einzelnen Ausführungsthread verwenden müssen, kann ein bestimmter Grad des Multitaskings durchgeführt werden, indem eine Callback-Funktion registriert wird, die die Funktion `db_register_a_callback` mit der Option `DB_CALLBACK_WAIT` verwendet. Ihre Callback-Funktion wird wiederholt von der Schnittstellenbibliothek aufgerufen, während der Datenbankserver oder die Clientbibliothek mit dem Abarbeiten Ihrer Datenbankanforderung beschäftigt sind.

In Ihrer Callback-Funktion können Sie keine weitere Datenbankanforderung starten, aber die aktuelle Anforderung mit der Funktion `db_cancel_request` abbrechen. Sie können die Funktion `db_is_working` in Ihren Message-Handlers verwenden, um festzustellen, ob gerade eine Anforderung an die Datenbank abgearbeitet wird.

Siehe auch

- „`db_register_a_callback`-Funktion“ auf Seite 546
- „`db_cancel_request`-Funktion“ auf Seite 538
- „`db_is_working`-Funktion“ auf Seite 543

Datenbanksicherung mit Embedded SQL

Das empfohlene Verfahren zum Sichern einer Datenbank ist die Verwendung der BACKUP-Anweisung.

Die `db_backup`-Funktion bietet eine andere Methode für eine Online-Sicherung in Embedded SQL-Anwendungen. Das Sicherungsdienstprogramm von SQL Anywhere nutzt diese Funktion ebenfalls.

Sie können auch direkt mit dem Sicherungs-Dienstprogramm von SQL Anywhere interagieren, indem Sie die Funktion `DBBackup` der Datenbanktools verwenden.

Ein eigenes Programm mit der `db_backup`-Funktion sollten Sie nur schreiben, wenn die anderen Sicherungsmethoden Ihre Sicherungsanforderungen nicht erfüllen.

Siehe auch

- „BACKUP-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Sicherungsdienstprogramm (dbbackup)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- `DBBackup`-Methode [Datenbanktools] auf Seite 879
- „`db_backup`-Funktion“ auf Seite 533

Referenz der Bibliotheksfunktionen

Der SQL-Präprozessor erzeugt Funktionsaufrufe in der Schnittstellenbibliothek oder DLL. Zusätzlich zu den vom SQL-Präprozessor erzeugten Funktionsaufrufen wird dem Benutzer eine Gruppe von Bibliotheksfunktionen zur Verfügung gestellt, mit denen Datenbankvorgänge leichter ausgeführt werden können. Prototypen dieser Funktionen werden durch die Anweisung EXEC SQL INCLUDE SQLCA eingefügt.

Dieser Abschnitt enthält eine Referenzbeschreibung der verschiedenen Funktionen, geordnet nach Kategorien.

DLL-Eintrittspunkte

Die DLL-Eintrittspunkte sind gleich, abgesehen davon, dass die Prototypen einen für DLLs passenden Zusatz haben.

Sie können die Eintrittspunkte auf portierbare Weise unter Verwendung von **_esqlentry_** deklarieren, was in *sqlca.h* definiert ist. Es wird in den Wert `__stdcall` aufgelöst.

alloc_sqllda-Funktion

Syntax

```
struct sqllda * alloc_sqllda( unsigned numvar );
```

Parameter

- **numvar** Die Anzahl der zuzuweisenden Variablendeskriptoren.

Rückgabe

Zeiger auf SQLDA bei Erfolg und Rückgabe eines Null-Zeigers, falls nicht genügend Speicherplatz zur Verfügung stand.

Bemerkungen

Diese Funktion weist einen SQLDA-Bereich mit Deskriptoren für *numvar*-Variablen zu. Das Feld *sqln* des SQLDA-Bereichs wird mit *numvar* initialisiert. Den Indikatorvariablen wird Speicherplatz zugewiesen, die Indikatorzeiger werden auf diesen Speicherplatz gesetzt und der Indikatorwert wird mit 0 (Null) initialisiert. Ein Null-Zeiger wird zurückgegeben, falls kein Speicher zugewiesen werden konnte. Es wird empfohlen, diese Funktion anstelle der Funktion `alloc_sqllda_noind` zu verwenden.

Siehe auch

- „Der SQL-Deskriptor-Bereich (SQLDA)“ auf Seite 504
- „fill_s_sqllda-Funktion“ auf Seite 556
- „fill_sqllda-Funktion“ auf Seite 557
- „fill_sqllda_ex-Funktion“ auf Seite 557
- „free_sqllda-Funktion“ auf Seite 558

alloc_sqllda_noind-Funktion

Syntax

```
struct sqllda * alloc_sqllda_noind( unsigned numvar );
```

Parameter

- **numvar** Die Anzahl der zuzuweisenden Variablendeskriptoren.

Rückgabe

Zeiger auf SQLDA bei Erfolg und Rückgabe eines Null-Zeigers, falls nicht genügend Speicherplatz zur Verfügung stand.

Bemerkungen

Diese Funktion weist einen SQLDA-Bereich mit Deskriptoren für *numvar*-Variablen zu. Das Feld *sqln* des SQLDA-Bereichs wird mit *numvar* initialisiert. Den Indikatorvariablen wird kein Speicherplatz zugewiesen; die Indikatorzeiger werden auf den Null-Zeiger gesetzt. Ein Null-Zeiger wird zurückgegeben, falls kein Speicher zugewiesen werden konnte.

Siehe auch

- „Der SQL-Deskriptor-Bereich (SQLDA)“ auf Seite 504
- „fill_s_sqllda-Funktion“ auf Seite 556
- „fill_sqllda-Funktion“ auf Seite 557
- „fill_sqllda_ex-Funktion“ auf Seite 557
- „free_sqllda_noind-Funktion“ auf Seite 559

db_backup-Funktion

Syntax

```
void db_backup(  
SQLCA * sqlca,  
int op,  
int file_num,  
unsigned long page_num,  
struct sqllda * sqllda);
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **op** Die auszuführende Aktion bzw. der durchzuführende Vorgang.
- **file_num** Die Dateinummer der Datenbank.
- **page_num** Die Seitenanzahl der Datenbank. Ein Wert im Bereich von 0 bis "maximale Seitenanzahl minus 1".
- **sqllda** Ein Zeiger auf eine SQLCA-Struktur.

Autorisierung

Muss als Benutzer mit BACKUP DATABASE-Systemprivileg verbunden sein oder die SYS_RUN_REPLICATION_ROLE-Systemrolle haben.

Bemerkungen

Obwohl diese Funktion eine Möglichkeit bietet, einer Anwendung Sicherungsfunktionen hinzuzufügen, wird empfohlen, diese Aufgabe über die BACKUP-Anweisung auszuführen.

Welche Aktion ausgeführt wird, hängt vom Wert des Parameters *op* ab:

- **DB_BACKUP_START** Muss aufgerufen werden, bevor eine Sicherung beginnen kann. Zu einem Zeitpunkt kann immer nur eine Sicherung pro Datenbank auf einem gegebenen Datenbankserver laufen. Datenbank-Checkpoints sind deaktiviert, bis die Sicherung vollständig ist (db_backup wird mit dem Wert *op* für DB_BACKUP_END aufgerufen). Kann die Sicherung nicht starten, wird der SQLCODE mit SQLE_BACKUP_NOT_STARTED belegt. Andernfalls wird das Feld SQLCOUNT des *sqlca* mit der Größe der Datenbankseiten belegt. Sicherungen werden Seite für Seite durchgeführt.

Die Parameter *file_num*, *page_num* und *sqlda* werden nicht beachtet.

- **DB_BACKUP_OPEN_FILE** Öffnet die mit *file_num* angegebene Datenbankdatei, sodass Seiten der angegebenen Datei mit DB_BACKUP_READ_PAGE gesichert werden können. Gültige Dateinummern sind 0 bis DB_BACKUP_MAX_FILE für die Stammverzeichnisdateien und 0 bis DB_BACKUP_TRANS_LOG_FILE für die Transaktionslogdatei. Falls die angegebene Datei nicht existiert, wird SQLCODE mit SQLE_NOTFOUND belegt. Andernfalls enthält SQLCOUNT die Anzahl der Seiten in der Datei, SQLIOESTIMATE enthält einen 32-Bit-Wert (POSIX time_t), der die Zeit angibt, zu der die Datenbankdatei erstellt wurde, und der Name der Betriebssystemdatei befindet sich im Feld *sqlerrmc* des SQLCA-Bereichs.

Die Parameter *page_num* und *sqlda* werden nicht beachtet.

- **DB_BACKUP_READ_PAGE** Eine Seite der Datenbankdatei, die durch *file_num* festgelegt wird. Der Wert für *page_num* muss zwischen 0 und einer Zahl unter der Seitenanzahl liegen, die von SQLCOUNT nach einem erfolgreichen Aufruf von db_backup mit dem DB_BACKUP_OPEN_FILE-Vorgang zurückgegeben wurde. Andernfalls wird SQLCODE mit SQLE_NOTFOUND belegt. Der *sqlda*-Deskriptor sollte mit einer Variablen vom Typ DT_BINARY oder DT_LONG_BINARY eingerichtet werden, die auf einen Puffer zeigt. Der Puffer sollte groß genug sein, um Binärdaten in der Größe zu speichern, wie sie im Feld SQLCOUNT beim Aufruf von db_backup mit dem Vorgang DB_BACKUP_START zurückgegeben werden.

Die Daten in DT_BINARY enthalten eine 2 Byte lange Längenangabe, gefolgt von den eigentlichen Binärdaten, sodass der Puffer mindestens 2 Byte länger sein muss als die Seitengröße.

Hinweis

Dieser Aufruf erzeugt eine der angegebenen Datenbankseiten im Puffer, aber es bleibt der Anwendung überlassen, den Pufferinhalt auf einem Sicherungsdatenträger zu speichern.

- **DB_BACKUP_READ_RENAME_LOG** Die gleiche Aktion wie bei DB_BACKUP_READ_PAGE, außer dass nach der letzten Seite das Transaktionslog zurückgegeben wurde, der Datenbankserver das bestehende Transaktionslog umbenennt und ein neues startet

Falls der Datenbankserver nicht in der Lage ist, das Log zu diesem Zeitpunkt umzubenennen (z.B. können in Datenbanken der Version 7.0.x oder älter unvollständige Transaktionen auftreten), tritt der Fehler `SQL_E_BACKUP_CANNOT_RENAME_LOG_YET` auf. In diesem Fall verwenden Sie nicht die zurückgegebene Seite, sondern erneuern Sie die Anforderung, bis Sie `SQL_NOERROR` erhalten, und schreiben Sie dann die Seite. Fahren Sie mit dem Lesen der Seiten fort, bis Sie den Zustand `SQL_NOTFOUND` erhalten.

Die Fehlermeldung `SQL_E_BACKUP_CANNOT_RENAME_LOG_YET` kann mehrere Male und für mehrere Seiten auftreten. In Ihrer Wiederholungsschleife sollten Sie eine Verzögerung einbauen, um den Server nicht mit zu vielen Anforderungen zu verlangsamen.

Erhalten Sie die Bedingung `SQL_NOTFOUND`, wurde das Transaktionslog erfolgreich gesichert und die Datei umbenannt. Der Name der alten Transaktionsdatei wird im Feld `sqlerrmc` des `SQLCA`-Bereichs zurückgegeben.

Sie sollten den Wert von `sqlda->sqlvar[0].sqlind` nach einem `db_backup`-Aufruf überprüfen. Ist dieser Wert größer als 0 (Null), wurde die letzte Logseite geschrieben und die Logdatei wurde umbenannt. Der neue Name befindet sich nach wie vor in `sqlca.sqlerrmc`, aber der Wert von `SQLCODE` ist `SQL_NOERROR`.

Danach sollten Sie `db_backup` nicht nochmals aufrufen, außer Sie möchten Dateien schließen und die Sicherung beenden. Falls Sie das tun, erhalten Sie eine zweite Kopie Ihrer gesicherten Logdatei und `SQL_NOTFOUND`.

- **DB_BACKUP_CLOSE_FILE** Muss aufgerufen werden, wenn die Bearbeitung einer Datei abgeschlossen ist, um die mit `file_num` angegebene Datenbankdatei zu schließen.

Die Parameter `page_num` und `sqlda` werden nicht beachtet.

- **DB_BACKUP_END** Muß beim Beenden der Sicherung aufgerufen werden. Keine andere Sicherung kann beginnen, bevor die letzte beendet wurde. Checkpoints werden wieder aktiviert.

Die Parameter `file_num`, `page_num` und `sqlda` werden nicht beachtet.

- **DB_BACKUP_PARALLEL_START** Startet eine parallele Sicherung. Wie bei `DB_BACKUP_START` kann auf einem Datenbankserver jeweils nur eine Sicherung für eine Datenbank ausgeführt werden. Datenbank-Checkpoints sind deaktiviert, bis die Sicherung vollständig ist (bis `db_backup` mit dem Wert `op` für `DB_BACKUP_END` aufgerufen wird). Wenn die Sicherung nicht starten kann, erhalten Sie `SQL_E_BACKUP_NOT_STARTED`. Andernfalls wird das Feld `SQLCOUNT` von `sqlca` mit der Größe der Datenbankseiten belegt.

Der Parameter `file_num` weist den Datenbankserver an, das Transaktionslog umzubenennen, und startet ein neues, nachdem die letzte Seite des Transaktionslogs zurückgegeben wurde. Wenn der Wert nicht Null ist, wird das Transaktionslog umbenannt oder neu gestartet. Andernfalls wird es nicht umbenannt und neu gestartet. Bei diesem Parameter ist der `DB_BACKUP_READ_RENAME_LOG`-Vorgang nicht erforderlich, der während eines parallelen Sicherungsvorgangs nicht zulässig ist.

Der Parameter `page_num` informiert den Datenbankserver über die maximale Größe des Clientpuffers, angegeben in Datenbankseiten. Auf der Serverseite versuchen die Reader der parallelen Sicherung, sequenzielle Blöcke von Seiten zu lesen. Dieser Wert informiert den Server darüber, wie viel

Speicherplatz diesen Blöcken zugewiesen werden muss. Bei der Übergabe des Werts *nnn* weiß der Server, dass der Client maximal *nnnn* Datenbankseiten gleichzeitig vom Server akzeptiert. Der Server kann Blöcke von Seiten zurückgeben, die kleiner als *nnn* sind, wenn er nicht genügend Speicherplatz für Blöcke von *nnn* Seiten zuweisen kann. Wenn der Client die Größe der Datenbankseiten erst nach dem Aufruf von `DB_BACKUP_PARALLEL_START` erfährt, kann dieser Wert dem Server mit `DB_BACKUP_INFO` bereitgestellt werden. Dieser Wert muss vor dem ersten Abruf von Sicherungsseiten (`DB_BACKUP_PARALLEL_READ`) bereitgestellt werden.

Hinweis

Wenn Sie mit `db_backup` eine parallele Sicherung starten, erstellt `db_backup` keine Schreib-Threads. Der Aufrufer von `db_backup` muss die Daten empfangen und als Schreiber agieren.

- **DB_BACKUP_INFO** Dieser Parameter liefert dem Datenbankserver zusätzliche Informationen zur parallelen Sicherung. Der Parameter *file_num* gibt den Typ der bereitgestellten Informationen an und der Parameter *page_num* liefert den Wert. Sie können folgende zusätzliche Informationen mit `DB_BACKUP_INFO` festlegen:
 - **DB_BACKUP_INFO_PAGES_IN_BLOCK** Das Argument *page_num* enthält die maximale Anzahl von Seiten, die in einem Block zurückgesendet werden sollen.
 - **DB_BACKUP_INFO_CHKPT_LOG** Dies ist die clientseitige Entsprechung zur Option `WITH CHECKPOINT LOG` der `BACKUP`-Anweisung. Der *page_num*-Wert von `DB_BACKUP_CHKPT_COPY` gibt `COPY` an, während der Wert `DB_BACKUP_CHKPT_NOCOPY` für `NO COPY` steht. Wenn dieser Wert nicht bereitgestellt wird, ist der Standardwert `COPY`.
- **DB_BACKUP_PARALLEL_READ** Dieser Vorgang liest einen Block von Seiten vom Datenbankserver. Bevor Sie diesen Vorgang aufrufen, verwenden Sie den `DB_BACKUP_OPEN_FILE`-Vorgang, um alle Dateien zu öffnen, die Sie sichern wollen. `DB_BACKUP_PARALLEL_READ` ignoriert die Argumente *file_num* und *page_num*.

Der `sqlda`-Deskriptor sollte mit einer Variablen vom Typ `DT_LONGBINARY` eingerichtet werden, die auf einen Puffer zeigt. Der Puffer muss groß genug sein, um Binärdaten der Größe *nnn* Seiten zu speichern (angegeben im `DB_BACKUP_START_PARALLEL`-Vorgang oder in einem `DB_BACKUP_INFO`-Vorgang).

Der Server gibt einen sequenziellen Block von Datenbankseiten für eine bestimmte Datenbankdatei zurück. Die Seitennummer der ersten Seite im Block wird im Feld `SQLCOUNT` zurückgegeben. Die Dateinummer, zu der die Seiten gehören, wird im Feld `SQLIOESTIMATE` zurückgegeben. Dieser Wert stimmt mit einer der Dateinummern überein, die in den `DB_BACKUP_OPEN_FILE`-Aufrufen verwendet wurden. Die Größe der zurückgegebenen Daten ist im Feld *stored_len* der Variablen `DT_LONGBINARY` verfügbar. Sie ist immer ein Mehrfaches der Datenbankseitengröße. Während die von diesem Aufruf zurückgegebenen Daten einen Block sequenzieller Seiten für eine bestimmte Datei enthalten, kann nicht sicher davon ausgegangen werden, dass einzelne Datenblöcke in einer sequenziellen Reihenfolge zurückgegeben werden oder dass alle Seiten einer Datenbankdatei vor den Seiten einer anderen Datenbankdatei zurückgegeben werden. Der Aufrufer muss damit rechnen, bei einem bestimmten Aufruf Teile einer anderen Datei zu erhalten, die nicht in einer sequenziellen Reihenfolge sind, oder Teile einer beliebigen geöffneten Datenbankdatei zu erhalten.

Eine Anwendung sollte wiederholte Aufrufe dieses Vorgangs ausführen, bis die Größe der Lesedaten 0 oder der Wert von `sqlca->sqlvar[0].sqlind` größer als 0 ist. Wenn die Sicherung mit einer Umbenennung und einem Neustart des Transaktionslogs gestartet wird, könnte `SQLERROR` auf `SQL Backup_Cannot_Rename_Log_Yet` gesetzt werden. In diesem Fall verwenden Sie nicht die zurückgegebenen Seiten, sondern erneuern Sie die Anforderung, bis Sie `SQL_NOERROR` erhalten, und schreiben Sie dann die Daten. Die Fehlermeldung `SQL Backup_Cannot_Rename_Log_Yet` kann mehrere Male und für mehrere Seiten auftreten. In Ihrer Wiederholungsschleife sollten Sie eine Verzögerung einbauen, damit der Server nicht mit zu vielen Anforderungen verlangsamt wird. Fahren Sie fort, die Seiten zu lesen, bis eine der beiden Bedingungen eingetreten ist.

Das Dienstprogramm `dbbackup` verwendet folgenden Algorithmus. Es handelt sich *nicht* um C-Code und es ist keine Fehlerprüfung enthalten.

```
sqlca->sqlc = 1;
sqlca->sqlvar[0].sqltype = DT_LONGBINARY

/* Allocate LONGBINARY value for page buffer. It MUST have */
/* enough room to hold the requested number (128) of database pages */
sqlca->sqlvar[0].sqldata = allocated buffer

/* Open the server files needing backup */
for file_num = 0 to DB_BACKUP_MAX_FILE
  db_backup( ... DB_BACKUP_OPEN_FILE, file_num ... )
  if SQLCODE == SQL_NO_ERROR
    /* The file exists */
    num_pages = SQLCOUNT
    file_time = SQLIOESTIMATE
    open backup file with name from sqlca.sqlerrmc
  end for

/* read pages from the server, write them locally */
while TRUE
  /* file_no and page_no are ignored */
  db_backup( &sqlca, DB_BACKUP_PARALLEL_READ, 0, 0, &sqlca );

  if SQLCODE != SQL_NO_ERROR
    break;

  if buffer->stored_len == 0 || sqlca->sqlvar[0].sqlind > 0
    break;

  /* SQLCOUNT contains the starting page number of the block */
  /* SQLIOESTIMATE contains the file number the pages belong to */
  write block of pages to appropriate backup file
end while

/* close the server backup files */
for file_num = 0 to DB_BACKUP_MAX_FILE
  /* close backup file */
  db_backup( ... DB_BACKUP_CLOSE_FILE, file_num ... )
end for

/* shut down the backup */
db_backup( ... DB_BACKUP_END ... )

/* cleanup */
free page buffer
```

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Der SQL-Deskriptor-Bereich (SQLDA)“ auf Seite 504
- „BACKUP-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Datentypen in Embedded SQL“ auf Seite 482

db_cancel_request-Funktion

Syntax

```
int db_cancel_request( SQLCA * sqlca );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.

Rückgabe

1, wenn die Abbruchanforderung gesendet wird, 0, wenn keine Anforderung gesendet wird.

Bemerkungen

Diese Funktion bricht die gerade aktive Anforderung an den Datenbankserver ab. Sie überprüft, ob die Anforderung an den Datenbankserver aktiv ist, bevor sie die Abbruchsanforderung sendet.

Ein Rückgabewert ungleich 0 bedeutet nicht, dass die Anforderung abgebrochen wurde. Es gibt einige kritische zeitliche Überschneidungen, wenn sich die Abbruchanforderung und die Antwort von der Datenbank oder vom Server überschneiden. In diesen Fällen findet einfach kein Abbruch statt, obwohl die Funktion weiterhin TRUE zurückgibt.

Die Funktion db_cancel kann asynchron aufgerufen werden. Diese und die Funktion db_is_working sind die einzigen Funktionen in der Schnittstellenbibliothek der Datenbank, die asynchron aufgerufen werden können, weil sie einen SQLCA-Bereich verwenden, der schon von einer anderen Anforderung verwendet werden könnte.

Wenn Sie eine Anforderung abbrechen, die einen Cursorvorgang durchführt, ist die Position des Cursors unbestimmt. Sie müssen den Cursor entweder mit seiner absoluten Position festlegen oder ihn nach dem Abbrechen schließen.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495

db_change_char_charset-Funktion

Syntax

```
unsigned int db_change_char_charset(  
SQLCA * sqlca,  
char * charset );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **charset** Eine Zeichenfolge, die den Zeichensatz darstellt.

Rückgabe

1, wenn die Änderung erfolgreich war, andernfalls 0.

Bemerkungen

Ändert den CHAR-Zeichensatz der Anwendung für diese Verbindung. Mit DT_FIXCHAR-, DT_VARCHAR-, DT_LONGVARCHAR- und DT_STRING-Typen gesendete und abgerufene Daten sind im CHAR-Zeichensatz enthalten.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Empfohlene Zeichensätze und Kollationen“ [[SQL Anywhere Server - Datenbankadministration](#)]

db_change_nchar_charset-Funktion

Syntax

```
unsigned int db_change_nchar_charset(  
SQLCA * sqlca,  
char * charset );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **charset** Eine Zeichenfolge, die den Zeichensatz darstellt.

Rückgabe

1, wenn die Änderung erfolgreich war, andernfalls 0.

Bemerkungen

Ändert den NCHAR-Zeichensatz der Anwendung für diese Verbindung. Mit den Hostvariablen-Typen DT_NFIXCHAR, DT_NVARCHAR, DT_LONGNVARCHAR und DT_NSTRING gesendete und abgerufene Daten sind im NCHAR-Zeichensatz enthalten.

Wenn die Funktion db_change_nchar_charset nicht aufgerufen wird, werden alle Daten mit dem CHAR-Zeichensatz gesendet und abgerufen. Gewöhnlich sollte eine Anwendung, die Unicode-Daten senden und abrufen will, den NCHAR-Zeichensatz auf UTF-8 setzen.

Wenn diese Funktion aufgerufen wird, ist der charset-Parameter üblicherweise UTF-8. Der NCHAR-Zeichensatz kann nicht auf UTF-16 gesetzt werden.

In Embedded SQL werden NCHAR, NVARCHAR and LONG NVARCHAR standardmäßig als DT_FIXCHAR, DT_VARCHAR bzw. DT_LONGVARCHAR beschrieben. Wenn die Funktion

db_change_nchar_charset aufgerufen wurde, werden diese Typen als DT_NFIXCHAR, DT_NVARCHAR bzw. DT_LONGNVARCHAR beschrieben.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Empfohlene Zeichensätze und Kollationen“ [*SQL Anywhere Server - Datenbankadministration*]

db_find_engine-Funktion

Syntax

```
unsigned short db_find_engine(  
    SQLCA * sqlca,  
    char * name );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **name** NULL oder eine Zeichenfolge mit dem Namen des Servers.

Rückgabe

Server Status als kurze Ganzzahl ohne Vorzeichen, oder 0, wenn kein Server über Shared Memory gefunden wurde.

Bemerkungen

Diese Funktion gibt eine kurze Ganzzahl ohne Vorzeichen zurück, die Statusinformationen zum lokalen Datenbankserver namens *name* anzeigt. Wenn kein Datenbankserver mit dem angegebenen Namen über den gemeinsam genutzten Speicher gefunden wird, ist der Rückgabewert 0. Ein Nicht-Null-Wert zeigt an, dass der lokale Server derzeit ausgeführt wird.

Falls ein NULL-Zeiger für *name* angegeben wird, gibt die Funktion Informationen zum voreingestellten Datenbankserver zurück.

Jedes Bit des Rückgabewerts enthält eine spezifische Information. Die Header-Datei *sqldef.h* definiert Konstanten für diese Informationen. Ihre Bedeutung wird unten beschrieben.

- **DB_ENGINE** Diese Markierung ist immer gesetzt.
- **DB_CLIENT** Diese Markierung ist immer gesetzt.
- **DB_CAN_MULTI_DB_NAME** Diese Markierung wird nicht mehr empfohlen.
- **DB_DATABASE_SPECIFIED** Diese Markierung ist immer gesetzt.
- **DB_ACTIVE_CONNECTION** Diese Markierung ist immer gesetzt.
- **DB_CONNECTION_DIRTY** Diese Markierung wird nicht mehr empfohlen.
- **DB_CAN_MULTI_CONNECT** Diese Markierung wird nicht mehr empfohlen.

- **DB_NO_DATABASES** Diese Markierung wird gesetzt, wenn der Datenbankserver keine gestarteten Datenbanken hat.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)

db_fini-Funktion

Syntax

```
int db_fini( SQLCA * sqlca );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.

Rückgabe

Von 0 verschiedener Wert bei Erfolg, ansonsten 0.

Bemerkungen

Diese Funktion gibt Ressourcen frei, die von der Datenbankschnittstelle benutzt wurden. Nachdem Sie db_fini aufgerufen haben, sind keine weiteren Bibliotheksaufrufe und keine weiteren Embedded SQL-Anweisungen erlaubt. Wenn während der Verarbeitung ein Fehler auftritt, wird der Fehlercode in SQLCA eingestellt und die Funktion gibt 0 zurück. Wenn es keine Fehler gegeben hat, wird ein Wert ungleich NULL zurückgegeben.

Sie müssen db_fini einmal für jeden benutzten SQLCA-Bereich aufrufen.

Die db_fini-Funktion darf in einer Windows-DLL nicht direkt oder indirekt von der DllMain-Funktion aufgerufen werden. Die DllMain-Eintrittspunktfunktion soll nur einfache Initialisierungs- und Beendigungsaufgaben ausführen. Der Aufruf von db_fini kann Deadlocks und Zirkeldefinitionen bewirken.

Für UltraLite-Anwendungen gibt es eine entsprechende db_fini-Methode.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)
- [„db_fini-Methode“ \[*UltraLite - C- und C++-Programmierung*\]](#)

db_get_property-Funktion

Syntax

```
unsigned int db_get_property(  
    SQLCA * sqlca,  
    a_db_property property,  
    char * value_buffer,  
    int value_buffer_size );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **property** Die angeforderte Eigenschaft, entweder DB_PROP_CLIENT_CHARSET, DB_PROP_SERVER_ADDRESS oder DB_PROP_DBLIB_VERSION.
- **value_buffer** Dieses Argument wird mit dem Eigenschaftswert als Zeichenfolge mit einer abschließenden Null gefüllt.
- **value_buffer_size** Die maximale Länge der Zeichenfolge value_buffer, einschließlich des Platzes für das abschließende Nullzeichen.

Rückgabe

1, wenn erfolgreich, sonst 0.

Bemerkungen

Diese Funktion wird verwendet, um Informationen zur Datenbankschnittstelle oder zum Server, mit dem eine Verbindung besteht, zu erhalten.

Folgende Eigenschaften werden unterstützt:

- **DB_PROP_CLIENT_CHARSET** Dieser Eigenschaftswert ruft den Client-Zeichensatz ab (z.B. "Windows-1252").
- **DB_PROP_SERVER_ADDRESS** Diese Eigenschaft ruft die Servernetzwerkadresse der derzeitigen Verbindung als druckbare Zeichenfolge ab. Das Protokoll für gemeinsam genutzten Speicher gibt immer die leere Zeichenfolge für die Adresse zurück. Das TCP/IP-Protokoll gibt nicht-leere Zeichenfolgenadressen zurück.
- **DB_PROP_DBLIB_VERSION** Dieser Eigenschaftswert ruft die Version der Datenbank-Schnittstellenbibliothek (z.B. "16.0.1297") ab.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)

db_init-Funktion

Syntax

```
int db_init( SQLCA * sqlca );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Diese Funktion initialisiert die Schnittstellenbibliothek der Datenbank. Diese Funktion muss aufgerufen werden, bevor ein anderer Bibliotheksaufruf erfolgen kann und bevor eine Embedded SQL-Anweisung ausgeführt wird. Die Ressourcen, die die Schnittstellenbibliothek für Ihr Programm braucht, werden bei diesem Aufruf zugewiesen und initialisiert.

Geben Sie mit `db_fini` am Ende Ihres Programms die Ressourcen frei. Falls während der Prozessverarbeitung Fehler auftreten, werden sie im SQLCA-Bereich zurückgegeben und der Rückgabewert ist 0 (Null). Treten keine Fehler auf, ist der Rückgabewert ungleich 0 (Null) und Sie können beginnen, Embedded SQL-Anweisungen und -Funktionen zu verwenden.

Normalerweise sollte diese Funktion nur einmal aufgerufen werden, um die Adresse der globalen Variablen `sqlca` weiterzugeben, die in der Header-Datei `sqlca.h` definiert ist. Falls Sie mit Embedded SQL eine DLL oder eine Anwendung schreiben, die mehrfache Threads hat, rufen Sie `db_init` einmal für jeden SQLCA-Bereich auf, der benutzt wird.

Für UltraLite-Anwendungen gibt es eine entsprechende `db_init`-Methode.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „SQLCA-Verwaltung für Code mit mehreren Threads oder "reentrant"-Code“ auf Seite 498
- „`db_init`-Methode“ [*UltraLite - C- und C++-Programmierung*]

db_is_working-Funktion

Syntax

```
unsigned short db_is_working( SQLCA * sqlca );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.

Rückgabe

Diese Funktion gibt 1 zurück, falls eine Datenbankanforderung Ihrer Anwendung läuft, die den angegebenen `sqlca` benutzt. Sie gibt 0 (Null) zurück, falls keine Anwendung läuft, die den angegebenen `sqlca` benutzt.

Bemerkungen

Diese Funktion kann asynchron aufgerufen werden. Diese und die Funktion `db_cancel_request` sind die einzigen Funktionen in der Schnittstellenbibliothek der Datenbank, die asynchron aufgerufen werden können, weil sie einen SQLCA-Bereich verwenden, der schon von einer anderen Anforderung verwendet werden könnte.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495

db_locate_servers-Funktion

Syntax

```
unsigned int db_locate_servers(
    SQLCA * sqlca,
    SQL_CALLBACK_PARM callback_address,
    void * callback_user_data );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **callback_address** Die Adresse einer Callback-Funktion.
- **callback_user_data** Die Adresse eines benutzerdefinierten Bereichs, in dem Daten gespeichert werden sollen.

Rückgabe

1, wenn erfolgreich, sonst 0.

Bemerkungen

Bietet Zugriff auf die vom Befehlszeilen-Dienstprogramm dblocate angezeigten Daten und listet alle SQL Anywhere-Datenbankserver mit TCP/IP-Anschluss im lokalen Netzwerk auf.

Die Callback-Funktion muss folgende Syntax haben:

```
int (*)( SQLCA * sqlca,
    a_server_address * server_addr,
    void * callback_user_data );
```

Die Callback-Funktion wird für jeden gefundenen Server ausgeführt. Wenn die Callback-Funktion den Wert 0 zurückgibt, stoppt db_locate_servers die Suche nach Servern.

Die an die Callback-Funktion übergebenen Parameter sqlca und callback_user_data sind dieselben, die an db_locate_servers übergeben wurden. Der zweite Parameter ist ein Zeiger auf eine a_server_address-Struktur. a_server_address wird mit folgender Definition in *sqlca.h* festgelegt:

```
typedef struct a_server_address {
    a_sql_uint32 port_type;
    a_sql_uint32 port_num;
    char        *name;
    char        *address;
} a_server_address;
```

- **port_type** Ist hier immer PORT_TYPE_TCP (laut Definition 6 in *sqlca.h*)
- **port_num** Ist die TCP-Portnummer, die der Server abhört
- **name** Zeigt auf einen Puffer, der den Servernamen enthält
- **address** Zeigt auf einen Puffer, der die IP-Adresse des Servers enthält

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Dienstprogramm für die Serverauflistung (dblocate)“ [*SQL Anywhere Server - Datenbankadministration*]

db_locate_servers_ex-Funktion**Syntax**

```
unsigned int db_locate_servers_ex(
    SQLCA * sqlca,
    SQL_CALLBACK_PARM callback_address,
    void * callback_user_data,
    unsigned int bitmask);
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **callback_address** Die Adresse einer Callback-Funktion.
- **callback_user_data** Die Adresse eines benutzerdefinierten Bereichs, in dem Daten gespeichert werden sollen.
- **bitmask** Eine Maske, die einen oder mehrere der folgenden Parameter enthält:
DB_LOOKUP_FLAG_NUMERIC, DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT und
DB_LOOKUP_FLAG_DATABASES.

Rückgabe

1, wenn erfolgreich, sonst 0.

Bemerkungen

Bietet programmatischen Zugriff auf die vom Befehlszeilen-Dienstprogramm dblocate angezeigten Daten und listet alle SQL Anywhere-Datenbankserver mit TCP/IP-Anschluss im lokalen Netzwerk auf, und bietet einen Mask-Parameter, der benutzt wird, um Adressen auszuwählen, die an die Callback-Funktion weitergegeben werden.

Die Callback-Funktion muss folgende Syntax haben:

```
int (*)( SQLCA * sqlca,
    a_server_address * server_addr,
    void * callback_user_data );
```

Die Callback-Funktion wird für jeden gefundenen Server ausgeführt. Wenn die Callback-Funktion den Wert "0" zurückgibt, stoppt db_locate_servers_ex die Suche nach Servern.

Die an die Callback-Funktion übergebenen Parameter sqlca und callback_user_data sind dieselben, die an db_locate_servers übergeben wurden. Der zweite Parameter ist ein Zeiger auf eine a_server_address-Struktur. a_server_address wird mit folgender Definition in *sqlca.h* festgelegt:

```
typedef struct a_server_address {
    a_sql_uint32    port_type;
```

```

    a_sql_uint32  port_num;
    char          *name;
    char          *address;
    char          *dbname;
} a_server_address;

```

- **port_type** Ist hier immer PORT_TYPE_TCP (laut Definition 6 in *sqlca.h*)
- **port_num** Ist die TCP-Portnummer, die der Server abhört
- **name** Zeigt auf einen Puffer, der den Servernamen enthält
- **address** Zeigt auf einen Puffer, der die IP-Adresse des Servers enthält
- **dbname** Zeigt auf einen Puffer, der den Datenbanknamen enthält

Drei Bitmask-Optionen werden unterstützt:

- DB_LOOKUP_FLAG_NUMERIC
- DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
- DB_LOOKUP_FLAG_DATABASES

Diese Optionen sind in *sqlca.h* definiert und können durch OR verknüpft werden.

DB_LOOKUP_FLAG_NUMERIC stellt sicher, dass Adressen, die an die Callback-Funktion übergeben werden, anstelle von Hostnamen IP-Adressen sind.

DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT legt fest, dass die Adresse in der Struktur *a_server_address*, die an die Callback-Funktion übergeben wird, die TCP/IP-Portnummer enthält.

DB_LOOKUP_FLAG_DATABASES legt fest, dass die Callback-Funktion für jede gefundene Datenbank einmal aufgerufen wird oder einmal für jeden gefundenen Datenbankserver, wenn der Datenbankserver das Senden von Datenbankinformationen nicht unterstützt (Version 9.0.2 und frühere Datenbankserver).

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Dienstprogramm für die Serveraufflistung (dblocate)“ [*SQL Anywhere Server - Datenbankadministration*]

db_register_a_callback-Funktion

Syntax

```

void db_register_a_callback(
    SQLCA * sqlca,
    a_db_callback_index index,
    ( SQL_CALLBACK_PARM ) callback );

```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.

- **index** Ein Index-Wert, der den Callback-Typ angibt, wie im Folgenden beschrieben.
- **callback** Die Adresse einer benutzerdefinierten Callback-Funktion.

Bemerkungen

Diese Funktion registriert Callback-Funktionen:

Falls Sie keine Callback-Funktion `DB_CALLBACK_WAIT` registrieren, ist die voreingestellte Reaktion, nichts zu tun. Ihre Anwendung wird blockiert, während sie auf die Antwort der Datenbank wartet. Sie müssen eine Callback-Funktion für die `MESSAGE TO CLIENT`-Anweisung registrieren.

Um einen Callback zu löschen, übergeben Sie einen Null-Zeiger als *callback*-Funktion.

Die folgenden Werte sind für den Parameter *index* erlaubt:

- **DB_CALLBACK_DEBUG_MESSAGE** Die bereitgestellte Funktion wird für jede Debug-Meldung einmal aufgerufen. Ihr wird eine auf NULL endende Zeichenfolge übergeben, die den Text der Debug-Meldung enthält. Eine Debug-Meldung ist eine Meldung, die in der LogFile-Datei protokolliert wird. Damit eine Debug-Meldung an dieses Callback übergeben wird, muss der LogFile-Verbindungsparameter verwendet werden. Die Zeichenfolge enthält gewöhnlich eine Zeilenendmarke (`\n`) unmittelbar vor dem beendenden NULL-Zeichen. Der Prototyp der Callback-Funktion lautet folgendermaßen:

```
void SQL_CALLBACK debug_message_callback(
SQLCA * sqlca,
char * message_string );
```

- **DB_CALLBACK_START** Der Prototyp sieht wie folgt aus:

```
void SQL_CALLBACK start_callback( SQLCA * sqlca );
```

Diese Funktion wird unmittelbar vor dem Absenden einer Datenbankanforderung an den Server aufgerufen. `DB_CALLBACK_START` wird nur unter Windows verwendet.

- **DB_CALLBACK_FINISH** Der Prototyp sieht wie folgt aus:

```
void SQL_CALLBACK finish_callback( SQLCA * sqlca );
```

Diese Funktion wird aufgerufen, nachdem die Antwort auf eine Datenbankanforderung von der DBLIB-Schnittstellen-DLL empfangen wurde. `DB_CALLBACK_FINISH` wird nur unter Windows-Betriebssystemen verwendet.

- **DB_CALLBACK_CONN_DROPPED** Der Prototyp sieht wie folgt aus:

```
void SQL_CALLBACK conn_dropped_callback (
SQLCA * sqlca,
char * conn_name );
```

Diese Funktion wird aufgerufen, wenn der Datenbankserver im Begriff ist, die Verbindung aufgrund eines Verfügbarkeits-Timeouts zu verlieren, von einer `DROP CONNECTION`-Anweisung oder weil der Server heruntergefahren wird. Der Verbindungsname *conn_name* wird übergeben, sodass Sie

zwischen den Verbindungen unterscheiden können. Wenn die Verbindung nicht benannt war, hat sie den Wert NULL.

- **DB_CALLBACK_WAIT** Der Prototyp sieht wie folgt aus:

```
void SQL_CALLBACK wait_callback( SQLCA * sqlca );
```

Diese Funktion wird wiederholt von der Schnittstellenbibliothek aufgerufen, während der Datenbankserver oder die Clientbibliothek mit dem Abarbeiten Ihrer Datenbankanforderung beschäftigt sind.

So würden Sie diesen Callback registrieren:

```
db_register_a_callback( &sqlca,
    DB_CALLBACK_WAIT,
    (SQL_CALLBACK_PARM)&db_wait_request );
```

- **DB_CALLBACK_MESSAGE** Diese Funktion ermöglicht es der Anwendung, Nachrichten vom Server zu verarbeiten, während dieser eine Anforderung abarbeitet. Nachrichten können vom Datenbankserver an die Clientanwendung unter Verwendung der SQL MESSAGE-Anweisung gesendet werden. Nachrichten können auch von lang laufenden Datenbankserver-Anweisungen generiert werden.

Der Callback-Prototyp sieht wie folgt aus:

```
void SQL_CALLBACK message_callback(
    SQLCA * sqlca,
    unsigned char msg_type,
    an_sql_code code,
    unsigned short length,
    char * msg
);
```

Der *msg_type*-Parameter gibt an, wie wichtig die Nachricht ist. Unterschiedliche Nachrichtentypen können verschieden behandelt werden. Die folgenden zulässigen Werte für *Nachrichtentyp* sind in *sqldef.h* festgelegt.

- **MESSAGE_TYPE_INFO** Der Nachrichtentyp war INFO.
- **MESSAGE_TYPE_WARNING** Der Nachrichtentyp war WARNING.
- **MESSAGE_TYPE_ACTION** Der Nachrichtentyp war ACTION.
- **MESSAGE_TYPE_STATUS** Der Nachrichtentyp war STATUS.
- **MESSAGE_TYPE_PROGRESS** Der Nachrichtentyp war PROGRESS. Dieser Nachrichtentyp wird von lang laufenden Datenbankserver-Anweisungen wie BACKUP DATABASE und LOAD TABLE generiert.

Das Feld *code* kann einen SQLCODE übergeben, der der Nachricht zugeordnet ist. Anderfalls ist der Wert "0". Das Feld *length* gibt die Länge der Nachricht an. Die Meldung endet *nicht* mit NULL. SQL Anywhere DBLIB und ODBC-Clients können mithilfe der DB_CALLBACK_MESSAGE-Parameter Meldungen zum Verarbeitungsfortschritt empfangen.

Zum Beispiel zeigt die Interactive SQL Callback-Funktion STATUS- und INFO-Nachrichten im Nachrichtenfenster an, während Nachrichten vom Typ ACTION und WARNING in einem Fenster erscheinen. Wenn eine Anwendung diese Callback-Funktion nicht registriert, gibt es eine Standard-Callback-Funktion, die alle Nachrichten in die Serverlogdatei schreibt (wenn der Debug-Modus eingestellt und eine Logdatei angegeben ist). Zusätzlich werden Nachrichten vom Typ MESSAGE_TYPE_WARNING und MESSAGE_TYPE_ACTION betriebssystemabhängig auffälliger dargestellt.

Wenn ein Nachrichten-Callback von der Anwendung nicht registriert wird, werden an den Client gesendete Nachrichten in der Logdatei gespeichert, sofern der LogFile-Verbindungsparameter angegeben ist. Außerdem werden auf Windows-Betriebssystemen an den Client gesendete ACTION- oder STATUS-Nachrichten in einem Fenster angezeigt, und auf Unix-Betriebssystemen in stderr protokolliert.

- **DB_CALLBACK_VALIDATE_FILE_TRANSFER** Dies wird verwendet, um eine Callback-Funktion zur Dateiübertragungsvalidierung zu registrieren. Bevor sie eine Übertragung zulässt, ruft die Clientbibliothek das Validierungs-Callback auf, falls vorhanden. Falls die Client-Datenübertragung während der Ausführung von indirekten Anweisungen angefordert wird, z.B. aus einer gespeicherten Prozedur heraus, lässt die Clientbibliothek die Übertragung nur zu, wenn die Clientanwendung ein Validierungs-Callback registriert hat. Die Bedingungen, unter denen ein Validierungsaufruf durchgeführt wird, werden unten ausführlich beschrieben.

Der Callback-Prototyp sieht wie folgt aus:

```
int SQL_CALLBACK file_transfer_callback(
SQLCA * sqlca,
char * file_name,
int is_write
);
```

Der Parameter *file_name* ist der Name der zu lesenden oder zu schreibenden Datei. Der Parameter *is_write* ist 0, wenn ein Lesevorgang (Übertragung vom Client zum Datenbankserver) angefordert wird, und Nicht-Null bei einem Schreibvorgang. Die Callback-Funktion sollte 0 zurückgeben, wenn die Übertragung nicht zulässig ist, ansonsten Nicht-Null.

Aus Gründen der Datensicherheit protokolliert der Datenbankserver den Ursprung von Anweisungen, die eine Dateiübertragung anfordern. Der Datenbankserver ermittelt, ob die Anweisung direkt von der Clientanwendung empfangen wurde. Wenn er die Datenübertragung vom Client initiiert, sendet der Datenbankserver die Informationen über den Ursprung der Anweisung an die Clientsoftware. Seinerseits erlaubt die Embedded SQL Clientbibliothek nur dann eine bedingungslose Übertragung, wenn die Datenübertragung aufgrund der Ausführung einer Anweisung angefordert wird, die direkt von der Clientanwendung gesendet wird. Ansonsten muss die Anwendung den oben beschriebenen Validierungs-Callback registriert haben. Wenn dieser fehlt, wird die Übertragung verweigert und die Anweisung schlägt mit einem Fehler fehl. Wenn die Clientanweisung eine bereits in der Datenbank vorhandene gespeicherte Prozedur aufruft, wird die Ausführung der gespeicherten Prozedur selbst nicht als vom Client initiierte Anweisung angesehen. Wenn die Clientanwendung allerdings explizit eine temporäre gespeicherte Prozedur erstellt, führt die Ausführung der gespeicherten Prozedur dazu, dass der Datenbankserver die Prozedur als eine vom Client initiierte behandelt. Wenn die Clientanwendung eine Batchanweisung ausführt, wird auf gleiche Weise die Ausführung der Batchanweisung als eine angesehen, die direkt von der Clientanwendung durchgeführt wird.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter LogFile (LOG)“ [*SQL Anywhere Server - Datenbankadministration*]
- „MESSAGE-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „progress_messages-Option“ [*SQL Anywhere Server - Datenbankadministration*]

db_start_database-Funktion

Syntax

```
unsigned int db_start_database( SQLCA * sqlca, char * parms );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **parms** Eine auf NULL endende Zeichenfolge, die eine durch Semikolons getrennte Liste mit Parametereinstellungen enthält, und zwar in der Form KEYWORD=Wert. Beispiel:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Die Datenbank wird auf einem bestehenden Server gestartet, wenn dies möglich ist. Sonst wird ein neuer Server gestartet.

Wenn die Datenbank bereits lief oder erfolgreich gestartet wurde, ist der Rückgabewert TRUE (von Null verschieden) und SQLCODE wird auf 0 gesetzt. Fehlerinformationen werden in SQLCA zurückgegeben.

Falls eine Benutzer-ID und ein Kennwort in den Parametern übergeben wurden, werden sie ignoriert.

Welches Privileg zum Starten und Stoppen einer Datenbank erforderlich ist, wird mit der Option -gd in der Serverbefehlszeile festgelegt.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]
- „Fehlerbehandlung: Datenbankserver suchen“ [*SQL Anywhere Server - Datenbankadministration*]
- „Datenbankserveroption -gd“ [*SQL Anywhere Server - Datenbankadministration*]

db_start_engine-Funktion

Syntax

```
unsigned int db_start_engine( SQLCA * sqlca, char * parms );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **parms** Eine auf NULL endende Zeichenfolge, die eine durch Semikolons getrennte Liste mit Parametereinstellungen enthält, und zwar in der Form **KEYWORD=Wert**. Beispiel:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Startet den Datenbankserver, falls er nicht läuft.

Wenn der Datenbankserver bereits lief oder erfolgreich gestartet wurde, ist der Rückgabewert TRUE (von Null verschieden) und SQLCODE wird auf 0 gesetzt. Fehlerinformationen werden in SQLCA zurückgegeben.

Der folgende Aufruf von db_start_engine startet den Datenbankserver, lädt die angegebene Datenbank und benennt den Server "demo".

```
db_start_engine( &sqlca, "DBF=demo.db;START=dbsrv16" );
```

Sofern nicht der ForceStart-Verbindungsparameter (FORCE) verwendet und auf YES gesetzt wurde, versucht die Funktion db_start_engine, eine Verbindung mit einem Server herzustellen, bevor einer gestartet wird, um zu verhindern, dass ein Serverstart für einen Server versucht wird, der bereits läuft.

Wenn der ForceStart-Verbindungsparameter auf YES gesetzt ist, wird nicht versucht, eine Verbindung zu einem Server herzustellen, bevor einer gestartet wird. Dadurch kann das folgende Befehlspaar wie erwartet arbeiten:

1. Starten eines Datenbankservers mit dem Namen server_1:

```
dbsrv16 -n server_1 demo.db
```

2. Der Start eines neuen Servers wird erzwungen und es wird eine Verbindung zu ihm hergestellt:

```
db_start_engine( &sqlca,
  "START=dbsrv16 -n server_2 mydb.db;ForceStart=YES" )
```

Wenn ForceStart (FORCE) nicht verwendet wird und der ServerName-Parameter (Server) nicht benutzt wird, würde der zweite Befehl versuchen, eine Verbindung zu server_1 herzustellen. Die Funktion db_start_engine übernimmt nicht den Servernamen von der Option -n des Parameters StartLine (START).

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]
- „Fehlerbehandlung: Datenbankserver suchen“ [*SQL Anywhere Server - Datenbankadministration*]

db_stop_database-Funktion

Syntax

```
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **parms** Eine auf NULL endende Zeichenfolge, die eine durch Semikolons getrennte Liste mit Parametereinstellungen enthält, und zwar in der Form KEYWORD=Wert. Beispiel:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Stoppt die Datenbank, die von DatabaseName (DBN) auf dem von ServerName (Server) angegebenen Server identifiziert wurde. Falls ServerName nicht angegeben ist, bezieht sich die Funktion auf den voreingestellten Server.

Als Voreinstellung beendet diese Funktion eine Datenbank nicht, solange noch eine Verbindung zu dieser Datenbank offen ist. Wenn Unconditional (UNC) den Wert **yes** hat, wird die Datenbank beendet, auch wenn noch Verbindungen offen sind.

Ist der Rückgabewert TRUE, sind keine Fehler aufgetreten.

Welches Privileg zum Starten und Stoppen einer Datenbank erforderlich ist, wird mit der Option -gd in der Serverbefehlszeile festgelegt.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]
- „Datenbankserveroption -gd“ [*SQL Anywhere Server - Datenbankadministration*]

db_stop_engine-Funktion

Syntax

```
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **parms** Eine auf NULL endende Zeichenfolge, die eine durch Semikolons getrennte Liste mit Parametereinstellungen enthält, und zwar in der Form KEYWORD=Wert. Beispiel:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Diese Funktion stoppt die Ausführung des Datenbankservers. Folgende Schritte werden von dieser Funktion ausgeführt:

- Sie sucht den lokalen Datenbankserver, dessen Name dem Wert des Parameters `ServerName` (Server) entspricht. Falls `ServerName` nicht angegeben wurde, sucht sie nach dem voreingestellten lokalen Datenbankserver.
- Wenn kein übereinstimmender Server gefunden wird, wird diese Funktion erfolgreich beendet.
- Die Funktion sendet eine Anforderung an den Server, alle Datenbanken zu finden und zu beenden.
- Sie entlädt den Datenbankserver.

Als Voreinstellung beendet diese Funktion einen Datenbankserver nicht, solange noch eine Verbindung zu dieser Datenbank offen ist. Wenn der Verbindungsparameter `Unconditional=yes` lautet, wird der Datenbankserver beendet, auch wenn noch Verbindungen offen sind.

Ein C-Programm kann diese Funktion nutzen, anstatt `dbstop` aufzurufen. Ist der Rückgabewert `TRUE`, sind keine Fehler aufgetreten.

Die Verwendung von `db_stop_engine` hängt von den Privilegien ab, die mit der Serveroption `-gk` festgelegt wurden.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]
- „Datenbankserveroption `-gk`“ [*SQL Anywhere Server - Datenbankadministration*]

db_string_connect-Funktion

Syntax

```
unsigned int db_string_connect( SQLCA * sqlca, char * parms );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **parms** Eine auf NULL endende Zeichenfolge, die eine durch Semikolons getrennte Liste mit Parametereinstellungen enthält, und zwar in der Form `KEYWORD=Wert`. Beispiel:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Liefert zusätzliche Funktionalität über die Embedded SQL-Anweisung CONNECT hinaus.

Der von dieser Funktion verwendeten Algorithmus wird unter dem Thema "Fehlerbehandlung" beschrieben.

Der Rückgabewert ist TRUE (ungleich Null), falls eine Verbindung zustande kam, andernfalls ist er FALSE (Null). Informationen zu Fehlern beim Starten des Servers, beim Starten der Datenbank oder beim Verbindungsaufbau werden im SQLCA-Bereich zurückgegeben.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]
- „Fehlerbehandlung: Verbindungen“ [*SQL Anywhere Server - Datenbankadministration*]

db_string_disconnect-Funktion

Syntax

```
unsigned int db_string_disconnect(  
    SQLCA * sqlca,  
    char * parms );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **parms** Eine auf NULL endende Zeichenfolge, die eine durch Semikolons getrennte Liste mit Parametereinstellungen enthält, und zwar in der Form KEYWORD=Wert. Beispiel:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

Rückgabe

Ein von 0 verschiedener Wert, wenn erfolgreich, andernfalls 0.

Bemerkungen

Diese Funktion trennt die Verbindung, die mit dem Parameter ConnectionName angegeben wurde. Alle anderen Parameter werden ignoriert.

Falls der ConnectionName-Parameter in der Zeichenfolge fehlt, wird die unbenannte Verbindung getrennt. Dies entspricht der Embedded SQL-Anweisung DISCONNECT. Der Rückgabewert ist TRUE, falls eine Verbindung erfolgreich getrennt wurde. Informationen zu Fehlern werden im SQLCA-Bereich zurückgegeben.

Diese Funktion fährt die Datenbank herunter, falls sie mit dem Verbindungsparameter AutoStop=yes gestartet wurde, und falls keine anderen Verbindungen mit der Datenbank bestehen. Es beendet auch den Serverbetrieb, falls der Server mit dem Parameter AutoStop=yes gestartet wurde, und falls keine weitere Datenbank mehr auf dem Server läuft.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]

db_string_ping_server-Funktion

Syntax

```
unsigned int db_string_ping_server(  
SQLCA * sqlca,  
char * connect_string,  
unsigned int connect_to_db );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **connect_string** Die *connect_string* ist eine normale Verbindungszeichenfolge, die eventuell Server- und Datenbankinformationen enthält.
- **connect_to_db** Wenn *connect_to_db* nicht gleich Null (d.h. TRUE) ist, dann versucht die Funktion eine Verbindung zu einer Datenbank auf einem Server herzustellen. Der Wert TRUE wird nur zurückgegeben, wenn die Verbindungszeichenfolge ausreicht, um eine Verbindung zu einer benannten Datenbank auf dem benannten Server herzustellen.

Wenn *connect_to_db* gleich Null ist, versucht die Funktion lediglich, die Position eines Servers zu ermitteln. Der Wert TRUE wird nur zurückgegeben, wenn die Verbindungszeichenfolge ausreicht, um die Position eines Servers zu ermitteln. Es wird nicht versucht, eine Verbindung zur Datenbank herzustellen.

Rückgabe

TRUE (ungleich Null), falls der Server oder die Datenbank wurde, sonst FALSE (Null). Fehlerinformationen bei der Suche nach dem Server oder der Datenbank werden in den SQLCA zurückgegeben.

Bemerkungen

Diese Funktion kann verwendet werden, um zu ermitteln, ob ein Server gefunden werden kann und optional ob eine erfolgreiche Verbindung zu einer Datenbank hergestellt werden kann.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]

db_time_change-Funktion

Syntax

```
unsigned int db_time_change(  
SQLCA * sqlca);
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.

Rückgabe

TRUE, wenn erfolgreich, andernfalls FALSE.

Bemerkungen

Diese Funktion gestattet es dem Client, den Server darüber zu informieren, dass die Zeit auf dem Client geändert wurde. Diese Funktion berechnet die Zeitzonenanpassung und sendet sie an den Server. Auf Windows-Plattformen sollten Anwendungen diese Funktion aufrufen, wenn sie die Nachricht WM_TIMECHANGE empfangen. Damit stellen Sie sicher, dass UTC-Zeitstempel auch bei Zeitänderungen, Änderungen von Zeitzonen oder den Wechsel zur Sommerzeit konsistent sind.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)

fill_s_sqlda-Funktion

Syntax

```
struct sqlda * fill_s_sqlda(  
    struct sqlda * sqlda,  
    unsigned int maxlen );
```

Parameter

- **sqlda** Ein Zeiger auf eine SQLCA-Struktur.
- **maxlen** Die maximale Anzahl der für die Zeichenfolge zuzuweisenden Bytes.

Rückgabe

sqlda, wenn erfolgreich, und NULL, wenn nicht genügend Speicherplatz zur Verfügung stand.

Bemerkungen

Diese Funktion entspricht fill_sqlda, abgesehen davon, dass sie alle Datentypen in *sqlda* in den Datentyp DT_STRING umwandelt. Es wird genügend Speicherplatz zugewiesen, um eine Repräsentation des Datentyps als Zeichenfolge zu speichern, der ursprünglich im SQLDA-Bereich angegeben wurde, und zwar bis zu maximal *maxlen* Byte. Die Längenfelder im SQLDA-Bereich (sqllen) werden dementsprechend geändert.

Die SQLDA sollte mit der Funktion free_filled_sqlda freigegeben werden.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)

fill_sqlda-Funktion

Syntax

```
struct sqlda * fill_sqlda( struct sqlda * sqlda );
```

Parameter

- **sqlda** Ein Zeiger auf eine SQLCA-Struktur.

Rückgabe

sqlda, wenn erfolgreich, und NULL, wenn nicht genügend Speicherplatz zur Verfügung stand.

Bemerkungen

Diese Funktion weist allen Variablen, die in den Deskriptoren von *sqlda* beschrieben sind, Speicherplatz zu und ordnet die Speicheradressen dem Feld *sqlda* des entsprechenden Deskriptors zu. Es wird genügend Speicherplatz zugewiesen für den im Deskriptor angegebenen Datenbanktyp und die angegebene Länge.

Die SQLDA sollte mit der Funktion *free_filled_sqlda* freigegeben werden.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „fill_sqlda_ex-Funktion“ auf Seite 557
- „free_filled_sqlda-Funktion“ auf Seite 558

fill_sqlda_ex-Funktion

Syntax

```
struct sqlda * fill_sqlda_ex( struct sqlda * sqlda , unsigned int flags);
```

Parameter

- **sqlda** Ein Zeiger auf eine SQLCA-Struktur.
- **flags** 0 oder FILL_SQLDA_FLAG_RETURN_DT_LONG

Rückgabe

sqlda, wenn erfolgreich, und NULL, wenn nicht genügend Speicherplatz zur Verfügung stand.

Bemerkungen

Diese Funktion weist allen Variablen, die in den Deskriptoren von *sqlda* beschrieben sind, Speicherplatz zu und ordnet die Speicheradressen dem Feld *sqlda* des entsprechenden Deskriptors zu. Es wird genügend Speicherplatz zugewiesen für den im Deskriptor angegebenen Datenbanktyp und die angegebene Länge.

Die SQLDA sollte mit der Funktion *free_filled_sqlda* freigegeben werden.

Ein Parameter-Bit wird unterstützt: FILL_SQLDA_FLAG_RETURN_DT_LONG. Dieser Parameter wird in *sqlca.h* definiert.

FILL_SQLDA_FLAG_RETURN_DT_LONG behält die Datentypen DT_LONGVARCHAR, DT_LONGNVARCHAR und DT_LONGBINARY im ausgefüllten Deskriptor bei. Wenn dieses Parameter-Bit nicht angegeben wird, konvertiert fill_sqlda_ex die Datentypen DT_LONGVARCHAR, DT_LONGNVARCHAR und DT_LONGBINARY in DT_VARCHAR, DT_NVARCHAR und DT_BINARY. Die Verwendung der Typen DT_LONGxyz ermöglicht das Abrufen von 32767 Byte, und nicht der 32765 Byte, auf die DT_VARCHAR, DT_NVARCHAR und DT_BINARY beschränkt sind.

fill_sqlda(SQLDA) ist gleichwertig mit fill_sqlda_ex(sqlda, 0).

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „fill_sqlda-Funktion“ auf Seite 557
- „free_filled_sqlda-Funktion“ auf Seite 558

free_filled_sqlda-Funktion

Syntax

```
void free_filled_sqlda( struct sqlda * sqlda );
```

Parameter

- **sqlda** Ein Zeiger auf eine SQLCA-Struktur.

Bemerkungen

Diese Funktion gibt den Speicherplatz frei, der einem sqldata-Zeiger und dem SQLDA-Bereich selbst zugewiesen war. Null-Zeiger werden nicht freigegeben.

Diese Funktion sollte nur aufgerufen werden, wenn fill_sqlda, fill_sqlda_ex oder fill_s_sqlda verwendet wurde, um die sqldata-Felder der SQLDA zuzuweisen.

Durch Aufrufen dieser Funktion wird automatisch auch free_sqlda aufgerufen und alle Deskriptoren, die durch alloc_sqlda zugewiesen waren, werden freigegeben.

Siehe auch

- „SQL-Kommunikationsbereich (SQLCA)“ auf Seite 495
- „alloc_sqlda-Funktion“ auf Seite 532
- „fill_s_sqlda-Funktion“ auf Seite 556
- „fill_sqlda-Funktion“ auf Seite 557
- „fill_sqlda_ex-Funktion“ auf Seite 557
- „free_sqlda-Funktion“ auf Seite 558

free_sqlda-Funktion

Syntax

```
void free_sqlda( struct sqlda * sqlda );
```

Parameter

- **sqllda** Ein Zeiger auf eine SQLCA-Struktur.

Bemerkungen

Diese Funktion gibt Speicherplatz, der diesem *sqllda* zugewiesen wurde, und den Speicherplatz der Indikatorvariablen, der in *fill_sqllda* zugewiesen wurde, frei. Sie sollten den von den *sqldata*-Zeigern referenzierten Speicherplatz nicht freigeben.

Siehe auch

- „Der SQL-Deskriptor-Bereich (SQLDA)“ auf Seite 504
- „*alloc_sqllda*-Funktion“ auf Seite 532
- „*fill_s_sqllda*-Funktion“ auf Seite 556
- „*fill_sqllda*-Funktion“ auf Seite 557
- „*fill_sqllda_ex*-Funktion“ auf Seite 557

free_sqllda_noind-Funktion

Syntax

```
void free_sqllda_noind( struct sqllda * sqllda );
```

Parameter

- **sqllda** Ein Zeiger auf eine SQLCA-Struktur.

Bemerkungen

Diese Funktion gibt den Speicherplatz frei, der dem *sqllda* zugewiesen war. Sie sollten den von den *sqldata*-Zeigern referenzierten Speicherplatz nicht freigeben. Die Indikatorvariablen-Zeiger werden nicht beachtet.

Siehe auch

- „Der SQL-Deskriptor-Bereich (SQLDA)“ auf Seite 504
- „*alloc_sqllda*-Funktion“ auf Seite 532
- „*fill_s_sqllda*-Funktion“ auf Seite 556
- „*fill_sqllda*-Funktion“ auf Seite 557
- „*fill_sqllda_ex*-Funktion“ auf Seite 557

sql_needs_quotes-Funktion

Syntax

```
unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **str** Eine Zeichenfolge, die als SQL-Bezeichner verwendet werden könnte.

Rückgabe

TRUE oder FALSE, womit angezeigt wird, ob eine Zeichenfolge in Anführungszeichen eingeschlossen werden muss, wenn sie als SQL-Bezeichner benutzt wird.

Bemerkungen

Sie formuliert eine Anforderung an den Datenbankserver, um festzustellen, ob Anführungszeichen nötig sind. Die relevante Information wird im Feld `sqlcode` gespeichert.

Wir unterscheiden drei unterschiedliche Kombinationen von Rückgabewert und Code:

- **return = FALSE, sqlcode = 0** Die Zeichenfolge benötigt keine Anführungszeichen.
- **return = TRUE** Der `sqlcode` ist immer `SQLE_WARNING`, und die Zeichenfolge erfordert Anführungszeichen.
- **return = FALSE** Falls `sqlcode` mit etwas anderem als 0 oder `SQLE_WARNING` belegt ist, ist das Testergebnis unklar.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)

sqlda_storage-Funktion

Syntax

```
a_sql_uint32 sqlda_storage( struct sqlda * sqlda, int varno );
```

Parameter

- **sqlda** Ein Zeiger auf eine SQLCA-Struktur.
- **varno** Ein Index für eine `sqlvar`-Hostvariable.

Rückgabe

32-Bit-Ganzzahlwert ohne Vorzeichen, der die Speichergröße angibt, die erforderlich wäre, um jeden möglichen Wert der in `sqlda->sqlvar[varno]` beschriebenen Variablen zu speichern.

Siehe auch

- [„Der SQL-Deskriptor-Bereich \(SQLDA\)“ auf Seite 504](#)

sqlda_string_length-Funktion

Syntax

```
a_sql_uint32 sqlda_string_length( struct sqlda * sqlda, int varno );
```

Parameter

- **sqlda** Ein Zeiger auf eine SQLCA-Struktur.

- **varno** Ein Index für eine sqlvar-Hostvariable.

Rückgabe

32-Bit-Ganzzahlwert, der die Länge der C-Zeichenfolge angibt (Typ DT_STRING), die erforderlich wäre, um die Variable `sqllda->sqlvar[varno]` zu speichern (unabhängig vom Datentyp).

Siehe auch

- [„Der SQL-Deskriptor-Bereich \(SQLDA\)“ auf Seite 504](#)

sqlerror_message-Funktion

Syntax

```
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

Parameter

- **sqlca** Ein Zeiger auf eine SQLCA-Struktur.
- **buffer** Der Puffer, in dem die Meldung (bis zu *max* Zeichen) gespeichert werden soll.
- **max** Die Maximallänge des Puffers.

Rückgabe

Ein Zeiger auf eine Zeichenfolge, die eine Fehlermeldung enthält, oder NULL, wenn kein Fehler angezeigt wurde.

Bemerkungen

Gibt einen Zeiger auf eine Zeichenfolge zurück, die eine Fehlermeldung enthält. Die Fehlermeldung enthält den Text für den Fehlercode im SQLCA-Bereich. Wurde kein Fehler gemeldet, wird ein Null-Zeiger zurückgegeben. Die Fehlermeldung wird in dem gelieferten Puffer gespeichert, falls erforderlich, gekürzt auf die Länge *max*.

Siehe auch

- [„SQL-Kommunikationsbereich \(SQLCA\)“ auf Seite 495](#)

Zusammenfassung der Embedded SQL-Anweisungen

Alle Embedded SQL-Anweisungen müssen mit den Worten EXEC SQL beginnen und mit einem Semikolon (;) enden.

Es gibt zwei Gruppen von Embedded SQL-Anweisungen: Standard-SQL-Anweisungen werden verwendet, indem sie einfach in ein C-Programm geschrieben werden, eingeschlossen von EXEC SQL und einem Semikolon (;). CONNECT, DELETE, SELECT, SET und UPDATE haben zusätzliche

Formate, die nur in Embedded SQL zur Verfügung stehen. Die zusätzlichen Formate gehören zur zweiten Gruppe der Embedded SQL-spezifischen Anweisungen.

Eine Beschreibung der Standard-SQL-Anweisungen finden Sie unter „SQL-Anweisungen“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Mehrere SQL-Anweisungen sind spezifisch für Embedded SQL und können nur innerhalb eines C-Programms benutzt werden. Siehe „SQL-Sprachelemente“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Standardmäßige Datenbearbeitung und Datendefinitions-Anweisungen können von Embedded SQL-Anwendungen verwendet werden. Darüber hinaus gelten die folgenden Anweisungen speziell für die Programmierung von Embedded SQL:

- **„ALLOCATE DESCRIPTOR-Anweisung [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Speicher für einen Deskriptor zuweisen.
- **„CLOSE-Anweisung [ESQL] [SP]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Einen Cursor schließen.
- **„CONNECT-Anweisung [ESQL] [Interactive SQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Mit der Datenbank verbinden.
- **„DEALLOCATE DESCRIPTOR-Anweisung [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Speicher freigeben, der für einen Deskriptor reserviert war.
- **„Deklarationsabschnitt [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Hostvariablen für die Kommunikation mit der Datenbank deklarieren.
- **„DECLARE CURSOR-Anweisung [ESQL] [SP]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Einen Cursor deklarieren.
- **„DELETE-Anweisung (positionsbasiert) [ESQL] [SP]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Die Zeilen an der aktuellen Cursorposition löschen.
- **„DESCRIBE-Anweisung [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Die Hostvariablen für eine bestimmte SQL-Anweisung beschreiben.
- **„DISCONNECT-Anweisung [ESQL] [Interactive SQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Die Verbindung mit dem Datenbankserver lösen.
- **„DROP STATEMENT-Anweisung [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Ressourcen freigeben, die von einem Prepared-Statement benutzt werden.
- **„EXECUTE-Anweisung [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Eine bestimmte SQL-Anweisung ausführen.
- **„EXPLAIN-Anweisung [ESQL]“** [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] Die Strategie zur Optimierung eines bestimmten Cursors erklären.

- **„FETCH-Anweisung [ESQL] [SP]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Eine Zeile aus einem Cursor abrufen.
- **„GET DATA-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Lange Werte aus einem Cursor abrufen.
- **„GET DESCRIPTOR-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Angaben zu einer Variablen in einen SQLDA-Bereich abrufen.
- **„GET OPTION-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Die Einstellung einer bestimmten Datenbankoption abholen.
- **„INCLUDE-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Eine Datei ins SQL-Preprocessing einschließen.
- **„OPEN-Anweisung [ESQL] [SP]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Einen Cursor öffnen.
- **„PREPARE-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Eine bestimmte SQL-Anweisung vorbereiten.
- **„PUT-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Eine Zeile in einen Cursor einfügen.
- **„SET CONNECTION-Anweisung [Interactive SQL] [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Die aktive Verbindung ändern.
- **„SET DESCRIPTOR-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Die Variablen in einem SQLDA-Bereich beschreiben und Daten im SQLDA-Bereich ablegen.
- **„SET SQLCA-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Einen anderen als den global voreingestellten SQLCA-Bereich verwenden.
- **„UPDATE-Anweisung (positionsbasiert) [ESQL] [SP]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Die Zeile an der aktuellen Cursorposition aktualisieren.
- **„WHENEVER-Anweisung [ESQL]“ [SQL Anywhere Server - SQL-Referenzhandbuch]** Angeben, welche Aktionen bei Fehlern in SQL-Anweisungen erfolgen sollen.

SQL Anywhere-Datenbank-API für C/C++

Die SQL Anywhere-C-API ist eine Anwendungsprogrammierschnittstelle für die C- / C++-Sprachen. Die C-API-Spezifikation definiert eine Gruppe von Funktionen, Variablen und Konventionen, die eine konsistente Datenbankschnittstelle unabhängig von der gerade benutzten Datenbank bereitstellen. Wenn die SQL Anywhere C-API verwendet wird, haben Ihre C- / C++-Anwendungen direkten Zugriff auf SQL Anywhere-Datenbankserver.

SQL Anywhere-C-API-Unterstützung

Die SQL Anywhere-C-API ist eine Anwendungsprogrammierschnittstelle für die C- / C++-Sprachen. Die C-API-Spezifikation definiert eine Gruppe von Funktionen, Variablen und Konventionen, die eine konsistente Datenbankschnittstelle unabhängig von der gerade benutzten Datenbank bereitstellen. Wenn die SQL Anywhere C-API verwendet wird, haben Ihre C- / C++-Anwendungen direkten Zugriff auf SQL Anywhere-Datenbankserver.

Die SQL Anywhere-C-API vereinfacht die Erstellung von C- und C++-Wrapper-Treibern für mehrere interpretierte Programmiersprachen, einschließlich PHP, Perl, Python und Ruby. Die SQL Anywhere C-API befindet sich eine Ebene über der DBLIB-Bibliothek und wurde mit Embedded SQL implementiert.

Auch wenn sie kein Ersatz für DBLIB ist, vereinfacht die SQL Anywhere C-API die Erstellung von Anwendungen mit C und C++. Sie benötigen keine umfassenden Kenntnisse von Embedded SQL, um die SQL Anywhere C-API zu verwenden.

API-Verteilung

Die API ist als DLL (Dynamic Link Library --dynamische Verknüpfungsbibliothek) *dbcapi.dll* auf Microsoft Windows-Systemen und als Shared Object *libdbcapi.so* auf Unix-Systemen verfügbar. Die DLL ist statisch mit dem DBLIB-Paket der SQL Anywhere-Version verbunden, auf der sie aufgebaut ist. Wenn die *dbcapi.dll*-Datei geladen wird, wird die entsprechende *dblibX.dll*-Datei vom Betriebssystem geladen. Anwendungen, die *dbcapi.dll* verwenden, können sich entweder direkt mit ihr verbinden oder sie dynamisch laden.

Beschreibungen der SQL Anywhere C-API-Datentypen und der Eintrittspunkte sind in der Header-Datei *sacapi.h* im Verzeichnis *sdk\dbcapi* Ihrer SQL Anywhere-Installation zu finden.

Threading-Unterstützung

Die SQL Anywhere C-API-Bibliothek ist "nicht thread-fähig", d.h. die Bibliothek führt keine Aufgaben aus, die einen gegenseitigen Ausschluss erfordern. Damit die Bibliothek in Thread-Anwendungen arbeiten kann, ist nur eine Anforderung in einer einzigen Verbindung zulässig. Basierend auf dieser Regel ist die Anwendung für den gegenseitigen Ausschluss verantwortlich, wenn sie auf eine verbindungsspezifische Ressource zugreift. Dies umfasst Verbindung-Handles, vorbereitete Anweisungen und Ergebnismengenobjekte.

Beispiele

Beispiele, die zeigen, wie Sie die SQL Anywhere C-API einsetzen können, befinden sich im Verzeichnis *sdk\dbcapi\examples* Ihrer SQL Anywhere-Installation.

- ***connecting.cpp*** Dieses Beispiel zeigt, wie Sie ein Verbindungsobjekt erstellen und mit dessen Hilfe eine Verbindung zu SQL Anywhere herstellen.
- ***dbcapi_isql.cpp*** Dieses Beispiel zeigt, wie Sie eine ISQL-ähnliche Anwendung erstellen.
- ***fetching_a_result_set.cpp*** Dieses Beispiel zeigt, wie Sie Daten aus einer Ergebnismenge abrufen.
- ***fetching_multiple_from_sp.cpp*** Dieses Beispiel zeigt, wie Sie mehrere Ergebnismengen aus einer gespeicherten Prozedur abrufen.
- ***preparing_statements.cpp*** Dieses Beispiel zeigt, wie Sie eine Anweisung vorbereiten und ausführen.
- ***send_retrieve_full_blob.cpp*** Dieses Beispiel zeigt, wie Sie einen Blob in einem Abschnitt einfügen und abrufen.
- ***send_retrieve_part_blob.cpp*** Dieses Beispiel zeigt, wie Sie einen Blob in Abschnitten einfügen und ihn auch in Abschnitten abrufen.

Dynamisches Laden der Schnittstellenbibliothek

Der Code zum dynamischen Laden der DLL ist in der Quelldatei *sacapidll.c* enthalten, die sich im Verzeichnis *sdk\dbcapi* Ihrer SQL Anywhere-Installation befindet. Anwendungen müssen die Header-Datei *sacapidll.h* verwenden und den Quellcode in *sacapidll.c* einbeziehen. Sie können die Methode *sqlany_initialize_interface* verwenden, um die DLL dynamisch zu laden und die Eintrittspunkte zu ermitteln. Beispiele werden mit der SQL Anywhere-Installation eingerichtet.

SQL Anywhere-C-API-Referenz

Headerdateien

- *sacapi.h*
- *sacapidll.h*

Bemerkungen

Die Headerdatei *sacapi.h* definiert die Eintrittspunkte für die SQL Anywhere-C-API.

Die Headerdatei *sacapidll.h* definiert die Initialisierungs- und Finalisierungsfunktionen für die C-API-Bibliothek. Sie müssen *sacapidll.h* in Ihre Quelldatei aufnehmen und den Quellcode aus *sacapidll.c* angeben.

sqlany_affected_rows-Methode

Gibt die Anzahl der Zeilen zurück, die von der Ausführung der vorbereiteten Anweisung betroffen sind.

Syntax

```
public sacapi_i32 sqlany_affected_rows(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die vorbereitet war und erfolgreich ausgeführt wurde und von der keine Ergebnismenge zurückgegeben wurde. Beispiel: Eine INSERT-, UPDATE- oder DELETE-Anweisung wurde ausgeführt.

Rückgabe

Die Anzahl der betroffenen Zeilen oder -1 bei Fehlschlag.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)

sqlany_bind_param-Methode

Bindet einen benutzerdefinierten Puffer als Parameter an die vorbereitete Anweisung.

Syntax

```
public sacapi_bool sqlany_bind_param(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_bind_param * param
)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare()` vorbereitet wurde.
- **index** Der Index des Parameters. Diese Zahl muss zwischen 0 und (`sqlany_num_params()` - 1) liegen.
- **param** Eine `a_sqlany_bind_param`-Strukturbeschreibung des zu bindenden Parameters.

Rückgabe

1 bei Erfolg oder 0 bei Fehlschlag.

Siehe auch

- [sqlany_describe_bind_param-Methode \[SQL Anywhere C\] auf Seite 570](#)

sqlany_cancel-Methode

Bricht eine ausstehende Anforderung auf einer Verbindung ab.

Syntax

```
public void sqlany_cancel(a_sqlany_connection * sqlany_conn)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect()` hergestellten Verbindung.

sqlany_clear_error-Methode

Löscht den letzten gespeicherten Fehlercode.

Syntax

```
public void sqlany_clear_error(a_sqlany_connection * sqlany_conn)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt, das von `sqlany_new_connection()` zurückgegeben wird.

Siehe auch

- [sqlany_new_connection-Methode \[SQL Anywhere C\] auf Seite 585](#)

sqlany_client_version-Methode

Gibt die aktuelle Clientversion zurück.

Syntax

```
public sacapi_bool sqlany_client_version(char * buffer, size_t len)
```

Parameter

- **buffer** Der Puffer, der mit der Clientversion-Zeichenfolge gefüllt werden soll.
- **len** Die Länge des bereitgestellten Puffers.

Rückgabe

1, wenn erfolgreich, oder 0, wenn nicht erfolgreich.

Bemerkungen

Diese Methode füllt den übergebenen Puffer mit der Hauptversions-, Unterversions-, Patch- und Build-Nummer der Clientbibliothek. Der Puffer wird mit einem Nullwert abgeschlossen.

sqlany_client_version_ex-Methode

Gibt die aktuelle Clientversion zurück.

Syntax

```
public sacapi_bool sqlany_client_version_ex(  
    a_sqlany_interface_context * context,  
    char * buffer,  
    size_t len  
)
```

Parameter

- **context** Ein mit `sqlany_init_ex()` erstelltes Objekt.
- **buffer** Der Puffer, der mit der Clientversion-Zeichenfolge gefüllt werden soll.
- **len** Die Länge des bereitgestellten Puffers.

Rückgabe

1, wenn erfolgreich, oder 0, wenn nicht erfolgreich.

Bemerkungen

Diese Methode füllt den übergebenen Puffer mit der Hauptversions-, Unterversions-, Patch- und Build-Nummer der Clientbibliothek. Der Puffer wird mit einem Nullwert abgeschlossen.

Siehe auch

- [sqlany_init_ex-Methode \[SQL Anywhere C\] auf Seite 583](#)

sqlany_commit-Methode

Schreibt die aktuelle Transaktion fest.

Syntax

```
public sacapi_bool sqlany_commit(a_sqlany_connection * sqlany_conn)
```

Parameter

- **sqlany_conn** Das Verbindungsobjekt, auf dem der Festschreibungsvorgang durchgeführt werden soll.

Rückgabe

1, wenn erfolgreich, oder 0, wenn nicht erfolgreich.

Siehe auch

- [sqlany_rollback-Methode \[SQL Anywhere C\] auf Seite 588](#)

sqlany_connect-Methode

Erstellt eine Verbindung zu einem SQL Anywhere-Datenbankserver, wobei das angegebene Verbindungsobjekt und die angegebene Verbindungszeichenfolge verwendet werden.

Syntax

```
public sacapi_bool sqlany_connect(  
    a_sqlany_connection * sqlany_conn,  
    const char * str  
)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt, das durch `sqlany_new_connection()` erstellt wurde.
- **str** Eine SQL Anywhere-Verbindungszeichenfolge.

Rückgabe

1, wenn die Verbindung erfolgreich hergestellt wird, oder 0, wenn die Verbindung fehlschlägt.
Verwenden Sie `sqlany_error()`, um den Fehlercode und die Fehlermeldung abzurufen.

Bemerkungen

Das bereitgestellte Verbindungsobjekt muss zunächst mit `sqlany_new_connection()` zugewiesen werden.

Das folgende Beispiel zeigt, wie Sie den Fehlercode für einen gescheiterten Verbindungsversuch abrufen:

```
a_sqlany_connection * sqlany_conn;  
sqlany_conn = sqlany_new_connection();  
if( !sqlany_connect( sqlany_conn, "uid=dba;pwd=sql" ) ) {  
    char reason[SACAPI_ERROR_SIZE];  
    sacapi_i32 code;  
    code = sqlany_error( sqlany_conn, reason, sizeof(reason) );  
    printf( "Connection failed. Code: %d Reason: %s\n", code, reason );  
} else {  
    printf( "Connected successfully!\n" );  
    sqlany_disconnect( sqlany_conn );  
}  
sqlany_free_connection( sqlany_conn );
```

Weitere Hinweise zum Herstellen der Verbindung mit einem SQL Anywhere-Datenbankserver finden Sie unter „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)] und „Datenbankverbindungen“ [[SQL Anywhere Server - Datenbankadministration](#)].

Siehe auch

- [sqlany_new_connection-Methode](#) [SQL Anywhere C] auf Seite 585
- [sqlany_error-Methode](#) [SQL Anywhere C] auf Seite 572

sqlany_describe_bind_param-Methode

Beschreibt die Bindungsparameter einer vorbereiteten Anweisung.

Syntax

```
public sacapi_bool sqlany_describe_bind_param(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 index,  
    a_sqlany_bind_param * param  
)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare()` vorbereitet wurde.
- **index** Der Index des Parameters. Diese Zahl muss zwischen 0 und (`sqlany_num_params()` - 1) liegen.
- **param** Die `a_sqlany_bind_param`-Struktur, die mit Informationen gefüllt wird.

Rückgabe

1, wenn erfolgreich, oder 0, wenn nicht erfolgreich.

Bemerkungen

Diese Funktion ermöglicht es dem Aufrufer, Informationen über vorbereitete Anweisungsparameter festzulegen. Der Typ der vorbereiteten Anweisung (gespeicherte Prozedur oder DML) bestimmt die Menge der bereitgestellten Informationen. Die Richtung der Parameter (Eingabe, Ausgabe oder Eingabe/Ausgabe) wird immer übergeben.

Siehe auch

- [sqlany_bind_param-Methode \[SQL Anywhere C\] auf Seite 567](#)
- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_disconnect-Methode

Trennt eine bereits eingerichtete SQL Anywhere-Verbindung.

Syntax

```
public sacapi_bool sqlany_disconnect(a_sqlany_connection * sqlany_conn)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect()` hergestellten Verbindung.

Rückgabe

1, wenn erfolgreich, oder 0, wenn nicht erfolgreich.

Bemerkungen

Alle nicht festgeschriebenen Transaktionen werden zurückgesetzt.

Siehe auch

- [sqlany_connect-Methode \[SQL Anywhere C\] auf Seite 569](#)
- [sqlany_new_connection-Methode \[SQL Anywhere C\] auf Seite 585](#)

sqlany_error-Methode

Gibt den letzten Fehlercode und die dazugehörige Meldung zurück, die im Verbindungsobjekt gespeichert sind.

Syntax

```
public sacapi_i32 sqlany_error(  
    a_sqlany_connection * sqlany_conn,  
    char * buffer,  
    size_t size  
)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt, das von `sqlany_new_connection()` zurückgegeben wird.
- **buffer** Ein Puffer, der mit der Fehlermeldung gefüllt wird.
- **size** Die Größe des bereitgestellten Puffers.

Rückgabe

Der letzte Fehlercode. Positive Werte sind Warnungen, negative Werte sind Fehler, und 0 bedeutet Erfolg.

Bemerkungen

Weitere Hinweise zu SQLCODE-Fehlermeldungen finden Sie unter „[SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE](#)“ [[Fehlermeldungen](#)].

Siehe auch

- [sqlany_connect-Methode](#) [SQL Anywhere C] auf Seite 569

sqlany_execute-Methode

Führt eine vorbereitete Anweisung aus.

Syntax

```
public sacapi_bool sqlany_execute(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare()` vorbereitet wurde.

Rückgabe

1, wenn die Anweisung erfolgreich abgeschlossen wurde, oder 0 bei Fehlschlag.

Bemerkungen

Sie können `sqlany_num_cols()` verwenden, um zu überprüfen, ob die Anweisung eine Ergebnismenge zurückgegeben hat.

Das folgende Beispiel zeigt, wie Sie eine Anweisung ausführen, die keine Ergebnismenge zurückgibt:

```

a_sqlany_stmt *      stmt;
int                 i;
a_sqlany_bind_param param;

stmt = sqlany_prepare( sqlany_conn, "insert into moe(id,value)
values( ?,? )" );
if( stmt ) {
    sqlany_describe_bind_param( stmt, 0, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
    sqlany_bind_param( stmt, 0, &param );

    sqlany_describe_bind_param( stmt, 1, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
    sqlany_bind_param( stmt, 1, &param );

    for( i = 0; i < 10; i++ ) {
        if( !sqlany_execute( stmt ) ) {
            // call sqlany_error()
        }
    }
    sqlany_free_stmt( stmt );
}

```

Siehe auch

- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_execute_direct-Methode

Führt die SQL-Anweisung aus, die durch das Zeichenfolgenargument angegeben ist, und gibt möglicherweise eine Ergebnismenge zurück.

Syntax

```

public a_sqlany_stmt * sqlany_execute_direct(
    a_sqlany_connection * sqlany_conn,
    const char * sql_str
)

```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect()` hergestellten Verbindung.
- **sql_str** Eine SQL-Zeichenfolge. Die SQL-Zeichenfolge sollte keine Parameter wie z.B. ? enthalten.

Rückgabe

Ein Anweisungshandle, wenn die Funktion erfolgreich ausgeführt wird, oder NULL, wenn die Funktion nicht erfolgreich ausgeführt wird.

Bemerkungen

Verwenden Sie diese Methode, um eine Anweisung vorzubereiten und auszuführen, oder anstelle von `sqlany_prepare()`, gefolgt von `sqlany_execute()`.

Das folgende Beispiel zeigt, wie Sie eine Anweisung ausführen, die eine Ergebnismenge zurückgibt:

```
a_sqlany_stmt * stmt;

stmt = sqlany_execute_direct( sqlany_conn, "select * from employees" );
if( stmt && sqlany_num_cols( stmt ) > 0 ) {
    while( sqlany_fetch_next( stmt ) ) {
        int i;
        for( i = 0; i < sqlany_num_cols( stmt ); i++ ) {
            // Get column i data
        }
    }
    sqlany_free_stmt( stmt );
}
```

Hinweis

Diese Funktion kann nicht zum Ausführen einer SQL-Anweisung mit Parametern verwendet werden.

Siehe auch

- [sqlany_fetch_absolute-Methode \[SQL Anywhere C\] auf Seite 575](#)
- [sqlany_fetch_next-Methode \[SQL Anywhere C\] auf Seite 575](#)
- [sqlany_num_cols-Methode \[SQL Anywhere C\] auf Seite 586](#)
- [sqlany_get_column-Methode \[SQL Anywhere C\] auf Seite 578](#)

sqlany_execute_immediate-Methode

Führt die angegebene SQL-Anweisung unmittelbar aus, ohne eine Ergebnismenge zurückzugeben.

Syntax

```
public sacapi_bool sqlany_execute_immediate(
    a_sqlany_connection * sqlany_conn,
    const char * sql
)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect()` hergestellten Verbindung.
- **sql** Eine Zeichenfolge, die die auszuführende SQL-Anweisung darstellt.

Rückgabe

1 bei Erfolg oder 0 bei Fehlschlag.

Bemerkungen

Diese Funktion ist nützlich für SQL-Anweisungen, die keine Ergebnismenge zurückgeben.

sqlany_fetch_absolute-Methode

Verschiebt die aktuelle Zeile in der Ergebnismenge an die angegebene Zeilennummer und ruft anschließend die Daten in dieser Zeile ab.

Syntax

```
public sacapi_bool sqlany_fetch_absolute(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_i32 row_num
)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.
- **row_num** Die abzurufende Zeilennummer. Die erste Zeile ist 1, die letzte Zeile ist -1.

Rückgabe

1, wenn der Abruf erfolgreich ist, 0, wenn der Abruf nicht erfolgreich ist.

Siehe auch

- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_error-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_fetch_next-Methode \[SQL Anywhere C\] auf Seite 575](#)

sqlany_fetch_next-Methode

Gibt die nächste Zeile aus der Ergebnismenge zurück.

Syntax

```
public sacapi_bool sqlany_fetch_next(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.

Rückgabe

1, wenn der Abruf erfolgreich ist, 0, wenn der Abruf nicht erfolgreich ist.

Bemerkungen

Diese Funktion ruft die nächste Zeile aus der Ergebnismenge ab. Wenn das Ergebnisobjekt zum ersten Mal erstellt wird, wird der Zeiger für die aktuelle Zeile zunächst vor die erste Zeile (Zeile 0) gesetzt. Diese Funktion verschiebt zuerst den Zeilenzeiger nach vorne und ruft anschließend die Daten in der neuen Zeile ab.

Siehe auch

- [sqlany_fetch_absolute-Methode \[SQL Anywhere C\] auf Seite 575](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_error-Methode \[SQL Anywhere C\] auf Seite 572](#)

sqlany_finalize_interface-Methode

Entlädt die DLL-Bibliothek der C-API und setzt die SQLAnywhereInterface-Struktur zurück.

Syntax

```
public void sqlany_finalize_interface(SQLAnywhereInterface * api)
```

Parameter

- **api** Eine zu finalisierende initialisierte Struktur.

Bemerkungen

Verwenden Sie die folgende Anweisung, um den Funktionsprototyp einzubeziehen:

```
#include "sacapidll.h"
```

Verwenden Sie diese Methode, um Ressourcen, die der DLL für die SQL Anywhere-C-API zugeordnet sind, zu finalisieren und freizugeben.

Beispiele dafür, wie die `sqlany_finalize_interface`-Methode verwendet wird, finden Sie unter den C-API-Beispielen im Verzeichnis `sdk\dbcapi\examples` Ihrer SQL Anywhere-Installation.

sqlany_fini-Methode

Finalisiert die Schnittstelle.

Syntax

```
public void sqlany_fini()
```

Bemerkungen

Gibt durch die API zugewiesene Ressourcen frei.

Siehe auch

- [sqlany_init-Methode \[SQL Anywhere C\] auf Seite 582](#)

sqlany_fini_ex-Methode

Finalisiert die Schnittstelle, die unter Verwendung des angegebenen Kontexts erstellt wurde.

Syntax

```
public void sqlany_fini_ex(a_sqlany_interface_context * context)
```

Parameter

- **context** Ein Kontextobjekt, das aus `sqlany_init_ex()` zurückgegeben wurde.

Siehe auch

- [sqlany_init_ex-Methode \[SQL Anywhere C\] auf Seite 583](#)

sqlany_free_connection-Methode

Gibt die Ressourcen frei, die mit einem Verbindungsobjekt verbunden sind.

Syntax

```
public void sqlany_free_connection(a_sqlany_connection * sqlany_conn)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt, das mit `sqlany_new_connection()` erstellt wurde.

Siehe auch

- [sqlany_new_connection-Methode \[SQL Anywhere C\] auf Seite 585](#)

sqlany_free_stmt-Methode

Gibt Ressourcen frei, die mit einem vorbereiteten Anweisungsobjekt verbunden sind.

Syntax

```
public void sqlany_free_stmt(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch die erfolgreiche Ausführung von `sqlany_prepare()` oder `sqlany_execute_direct()` zurückgegeben wird.

Siehe auch

- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)

sqlany_get_bind_param_info-Methode

Ruft Informationen über die Parameter ab, die mit `sqlany_bind_param()` gebunden wurden.

Syntax

```
public sacapi_bool sqlany_get_bind_param_info(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 index,  
    a_sqlany_bind_param_info * info  
)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare()` vorbereitet wurde.
- **index** Der Index des Parameters. Diese Zahl sollte zwischen 0 und (`sqlany_num_params()` - 1) liegen.
- **info** Der `sqlany_bind_param_info`-Puffer, der mit Informationen des gebundenen Parameters gefüllt wird.

Rückgabe

1 bei Erfolg oder 0 bei Fehlschlag.

Siehe auch

- [sqlany_bind_param-Methode \[SQL Anywhere C\] auf Seite 567](#)
- [sqlany_describe_bind_param-Methode \[SQL Anywhere C\] auf Seite 570](#)
- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_get_column-Methode

Füllt den bereitgestellten Puffer mit dem Wert, der für die angegebene Spalte abgerufen wird.

Syntax

```
public sacapi_bool sqlany_get_column(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 col_index,  
    a_sqlany_data_value * buffer  
)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.
- **col_index** Die Nummer der abzurufenden Spalte. Eine Spaltennummer zwischen 0 und `sqlany_num_cols()` - 1.
- **buffer** Ein `a_sqlany_data_value`-Objekt, das mit den Daten gefüllt wird, die für die Spalte `col_index` abgerufen werden.

Rückgabe

1 bei Erfolg oder 0 bei Fehlschlag. Ein Fehlschlag kann auftreten, wenn ein Parameter ungültig ist oder wenn nicht genügend Speicher vorhanden ist, um den vollständigen Wert aus dem SQL Anywhere-Datenbankserver abzurufen.

Bemerkungen

Bei A_BINARY und A_STRING*-Datentypen zeigt "value->buffer" auf einen internen Puffer, der mit der Ergebnismenge verknüpft ist. Verwenden und ändern Sie den Inhalt des Zeigers nicht, da er sich ändert, wenn eine neue Zeile abgerufen oder das Ergebnismengenobjekt freigegeben wird. Benutzer sollten die Daten aus diesen Zeigern in eigene Puffer kopieren.

Das Feld "value->length" gibt die Anzahl der gültigen Zeichen an, auf die "value->buffer" zeigt. Die in "value->buffer" zurückgegebenen Daten sind nicht mit NULL abgeschlossen. Diese Funktion ruft alle zurückgegebenen Werte aus dem SQL Anywhere-Datenbankserver ab. Wenn die Spalte beispielsweise einen BLOB-Wert enthält, versucht diese Funktion, genügend Speicherplatz für diesen Wert zuzuweisen. Wenn Sie keinen Speicher zuweisen wollen, verwenden Sie stattdessen `sqlany_get_data()`.

Siehe auch

- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_fetch_absolute-Methode \[SQL Anywhere C\] auf Seite 575](#)
- [sqlany_fetch_next-Methode \[SQL Anywhere C\] auf Seite 575](#)

sqlany_get_column_info-Methode

Ruft Informationen über Spaltenmetadaten ab und füllt die `a_sqlany_column_info`-Struktur mit Informationen über die Spalte.

Syntax

```
public sacapi_bool sqlany_get_column_info(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    a_sqlany_column_info * buffer
)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_prepare()` oder `sqlany_execute_direct()` erstellt wurde.
- **col_index** Die Spaltennummer zwischen 0 und (`sqlany_num_cols()` - 1).
- **buffer** Eine Spalten-Info-Struktur, die mit Spalteninformationen gefüllt wird.

Rückgabe

1 bei Erfolg oder 0, wenn der Spaltenindex nicht im gültigen Wertebereich ist oder wenn die Anweisung keine Ergebnismenge zurückgibt.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_get_data-Methode

Ruft die Daten ab, die für die angegebene Spalte in den bereitgestellten Pufferspeicher abgerufen wurden.

Syntax

```
public sacapi_i32 sqlany_get_data(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 col_index,  
    size_t offset,  
    void * buffer,  
    size_t size  
)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.
- **col_index** Die Nummer der abzurufenden Spalte. Eine Spaltennummer zwischen 0 und `sqlany_num_cols() - 1`.
- **offset** Start-Offset der abzurufenden Daten.
- **buffer** Ein Puffer, der mit dem Inhalt der Spalte gefüllt wird. Der Pufferzeiger muss korrekt für den Datentyp ausgerichtet sein, der kopiert wird.
- **size** Die Größe des Puffers in Byte. Die Funktion schlägt fehl, wenn Sie eine Größe von mehr als $2^{31}-1$ angeben.

Rückgabe

Die Anzahl der Byte, die erfolgreich in den bereitgestellten Puffer kopiert wurden. Diese Zahl darf $2^{31}-1$ nicht überschreiten. 0 zeigt an, dass keine weiteren Daten zu kopieren sind. -1 zeigt einen Fehlschlag an.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_fetch_absolute-Methode \[SQL Anywhere C\] auf Seite 575](#)
- [sqlany_fetch_next-Methode \[SQL Anywhere C\] auf Seite 575](#)

sqlany_get_data_info-Methode

Fragt Informationen über die Daten ab, die durch den letzten Fetch-Vorgang abgerufen wurden.

Syntax

```
public sacapi_bool sqlany_get_data_info(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    a_sqlany_data_info * buffer
)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.
- **col_index** Die Spaltennummer zwischen 0 und (`sqlany_num_cols()` - 1).
- **buffer** Ein Dateninformationspuffer, der mit den Metadaten über die abgerufenen Daten gefüllt wird.

Rückgabe

1 bei Erfolg oder 0 bei Fehler. Ein Fehler wird zurückgegeben, wenn einer der gelieferten Parameter ungültig ist.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_fetch_absolute-Methode \[SQL Anywhere C\] auf Seite 575](#)
- [sqlany_fetch_next-Methode \[SQL Anywhere C\] auf Seite 575](#)

sqlany_get_next_result-Methode

Geht bei einer Abfrage mit mehreren Ergebnismengen zur nächsten Ergebnismenge.

Syntax

```
public sacapi_bool sqlany_get_next_result(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.

Rückgabe

1, wenn die Anweisung erfolgreich zur nächsten Ergebnismenge wechselt, ansonsten 0.

Bemerkungen

Wenn eine Abfrage (wie zum Beispiel ein Aufruf einer gespeicherten Prozedur) mehrere Ergebnismengen zurückgibt, wechselt diese Funktion aus der aktuellen Ergebnismenge zur nächsten.

Das folgende Beispiel zeigt, wie Sie in einer Abfrage mit mehreren Ergebnismengen zur nächsten Ergebnismenge wechseln:

```
stmt = sqlany_execute_direct( sqlany_conn, "call
my_multiple_results_procedure()" );
if( result ) {
    do {
        while( sqlany_fetch_next( stmt ) ) {
            // get column data
        }
    } while( sqlany_get_next_result( stmt ) );
    sqlany_free_stmt( stmt );
}
```

Siehe auch

- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)

sqlany_init-Methode

Initialisiert die Schnittstelle.

Syntax

```
public sacapi_bool sqlany_init(
    const char * app_name,
    sacapi_u32 api_version,
    sacapi_u32 * version_available
)
```

Parameter

- **app_name** Eine Zeichenfolge mit dem Namen der Anwendung, die die API verwendet. Beispiel: "PHP", "PERL" oder "RUBY".
- **api_version** Die Version der kompilierten Anwendung.
- **version_available** Ein optionales Argument für die Rückgabe der höchsten unterstützten API-Version.

Rückgabe

1 bei Erfolg, ansonsten 0.

Bemerkungen

Das folgende Beispiel zeigt, wie Sie die DLL für die SQL Anywhere-C-API initialisieren:

```
sacapi_u32 api_version;
if( sqlany_init( "PHP", SQLANY_API_VERSION_1, &api_version ) ) {
    printf( "Interface initialized successfully!\n" );
} else {
    printf( "Failed to initialize the interface! Supported version=%d\n",
api_version );
}
```

Siehe auch

- [sqlany_fini-Methode \[SQL Anywhere C\] auf Seite 576](#)

sqlany_init_ex-Methode

Initialisiert die Schnittstelle mit einem Kontext.

Syntax

```
public a_sqlany_interface_context * sqlany_init_ex(
    const char * app_name,
    sacapi_u32 api_version,
    sacapi_u32 * version_available
)
```

Parameter

- **app_name** Eine Zeichenfolge mit dem Namen der verwendeten API, zum Beispiel "PHP", "PERL" oder "RUBY".
- **api_version** Die aktuelle API-Version, die von der Anwendung verwendet wird. Dies sollte normalerweise eines der SQLANY_API_VERSION_*-Makros sein.
- **version_available** Ein optionales Argument für die Rückgabe der höchsten unterstützten API-Version.

Rückgabe

Ein Kontextobjekt bei Erfolg oder NULL bei Fehlschlag.

Siehe auch

- [sqlany_fini_ex-Methode \[SQL Anywhere C\] auf Seite 576](#)

sqlany_initialize_interface-Methode

Initialisiert das SQLAnywhereInterface-Objekt und lädt die DLL dynamisch.

Syntax

```
public int sqlany_initialize_interface(
    SQLAnywhereInterface * api,
    const char * optional_path_to_dll
)
```

Parameter

- **api** Eine zu initialisierende API-Struktur.
- **optional_path_to_dll** Ein optionales Argument, das einen Pfad zur DLL der SQL Anywhere-C-API angibt.

Rückgabe

1 bei erfolgreicher Initialisierung und 0 bei Fehlschlag.

Bemerkungen

Verwenden Sie die folgende Anweisung, um den Funktionsprototyp einzubeziehen:

```
#include "sacapidll.h"
```

Diese Funktion versucht, die SQL Anywhere C-API-DLL dynamisch zu laden, und ermittelt alle Eintrittspunkte der DLL. Die Felder in der `SQLAnywhereInterface`-Struktur werden gefüllt, um auf die entsprechenden Funktionen in der DLL zu zeigen. Wenn das optionale Pfadargument `NULL` ist, wird die Umgebungsvariable `SQLANY_DLL_PATH` überprüft. Wenn die Variable gesetzt ist, versucht die Bibliothek, die DLL zu laden, die durch die Umgebungsvariable angegeben ist. Wenn dies fehlschlägt, versucht die Schnittstelle, die DLL direkt zu laden (hierzu ist erforderlich, dass die Umgebung korrekt eingerichtet ist).

Beispiele dafür, wie die `sqlany_initialize_interface`-Methode verwendet wird, finden Sie unter den C-API-Beispielen im Verzeichnis `sdk\dbcapi\examples` Ihrer SQL Anywhere-Installation.

sqlany_make_connection-Methode

Erstellt ein Verbindungsobjekt, basierend auf einem gelieferten DBLIB SQLCA-Zeiger.

Syntax

```
public a_sqlany_connection * sqlany_make_connection(void * arg)
```

Parameter

- **arg** Ein leerer *-Zeiger auf ein DBLIB SQLCA-Objekt.

Rückgabe

Ein Verbindungsobjekt.

Siehe auch

- [sqlany_new_connection-Methode \[SQL Anywhere C\] auf Seite 585](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute_immediate-Methode \[SQL Anywhere C\] auf Seite 574](#)
- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_make_connection_ex-Methode

Erstellt ein Verbindungsobjekt, basierend auf einem gelieferten DBLIB SQLCA-Zeiger und Kontext.

Syntax

```
public a_sqlany_connection * sqlany_make_connection_ex(  
    a_sqlany_interface_context * context,  
    void * arg  
)
```

Parameter

- **context** Ein gültiges Kontextobjekt, das durch `sqlany_init_ex()` erstellt wurde.
- **arg** Ein leerer `*`-Zeiger auf ein DBLIB SQLCA-Objekt.

Rückgabe

Ein Verbindungsobjekt.

Siehe auch

- [sqlany_init_ex-Methode \[SQL Anywhere C\] auf Seite 583](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute_immediate-Methode \[SQL Anywhere C\] auf Seite 574](#)
- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_new_connection-Methode

Erstellt ein Verbindungsobjekt.

Syntax

```
public a_sqlany_connection * sqlany_new_connection(void)
```

Rückgabe

Ein Verbindungsobjekt.

Bemerkungen

Sie müssen ein API-Verbindungsobjekt erstellen, bevor Sie eine Datenbankverbindung herstellen. Fehler können aus dem Verbindungsobjekt abgerufen werden. Nur jeweils eine Anforderung kann auf einer Verbindung abgearbeitet werden. Zusätzlich darf jeweils nur ein Thread auf ein Verbindungsobjekt zugreifen. Undefiniertes Verhalten oder ein Fehlschlag kann auftreten, wenn mehrere Threads versuchen, gleichzeitig auf ein Verbindungsobjekt zuzugreifen.

Siehe auch

- [sqlany_connect-Methode \[SQL Anywhere C\] auf Seite 569](#)
- [sqlany_disconnect-Methode \[SQL Anywhere C\] auf Seite 571](#)

sqlany_new_connection_ex-Methode

Erstellt ein Verbindungsobjekt unter Verwendung eines Kontexts.

Syntax

```
public a_sqlany_connection * sqlany_new_connection_ex(  
    a_sqlany_interface_context * context  
)
```

Parameter

- **context** Ein Kontextobjekt, das aus `sqlany_init_ex()` zurückgegeben wurde.

Rückgabe

Ein Verbindungsobjekt.

Siehe auch

- [sqlany_connect-Methode \[SQL Anywhere C\] auf Seite 569](#)
- [sqlany_disconnect-Methode \[SQL Anywhere C\] auf Seite 571](#)
- [sqlany_init_ex-Methode \[SQL Anywhere C\] auf Seite 583](#)

sqlany_num_cols-Methode

Gibt die Anzahl der Spalten in der Ergebnismenge zurück.

Syntax

```
public sacapi_i32 sqlany_num_cols(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_prepare()` oder `sqlany_execute_direct()` erstellt wurde.

Rückgabe

Die Anzahl der Spalten in der Ergebnismenge oder -1 bei Fehlschlag.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_num_params-Methode

Gibt die Anzahl der Parameter zurück, die bei einer vorbereiteten Anweisung erwartet werden.

Syntax

```
public sacapi_i32 sqlany_num_params(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch die erfolgreiche Ausführung von `sqlany_prepare()` zurückgegeben wird.

Rückgabe

Die erwartete Anzahl von Parametern oder -1, wenn das Anweisungsobjekt nicht gültig ist.

Siehe auch

- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_num_rows-Methode

Gibt die Anzahl der Zeilen in der Ergebnismenge zurück.

Syntax

```
public sacapi_i32 sqlany_num_rows(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Ein Anweisungsobjekt, das durch `sqlany_execute()` oder `sqlany_execute_direct()` ausgeführt wurde.

Rückgabe

Die Anzahl der Zeilen in der Ergebnismenge Wenn die Anzahl der Zeilen eine Schätzung ist, ist die zurückgegebene Anzahl negativ und die Schätzung der absolute Wert der zurückgegebenen Ganzzahl. Der zurückgegebene Wert ist positiv, wenn die Anzahl der Zeilen exakt ist.

Bemerkungen

Standardmäßig gibt diese Funktion nur eine Schätzung zurück. Um eine exakte Zählung zurückzugeben, setzen Sie die `row_counts`-Option für die Verbindung. Weitere Hinweise zur `row_counts`-Option finden Sie unter „[row_counts-Option](#)“ [*SQL Anywhere Server - Datenbankadministration*].

Siehe auch

- [sqlany_execute_direct-Methode \[SQL Anywhere C\] auf Seite 573](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)

sqlany_prepare-Methode

Bereitet die gelieferte SQL-Zeichenfolge vor.

Syntax

```
public a_sqlany_stmt * sqlany_prepare(  
    a_sqlany_connection * sqlany_conn,  
    const char * sql_str  
)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect()` hergestellten Verbindung.
- **sql_str** Die vorzubereitende SQL-Anweisung..

Rückgabe

Ein Handle zu einem SQL Anywhere-Anweisungsobjekt. Das Anweisungsobjekt kann von `sqlany_execute()` verwendet werden, um die Anweisung auszuführen.

Bemerkungen

Die Ausführung erfolgt erst, wenn `sqlany_execute()` aufgerufen wird. Das zurückgegebene Anweisungsobjekt sollte mit `sqlany_free_stmt()` freigegeben werden.

Die folgende Anweisung zeigt, wie Sie eine SQL SELECT-Zeichenfolge vorbereiten:

```
char * str;
a_sqlany_stmt * stmt;

str = "select * from employees where salary >= ?";
stmt = sqlany_prepare( sqlany_conn, str );
if( stmt == NULL ) {
    // Failed to prepare statement, call sqlany_error() for more info
}
```

Siehe auch

- [sqlany_free_stmt-Methode \[SQL Anywhere C\] auf Seite 577](#)
- [sqlany_connect-Methode \[SQL Anywhere C\] auf Seite 569](#)
- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)
- [sqlany_num_params-Methode \[SQL Anywhere C\] auf Seite 586](#)
- [sqlany_describe_bind_param-Methode \[SQL Anywhere C\] auf Seite 570](#)
- [sqlany_bind_param-Methode \[SQL Anywhere C\] auf Seite 567](#)

sqlany_reset-Methode

Setzt eine Anweisung in ihren vorbereiteten Zustand zurück.

Syntax

```
public sacapi_bool sqlany_reset(a_sqlany_stmt * sqlany_stmt)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare()` vorbereitet wurde.

Rückgabe

1 bei Erfolg, 0 bei Fehlschlag.

Siehe auch

- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_rollback-Methode

Setzt die aktuelle Transaktion zurück.

Syntax

```
public sacapi_bool sqlany_rollback(a_sqlany_connection * sqlany_conn)
```

Parameter

- **sqlany_conn** Das Verbindungsobjekt, auf dem der Rollback-Vorgang durchgeführt werden soll.

Rückgabe

1 bei Erfolg, ansonsten 0.

Siehe auch

- [sqlany_commit-Methode \[SQL Anywhere C\] auf Seite 569](#)

sqlany_send_param_data-Methode

Sendet Daten als Teil eines gebundenen Parameters.

Syntax

```
public sacapi_bool sqlany_send_param_data(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 index,  
    char * buffer,  
    size_t size  
)
```

Parameter

- **sqlany_stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare()` vorbereitet wurde.
- **index** Der Index des Parameters. Dies sollte eine Zahl zwischen 0 und `sqlany_num_params() - 1` sein.
- **buffer** Die zu sendenden Daten.
- **size** Die Anzahl der zu sendenden Bytes.

Rückgabe

1 bei Erfolg oder 0 bei Fehlschlag.

Bemerkungen

Mit dieser Methode können Sie eine große Menge von Daten für einen gebundenen Parameter in Abschnitten senden.

Siehe auch

- [sqlany_prepare-Methode \[SQL Anywhere C\] auf Seite 587](#)

sqlany_sqlstate-Methode

Ruft den aktuellen SQLSTATE ab.

Syntax

```
public size_t sqlany_sqlstate(  
    a_sqlany_connection * sqlany_conn,  
    char * buffer,  
    size_t size  
)
```

Parameter

- **sqlany_conn** Ein Verbindungsobjekt, das von sqlany_new_connection() zurückgegeben wird.
- **buffer** Ein Puffer, der mit dem aktuellen 5-stelligen SQLSTATE-Wert gefüllt wird.
- **size** Die Puffergröße.

Rückgabe

Die Anzahl der Byte, die in den Puffer kopiert wurden.

Bemerkungen

Weitere Hinweise zu SQLSTATE-Fehlermeldungen finden Sie unter „SQL Anywhere-Fehlermeldungen - sortiert nach SQLSTATE“ [[Fehlermeldungen](#)].

Siehe auch

- [sqlany_error-Methode](#) [SQL Anywhere C] auf Seite 572

a_sqlany_data_direction-Enumeration

Eine Datenrichtungsenumeration.

Syntax

```
public enum a_sqlany_data_direction
```

Mitglieder

Mitgliedsname	Beschreibung	Wert
DD_INVALID	Ungültige Datenrichtung.	0x0
DD_INPUT	Hostvariablen nur für die Eingabe.	0x1
DD_OUTPUT	Hostvariablen nur für die Ausgabe.	0x2
DD_INPUT_OUTPUT	Hostvariablen für Eingabe und Ausgabe.	0x3

a_sqlany_data_type-Enumeration

Gibt den Datentyp an, der übergeben oder abgerufen wird.

Syntax

```
public enum a_sqlany_data_type
```

Mitglieder

Mitgliedsname	Beschreibung
A_INVALID_TYPE	Ungültiger Datentyp.
A_BINARY	Binärdaten. Binärdaten werden verarbeitet, wie sie sind, und es wird keine Zeichensatzkonvertierung durchgeführt.
A_STRING	Textdaten. Die Daten, bei denen eine Zeichensatzkonvertierung durchgeführt wird.
A_DOUBLE	Double-Daten. Umfasst float-Werte.
A_VAL64	64-Bit-Ganzzahl.
A_UVAL64	64-Bit-Ganzzahl ohne Vorzeichen.
A_VAL32	32-Bit-Ganzzahl.
A_UVAL32	32-Bit Ganzzahl ohne Vorzeichen
A_VAL16	16-Bit-Ganzzahl.
A_UVAL16	16-Bit Ganzzahl ohne Vorzeichen
A_VAL8	8-Bit-Ganzzahl.
A_UVAL8	8-Bit Ganzzahl ohne Vorzeichen

a_sqlany_native_type-Enumeration

Eine Enumeration der nativen Datentypen von Werten gemäß der Beschreibung durch den Server.

Syntax

```
public enum a_sqlany_native_type
```

Mitglieder

Mitgliedsname	Beschreibung
DT_NOTYPE	Kein Datentyp.
DT_DATE	Mit NULL abgeschlossene Zeichenfolge, die ein gültiges Datum repräsentiert.
DT_TIME	Mit NULL abgeschlossene Zeichenfolge, die eine gültige Zeitangabe repräsentiert.
DT_TIMESTAMP	Mit NULL abgeschlossene Zeichenfolge, die einen gültigen Zeitstempel darstellt.
DT_VARCHAR	Zeichenfolge variabler Länge, im CHAR-Zeichensatz, mit einem 2-Byte-Längenfeld. Die maximale Länge beträgt 32765 Byte. Beim Senden von Daten müssen Sie das Längenfeld festlegen. Beim Abrufen von Daten setzt der Datenbankserver das Längenfeld. Die Daten werden nicht mit NULL abgeschlossen oder mit Leerzeichen aufgefüllt.
DT_FIXCHAR	Zeichenfolge mit fester Länge, die mit Leerzeichen aufgefüllt wird. Im CHAR-Zeichensatz. Die maximale Länge ist 32767 (in Byte). Die Daten enden nicht mit NULL.
DT_LONGVARCHAR	Lange Zeichenfolge von variabler Länge, im CHAR-Zeichensatz.
DT_STRING	Mit NULL abgeschlossene Zeichenfolge im CHAR-Zeichensatz. Die Zeichenfolge wird mit Leerzeichen aufgefüllt, wenn bei der Initialisierung der Datenbank mit Leerzeichen aufgefüllte Zeichenfolgen verwendet werden.
DT_DOUBLE	8-Byte-Gleitkommazahl.
DT_FLOAT	4-Byte-Gleitkommazahl.
DT_DECIMAL	Gepackte Dezimalzahl (systemeigenes Format)
DT_INT	32-Bit-Ganzzahl mit Vorzeichen
DT_SMALLINT	16-Bit-Ganzzahl mit Vorzeichen
DT_BINARY	Binärdaten variabler Länge mit einem 2-Byte-Längenfeld. Die maximale Länge beträgt 32765 Byte. Wenn Sie dem Datenbankserver Informationen liefern, müssen Sie das Längenfeld setzen. Wenn Sie Informationen vom Datenbankserver abrufen, setzt der Datenbankserver das Längenfeld.
DT_LONGBINARY	Lange Binärdaten.

Mitgliedsname	Beschreibung
DT_TINYINT	8-Bit-Ganzzahl mit Vorzeichen
DT_BIGINT	64-Bit-Ganzzahl mit Vorzeichen
DT_UNSINT	32-Bit Ganzzahl ohne Vorzeichen
DT_UNSSMALLINT	16-Bit Ganzzahl ohne Vorzeichen
DT_UNSBIGINT	64-Bit-Ganzzahl ohne Vorzeichen.
DT_BIT	8-Bit-Ganzzahl mit Vorzeichen
DT_LONGNVARCHAR	Lange Zeichenfolge von variabler Länge, im NCHAR-Zeichensatz.

Bemerkungen

Die Werttypen entsprechen den Embedded SQL-Datentypen. Weitere Hinweise zu Embedded SQL-Datentypen finden Sie unter „[Datentypen in Embedded SQL](#)“ auf Seite 482.

Siehe auch

- [sqlany_get_column_info-Methode \[SQL Anywhere C\]](#) auf Seite 579
- [a_sqlany_column_info-Struktur \[SQL Anywhere C\]](#) auf Seite 594

a_sqlany_bind_param-Struktur

Eine bind-Parameterstruktur, die für die Bindung von Parameter und vorbereiteten Anweisungen verwendet wird.

Syntax

```
public typedef struct a_sqlany_bind_param
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
direction	a_sqlany_data_direction	Die Richtung der Daten. (input, output, input_output)
name	char *	Der Name des bind-Parameters. Dies wird nur von sqlany_describe_bind_param() verwendet.
value	a_sqlany_data_value	Der tatsächliche Wert der Daten.

Bemerkungen

Beispiele für die Verwendung der `a_sqlany_bind_param`-Struktur finden Sie in den folgenden Beispieldateien im Verzeichnis `sdk\dbcapi\examples` Ihrer SQL Anywhere-Installation.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)

a_sqlany_bind_param_info-Struktur

Ruft Informationen über die derzeit gebundenen Parameter ab.

Syntax

```
public typedef struct a_sqlany_bind_param_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
direction	a_sqlany_data_direction	Die Richtung des Parameters.
input_value	a_sqlany_data_value	Informationen über den gebundenen Eingabewert.
name	char *	Ein Zeiger auf den Namen des Parameters.
output_value	a_sqlany_data_value	Informationen über den gebundenen Ausgabewert.

Bemerkungen

`sqlany_get_bind_param_info()` kann verwendet werden, um diese Struktur mit Daten zu füllen.

Beispiele für die Verwendung der `a_sqlany_bind_param_info`-Struktur finden Sie in den folgenden Beispieldateien im Verzeichnis `sdk\dbcapi\examples` Ihrer SQL Anywhere-Installation.

Siehe auch

- [sqlany_execute-Methode \[SQL Anywhere C\] auf Seite 572](#)

a_sqlany_column_info-Struktur

Gibt Informationen über Spaltenmetadaten zurück.

Syntax

```
public typedef struct a_sqlany_column_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
max_size	size_t	Die maximale Größe, die ein Datenwert in dieser Zeile annehmen kann.
name	char *	Der Name der Spalte (mit NULL abgeschlossen). Die Zeichenfolge kann referenziert werden, solange das Ergebnismengenobjekt nicht freigegeben wird.
native_type	a_sqlany_native_type	Der native Typ der Spalte in der Datenbank.
nullable	sacapi_bool	Zeigt an, ob ein Wert in der Spalte NULL sein kann.
precision	unsigned short	Die Gesamtstellenzahl.
scale	unsigned short	Die Anzahl der Dezimalstellen.
type	a_sqlany_data_type	Der Spaltendatentyp.

Bemerkungen

sqlany_get_column_info() kann verwendet werden, um diese Struktur mit Daten zu füllen.

Ein Beispiel für die Verwendung der a_sqlany_column_info-Struktur finden Sie in der folgenden Beispieldatei im Verzeichnis *sdk\dbcapi\examples* Ihrer SQL Anywhere-Installation.

- dbcapi_isql.cpp

a_sqlany_data_info-Struktur

Gibt Metadaten-Informationen über einen Spaltenwert in einer Ergebnismenge zurück.

Syntax

```
public typedef struct a_sqlany_data_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
data_size	size_t	Die Gesamtzahl der Byte, die zum Abrufen verfügbar sind. Dieses Feld ist nur nach einem erfolgreichen fetch-Vorgang gültig.

Mitgliedsname	Typ	Beschreibung
is_null	sacapi_bool	Gibt an, ob die zuletzt abgerufenen Daten NULL sind. Dieses Feld ist nur nach einem erfolgreichen fetch-Vorgang gültig.
type	a_sqlany_data_type	Der Typ der Daten in der Spalte.

Bemerkungen

sqlany_get_data_info() kann verwendet werden, um diese Struktur mit Informationen darüber zu füllen, was zuletzt durch einen fetch-Vorgang abgerufen wurde.

Ein Beispiel für die Verwendung der a_sqlany_info-Struktur finden Sie in der folgenden Beispieldatei im Verzeichnis *sdk\dbcapi\examples* Ihrer SQL Anywhere-Installation.

Siehe auch

- [sqlany_get_data_info-Methode \[SQL Anywhere C\] auf Seite 580](#)

a_sqlany_data_value-Struktur

Gibt eine Beschreibung der Attribute eines Datenwerts zurück.

Syntax

```
public typedef struct a_sqlany_data_value
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
buffer	char *	Ein Zeiger auf einen vom Benutzer bereitgestellten Puffer von Daten.
buffer_size	size_t	Die Größe des Puffers.
is_null	sacapi_bool *	Ein Zeiger auf einen Indikator, ob die zuletzt abgerufenen Daten NULL sind.
length	size_t *	Ein Zeiger auf die Anzahl gültiger Byte im Puffer. Dieser Wert muss kleiner sein als buffer_size.
type	a_sqlany_data_type	Der Typ der Daten.

Bemerkungen

Beispiele für die Verwendung der `a_sqlany_data_value`-Struktur finden Sie in den folgenden Beispieldateien im `sdk\dbcapi\examples` -Verzeichnis Ihrer SQL Anywhere-Installation.

- `dbcapi_isql.cpp`
- `fetching_a_result_set.cpp`
- `send_retrieve_full_blob.cpp`
- `preparing_statements.cpp`

SQLAnywhereInterface-Struktur

Die Schnittstellenstruktur der SQL Anywhere-C-API.

Syntax

```
public typedef struct SQLAnywhereInterface
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
<code>dll_handle</code>	<code>void *</code>	DLL-Handle.
<code>initialized</code>	<code>int</code>	Parameter, der angibt, ob initialisiert oder nicht.
<code>sqlany_affected_rows</code>	<code>void *</code>	Zeiger auf die <code>sqlany_affected_rows()</code> -Funktion.
<code>sqlany_bind_param</code>	<code>void *</code>	Zeiger auf die <code>sqlany_bind_param()</code> -Funktion.
<code>sqlany_cancel</code>	<code>void *</code>	Zeiger auf die <code>sqlany_cancel()</code> Funktion.
<code>sqlany_clear_error</code>	<code>void *</code>	Zeiger auf die <code>sqlany_clear_error()</code> -Funktion.
<code>sqlany_client_version</code>	<code>void *</code>	Zeiger auf die <code>sqlany_client_version()</code> -Funktion.
<code>sqlany_client_version_ex</code>	<code>void *</code>	Zeiger auf die <code>sqlany_client_version_ex()</code> -Funktion.
<code>sqlany_commit</code>	<code>void *</code>	Zeiger auf die <code>sqlany_commit()</code> -Funktion.
<code>sqlany_connect</code>	<code>void *</code>	Zeiger auf die <code>sqlany_connect()</code> -Funktion.
<code>sqlany_describe_bind_param</code>	<code>void *</code>	Zeiger auf die <code>sqlany_describe_bind_param()</code> -Funktion.
<code>sqlany_disconnect</code>	<code>void *</code>	Zeiger auf die <code>sqlany_disconnect()</code> -Funktion.
<code>sqlany_error</code>	<code>void *</code>	Zeiger auf die <code>sqlany_error()</code> -Funktion.

Mitgliedsname	Typ	Beschreibung
sqlany_execute	void *	Zeiger auf die sqlany_execute()-Funktion.
sqlany_execute_direct	void *	Zeiger auf die sqlany_execute_direct()-Funktion.
sqlany_execute_immediate	void *	Zeiger auf die sqlany_execute_immediate()-Funktion.
sqlany_fetch_absolute	void *	Zeiger auf die sqlany_fetch_absolute()-Funktion.
sqlany_fetch_next	void *	Zeiger auf die sqlany_fetch_next()-Funktion.
sqlany_fini	void *	Zeiger auf die sqlany_fini()-Funktion.
sqlany_fini_ex	void *	Zeiger auf die sqlany_fini_ex() Funktion.
sqlany_free_connection	void *	Zeiger auf die sqlany_free_connection()-Funktion.
sqlany_free_stmt	void *	Zeiger auf die sqlany_free_stmt()-Funktion.
sqlany_get_bind_param_info	void *	Zeiger auf die sqlany_get_bind_param_info()-Funktion.
sqlany_get_column	void *	Zeiger auf die sqlany_get_column()-Funktion.
sqlany_get_column_info	void *	Zeiger auf die sqlany_get_column_info()-Funktion.
sqlany_get_data	void *	Zeiger auf die sqlany_get_data()-Funktion.
sqlany_get_data_info	void *	Zeiger auf die sqlany_get_data_info()-Funktion.
sqlany_get_next_result	void *	Zeiger auf die sqlany_get_next_result()-Funktion.
sqlany_init	void *	Zeiger auf die sqlany_init()-Funktion.
sqlany_init_ex	void *	Zeiger auf die sqlany_init_ex()-Funktion.
sqlany_make_connection	void *	Zeiger auf die sqlany_make_connection()-Funktion.
sqlany_make_connection_ex	void *	Zeiger auf die sqlany_make_connection_ex() Funktion.
sqlany_new_connection	void *	Zeiger auf die sqlany_new_connection()-Funktion.
sqlany_new_connection_ex	void *	Zeiger auf die sqlany_new_connection_ex()-Funktion.
sqlany_num_cols	void *	Zeiger auf die sqlany_num_cols()-Funktion.
sqlany_num_params	void *	Zeiger auf die sqlany_num_params()-Funktion.
sqlany_num_rows	void *	Zeiger auf die sqlany_num_rows()-Funktion.

Mitgliedsname	Typ	Beschreibung
sqlany_prepare	void *	Zeiger auf die sqlany_prepare()-Funktion.
sqlany_reset	void *	Zeiger auf die sqlany_reset()-Funktion.
sqlany_rollback	void *	Zeiger auf die sqlany_rollback()-Funktion.
sqlany_send_param_data	void *	Zeiger auf die sqlany_send_param_data()-Funktion.
sqlany_sqlstate	void *	Zeiger auf die sqlany_sqlstate()-Funktion.

Siehe auch

- [sqlany_initialize_interface-Methode \[SQL Anywhere C\] auf Seite 583](#)

SACAPI_ERROR_SIZE-Variable

Gibt die minimale Fehlerpuffergröße zurück.

Syntax

```
#define SACAPI_ERROR_SIZE
```

SQLANY_API_VERSION_1-Variable

Legt fest, dass die API-Versionen angezeigt werden.

Syntax

```
#define SQLANY_API_VERSION_1
```

SQLANY_API_VERSION_2-Variable

Mit Version 2 wurden die "_ex"-Funktionen und die Möglichkeit zum Abbrechen von Anforderungen eingeführt.

Syntax

```
#define SQLANY_API_VERSION_2
```

SQL Anywhere-Schnittstelle für externe Aufrufe

Sie können eine Funktion in einer externen Bibliothek aus einer gespeicherten Prozedur oder Funktion aufrufen. Sie können Funktionen in einer DLL unter Windows und in einem Shared Object unter Unix aufrufen. Unter Windows Mobile können keine externen Funktionen aufgerufen werden.

Dieser Abschnitt beschreibt die Verwendung der Schnittstelle für externe Funktionsaufrufe. Externe gespeicherte Beispielsprozeduren sowie die Dateien, die zum Erstellen einer sie enthaltenden DLL erforderlich sind, befinden sich im Ordner `%SQLANYSAMPI6%\SQLAnywhere\ExternalProcedures`.

Vorsicht

Externe Bibliotheken, die aus Prozeduren aufgerufen werden, nutzen den Speicher des Servers gemeinsam. Wenn Sie eine externe Bibliothek aus einer Prozedur aufrufen, und die externe Bibliothek Speicherverwaltungsfehler enthält, kann der Server abstürzen und die Datenbank beschädigen. Achten Sie daher darauf, Ihre Bibliotheken umfassend auszutesten, bevor Sie sie in Produktionsdatenbanken einsetzen.

Die in diesem Abschnitt beschriebene Schnittstelle ersetzt eine ältere Schnittstelle, die nicht mehr empfohlen wird. Bibliotheken, die für die ältere, in Versionen vor 7.0.x eingesetzte Schnittstelle geschrieben wurden, werden weiterhin unterstützt. Wenn Sie jedoch eine neue Anwendung entwickeln, sollten Sie die neue Schnittstelle verwenden. Die neue Schnittstelle muss für alle Unix-Plattformen und für alle 64-Bit-Plattformen einschließlich 64-Bit-Windows verwendet werden.

SQL Anywhere enthält eine Serie von Systemprozeduren, die diese Möglichkeiten nutzen, um beispielsweise MAPI-E-Mail-Nachrichten zu versenden. Siehe „[MAPI- und SMTP-Systemprozeduren](#)“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Prozeduren und Funktionen, die externe Aufrufe verwenden

In diesem Abschnitt werden einige Beispiele für Prozeduren und Funktionen mit externen Aufrufen beschrieben.

Hinweis

Sie müssen das CREATE EXTERNAL REFERENCE-Systemprivileg haben, um Prozeduren oder Funktionen erstellen zu können, die externe Bibliotheken referenzieren.

Syntax

Sie können eine gespeicherte SQL-Prozedur, die eine C/C++-Funktion in einer Bibliothek (einer DLL(Dynamic Link Library --dynamische Verknüpfungsbibliothek) oder Shared Object) aufruft, folgendermaßen erstellen:

```
CREATE PROCEDURE coverProc( parameter-list )  
  EXTERNAL NAME 'myFunction@myLibrary'  
  LANGUAGE C_ESQL32;
```

Wenn Sie eine gespeicherte Prozedur oder Funktion auf diese Weise definieren, erstellen Sie eine Brücke zu der Funktion in der externen DLL. Die gespeicherte Prozedur oder Funktion kann keine anderen Aufgaben ausführen.

Auf ähnliche Weise können Sie folgendermaßen eine gespeicherte SQL-Funktion erstellen, die eine C/C++-Funktion in einer Bibliothek aufruft:

```
CREATE FUNCTION coverFunc( parameter-list )  
  RETURNS data-type  
  EXTERNAL NAME 'myFunction@myLibrary'  
  LANGUAGE C_ESQL32;
```

In diesen Anweisungen gibt die EXTERNAL NAME-Klausel den Namen der Funktion und die Bibliothek an, in der sie sich befindet. Im Beispiel ist `myFunction` der exportierte Name einer Funktion in der Bibliothek und `myLibrary` ist der Name der Bibliothek (z.B. `myLibrary.dll` oder `myLibrary.so`).

Die LANGUAGE-Klausel gibt an, dass die Funktion in einer externen Umgebung aufgerufen werden soll. Die LANGUAGE-Klausel kann Folgendes angeben: `C_ESQL32`, `C_ESQL64`, `C_ODBC32` oder `C_ODBC64`. Das Suffix **32** oder **64** gibt an, dass die Funktion als 32-Bit- oder 64-Bit-Anwendung kompiliert ist. Die Angabe von **ODBC** zeigt an, dass die Anwendung die ODBC-API verwendet. Die Angabe von **ESQL** zeigt an, dass die Anwendung die Embedded SQL-API, die SQL Anywhere C-API, jede andere Nicht-ODBC-API oder keine API verwendet.

Wenn die LANGUAGE-Klausel nicht angegeben ist, wird die Bibliothek, die die Funktion enthält, in den Adressraum des Datenbankservers geladen. Wenn sie aufgerufen wird, wird die externe Funktion als Teil des Datenbankservers ausgeführt. In diesem Fall wird, falls die Funktion einen Fehler verursacht, der Datenbankserver beendet. Aus diesem Grund wird empfohlen, Funktionen in einer externen Umgebung zu laden und auszuführen. Wenn eine Funktion einen Fehler in einer externen Umgebung verursacht, läuft der Datenbankserver weiter.

Die Argumente in *Parameterliste* müssen in Typ und Reihenfolge den Argumenten entsprechen, die von der Bibliotheksfunktion erwartet werden. Die Bibliotheksfunktion greift auf die Prozedurargumente unter Verwendung einer speziellen Schnittstelle zu. Siehe „[Prototypen externer Funktionen](#)“ auf Seite 603.

Jeder von der externen Funktion zurückgegebene Wert bzw. jede Ergebnismenge kann von der gespeicherten Funktion oder Prozedur an die aufrufende Umgebung zurückgegeben werden.

Keine anderen Anweisungen zulässig

Eine gespeicherte Prozedur oder Funktion, die eine externe Funktion referenziert, kann keine anderen Anweisungen enthalten. Ihr einziger Zweck ist die Übernahme von Argumenten für eine Funktion, der Aufruf der Funktion und die Rückgabe der Werte und der von der Funktion zurückgegebenen Argumente an die aufrufende Umgebung. Sie können die Parameter `IN`, `INOUT` oder `OUT` im Prozeduraufruf genauso wie bei anderen Prozeduren verwenden. Die Eingabewerte werden an die externe Funktion übergeben, und von der Funktion veränderte Parameter werden an die aufrufende Umgebung in `OUT`- oder `INOUT`-Parametern oder als `RETURNS`-Ergebnis der gespeicherten Funktion zurückgegeben.

Systemabhängige Aufrufe

Sie können Betriebssystem-abhängige Aufrufe angeben, sodass eine Prozedur eine Funktion auf einem Betriebssystem und eine andere Funktion (wahrscheinlich eine dazu analoge) auf einem anderen Betriebssystem aufruft. Die Syntax für solche Aufrufe verlangt den Betriebssystemnamen als Präfix vor dem Funktionsnamen. Die Betriebssystem-ID muss Unix sein. Beispiel:

```
CREATE FUNCTION func ( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'Unix:function-name@library.so;function-name@library.dll';
```

Wenn die Liste der Funktionen keinen Eintrag für das Betriebssystem enthält, auf dem der Server läuft, aber ein Eintrag ohne Betriebssystem vorhanden ist, ruft der Datenbankserver die Funktion in diesem Eintrag auf.

Siehe auch

- „CREATE FUNCTION-Anweisung [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [externer Aufruf]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Unterstützung für externe Umgebungen in SQL Anywhere“ auf Seite 621

Prototypen externer Funktionen

Dieser Abschnitt beschreibt die Schnittstelle, die Sie bei in C oder C++ geschriebenen Funktionen verwenden.

Die Schnittstelle ist in der Header-Datei *extfnapi.h* im Unterverzeichnis *SDK\Include* des SQL Anywhere-Installationsverzeichnisses definiert. Diese Header-Datei verarbeitet die plattformabhängigen Funktionen der Prototypen externer Funktionen.

Prototypen für Funktionen

Der Name der Funktion muss zu der Funktion passen, die in der Anweisung CREATE PROCEDURE oder CREATE FUNCTION referenziert wird. Nehmen wir an, die folgende CREATE FUNCTION-Anweisung wurde ausgeführt.

```
CREATE FUNCTION cover-name ( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'function-name@library.dll'
  LANGUAGE C_ESQL32;
```

Die C/C++-Funktionsdeklaration muss wie folgt aussehen:

```
void function-name(an_extfn_api *api, void *argument-handle )
```

Die Funktion muss "void" zurückgeben und als Argumente einen Zeiger auf eine Struktur haben, die zum Aufrufen einer Gruppe von Callback-Funktionen benutzt wird, und ein Handle für die Argumente, die von der SQL-Prozedur bereitgestellt werden.

extfn_use_new_api-Methode

Um den Datenbankserver darüber zu informieren, dass die externe Bibliothek mithilfe der Schnittstelle für externe Funktionsaufrufe geschrieben wurde, muss die externe Bibliothek die folgende Funktion exportieren:

Syntax

```
extern "C" a_sql_uint32 SQL_CALLBACK extfn_use_new_api( void );
```

Rückgabe

Die Funktion gibt eine vorzeichenlose 32-Bit-Ganzzahl zurück. Der zurückgegebene Wert muss die Versionsnummer der Schnittstelle, EXTFN_API_VERSION, sein, die in *extfnapi.h* definiert wurde. Ein Rückgabewert von 0 bedeutet, dass die alte, nicht mehr empfohlene Schnittstelle benutzt wird.

Bemerkungen

Wenn die Funktion von der Bibliothek nicht exportiert wird, nimmt der Datenbankserver an, dass die alte Schnittstelle benutzt wird. Die neue Schnittstelle muss für alle Unix-Plattformen und für alle 64-Bit-Plattformen einschließlich 64-Bit-Windows benutzt werden.

Im Folgenden wird eine typische Implementierung dieser Funktion gezeigt:

```
extern "C" a_sql_uint32 SQL_CALLBACK extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}
```

Siehe auch

- „an_extfn_api-Struktur“ auf Seite 606

extfn_cancel-Methode

Um den Datenbankserver darüber zu informieren, dass die externe Bibliothek die Abbruchverarbeitung unterstützt, muss Ihre externe Bibliothek die folgenden Funktionen exportieren:

Syntax

```
extern "C" void SQL_CALLBACK extfn_cancel( void *cancel_handle );
```

Parameter

- **cancel_handle** Ein Zeiger auf eine zu ändernde Variable

Bemerkungen

Diese Funktion wird vom Datenbankserver immer dann asynchron aufgerufen, wenn die derzeit ausgeführte SQL-Anweisung abgebrochen wird.

Die Funktion setzt mit cancel_handle einen Parameter, der den externen Bibliotheksfunktionen anzeigt, dass die SQL-Anweisung abgebrochen wurde.

Wenn die Funktion von der Bibliothek nicht exportiert wird, nimmt der Datenbankserver an, dass die Abbruchverarbeitung nicht unterstützt wird.

Im Folgenden wird eine typische Implementierung dieser Funktion gezeigt:

```
extern "C" void SQL_CALLBACK extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}
```

Siehe auch

- „an_extfn_api-Struktur“ auf Seite 606

extfn_post_load_library-Methode

Wenn diese Funktion implementiert ist und in der externen Bibliothek bereitgestellt wird, wird sie vom Datenbankserver ausgeführt, nachdem die externe Bibliothek geladen und die Versionsprüfung durchgeführt wurde und bevor eine andere der in der externen Bibliothek definierten Funktionen aufgerufen wird.

Syntax

```
extern "C" void SQL_CALLBACK extfn_post_load_library( void );
```

Bemerkungen

Diese Funktion ist nur dann erforderlich, wenn es eine Bibliothek-spezifische Anforderung gibt, ein sich über die gesamte Bibliothek erstreckendes Setup auszuführen, bevor eine der Funktionen in der Bibliothek aufgerufen wird.

Diese Funktion wird vom Datenbankserver asynchron aufgerufen, nachdem die externe Bibliothek geladen und die Versionsprüfung durchgeführt wurde und bevor eine andere der in der externen Bibliothek definierten Funktionen aufgerufen wird.

Beispiel

```
extern "C" __declspec( dllexport )
void SQL_CALLBACK extfn_post_load_library( void )
{
    MessageBox( NULL, "Library loaded",
                "SQL Anywhere",
                MB_OK | MB_TASKMODAL );
}
```

Siehe auch

- „an_extfn_api-Struktur“ auf Seite 606

extfn_pre_unload_library-Methode

Wenn diese Funktion implementiert ist und in der externen Bibliothek bereitgestellt wird, wird sie vom Datenbankserver unmittelbar vor dem Entladen der externen Bibliothek ausgeführt.

Syntax

```
extern "C" void SQL_CALLBACK extfn_pre_unload_library( void );
```

Bemerkungen

Diese Funktion ist nur dann erforderlich, wenn es eine Bibliothek-spezifische Anforderung gibt, eine sich über die gesamte Bibliothek erstreckende Bereinigung auszuführen, bevor die Bibliothek entladen wird.

Diese Funktion wird vom Datenbankserver unmittelbar vor dem Entladen der externen Bibliothek asynchron aufgerufen.

Beispiel

```
extern "C" __declspec( dllexport )
void SQL_CALLBACK extfn_pre_unload_library( void )
{
    MessageBox( NULL, "Library unloading",
                "SQL Anywhere",
                MB_OK | MB_TASKMODAL );
}
```

Siehe auch

- „an_extfn_api-Struktur“ auf Seite 606

an_extfn_api-Struktur

Wird verwendet, um mit der aufrufenden SQL-Umgebung zu kommunizieren.

Syntax

```
typedef struct an_extfn_api {
    short (SQL_CALLBACK *get_value)(
        void *      arg_handle,
        a_sql_uint32 arg_num,
        an_extfn_value *value
    );
    short (SQL_CALLBACK *get_piece)(
        void *      arg_handle,
        a_sql_uint32 arg_num,
        an_extfn_value *value,
        a_sql_uint32 offset
    );
    short (SQL_CALLBACK *set_value)(
        void *      arg_handle,
        a_sql_uint32 arg_num,
        an_extfn_value *value,
        short      append
    );
    void (SQL_CALLBACK *set_cancel)(
        void *      arg_handle,
        void *      cancel_handle
    );
} an_extfn_api;
```

Eigenschaften

- **get_value** Verwenden Sie diese Callback-Funktion, um den Wert des angegebenen Parameters abzurufen. Das folgende Beispiel ruft den Wert des Parameters 1 ab.

```
result = extapi->get_value( arg_handle, 1, &arg )
if( result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}
```

- **get_piece** Verwenden Sie diese Callback-Funktion, um den nächsten Abschnitt des Werts des angegebenen Parameters abzurufen (falls vorhanden). Das folgende Beispiel ruft die verbleibenden Abschnitte des Parameters 1 ab.

```
cmd = (char *)malloc( arg.len.total_len + 1 );
offset = 0;
for( ; result != 0; )
{
    if( arg.data == NULL ) break;
    memcpy( &cmd[offset], arg.data, arg.piece_len );
    offset += arg.piece_len;
    cmd[offset] = '\0';
    if( arg.piece_len == 0 ) break;
    result = extapi->get_piece( arg_handle, 1, &arg, offset );
}
```

- **set_value** Verwenden Sie diese Callback-Funktion, um den Wert des angegebenen Parameters festzulegen. Das folgende Beispiel legt den Rückgabewert (Parameter 0) für eine RETURNS-Klausel einer FUNCTION fest.

```
an_extfn_value      retval;
int ret = -1;

// set up the return value struct
retval.type = DT_INT;
retval.data = (void*) &ret;
retval.piece_len = retval.len.total_len =
    (a_sql_uint32) sizeof( int );
extapi->set_value( arg_handle, 0, &retval, 0 );
```

- **set_cancel** Verwenden Sie diese Callback-Funktion, um einen Zeiger auf eine Variable einzurichten, die durch die **extfn_cancel**-Methode festgelegt werden kann. Im Folgenden finden Sie ein Beispiel.

```
short               canceled = 0;
extapi->set_cancel( arg_handle, &canceled );
```

Bemerkungen

Ein Zeiger auf die **an_extfn_api**-Struktur wird durch den Aufrufer an Ihre externe Funktion übergeben. Hier sehen Sie ein Beispiel.

```
extern "C" __declspec( dllexport )
void my_external_proc( an_extfn_api *extapi, void *arg_handle )
{
    short          result;
    short          canceled;
    an_extfn_value arg;
```

```
canceled = 0;
extapi->set_cancel( arg_handle, &canceled );

result = extapi->get_value( arg_handle, 1, &arg );
if( canceled || result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}

.
.
}
```

Wenn Sie eine der Callback-Funktionen verwenden, müssen Sie das Argument-Handle zurück übergeben, das an Ihre externe Funktion als zweiter Parameter übergeben wurde.

Siehe auch

- „an_extfn_value-Struktur“ auf Seite 608
- „extfn_cancel-Methode“ auf Seite 604

an_extfn_value-Struktur

Wird verwendet, um auf Parameterdaten von der aufrufenden SQL-Umgebung aus zuzugreifen.

Syntax

```
typedef struct an_extfn_value {
    void *      data;
    a_sql_uint32 piece_len;
    union {
        a_sql_uint32 total_len;
        a_sql_uint32 remain_len;
    } len;
    a_sql_data_type type;
} an_extfn_value;
```

Eigenschaften

- **data** Ein Zeiger auf die Daten für diesen Parameter.
- **piece_len** Die Länge dieses Segments des Parameters. Dies ist kleiner oder gleich **total_len**.
- **total_len** Die Gesamtlänge des Parameters. Bei Zeichenfolgen stellt dies die Länge der Zeichenfolge dar, ohne das Nullabschlusszeichen einzubeziehen. Diese Eigenschaft wird nach einem Aufruf der **get_value**-Callback-Funktion gesetzt. Diese Eigenschaft ist nach einem Aufruf der **get_piece**-Callback-Funktion nicht mehr gültig.
- **remain_len** Wenn der Parameter in Segmenten übernommen wird, ist dies die Länge des Teils, der noch nicht übernommen wurde. Diese Eigenschaft wird nach jedem Aufruf der **get_piece**-Callback-Funktion gesetzt.
- **type** Zeigt den Typ des Parameters an. Dies ist einer der Embedded SQL-Datentypen, wie z.B. **DT_INT**, **DT_FIXCHAR** oder **DT_BINARY**. Siehe „Datentypen in Embedded SQL“ auf Seite 482.

Bemerkungen

Angenommen, die Schnittstelle Ihrer externen Funktion wurde unter Verwendung der folgenden Anweisung geschrieben.

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@c:\\project\\mystring.dll';
```

Das folgende Codefragment zeigt, wie Sie auf die Eigenschaften für Objekte des Typs **an_extfn_value** zugreifen. In dem Beispiel wird erwartet, dass der Eingabeparameter 1 (instr) für diese Funktion (mystring) eine SQL LONGVARCHAR-Zeichenfolge ist.

```
an_extfn_value      arg;

result = extapi->get_value( arg_handle, 1, &arg );
if( result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}

if( arg.type != DT_LONGVARCHAR )
{
    return; // unexpected type of parameter
}

cmd = (char *)malloc( arg.len.total_len + 1 );
offset = 0;
for( ; result != 0; )
{
    if( arg.data == NULL ) break;
    memcpy( &cmd[offset], arg.data, arg.piece_len );
    offset += arg.piece_len;
    cmd[offset] = '\0';
    if( arg.piece_len == 0 ) break;
    result = extapi->get_piece( arg_handle, 1, &arg, offset );
}
```

Siehe auch

- „an_extfn_api-Struktur“ auf Seite 606

an_extfn_result_set_info-Struktur

Erleichtert die Rückgabe von Ergebnismengen an die aufrufende SQL-Umgebung.

Syntax

```
typedef struct an_extfn_result_set_info {
    a_sql_uint32          number_of_columns;
    an_extfn_result_set_column_info *column_infos;
    an_extfn_result_set_column_data *column_data_values;
} an_extfn_result_set_info;
```

Eigenschaften

- **number_of_columns** Die Anzahl der Spalten in der Ergebnismenge.

- **column_infos** Verknüpfung zu einer Beschreibung der Spalten in der Ergebnismenge. Siehe [„an_extfn_result_set_column_info-Struktur“ auf Seite 610](#).
- **column_data_values** Verknüpfung zu einer Beschreibung der Spaltendaten in der Ergebnismenge. Siehe [„an_extfn_result_set_column_data-Struktur“ auf Seite 611](#).

Bemerkungen

Das folgende Codefragment zeigt, wie Sie die Eigenschaften für Objekte des Typs setzen.

```
int columns = 2;
an_extfn_result_set_info rs_info;

an_extfn_result_set_column_info *col_info =
    (an_extfn_result_set_column_info *)
    malloc( columns * sizeof(an_extfn_result_set_column_info) );

an_extfn_result_set_column_data *col_data =
    (an_extfn_result_set_column_data *)
    malloc( columns * sizeof(an_extfn_result_set_column_data) );

rs_info.number_of_columns = columns;
rs_info.column_infos = col_info;
rs_info.column_data_values = col_data;
```

Siehe auch

- [„an_extfn_result_set_column_info-Struktur“ auf Seite 610](#)
- [„an_extfn_result_set_column_data-Struktur“ auf Seite 611](#)

an_extfn_result_set_column_info-Struktur

Wird verwendet, um eine Ergebnismenge zu beschreiben.

Syntax

```
typedef struct an_extfn_result_set_column_info {
    char *                column_name;
    a_sql_data_type        column_type;
    a_sql_uint32           column_width;
    a_sql_uint32           column_index;
    short int              column_can_be_null;
} an_extfn_result_set_column_info;
```

Eigenschaften

- **column_name** Zeigt auf den Namen der Spalte, der eine mit NULL abgeschlossene Zeichenfolge ist.
- **column_type** Zeigt den Typ der Spalte an. Dies ist einer der Embedded SQL-Datentypen, wie z.B. **DT_INT**, **DT_FIXCHAR** oder **DT_BINARY**. Siehe [„Datentypen in Embedded SQL“ auf Seite 482](#).
- **column_width** Legt die maximale Breite für char(n)-, varchar(n)- und binary(n)-Deklarationen fest und wird bei allen anderen Typen auf 0 gesetzt.
- **column_index** Die Ordinalposition der Spalte, die mit 1 beginnt.

- **column_can_be_null** Wenn die Spalte nullwertfähig ist 1, sonst 0.

Bemerkungen

Das folgende Codefragment zeigt, wie Sie die Eigenschaften für Objekte dieses Typs festlegen und die Ergebnismenge der aufrufenden SQL-Umgebung beschreiben.

```
// set up column descriptions
// DepartmentID      INTEGER NOT NULL
col_info[0].column_name = "DepartmentID";
col_info[0].column_type  = DT_INT;
col_info[0].column_width = 0;
col_info[0].column_index = 1;
col_info[0].column_can_be_null = 0;

// DepartmentName    CHAR(40) NOT NULL
col_info[1].column_name = "DepartmentName";
col_info[1].column_type  = DT_FIXCHAR;
col_info[1].column_width = 40;
col_info[1].column_index = 2;
col_info[1].column_can_be_null = 0;

extapi->set_value( arg_handle,
                  EXTFN_RESULT_SET_ARG_NUM,
                  (an_extfn_value *)&rs_info,
                  EXTFN_RESULT_SET_DESCRIBE );
```

Siehe auch

- „an_extfn_result_set_info-Struktur“ auf Seite 609
- „an_extfn_result_set_column_data-Struktur“ auf Seite 611

an_extfn_result_set_column_data-Struktur

Wird verwendet, um die Datenwerte von Spalten zurückzugeben.

Syntax

```
typedef struct an_extfn_result_set_column_data {
    a_sql_uint32      column_index;
    void *            column_data;
    a_sql_uint32      data_length;
    short             append;
} an_extfn_result_set_column_data;
```

Eigenschaften

- **column_index** Die Ordinalposition der Spalte, die mit 1 beginnt.
- **column_data** Zeiger auf einen Puffer, der die Spaltendaten enthält.
- **data_length** Die tatsächliche Länge der Daten.
- **append** Wird verwendet, um den Spaltenwert in Abschnitten zurückzugeben. Mit 1 festgelegt, wenn ein Spaltenwert abschnittsweise zurückgegeben wird; sonst 0.

Bemerkungen

Das folgende Codefragment zeigt, wie Sie die Eigenschaften für Objekte dieses Typs festlegen und die Ergebnismengenzeile an die aufrufenden SQL-Umgebung zurückgeben.

```
int DeptNumber = 400;
char * DeptName = "Marketing";

col_data[0].column_index = 1;
col_data[0].column_data = &DeptNumber;
col_data[0].data_length = sizeof( DeptNumber );
col_data[0].append = 0;

col_data[1].column_index = 2;
col_data[1].column_data = DeptName;
col_data[1].data_length = strlen(DeptName);
col_data[1].append = 0;

extapi->set_value( arg_handle,
                  EXTFN_RESULT_SET_ARG_NUM,
                  (an_extfn_value *)&rs_info,
                  EXTFN_RESULT_SET_NEW_ROW_FLUSH );
```

Siehe auch

- „an_extfn_result_set_info-Struktur“ auf Seite 609
- „an_extfn_result_set_column_info-Struktur“ auf Seite 610

Schnittstellenmethoden für externen Funktionsaufruf

get_value-Callback

```
short (SQL_CALLBACK *get_value)
(
    void *      arg_handle,
    a_sql_uint32 arg_num,
    an_extfn_value *value
);
```

Mit der **get_value**-Callback-Funktion können Sie den Wert eines Parameters abfragen, der an die gespeicherte Prozedur oder Funktion übergeben wurde, die als Schnittstelle zu der externen Funktion agiert. Wenn die Funktion nicht erfolgreich ist, wird 0 zurückgegeben, andernfalls ist der Rückgabewert Nicht-Null. Nach dem Aufruf von **get_value** enthält das Feld **total_len** der Struktur **an_extfn_value** die Länge des gesamten Werts. Das Feld **piece_len** enthält die Länge des Teils, der als Ergebnis des Aufrufs von **get_value** abgerufen wurde. Das **piece_len**-Feld ist immer kleiner oder gleich **total_len**. Wenn **piece_len** kleiner ist, kann als zweite Funktion **get_piece** aufgerufen werden, um die übrigen Teile abzurufen. Das **total_len**-Feld ist erst nach dem ersten Aufruf von **get_value** gültig. Dieses Feld wird vom Feld **remain_len** überlagert, das von Aufrufen von **get_piece** geändert wird. Der Wert des Felds **total_len** unmittelbar nach dem Aufruf von **get_value** muss bewahrt werden, wenn geplant wird, den Wert zu einem späteren Zeitpunkt zu verwenden.

get_piece-Callback

```
short (SQL_CALLBACK *get_piece)
(
```

```

    void *      arg_handle,
    a_sql_uint32 arg_num,
    an_extfn_value *value,
    a_sql_uint32 offset
);

```

Wenn es nicht möglich ist, den gesamten Parameterwert in einem Stück zurückzugeben, kann die **get_piece**-Funktion wiederholt aufgerufen werden, um die übrigen Teile des Parameterwerts abzurufen.

Die Summe aller von den beiden Aufrufen von `get_value` und `get_piece` zurückgegebenen `piece_len`-Werte wird zum Ausgangswert addiert, der im Feld `total_len` nach dem Aufruf von `get_value` zurückgegeben wurde. Nach dem Aufruf von `get_piece` repräsentiert das Feld `remain_len`, das `total_len` überlagert, die noch nicht erhaltene Länge.

get_value- und get_piece-Callback verwenden

Im folgenden Beispiel wird gezeigt, wie Sie mit `get_value` und `get_piece` den Wert eines Zeichenfolgeparameters wie z.B. eines `long varchar`-Parameters erhalten.

Angenommen, der Wrapper für eine externe Funktion wurde folgendermaßen deklariert:

```

CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mystring.dll';

```

Um die externe Funktion aus SQL aufzurufen, verwenden Sie eine Anweisung ähnlich der folgenden.

```
call mystring('Hello world!');
```

Im Folgenden wird eine in C geschriebene Beispielimplementierung der Funktion `mystring` für Windows gezeigt:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                      DWORD  ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    return TRUE;
}

extern "C" __declspec( dllexport )
a_sql_uint32 extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}

extern "C" __declspec( dllexport )
void mystring( an_extfn_api *extapi, void *arg_handle )
{
    short      result;
    an_extfn_value arg;
    unsigned   offset;
    char       *string;

```

```
result = extapi->get_value( arg_handle, 1, &arg );
if( result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}
string = (char *)malloc( arg.len.total_len + 1 );
offset = 0;
for( ; result != 0; ) {
    if( arg.data == NULL ) break;
    memcpy( &string[offset], arg.data, arg.piece_len );
    offset += arg.piece_len;
    string[offset] = '\\0';
    if( arg.piece_len == 0 ) break;
    result = extapi->get_piece( arg_handle, 1, &arg, offset );
}
MessageBoxA( NULL, string,
    "SQL Anywhere",
    MB_OK | MB_TASKMODAL );
free( string );
return;
}
```

set_value-Callback

```
short (SQL_CALLBACK *set_value)
(
    void *          arg_handle,
    a_sql_uint32    arg_num,
    an_extfn_value *value
    short          append
);
```

Mit der Callback-Funktion `set_value` können die Werte von OUT-Parametern und das RETURNS-Ergebnis einer gespeicherten Funktion festgelegt werden. Legen Sie den RETURNS-Wert mit dem `arg_num`-Wert 0 fest. Im Folgenden finden Sie ein Beispiel.

```
an_extfn_value    retval;

retval.type = DT_LONGVARCHAR;
retval.data = result;
retval.piece_len = retval.len.total_len = (a_sql_uint32) strlen( result );
extapi->set_value( arg_handle, 0, &retval, 0 );
```

Das `append`-Argument von `set_value` legt fest, ob die gelieferten Daten die bestehenden Daten ersetzen (FALSE) oder ergänzen (TRUE). Sie müssen `set_value` mit `append=FALSE` aufrufen, bevor Sie es mit `append=TRUE` für dasselbe Argument aufrufen. Das `append`-Argument wird bei Datentypen mit fester Länge ignoriert.

Damit NULL zurückgegeben wird, setzen Sie das Datenfeld der Struktur `an_extfn_value` auf NULL.

set_cancel-Callback

```
void (SQL_CALLBACK *set_cancel)
(
    void *arg_handle,
    void *cancel_handle
);
```

Externe Funktionen können die Werte von IN- oder INOUT-Parametern erhalten und können die Werte von OUT-Parametern und das RETURNS-Ergebnis einer gespeicherten Funktion festlegen. In einem

bestimmten Fall sind die erhaltenen Parameterwerte jedoch möglicherweise nicht mehr gültig oder die Einstellung der Werte ist nicht mehr erforderlich. Dieser Fall tritt ein, wenn eine ausführende SQL-Anweisung abgebrochen wird. Der Grund hierfür kann sein, dass eine Anwendung plötzlich vom Datenbankserver getrennt wird. Sie können für diese Situation einen speziellen Eintrittspunkt in der Bibliothek mit dem Namen `extfn_cancel` definieren. Wenn diese Funktion definiert wurde, wird sie vom Server immer dann aufgerufen, wenn eine laufende SQL-Anweisung abgebrochen wird.

Die Funktion `extfn_cancel` wird mit einem Handle aufgerufen, das auf jede gewünschte Weise verwendet werden kann. Eine typische Verwendung des Handles ist das indirekte Setzen eines Parameters, der anzeigt, dass die aufrufende SQL-Anweisung abgebrochen wurde.

Der übergebene Wert des Handles kann von Funktionen in der externen Bibliothek mit der Callback-Funktion `set_cancel` festgelegt werden. Dieser Vorgang wird im folgenden Codefragment veranschaulicht.

```
extern "C" __declspec( dllexport )
void extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}

extern "C" __declspec( dllexport )
void mystring( an_extfn_api *api, void *arg_handle )
{
    .
    .
    .
    short   canceled = 0;

    extapi->set_cancel( arg_handle, &canceled );

    .
    .
    .
    if( canceled )
```

Das Festlegen einer statischen globalen Variablen "canceled" ist ungeeignet, da dies als Abbruch aller SQL-Anweisungen auf allen Verbindungen interpretiert würde, was normalerweise nicht der Fall ist. Aus diesem Grund wird die `set_cancel`-Callback-Funktion bereitgestellt. Stellen Sie sicher, dass die `canceled`-Variable vor dem Aufruf von `set_cancel` initialisiert wurde.

Es ist wichtig, die Einstellung der `canceled`-Variablen an strategischen Punkten der externen Funktion zu überprüfen. Zu den strategischen Punkten zählen Punkte vor und nach dem Aufruf von Schnittstellenfunktionen für externe Bibliotheksaufrufe wie `get_value` und `set_value`. Wenn die Variable festgelegt wird (als Ergebnis des Aufrufs von `extfn_cancel`), kann die externe Funktion entsprechende Beendigungsaktionen ausführen. Im Folgenden wird der Vorgang durch ein Codefragment basierend auf dem obigen Beispiel veranschaulicht:

```
if( canceled )
{
    free( string );
    return;
}
```

Hinweise

Die `get_piece`-Funktion für ein bestimmtes Argument kann nur sofort nach der `get_value`-Funktion für dasselbe Argument aufgerufen werden.

Der Aufruf eines OUT-Parameter gibt das Feld type der Struktur an_extfn_value zurück, das den Datentyp des Arguments enthält, sowie das Feld data der Struktur an_extfn_value, das auf NULL gesetzt ist.

Die Header-Datei extfnapi.h im SQL Anywhere-Installationsverzeichnis SDK\Include enthält einige zusätzliche Hinweise.

In der folgenden Tabelle werden die Bedingungen aufgeführt, unter welchen die in an_extfn_api definierten Funktionen den Wert FALSE zurückgeben:

Funktion	Gibt 0 zurück, wenn das Folgende zutrifft, andernfalls 1
get_value()	<ul style="list-style-type: none">arg_num ist ungültig. Beispiel: arg_num ist größer als die Anzahl der Argumente für die externe Funktion.
get_piece()	<ul style="list-style-type: none">arg_num ist ungültig. Beispiel: arg_num entspricht nicht der Anzahl der Argumente, die im vorherigen Aufruf von get_value verwendet wurden.Der Offset ist größer als die Gesamtlänge des Werts für das Argument arg_num.Es wird vor dem Aufruf von get_value aufgerufen.
set_value()	<ul style="list-style-type: none">arg_num ist ungültig. Beispiel: arg_num ist größer als die Anzahl der Argumente für die externe Funktion.arg_num ist nur ein Eingabeargument.Der angegebene Wertetyp stimmt nicht mit dem des Arguments arg_num überein.

Behandlung von Datentypen

Datentypen

Die folgenden Datentypen können an eine externe Bibliothek übergeben werden:

SQL-Datentyp	sqldef.h	C-Typ
CHAR	DT_FIXCHAR	Zeichendaten mit einer definierten Länge
VARCHAR	DT_VARCHAR	Zeichendaten mit einer definierten Länge
LONG VARCHAR, TEXT	DT_LONG-VARCHAR	Zeichendaten mit einer definierten Länge
UNIQUEIDENTIFIERSTR	DT_FIXCHAR	Zeichendaten mit einer definierten Länge

SQL-Datentyp	sqldef.h	C-Typ
XML	DT_LONG-VARCHAR	Zeichendaten mit einer definierten Länge
NCHAR	DT_NFIX-CHAR	UTF-8-Zeichendaten mit einer definierten Länge
NVARCHAR	DT_NVAR-CHAR	UTF-8-Zeichendaten mit einer definierten Länge
LONG NVARCHAR, NTEXT	DT_LONG-NVARCHAR	UTF-8-Zeichendaten mit einer definierten Länge
UNIQUEIDENTIFIER	DT_BINARY	Binärdaten mit einer Länge von 16 Byte
BINARY	DT_BINARY	Binärdaten mit einer definierten Länge
VARBINARY	DT_BINARY	Binärdaten mit einer definierten Länge
LONG BINARY	DT_LONGBI-NARY	Binärdaten mit einer definierten Länge
TINYINT	DT_TINYINT	1-Byte-Ganzzahl
[UNSIGNED] SMALLINT	DT_SMAL-LINT, DT_UNSSMAL LINT	[Vorzeichenlose] 2-Byte-Ganzzahl
[UNSIGNED] INT	DT_INT, DT_UNSENT	[Vorzeichenlose] 4-Byte-Ganzzahl
[UNSIGNED] BIGINT	DT_BIGINT, DT_UNSBIGINT	[Vorzeichenlose] 8-Byte-Ganzzahl
REAL, FLOAT(1-24)	DT_FLOAT	Einfachgenaue Gleitkommazahl
DOUBLE, FLOAT(25-53)	DT_DOUBLE	Doppeltgenaue Gleitkommazahl

Es ist nicht möglich, die date- oder time-Datentypen oder die Datentypen DECIMAL oder NUMERIC (einschließlich der money-Datentypen) zu verwenden.

Um Werte für INOUT oder OUT-Parameter zu liefern, verwenden Sie die Funktion set_value. Um IN- und INOUT-Parameter zu lesen, verwenden Sie die Funktion get_value.

Parameter-Datentypen ermitteln

Nach einem Aufruf von `get_value` kann mit der Struktur `an_extfn_value` der Datentyp des Parameters abgefragt werden. Das folgende Codefragment zeigt ein Beispiel zur Identifizierung des Parameterdatentyps.

```
an_extfn_value      arg;
a_sql_data_type     data_type;

extapi->get_value( arg_handle, 1, &arg );
data_type = arg.type & DT_TYPES;
switch( data_type )
{
case DT_FIXCHAR:
case DT_VARCHAR:
case DT_LONGVARCHAR:
    break;
default:
    return;
}
```

Weitere Hinweise zu Datentypen finden Sie unter „[Hostvariablen in Embedded SQL](#)“ auf Seite 485.

UTF-8-Datentypen

Die UTF-8-Datentypen wie `NCHAR`, `NVARCHAR`, `LONG NVARCHAR` und `NTEXT` werden als UTF-8-kodierte Zeichenfolgen übergeben. Mit einer Funktion wie der Windows-Funktion `MultiByteToWideChar` kann eine UTF-8-Zeichenfolge in eine weite (Unicode-) Zeichenfolge konvertiert werden.

NULL übergeben

Sie können `NULL` als gültigen Wert für alle Argumente übergeben. Funktionen in externen Bibliotheken können `NULL` als Rückgabewert für jeden Datentyp übergeben.

Rückgabewerte

Um einen Rückgabewert in einer externen Funktion festzulegen, rufen Sie die Funktion `set_value` mit diesem `arg_num`-Parameterwert 0 auf. Wenn `set_value` nicht mit dem Wert für `arg_num` aufgerufen wird, ist das Funktionsergebnis `NULL`.

Auch der Datentyp eines Rückgabewerts eines Aufrufs einer gespeicherten Funktion sollte unbedingt festgelegt werden. Das folgende Codefragment zeigt, wie Sie den Rückgabe-Datentyp festlegen.

```
an_extfn_value      retval;

retval.type = DT_LONGVARCHAR;
retval.data = result;
retval.pieces_len = retval.len.total_len = (a_sql_uint32) strlen( result );
extapi->set_value( arg_handle, 0, &retval, 0 );
```

Externe Vorfilterbibliotheken entladen

Mit der `sa_external_library_unload`-Systemprozedur kann eine externe Bibliothek entladen werden, wenn die Bibliothek nicht verwendet wird. Die Prozedur hat einen optionalen Parameter vom Datentyp `long`

varchar. Der Parameter gibt den Namen der zu entladenden Bibliothek an. Wenn kein Parameter angegeben wird, werden alle zur Zeit nicht verwendeten externen Bibliotheken entladen.

Im folgenden Beispiel wird eine externe Funktionsbibliothek entladen.

```
CALL sa_external_library_unload( 'library.dll' );
```

Diese Funktion ist bei der Entwicklung einer Gruppe externer Funktionen nützlich, da der Datenbankserver nicht heruntergefahren werden muss, um eine neuere Version der Bibliothek zu installieren.

Unterstützung für externe Umgebungen in SQL Anywhere

SQL Anywhere stellt Unterstützung für sechs externe Laufzeitumgebungen bereit. Diese umfassen Embedded SQL und ODBC-Anwendungen, die in C/C++ geschrieben wurden, sowie in Java, Perl, PHP oder in Sprachen wie C# und Visual Basic geschriebene Anwendungen, die auf der Common Language Runtime (CLR) von Microsoft basieren.

SQL Anywhere besaß die Fähigkeit, kompilierte native Funktionen, die in C oder C++ geschrieben waren, aufzurufen. Wenn diese Prozeduren vom Server ausgeführt wurden, wurde die DLL oder das Shared Object immer vom Datenbankserver geladen und die Aufrufe der nativen Funktion wurden immer vom Datenbankserver durchgeführt. Dadurch bestand das Risiko, dass der Datenbankserver abstürzte, wenn die native Funktion einen Fehler verursachte. Durch die Ausführung kompilierter nativer Funktionen außerhalb des Datenbankservers in einer externen Umgebung wird dieses Risiko nun vermieden.

Es folgt ein Überblick über die Unterstützung von externen Umgebungen in SQL Anywhere.

- Die Anweisungen `START EXTERNAL ENVIRONMENT` und `STOP EXTERNAL ENVIRONMENT` werden verwendet, um eine externe Umgebung bei Bedarf zu starten bzw. zu stoppen. Diese Anweisungen sind optional, da externe Umgebungen automatisch gestartet und gestoppt werden, wenn sie benötigt werden.
- Die `ALTER EXTERNAL ENVIRONMENT`-Anweisung wird verwendet, um den Speicherort einer externen Umgebung festzulegen oder zu ändern.
- Die `COMMENT ON EXTERNAL ENVIRONMENT`-Anweisung wird verwendet, um einer externen Umgebung einen Kommentar hinzuzufügen.
- Wenn eine externe Umgebung für die Verwendung auf dem Datenbankserver eingerichtet ist, können Sie Objekte in der Datenbank installieren und gespeicherte Prozeduren und Funktionen erstellen, die diese Objekte innerhalb der externen Umgebung nutzen.
- Die `INSTALL EXTERNAL OBJECT`-Anweisung wird verwendet, um ein externes Perl- oder PHP-Objekt (z.B. ein Perl-Skript) aus einer Datei oder einem Ausdruck in der Datenbank zu installieren. Sobald die externen Objekte in der Datenbank installiert sind, können sie in Definitionen für externe gespeicherte Prozeduren und Funktionen verwendet werden.
- Die `COMMENT ON EXTERNAL ENVIRONMENT OBJECT`-Anweisung wird verwendet, um einem externen Umgebungsobjekt einen Kommentar hinzuzufügen.
- Wenn Sie ein installiertes externes Perl- oder PHP-Objekt aus der Datenbank entfernen möchten, verwenden Sie die `REMOVE EXTERNAL OBJECT`-Anweisung.
- Die Anweisungen `CREATE PROCEDURE` und `CREATE FUNCTION` werden verwendet, um Definitionen für externe gespeicherte Prozeduren und Funktionen zu erstellen. Sie können wie jede andere gespeicherte Prozedur oder Funktion in der Datenbank verwendet werden. Wenn der Datenbankserver auf eine gespeicherte Prozedur oder Funktion einer externen Umgebung trifft, startet

er automatisch die externe Umgebung (falls sie noch nicht gestartet ist) und übersendet alle erforderlichen Informationen, damit die externe Umgebung das externe Objekt in der Datenbank abrufen und ausführt. Daraus resultierende Ergebnismengen und Rückgabewerte werden nach Bedarf zurückgegeben.

Siehe auch

- „ALTER EXTERNAL ENVIRONMENT-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „COMMENT-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [externer Aufruf]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE PROCEDURE-Anweisung [externer Aufruf]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „INSTALL EXTERNAL OBJECT-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „REMOVE EXTERNAL OBJECT-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „START EXTERNAL ENVIRONMENT-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „STOP EXTERNAL ENVIRONMENT-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Die externe CLR-Umgebung

SQL Anywhere stellt Unterstützung für gespeicherte CLR-Prozeduren und -Funktionen bereit. Eine gespeicherte CLR-Prozedur oder -Funktion verhält sich wie eine gespeicherte SQL-Prozedur oder -Funktion, abgesehen davon, dass der Code für die Prozedur oder Funktion in einer .NET-Sprache wie C# oder Visual Basic geschrieben ist und die Ausführung der Prozedur oder Funktion außerhalb des Datenbankservers stattfindet (d.h. innerhalb eines separaten .NET-Programms). Es gibt nur eine Instanz dieses .NET-Programms pro Datenbank. Alle Verbindungen, die CLR-Funktionen und gespeicherte Prozeduren ausführen, verwenden dieselbe .NET-Programminstanz, aber die Namensbereiche für die einzelnen Verbindungen sind getrennt. Statistiken werden für die Dauer der Verbindung aufrechterhalten, können aber nicht über Verbindungen hinweg gemeinsam genutzt werden. Es wird nur .NET Version 2.0 unterstützt.

Um eine externe CLR-Funktion oder -Prozedur aufzurufen, definieren Sie eine entsprechende gespeicherte Prozedur oder Funktion mit einer EXTERNAL NAME-Zeichenfolge, die festlegt, welche DLL geladen und welche Funktion innerhalb der Assembly aufgerufen werden soll. Sie müssen auch LANGUAGE CLR angeben, wenn Sie die gespeicherte Prozedur oder Funktion definieren. Hier ist eine Beispieldeklaration:

```
CREATE PROCEDURE clr_stored_proc(  
    IN p1 INT,  
    IN p2 UNSIGNED SMALLINT,  
    OUT p3 LONG VARCHAR)  
EXTERNAL NAME 'MyCLRTest.dll::MyCLRTest.Run( int, ushort, out string )'  
LANGUAGE CLR;
```

In diesem Beispiel ruft die gespeicherte Prozedur bei der Ausführung `clr_stored_proc` auf, lädt die DLL `MyCLRTest.dll` und ruft die Funktion `MyCLRTest.Run` auf. Die Prozedur `clr_stored_proc` übernimmt drei SQL-Parameter: zwei IN-Parameter, einen vom Typ INT und einen vom Typ

UNSIGNED SMALLINT, und einen OUT-Parameter des Typs LONG VARCHAR. Auf der .NET-Seite werden diese drei Parameter in Eingabeargumente des Typs "int" und "ushort" sowie in ein Ausgabeargument des Typs "string" konvertiert. Zusätzlich zu out-Argumenten kann die CLR-Funktion auch ref-Argumente haben. Ein Benutzer muss ein Argument "ref CLR" deklarieren, wenn die entsprechende gespeicherte Prozedur einen INOUT-Parameter hat.

Die folgende Tabelle enthält die verschiedenen CLR-Argumenttypen und den entsprechenden empfohlenen SQL-Datentyp:

CLR-Typ	Empfohlener SQL-Datentyp
bool	bit
byte	tinyint
short	smallint
ushort	unsigned smallint
int	int
uint	unsigned int
long	bigint
ulong	unsigned bigint
decimal	numeric
float	real
double	double
DateTime	timestamp
string	long varchar
byte[]	long binary

Die Deklaration der DLL kann ein relativer oder absoluter Pfad sein. Wird ein relativer Pfad angegeben, durchsucht das externe .NET-Programm den Pfad und andere Positionen nach der DLL. Das Programm durchsucht nicht den Global Assembly Cache (GAC) nach der DLL.

Wie die bestehenden gespeicherten Java-Prozeduren und Funktionen können gespeicherte CLR-Prozeduren und -Funktionen serverseitige Anforderungen an die Datenbank und Ergebnismengen zurückgeben. Außerdem werden wie bei Java alle an Console.Out und Console.Error übermittelten Informationen automatisch an das Meldungsfenster des Datenbankservers weitergeleitet.

Weitere Hinweise zur Durchführung von serverseitigen Anforderungen und zur Rückgabe von Ergebnismengen einer CLR-Funktion oder gespeicherten Prozedur finden Sie in den Beispielen unter `%SQLANY16%\SQLAnywhere\ExternalEnvironments\CLR`.

Um CLR in der Datenbank zu verwenden, vergewissern Sie sich, dass der Datenbankserver in der Lage ist, die Position des CLR-Programms zu ermitteln und es zu starten. Sie können überprüfen, ob der Datenbankserver in der Lage ist, die Position des CLR-Programms zu ermitteln und es zu starten, indem Sie die folgende Anweisung ausführen:

```
START EXTERNAL ENVIRONMENT CLR;
```

Wenn der Datenbankserver CLR nicht starten kann, findet er wahrscheinlich die CLR-Programmdatei nicht. Die CLR-Programmdatei ist `dbextclr16.exe`. Vergewissern Sie sich, dass diese Datei im Ordner `%SQLANY16%\Bin32` oder `%SQLANY16%\Bin64` vorhanden ist, je nach verwendeter Datenbankserver-Version.

Die `START EXTERNAL ENVIRONMENT CLR`-Anweisung wird nur benötigt, um zu überprüfen, ob der Datenbankserver CLR-Programmdateien starten kann. Üblicherweise startet CLR automatisch, wenn Sie einen Aufruf einer gespeicherten CLR-Prozedur oder Funktion durchführen.

Die Anweisung `STOP EXTERNAL ENVIRONMENT CLR` ist auch nicht erforderlich, um eine Instanz von CLR zu stoppen, weil die Instanz mit dem Ende der Verbindung automatisch beendet wird. Wenn Sie allerdings die Arbeit mit CLR abgeschlossen haben und Ressourcen freigeben wollen, entfernt die Anweisung `STOP EXTERNAL ENVIRONMENT CLR` die CLR-Instanz für Ihre Verbindung.

Im Gegensatz zu den externen Perl-, PHP- und Java-Umgebungen muss bei der CLR-Umgebung kein Objekt in der Datenbank installiert werden. Daher müssen Sie keine `INSTALL`-Anweisungen ausführen, bevor Sie die externe CLR-Umgebung verwenden.

Hier ist ein Beispiel einer in C# geschriebenen Funktion, die in einer externen Umgebung ausgeführt werden kann.

```
public class StaticTest
{
    private static int val = 0;

    public static int GetValue() {
        val += 1;
        return val;
    }
}
```

Wenn sie in eine dynamische Verknüpfungsbibliothek kompiliert wird, kann diese Funktion von einer externen Umgebung aufgerufen werden. Ein ausführbares Image namens `dbextclr16.exe` wird vom Datenbankserver gestartet und lädt die dynamische Verknüpfungsbibliothek. In SQL Anywhere stehen verschiedene Versionen dieses Programms zur Verfügung. Unter Windows beispielsweise gibt es sowohl 32-Bit-Programme als auch 64-Bit-Programme. Die eine Version ist für die Verwendung mit der 32-Bit-Version und die andere für die Verwendung mit der 64-Bit-Version des Datenbankservers bestimmt.

Um diese Anwendung in eine dynamische Verknüpfungsbibliothek unter Verwendung des C#-Compilers von Microsoft zu kompilieren, verwenden Sie einen ähnlichen Befehl wie den folgenden: Hierbei wird angenommen, dass sich der Quellcode für das obenstehende Beispiel in einer Datei namens `StaticTest.cs` befindet.

```
csc /target:library /out:clrtest.dll StaticTest.cs
```

Dieser Befehl platziert den kompilierten Code in eine DLL namens *clrtest.dll*. Um die kompilierte C#-Funktion aufzurufen, wird ein Wrapper unter Verwendung von Interactive SQL folgendermaßen festgelegt:

```
CREATE FUNCTION stc_get_value()  
RETURNS INT  
EXTERNAL NAME 'clrtest.dll::StaticTest.GetValue() int'  
LANGUAGE CLR;
```

Bei CLR wird die Zeichenfolge EXTERNAL NAME in einer einzigen Zeile mit SQL-Code angegeben. Möglicherweise müssen Sie den Pfad zur DLL als Teil der Zeichenfolge EXTERNAL NAME aufnehmen, damit die DLL gefunden wird. Im Fall von abhängigen Assemblys (z.B. wenn *myLib.dll* Code enthält, der Funktionen in *myOtherLib.dll* aufruft oder auf andere Weise von dieser abhängt) liegt es am .NET-Framework, die Abhängigkeiten aufzulösen. Die externe CLR-Umgebung führt das Laden der angegebenen Assembly durch, aber möglicherweise sind zusätzliche Schritte erforderlich, um zu gewährleisten, dass abhängige Assemblys geladen werden. Eine Möglichkeit besteht darin, alle Abhängigkeiten im globalen Assembly-Cache (**GAC**) zu registrieren, indem Sie das **gacutil**-Dienstprogramm von Microsoft verwenden, das mit dem .NET-Framework installiert wird. In Fall von benutzerdefinierten Bibliotheken erfordert gacutil, dass diese mit einem starken Namensschlüssel signiert werden, bevor sie im **GAC** registriert werden können.

Für die kompilierte C#-Beispielfunktion führen Sie die folgende Anweisung aus.

```
SELECT stc_get_value();
```

Bei jedem Aufruf der C#-Funktion wird ein neues Ganzzahl-Ergebnis erzeugt. Die Sequenz der zurückgegebenen Werte ist 1, 2, 3 usw.

Weitere Hinweise und Beispiele zur Unterstützung für CLR in der Datenbank finden Sie unter den Beispielen, die sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\ExternalEnvironments\CLR` befinden.

Siehe auch

- „SQL-Funktionen“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Benutzerdefinierte Funktionen“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)]
- „Systemprozeduren“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Prozeduren“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)]
- „Benannte Parameter“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Die externen ESQL- und ODBC-Umgebungen

SQL Anywhere besaß die Fähigkeit, kompilierte native Funktionen, die in C oder C++ geschrieben waren, aufzurufen. Wenn diese Prozeduren vom Server ausgeführt wurden, wurde die DLL oder das Shared Object immer vom Datenbankserver geladen und die Aufrufe der nativen Funktion wurden immer vom Datenbankserver durchgeführt. Der native Aufruf durch den Datenbankserver ist zwar am effizientesten, kann aber schwerwiegende Konsequenzen haben, wenn eine native Funktion Probleme verursacht. Dies gilt besonders, wenn die native Funktion in eine Endlosschleife gerät, weil dann der

Datenbankserver hängen würde, oder wenn die native Funktion einen Fehler bewirkt, weil dann der Datenbankserver abstürzen würde. Aus diesem Grund haben Sie die Möglichkeit, kompilierte native Funktionen außerhalb des Datenbankservers in einer externen Umgebung auszuführen. Es gibt einige wichtige Vorteile, eine kompilierte native Funktion in einer externen Umgebung auszuführen:

1. Der Datenbankserver hängt nicht oder stürzt nicht ab, wenn eine native Funktion Probleme verursacht.
2. Die native Funktion kann so geschrieben werden, dass sie ODBC, Embedded SQL (ESQL) oder die SQL Anywhere-C-API verwendet und in der Lage ist, serverseitige Aufrufe zurück zum Datenbankserver durchzuführen, ohne eine Verbindung herstellen zu müssen.
3. Die native Funktion kann an den Datenbankserver eine Ergebnismenge zurückgeben.
4. In einer externen Umgebung kann ein 32-Bit-Datenbankserver mit einer kompilierten nativen 64-Bit-Funktion kommunizieren, bzw. umgekehrt. Dies ist nicht möglich, wenn die kompilierten nativen Funktionen direkt in den Adressraum des Datenbankservers geladen werden. Eine 32-Bit-Bibliothek kann nur durch einen 32-Bit-Datenbankserver und eine 64-Bit-Bibliothek kann nur durch einen 64-Bit-Datenbankserver geladen werden.

Das Ausführen einer kompilierten nativen Funktion in einer externen Umgebung statt im Datenbankserver führt zu einer gewissen Performanceeinbuße.

Außerdem muss die kompilierte native Funktion die API für native Funktionsaufrufe verwenden, um Informationen an die native Funktion zu übergeben und aus dieser zurückzugeben. Eine Beschreibung dieser API finden Sie unter „[SQL Anywhere-Schnittstelle für externe Aufrufe](#)“ auf Seite 601.

Um eine kompilierte native C-Funktion in einer externen Umgebung statt im Datenbankserver auszuführen, wird die gespeicherte Prozedur oder Funktion mit der Klausel EXTERNAL NAME definiert, gefolgt vom Attribut LANGUAGE, das C_ESQL32, C_ESQL64, C_ODBC32 oder C_ODBC64 angibt.

Anders als bei den externen Perl-, PHP- und Java-Umgebungen müssen Sie keinen Quellcode oder kompilierte Objekte in der Datenbank installieren. Daher müssen Sie keine INSTALL-Anweisungen ausführen, bevor Sie die externen ESQL- und ODBC-Umgebungen verwenden.

Nachfolgend finden Sie ein Beispiel einer in C++ geschriebenen Funktion, die im Datenbankserver oder in einer externen Umgebung ausgeführt werden kann.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    return TRUE;
}
```

```

// Note: extfn_use_new_api used only for
// execution in the database server

extern "C" __declspec( dllexport )
a_sql_uint32 extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}

extern "C" __declspec( dllexport )
void SimpleCFunction(
    an_extfn_api *api,
    void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    an_extfn_value retval;
    int *          intptr;
    int            i, j, k;

    j = 1000;
    k = 0;
    for( i = 1; i <= 4; i++ )
    {
        result = api->get_value( arg_handle, i, &arg );
        if( result == 0 || arg.data == NULL ) break;
        if( arg.type & DT_TYPES != DT_INT ) break;
        intptr = (int *) arg.data;
        k += *intptr * j;
        j = j / 10;
    }
    retval.type = DT_INT;
    retval.data = (void*)&k;
    retval.piece_len = retval.len.total_len =
        (a_sql_uint32) sizeof( int );
    api->set_value( arg_handle, 0, &retval, 0 );
    return;
}

```

Wenn sie in eine DLL oder in ein Shared Object kompiliert wird, kann diese Funktion von einer externen Umgebung aufgerufen werden. Ein ausführbares Image namens *dbexternc16* wird vom Datenbankserver gestartet und lädt die DLL bzw. das Shared Object. In SQL Anywhere stehen verschiedene Versionen dieses Programms zur Verfügung. Unter Windows gibt es beispielsweise sowohl 32-Bit-Programme als auch 64-Bit-Programme.

Die 32- oder 64-Bit-Version des Datenbankservers kann verwendet werden und beide Versionen können die 32- oder 64-Bit-Version von *dbexternc16* starten. Dies ist einer der Vorteile bei der Verwendung der externen Umgebung. Nachdem *dbexternc16* vom Datenbankserver gestartet wurde, wird es erst beendet, wenn die Verbindung beendet oder die STOP EXTERNAL ENVIRONMENT-Anweisung (mit dem korrekten Umgebungsnamen) ausgeführt wird. Jede Verbindung, die einen externen Umgebungsaufufruf ausführt, erhält ihre eigene Kopie von *dbexternc16*.

Um die kompilierte native Funktion SimpleCFunction aufzurufen, wird ein Wrapper folgendermaßen definiert:

```

CREATE FUNCTION SimpleCDemo(
    IN arg1 INT,
    IN arg2 INT,
    IN arg3 INT,

```

```
IN arg4 INT )
RETURNS INT
EXTERNAL NAME 'SimpleCFFunction@c:\\c\\extdemo.dll'
LANGUAGE C_ODBC32;
```

Dies ist beinahe identisch mit der Art der Beschreibung einer kompilierten nativen Funktion, wenn sie in den Adressraum des Datenbankservers geladen wird. Der einzige Unterschied ist die Verwendung der LANGUAGE C_ODBC32-Klausel. Diese Klausel gibt an, dass SimpleCDemo eine Funktion ist, die in einer externen Umgebung ausgeführt wird und 32-Bit-ODBC-Aufrufe verwendet. Die Sprachspezifikation C_ESQL32, C_ESQL64, C_ODBC32 oder C_ODBC64 teilt dem Datenbankserver mit, ob die externe C-Funktion 32-Bit- bzw. 64-Bit-ODBC-, ESQL- oder SQL Anywhere-C-API-Aufrufe ausgibt, wenn serverseitige Anforderungen erfolgen.

Wenn die native Funktion keine ODBC-, ESQL- oder SQL Anywhere-C-API-Aufrufe verwendet, um serverseitige Anforderungen durchzuführen, kann entweder C_ODBC32 oder C_ESQL32 bei 32-Bit-Anwendungen und entweder C_ODBC64 oder C_ESQL64 bei 64-Bit-Anwendungen verwendet werden. Dies ist in der oben gezeigten externen C-Funktion der Fall. Sie verwendet keine dieser APIs.

Zum Ausführen der kompilierten nativen Beispielfunktion verwenden Sie die folgende Anweisung.

```
SELECT SimpleCDemo(1,2,3,4);
```

Für eine serverseitige ODBC-Anbindung muss der C-/C++-Code die Standarddatenbankverbindung verwenden. Um ein Handle für die Datenbankverbindung zu erhalten, rufen Sie get_value mit einem EXTFN_CONNECTION_HANDLE_ARG_NUM-Argument auf. Das Argument teilt dem Datenbankserver mit, die aktuelle externe Umgebungsverbindung zurückzugeben, anstatt eine neue zu öffnen.

```
#include <windows.h>
#include <stdio.h>
#include "odbc.h"
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    return TRUE;
}

extern "C" __declspec( dllexport )
void ServerSideFunction( an_extfn_api *api, void *arg_handle )
{
    short          result;
    an_extfn_value  arg;
    an_extfn_value  retval;
    SQLRETURN       ret;

    ret = -1;
    // set up the return value struct
    retval.type = DT_INT;
    retval.data = (void*) &ret;
    retval.piece_len = retval.len.total_len =
        (a_sql_uint32) sizeof( int );

    result = api->get_value( arg_handle,
        EXTFN_CONNECTION_HANDLE_ARG_NUM,
```

```

        &arg );
if( result == 0 || arg.data == NULL )
{
    api->set_value( arg_handle, 0, &retval, 0 );
    return;
}

HDBC dbc = (HDBC)arg.data;
HSTMT stmt = SQL_NULL_HSTMT;
ret = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( ret != SQL_SUCCESS ) return;
ret = SQLExecDirect( stmt,
    (SQLCHAR *) "INSERT INTO odbcTab "
        "SELECT table_id, table_name "
        "FROM SYS.SYSTAB", SQL_NTS );
if( ret == SQL_SUCCESS )
{
    SQLExecDirect( stmt,
        (SQLCHAR *) "COMMIT", SQL_NTS );
}
SQLFreeHandle( SQL_HANDLE_STMT, stmt );

api->set_value( arg_handle, 0, &retval, 0 );
return;
}

```

Wenn der vorstehende ODBC-Code in der Datei *extodbc.cpp* gespeichert ist, kann er für Windows unter Verwendung der folgenden Befehle kompiliert werden (unter der Annahme, dass sich die SQL Anywhere-Software im Ordner *c:\sa16* befindet und Microsoft Visual C++ installiert ist).

```
cl extodbc.cpp /LD /Ic:\sa16\sdk\include odbc32.lib
```

Das folgende Beispiel erstellt eine Tabelle, definiert eine gespeicherte Prozedur als Wrapper für die kompilierte native Funktion und ruft diese an auf, um die Tabelle mit Daten zu füllen.

```

CREATE TABLE odbcTab(c1 int, c2 char(128));

CREATE FUNCTION ServerSideODBC( )
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extodbc.dll'
LANGUAGE C_ODBC32;

SELECT ServerSideODBC();

// The following statement should return two identical rows
SELECT COUNT(*) FROM odbcTab
UNION ALL
SELECT COUNT(*) FROM SYS.SYSTAB;

```

In diesem Sinne gilt: Für eine serverseitige ESQL-Anbindung muss der C-/C++-Code die Standarddatenbankverbindung verwenden. Um ein Handle für die Datenbankverbindung zu erhalten, rufen Sie `get_value` mit einem `EXTFN_CONNECTION_HANDLE_ARG_NUM`-Argument auf. Das Argument teilt dem Datenbankserver mit, die aktuelle externe Umgebungsverbindung zurückzugeben, anstatt eine neue zu öffnen.

```

#include <windows.h>
#include <stdio.h>

#include "sqlca.h"
#include "sqlda.h"

```

```
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    return TRUE;
}

EXEC SQL INCLUDE SQLCA;
static SQLCA *_sqlc;
EXEC SQL SET SQLCA "_sqlc";
EXEC SQL WHENEVER SQLERROR { ret = _sqlc->sqlcode; };

extern "C" __declspec( dllexport )
void ServerSideFunction( an_extfn_api *api, void *arg_handle )
{
    short          result;
    an_extfn_value  arg;
    an_extfn_value  retval;

    EXEC SQL BEGIN DECLARE SECTION;
    char *stmt_text =
        "INSERT INTO esqlTab "
        "SELECT table_id, table_name "
        "FROM SYS.SYSTAB";
    char *stmt_commit =
        "COMMIT";
    EXEC SQL END DECLARE SECTION;

    int ret = -1;

    // set up the return value struct
    retval.type = DT_INT;
    retval.data = (void*) &ret;
    retval.piece_len = retval.len.total_len =
        (a_sql_uint32) sizeof( int );

    result = api->get_value( arg_handle,
        EXTFN_CONNECTION_HANDLE_ARG_NUM,
        &arg );
    if( result == 0 || arg.data == NULL )
    {
        api->set_value( arg_handle, 0, &retval, 0 );
        return;
    }
    ret = 0;
    _sqlc = (SQLCA *)arg.data;

    EXEC SQL EXECUTE IMMEDIATE :stmt_text;
    EXEC SQL EXECUTE IMMEDIATE :stmt_commit;

    api->set_value( arg_handle, 0, &retval, 0 );
}
```

Wenn die vorstehenden Embedded SQL-Anweisungen in der Datei *extesql.cpp* gespeichert sind, können sie für Windows unter Verwendung der folgenden Befehle kompiliert werden (unter der Annahme, dass sich die SQL Anywhere-Software im Ordner *c:\sa16* befindet und Microsoft Visual C++ installiert ist).

```
sqlpp extesql.sqc extesql.cpp
cl extesql.cpp /LD /Ic:\sa16\sdk\include c:\sa16\sdk\lib\x86\dblibtm.lib
```

Das folgende Beispiel erstellt eine Tabelle, definiert eine gespeicherte Prozedur als Wrapper für die kompilierte native Funktion und ruft diese an, um die Tabelle mit Daten zu füllen.

```
CREATE TABLE esqlTab(c1 int, c2 char(128));

CREATE FUNCTION ServerSideESQL( )
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extesql.dll'
LANGUAGE C_ESQL32;

SELECT ServerSideESQL();

// The following statement should return two identical rows
SELECT COUNT(*) FROM esqlTab
UNION ALL
SELECT COUNT(*) FROM SYS.SYSTAB;
```

Wie in den vorherigen Beispielen muss der C/C++-Code die Standard-Datenbankverbindung benutzen, um SQL Anywhere-C-API-Aufrufe zu verwenden. Um ein Handle für die Datenbankverbindung zu erhalten, rufen Sie `get_value` mit einem `EXTFN_CONNECTION_HANDLE_ARG_NUM`-Argument auf. Das Argument teilt dem Datenbankserver mit, die aktuelle externe Umgebungsverbindung zurückzugeben, anstatt eine neue zu öffnen. Das folgende Beispiel zeigt das Framework, um den Verbindungshandle zu erhalten, die C-API-Umgebung zu initialisieren und den Verbindungshandle in ein Verbindungsobjekt (**a_sqlany_connection**) umzuwandeln, das mit der SQL Anywhere-C-API verwendet werden kann.

```
#include <windows.h>
#include "sacapidll.h"
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    return TRUE;
}

extern "C" __declspec( dllexport )
void ServerSideFunction( an_extfn_api *extapi, void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    an_extfn_value retval;
    unsigned       offset;
    char           *cmd;

    SQLAnywhereInterface capi;
    a_sqlany_connection * sqlany_conn;
    unsigned int      max_api_ver;

    result = extapi->get_value( arg_handle,
        EXTFN_CONNECTION_HANDLE_ARG_NUM,
        &arg );

    if( result == 0 || arg.data == NULL )
    {
        return;
    }
    if( !sqlany_initialize_interface( &capi, NULL ) )
    {

```

```
        return;
    }
    if( !capi.sqlany_init( "MyApp",
        SQLANY_CURRENT_API_VERSION,
        &max_api_ver ) )
    {
        sqlany_finalize_interface( &capi );
        return;
    }
    sqlany_conn = sqlany_make_connection( arg.data );

    // processing code goes here

    capi.sqlany_fini();

    sqlany_finalize_interface( &capi );
    return;
}
```

Wenn der vorstehende C-Code in der Datei *extcapi.c* gespeichert ist, kann er für Windows unter Verwendung der folgenden Befehle kompiliert werden (unter der Annahme, dass sich die SQL Anywhere-Software im Ordner *c:\sa16* befindet und Microsoft Visual C++ installiert ist).

```
cl /LD /Tp extcapi.c /Tp c:\sa16\SDK\C\sacapidll.c
/Ic:\sa16\SDK\Include c:\sa16\SDK\Lib\X86\dbcapi.lib
```

Das folgende Beispiel definiert eine gespeicherte Prozedur als Wrapper für die kompilierte native Funktion und ruft diese dann auf.

```
CREATE FUNCTION ServerSideC()
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extcapi.dll'
LANGUAGE C_ESQL32;

SELECT ServerSideC();
```

Das LANGUAGE-Attribut im Beispiel oben gibt C_ESQL32 an. Bei 64-Bit-Anwendungen würden Sie C_ESQL64 verwenden. Sie müssen das Embedded SQL-Sprachattribut verwenden, weil die SQL Anywhere-C-API auf derselben Ebene (Bibliothek) wie ESQL kompiliert ist.

Wie bereits erwähnt, startet jede Verbindung, die einen externen Umgebungsaufruf ausführt, ihre eigene Kopie von *dbexternc12*. Dieses Programm wird automatisch vom Datenbankserver geladen, wenn zum ersten Mal ein externer Umgebungsaufruf durchgeführt wird. Sie können allerdings die Anweisung **START EXTERNAL ENVIRONMENT** verwenden, um *dbexternc16* vorzuladen. Dies ist nützlich, um die geringfügige Verzögerung zu umgehen, die auftritt, wenn ein externer Umgebungsaufruf zum ersten Mal ausgeführt wird. Hier ist ein Beispiel für die Anweisung.

```
START EXTERNAL ENVIRONMENT C_ESQL32
```

Ein weiterer Fall, in dem ein Vorausladen von *dbexternc16* nützlich ist, liegt vor, wenn Sie eine Fehlersuche in Ihrer externen Funktion durchführen wollen. Sie können den Debugger so einsetzen, dass er sich an den laufenden *dbexternc16* anhängt und Breakpoints in Ihrer externen Funktion setzt.

Die Anweisung **STOP EXTERNAL ENVIRONMENT** ist nützlich, wenn Sie eine DLL oder ein Shared Object aktualisieren. Sie beendet das native Bibliotheksladeprogramm *dbexternc16* für die laufende Verbindung und gibt dadurch den Zugriff auf die DLL bzw. das Shared Object frei. Wenn mehrere Verbindungen die DLL bzw. das Shared Object verwenden, müssen alle ihre Kopien von *dbexternc16*

beendet werden. Der entsprechende externe Umgebungsname muss in der Anweisung STOP EXTERNAL ENVIRONMENT angegeben werden.. Hier ist ein Beispiel für die Anweisung.

```
STOP EXTERNAL ENVIRONMENT C_ESQL32
```

Um eine Ergebnismenge aus einer externen Funktion zurückzugeben, muss die kompilierte native Funktion die API für native Funktionsaufrufe verwenden. Eine umfassende Beschreibung dieser API finden Sie unter „[SQL Anywhere-Schnittstelle für externe Aufrufe](#)“ auf Seite 601.

Das folgende Codefragment zeigt, wie Sie eine Informationsstruktur für Ergebnismengen einrichten. Es enthält eine Spaltenzählung, einen Zeiger auf ein Array von Spalteninformationsstrukturen und einen Zeiger auf ein Array von Spaltendatenwert-Strukturen. Das Beispiel verwendet auch die SQL Anywhere C-API.

```
an_extfn_result_set_info    rs_info;

int columns = capi.sqlany_num_cols( sqlany_stmt );

an_extfn_result_set_column_info *col_info =
    (an_extfn_result_set_column_info *)
    malloc( columns * sizeof(an_extfn_result_set_column_info) );

an_extfn_result_set_column_data *col_data =
    (an_extfn_result_set_column_data *)
    malloc( columns * sizeof(an_extfn_result_set_column_data) );

rs_info.number_of_columns   = columns;
rs_info.column_infos        = col_info;
rs_info.column_data_values  = col_data;
```

Das folgende Codefragment zeigt, wie Sie die Ergebnismenge beschreiben. Es verwendet die SQL Anywhere C-API, um Spalteninformationen für eine SQL-Abfrage zu erhalten, die vorher durch die C-API ausgeführt wurde. Die Informationen, die von der SQL Anywhere C-API für jede Spalte bezogen werden, werden in einen Spaltennamen-, Typ-, Breiten-, Index- und Nullwert-Indikator umgewandelt, der verwendet wird, um die Ergebnismenge zu beschreiben.

```
a_sqlany_column_info    info;
for( int i = 0; i < columns; i++ )
{
    if( sqlany_get_column_info( sqlany_stmt, i, &info ) )
    {
        // set up a column description
        col_info[i].column_name = info.name;
        col_info[i].column_type = info.native_type;
        switch( info.native_type )
        {
            case DT_DATE:           // DATE is converted to string by C API
            case DT_TIME:           // TIME is converted to string by C API
            case DT_TIMESTAMP:      // TIMESTAMP is converted to string by C API
            case DT_DECIMAL:        // DECIMAL is converted to string by C API
                col_info[i].column_type = DT_FIXCHAR;
                break;
            case DT_FLOAT:          // FLOAT is converted to double by C API
                col_info[i].column_type = DT_DOUBLE;
                break;
            case DT_BIT:            // BIT is converted to tinyint by C API
                col_info[i].column_type = DT_TINYINT;
                break;
        }
    }
}
```

```
        col_info[i].column_width = info.max_size;
        col_info[i].column_index = i + 1; // column indices are origin 1
        col_info[i].column_can_be_null = info.nullable;
    }
}
// send the result set description
if( extapi->set_value( arg_handle,
                      EXTFN_RESULT_SET_ARG_NUM,
                      (an_extfn_value *)&rs_info,
                      EXTFN_RESULT_SET_DESCRIBE ) == 0 )
{
    // failed
    free( col_info );
    free( col_data );
    return;
}
```

Sobald die Ergebnismenge beschrieben wurde, können die Ergebnismengenzeilen zurückgegeben werden. Das folgende Codefragment zeigt, wie die Zeilen der Ergebnismenge zurückgegeben werden. Es verwendet die SQL Anywhere C-API, um die Zeilen für eine SQL-Abfrage abzurufen, die vorher durch die C-API ausgeführt wurde. Die von der SQL Anywhere-C-API zurückgegebenen Zeilen werden einzeln an die aufrufende Umgebung zurückgesendet. Das Array von Spaltendatenwert-Strukturen muss aufgefüllt werden, bevor die einzelnen Zeilen zurückgegeben werden. Die Spaltendatenwert-Struktur besteht aus einem Spaltenindex, einem Zeiger auf einen Datenwert, einer Datenlänge und einem Append-Parameter (Einfügeindikator).

```
a_sqlany_data_value *value = (a_sqlany_data_value *)
    malloc( columns * sizeof(a_sqlany_data_value) );

while( capi.sqlany_fetch_next( sqlany_stmt ) )
{
    for( int i = 0; i < columns; i++ )
    {
        if( capi.sqlany_get_column( sqlany_stmt, i, &value[i] ) )
        {
            col_data[i].column_index = i + 1;
            col_data[i].column_data = value[i].buffer;
            col_data[i].data_length = (a_sql_uint32)*(value[i].length);
            col_data[i].append = 0;
            if( *(value[i].is_null) )
            {
                // Received a NULL value
                col_data[i].column_data = NULL;
            }
        }
    }
    if( extapi->set_value( arg_handle,
                          EXTFN_RESULT_SET_ARG_NUM,
                          (an_extfn_value *)&rs_info,
                          EXTFN_RESULT_SET_NEW_ROW_FLUSH ) == 0 )
    {
        // failed
        free( value );
        free( col_data );
        free( col_data );
        extapi->set_value( arg_handle, 0, &retval, 0 );
        return;
    }
}
```

Weitere Hinweise zur Durchführung von serverseitigen Anforderungen und zur Rückgabe von Ergebnismengen durch eine externe Funktionen finden Sie in den Beispielen unter `%SQLANYSP16%\SQLAnywhere\ExternalEnvironments\ExternC`.

Siehe auch

- „SQL Anywhere-Schnittstelle für externe Aufrufe“ auf Seite 601

Die externe Java-Umgebung

SQL Anywhere stellt Unterstützung für gespeicherte Java-Prozeduren und -Funktionen bereit. Eine gespeicherte Java-Prozedur oder Funktion verhält sich wie eine gespeicherte SQL-Prozedur oder Funktion, abgesehen davon, dass der Code für die Prozedur oder Funktion in Java geschrieben ist und die Ausführung der Prozedur oder Funktion außerhalb des Datenbankservers stattfindet (d.h. in einer Java VM-Umgebung). Es gibt eine Instanz der Java VM für jede Datenbank statt einer Instanz für jede Verbindung. Gespeicherte Java-Prozeduren können Ergebnismengen zurückgeben.

Es gibt einige Voraussetzungen für die Verwendung der Unterstützung für Java in der Datenbank:

1. Eine Kopie der Java-Laufzeitumgebung muss auf dem Datenbankserversystem installiert sein.
2. Der SQL Anywhere-Datenbankserver muss in der Lage sein, die Position des Java-Programms (der Java VM) zu ermitteln.

Um Java in der Datenbank zu verwenden, achten Sie darauf, dass der Datenbankserver in der Lage ist, das Java-Programm aufzufinden und zu starten. Sie überprüfen dies, indem Sie Folgendes ausführen:

```
START EXTERNAL ENVIRONMENT JAVA;
```

Wenn der Datenbankserver Java nicht starten kann, liegt die Ursache des Problems wahrscheinlich darin, dass der Datenbankserver das Java-Programm nicht finden kann. In diesem Fall sollten Sie die Anweisung `ALTER EXTERNAL ENVIRONMENT` ausführen, um den Speicherort des Java-Programms explizit einzustellen. Achten Sie darauf, den Programmdateinamen einzubeziehen.

```
ALTER EXTERNAL ENVIRONMENT JAVA
LOCATION 'java-path';
```

Beispiel:

```
ALTER EXTERNAL ENVIRONMENT JAVA
LOCATION 'c:\\jdk1.7.0\\jre\\bin\\java.exe';
```

Sie können den Speicherort der Java VM abfragen, die der Datenbankserver verwenden wird, indem Sie die folgende SQL-Abfrage ausführen:

```
SELECT db_property('JAVAVM');
```

Die `START EXTERNAL ENVIRONMENT JAVA`-Anweisung wird nur benötigt, um zu überprüfen, ob der Datenbankserver die Java VM starten kann. Im Allgemeinen führt das Ausführen eines gespeicherten Java-Prozedur- oder -Funktionsaufrufs dazu, dass die Java VM automatisch gestartet wird.

Gleichmaßen ist die Anweisung `STOP EXTERNAL ENVIRONMENT JAVA` nicht erforderlich, um eine Instanz von Java zu stoppen, da die Instanz automatisch entfernt wird, wenn alle Verbindungen zur

Datenbank beendet wurden. Wenn Sie allerdings die Arbeit mit Java vollständig abgeschlossen haben und einige Ressourcen freigeben wollen, vermindert die Anweisung `STOP EXTERNAL ENVIRONMENT JAVA` die Verwendungsanzahl für die Java VM.

Nachdem Sie überprüft haben, dass der Datenbankserver das Java VM-Programm starten kann, besteht der nächste Schritt darin, den erforderlichen Java-Klassencode in der Datenbank zu installieren. Sie erreichen dies, indem Sie die Anweisung `INSTALL JAVA` verwenden. Sie können beispielsweise die folgende Anweisung ausführen, um eine Java-Klasse von einer Datei in der Datenbank zu installieren.

```
INSTALL JAVA
NEW
FROM FILE 'java-class-file';
```

Sie können auch eine Java JAR-Datei in der Datenbank installieren.

```
INSTALL JAVA
NEW
JAR 'jar-name'
FROM FILE 'jar-file';
```

Java-Klassen können wie folgt von einer Variablen installiert werden:

```
CREATE VARIABLE JavaClass LONG VARCHAR;
SET JavaClass = xp_read_file('java-class-file')
INSTALL JAVA
NEW
FROM JavaClass;
```

Java JAR-Dateien können wie folgt von einer Variablen installiert werden:

```
CREATE VARIABLE JavaJar LONG VARCHAR;
SET JavaJar = xp_read_file('jar-file')
INSTALL JAVA
NEW
JAR 'jar-name'
FROM JavaJar;
```

Um eine Java-Klasse aus der Datenbank zu entfernen, verwenden Sie die Anweisung `REMOVE JAVA`:

```
REMOVE JAVA CLASS java-class
```

Um eine Java JAR-Datei aus der Datenbank zu entfernen, verwenden Sie die Anweisung `REMOVE JAVA`:

```
REMOVE JAVA JAR 'jar-name'
```

Um bestehende Java-Klassen zu ändern, können Sie die `UPDATE`-Klausel der Anweisung `INSTALL JAVA` verwenden:

```
INSTALL JAVA
UPDATE
FROM FILE 'java-class-file'
```

Sie können auch bestehende Java JAR-Dateien in der Datenbank aktualisieren.

```
INSTALL JAVA
UPDATE
```

```
JAR 'jar-name'
FROM FILE 'jar-file';
```

Java-Klassen können wie folgt von einer Variablen aktualisiert werden:

```
CREATE VARIABLE JavaClass LONG VARCHAR;
SET JavaClass = xp_read_file('java-class-file')
INSTALL JAVA
UPDATE
FROM JavaClass;
```

Java JAR-Dateien können wie folgt von einer Variablen aktualisiert werden:

```
CREATE VARIABLE JavaJar LONG VARCHAR;
SET JavaJar = xp_read_file('jar-file')
INSTALL JAVA
UPDATE
FROM JavaJar;
```

Nachdem die Java-Klasse in der Datenbank installiert ist, können Sie gespeicherte Prozeduren und Funktionen erstellen, um eine Schnittstelle zu den Java-Methoden herzustellen. Die Zeichenfolge der EXTERNAL NAME-Klausel enthält die erforderlichen Informationen, um die Java-Methode aufzurufen und um OUT-Parameter und Rückgabewerte zurückzugeben. Das LANGUAGE-Attribut der EXTERNAL NAME-Klausel muss JAVA angeben. Das Format der EXTERNAL NAME-Klausel ist folgendes:

EXTERNAL NAME '*java-call*' LANGUAGE JAVA

java-call :
[*package-name*.]*class-name.method-name method-signature*

method-signature :
([*field-descriptor*, ...]) *return-descriptor*

field-descriptor und *return-descriptor* :

```
Z
| B
| S
| I
| J
| F
| D
| C
| V
| descriptor
| class-name;
```

Eine Java-Methodensignatur ist die kompakte Zeichendarstellung der Parametertypen und des Rückgabewerttyps. Wenn die Anzahl der Parameter geringer ist als die in der Methodensignatur angegebene Zahl, muss die Differenz der in DYNAMIC RESULT SETS angegebenen Zahl entsprechen. Jeder Parameter in der Methodensignatur, der nicht in der Parameterliste für die Prozedur enthalten ist, muss die Methodensignatur [Ljava/sql/ResultSet; haben.

Der Felddeskriptor (*field-descriptor*) und der Rückgabedeskriptor (*return-descriptor*) haben die folgenden Bedeutungen:

Feldtyp	Java-Datentyp
B	byte
C	char
D	double
F	float
I	int
J	long
L <i>class-name</i> ;	Eine Instanz der Klasse <i>class-name</i> . Die Klasse muss voll qualifiziert sein und alle Punkte im Namen müssen durch das Zeichen / ersetzt werden. Beispiel: java/lang/String
S	short
V	void
Z	Boolescher Wert
[Verwenden Sie jeweils eine für jede Felddimension.

Beispiel:

```
double some_method(  
    boolean a,  
    int b,  
    java.math.BigDecimal c,  
    byte [][] d,  
    java.sql.ResultSet[] rs ) {  
}
```

Dieses Beispiel hätte folgende Signatur:

```
'(ZILjava/math/BigDecimal;[[B[Ljava/sql/ResultSet;)D'
```

Die folgende Prozedur erstellt eine Schnittstelle zu einer Java-Methode. Die Java-Methode gibt keinen Wert zurück (V).

```
CREATE PROCEDURE insertfix()  
EXTERNAL NAME 'JDBCExample.InsertFixed()V'  
LANGUAGE JAVA;
```

Die folgende Prozedur erstellt eine Schnittstelle zu einer Java-Methode, die ein String-Eingabeargument ([Ljava/lang/String;) hat. Die Java-Methode gibt keinen Wert zurück (V).

```
CREATE PROCEDURE InvoiceMain( IN arg1 CHAR(50) )  
EXTERNAL NAME 'Invoice.main([Ljava/lang/String;)V'  
LANGUAGE JAVA;
```

Die folgende Prozedur erstellt eine Schnittstelle zu einer Java-Methode `Invoice.init`, die ein String-Argument (`Ljava/lang/String;`) übernimmt, sowie ein Double (D), ein weiteres String-Argument (`Ljava/lang/String;`) und ein weiteres Double (D), und die keinen Wert zurückgibt (V).

```
CREATE PROCEDURE init( IN arg1 CHAR(50),
                      IN arg2 DOUBLE,
                      IN arg3 CHAR(50),
                      IN arg4 DOUBLE)
EXTERNAL NAME 'Invoice.init(Ljava/lang/String;DLjava/lang/String;D)V'
LANGUAGE JAVA
```

Das folgende Java-Beispiel enthält die Funktion `main`, die eine Zeichenfolge übernimmt und sie in das Meldungsfenster des Datenbankservers schreibt. Außerdem enthält es die Funktion `whoAreYou`, die eine Java-Zeichenfolge zurückgibt.

```
import java.io.*;

public class Hello
{
    public static void main( String[] args )
    {
        System.out.print( "Hello" );
        for ( int i = 0; i < args.length; i++ )
            System.out.print( " " + args[i] );
        System.out.println();
    }
    public static String whoAreYou()
    {
        return( "I am SQL Anywhere." );
    }
}
```

Der obenstehende Java-Code wird in die Datei `Hello.java` abgelegt und mit dem Java-Compiler kompiliert. Die resultierende Klassendatei wird folgendermaßen in die Datenbank geladen:

```
INSTALL JAVA
NEW
FROM FILE 'Hello.class';
```

Unter Verwendung von Interactive SQL wird die gespeicherte Prozedur, die eine Schnittstelle zur Methode `main` in der Klasse `Hello` herstellt, folgendermaßen erstellt:

```
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

Das Argument zu `main` wird als Array von `java.lang.String` beschrieben. Unter Verwendung von Interactive SQL testen Sie die Schnittstelle, indem Sie die folgende SQL-Anweisung ausführen.

```
CALL HelloDemo('SQL Anywhere');
```

Wenn Sie das Meldungsfenster des Datenbankservers überprüfen, werden Sie dort die Meldung angezeigt sehen. Die gesamte Ausgabe an `System.out` wird zum Fenster "Datenbankservermeldungen" umgeleitet.

Unter Verwendung von Interactive SQL wird die Funktion, die eine Schnittstelle zur Methode `whoAreYou` in der Klasse `Hello` herstellt, folgendermaßen erstellt:

```
CREATE FUNCTION WhoAreYou()
RETURNS LONG VARCHAR
```

```
EXTERNAL NAME 'Hello.whoAreYou(V)Ljava/lang/String;'
LANGUAGE JAVA;
```

Die Funktion `whoAreYou` gibt gemäß ihrer Beschreibung einen `java.lang.String`-Wert zurück. Unter Verwendung von Interactive SQL testen Sie die Schnittstelle, indem Sie die folgende SQL-Anweisung ausführen.

```
SELECT WhoAreYou();
```

Sie sollten die Antwort im Interactive SQL-Fenster **Ergebnisse** sehen.

Bei dem Versuch einer Fehlerbehandlung für den Fall, dass eine externe Java-Umgebung nicht gestartet wurde (d.h. die Anwendung bei einem Java-Aufruf einen Fehler erhält, wonach der Haupt-Thread nicht gefunden wurde), sollte der DBA Folgendes überprüfen:

- Wenn die Java VM einen anderen Bitwert aufweist als der Datenbankserver, sorgen Sie dafür, dass auf dem Datenbankserver-Computer die Clientbibliotheken mit demselben Bitwert wie die VM installiert werden.
- Achten Sie darauf, dass die Shared Objects *sajdbc4.jar* und *dbjdbc16/libdbjdbc16* aus demselben Software-Build stammen.
- Wenn sich auf dem Datenbankserver-Computer mehrere Kopien von *sajdbc4.jar* befinden, sorgen Sie dafür, dass alle mit derselben Softwareversion synchronisiert werden.
- Wenn der Datenbankserver-Computer stark ausgelastet ist, besteht die Möglichkeit, dass der Fehler aufgrund eines Timeouts gemeldet wird.

Siehe auch

- [„Lektion 4: Methoden in Java-Klassen aufrufen“ auf Seite 408](#)
- [„Ergebnismengen aus Java-Methoden zurückgeben“ auf Seite 415](#)
- [„Java in der Datenbank“ auf Seite 401](#)

Die externe PERL-Umgebung

SQL Anywhere stellt Unterstützung für gespeicherte Perl-Prozeduren und -Funktionen bereit. Eine gespeicherte Perl-Prozedur oder Funktion verhält sich wie eine gespeicherte SQL-Prozedur oder Funktion, abgesehen davon, dass der Code für die Prozedur oder Funktion in Perl geschrieben ist und die Ausführung der Prozedur oder Funktion außerhalb des Datenbankservers stattfindet (d.h. innerhalb einer Perl-Programminstanz). Es gibt eine separate Instanz der Perl-Programmdatei für jede Verbindung, die gespeicherte Perl-Prozeduren und Perl-Funktionen verwendet. Dieses Verhalten unterscheidet sich von gespeicherten Java- Prozeduren und Funktionen. Bei Java gibt es eine Instanz der Java VM für jede Datenbank statt einer Instanz für jede Verbindung. Der andere Hauptunterschied zwischen Perl und Java besteht darin, dass gespeicherte Perl-Prozeduren keine Ergebnismengen zurückgeben, während gespeicherte Java-Prozeduren Ergebnismengen zurückgeben können.

Es gibt einige Voraussetzungen für die Verwendung der Unterstützung für Perl in der Datenbank:

1. Perl muss auf dem Datenbankserver-Computer installiert sein und der SQL Anywhere-Datenbankserver muss in der Lage sein, die Position des Perl-Programms zu ermitteln.

2. Der DBD::SQLAnywhere-Treiber muss auf dem Datenbankserver-Computer installiert sein.
3. Unter Windows muss Microsoft Visual Studio installiert sein. Dies ist eine Voraussetzung, um den DBD::SQLAnywhere-Treiber installieren zu können.

Weitere Hinweise zur Installation des DBD::SQLAnywhere-Treibers finden Sie unter [„Perl/DBI-Unterstützung“ auf Seite 651](#).

Zusätzlich zu den oben genannten Voraussetzungen muss der Datenbankadministrator auch das SQL Anywhere Perl External Environment-Modul installieren.

Unter Windows führen Sie die folgenden Befehle aus dem *SDK\PerlEnv*-Unterverzeichnis Ihrer SQL Anywhere-Installation aus, um das externe Umgebungsmodul zu installieren:

```
perl Makefile.PL
nmake
nmake install
```

Unter Unix führen Sie die folgenden Befehle aus dem *sdk/perlenv*-Unterverzeichnis Ihrer SQL Anywhere-Installation aus, um das externe Umgebungsmodul zu installieren:

```
perl Makefile.PL
make
make install
```

Sobald das Modul für die externe Perl-Umgebung kompiliert und installiert ist, kann die Unterstützung für Perl in der Datenbank verwendet werden. Die Unterstützung für Perl in der Datenbank ist nur für SQL Anywhere-Datenbanken ab Version 11 verfügbar. Wenn eine SQL Anywhere 10-Datenbank geladen ist und Sie versuchen, die Unterstützung für Perl in der Datenbank zu verwenden, wird ein Fehler zurückgegeben, der angibt, dass externe Umgebungen nicht unterstützt werden.

Um Perl in der Datenbank zu verwenden, achten Sie darauf, dass der Datenbankserver in der Lage ist, die Position des Perl-Programms zu ermitteln und es zu starten. Sie überprüfen dies, indem Sie Folgendes ausführen:

```
START EXTERNAL ENVIRONMENT PERL;
```

Wenn der Datenbankserver Perl nicht starten kann, liegt das Problem wahrscheinlich daran, dass der Datenbankserver nicht in der Lage ist, das Perl-Programm zu finden. In diesem Fall sollten Sie eine Anweisung ALTER EXTERNAL ENVIRONMENT ausführen, um den Speicherort des Perl-Programms explizit einzustellen. Achten Sie darauf, den Programmdateinamen einzubeziehen.

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'perl-path';
```

Beispiel:

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'c:\\Perl\\bin\\perl.exe';
```

Die START EXTERNAL ENVIRONMENT PERL-Anweisung wird nur benötigt, um zu überprüfen, ob der Datenbankserver Perl starten kann. Üblicherweise startet Perl automatisch, wenn Sie einen Aufruf einer gespeicherten Perl-Prozedur oder Funktion durchführen.

Die Anweisung `STOP EXTERNAL ENVIRONMENT PERL` ist auch nicht erforderlich, um eine Instanz von Perl zu stoppen, weil die Instanz mit dem Ende der Verbindung automatisch beendet wird. Wenn Sie allerdings die Arbeit mit Perl abgeschlossen haben und Ressourcen freigeben wollen, entfernt die Anweisung `STOP EXTERNAL ENVIRONMENT PERL` die Perl-Instanz für Ihre Verbindung.

Wenn Sie überprüft haben, dass der Datenbankserver das Perl-Programm starten kann, besteht der nächste Schritt darin, den erforderlichen Perl-Code in der Datenbank zu installieren. Sie erreichen dies, indem Sie die Anweisung `INSTALL` verwenden. Sie können beispielsweise die folgende Anweisung ausführen, um ein Perl-Skript von einer Datei in der Datenbank zu installieren.

```
INSTALL EXTERNAL OBJECT 'perl-script'  
NEW  
FROM FILE 'perl-file'  
ENVIRONMENT PERL;
```

Perl-Code kann wie folgt aus einem Ausdruck extrahiert und installiert werden:

```
INSTALL EXTERNAL OBJECT 'perl-script'  
NEW  
FROM VALUE 'perl-statements'  
ENVIRONMENT PERL;
```

Perl-Code kann wie folgt aus einer Variablen extrahiert und installiert werden:

```
CREATE VARIABLE PerlVariable LONG VARCHAR;  
SET PerlVariable = 'perl-statements';  
INSTALL EXTERNAL OBJECT 'perl-script'  
NEW  
FROM VALUE PerlVariable  
ENVIRONMENT PERL;
```

Um Perl aus der Datenbank zu entfernen, verwenden Sie die Anweisung `REMOVE`:

```
REMOVE EXTERNAL OBJECT 'perl-script'
```

Um bestehenden Perl-Code zu ändern, können Sie die `UPDATE`-Klausel der Anweisung `INSTALL EXTERNAL OBJECT` verwenden:

```
INSTALL EXTERNAL OBJECT 'perl-script'  
UPDATE  
FROM FILE 'perl-file'  
ENVIRONMENT PERL  
  
INSTALL EXTERNAL OBJECT 'perl-script'  
UPDATE  
FROM VALUE 'perl-statements'  
ENVIRONMENT PERL  
  
SET PerlVariable = 'perl-statements';  
INSTALL EXTERNAL OBJECT 'perl-script'  
UPDATE  
FROM VALUE PerlVariable  
ENVIRONMENT PERL
```

Wenn der Perl-Code in der Datenbank installiert ist, können Sie die erforderlichen gespeicherten Perl-Prozeduren und -Funktionen erstellen. Wenn Sie gespeicherte Perl-Prozeduren und Funktionen erstellen, ist `LANGUAGE` immer `PERL` und die Zeichenfolge `EXTERNAL NAME` enthält die Informationen, die

erforderlich sind, um die Perl-Subroutinen aufzurufen und OUT-Parameter sowie Rückgabewerte zurückzugeben. Die folgenden globalen Variablen stehen für den Perl-Code bei jedem Aufruf zur Verfügung:

- **\$sa_perl_return** Wird verwendet, um einen Rückgabewert für einen Funktionsaufruf festzulegen.
- **\$sa_perl_argN** Dabei gilt: N ist eine positive Ganzzahl [0 .. n]. Wird verwendet, um die SQL Argumente an den Perl-Code zu übergeben. Beispiel: \$sa_perl_arg0 bezieht sich auf Argument 0, \$sa_perl_arg1 bezieht sich auf Argument 1, usw.
- **\$sa_perl_default_connection** Wird verwendet, um serverseitige Perl-Aufrufe durchzuführen.
- **\$sa_output_handle** Wird verwendet, um Ausgaben des Perl-Codes an das Meldungsfenster des Datenbankservers zu senden.

Eine gespeicherte Perl-Prozedur kann mit allen Datentypen für Eingabe- und Ausgabeargumente sowie für den Rückgabewert erstellt werden. Allerdings werden bei der Durchführung des Perl-Aufrufs alle nicht-binären Datentypen Zeichenfolgen zugeordnet, während Binärdaten einem Array von Zahlen zugeordnet werden. Es folgt ein einfaches Perl-Beispiel:

```
INSTALL EXTERNAL OBJECT 'SimplePerlExample'
NEW
FROM VALUE 'sub SimplePerlSub{
    return( ($_[0] * 1000) +
            ($_[1] * 100) +
            ($_[2] * 10) +
            $_[3] );
}'
ENVIRONMENT PERL;

CREATE FUNCTION SimplePerlDemo(
    IN thousands INT,
    IN hundreds INT,
    IN tens INT,
    IN ones INT)
RETURNS INT
EXTERNAL NAME '<file=SimplePerlExample>'
    $sa_perl_return = SimplePerlSub(
        $sa_perl_arg0,
        $sa_perl_arg1,
        $sa_perl_arg2,
        $sa_perl_arg3)'
LANGUAGE PERL;

// The number 1234 should appear
SELECT SimplePerlDemo(1,2,3,4);
```

Das folgende Perl-Beispiel nimmt eine Zeichenfolge und schreibt sie in das Meldungsfenster des Datenbankservers:

```
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE 'sub WriteToServerConsole { print $sa_output_handle $_[0]; }'
ENVIRONMENT PERL;

CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
EXTERNAL NAME '<file=PerlConsoleExample>'
    WriteToServerConsole( $sa_perl_arg0 )'
```

```
LANGUAGE PERL;

// 'Hello world' should appear in the database server messages window
CALL PerlWriteToConsole( 'Hello world' );
```

Um serverseitiges Perl zu verwenden, muss der Perl-Code die Variable `$sa_perl_default_connection` verwenden. Das folgende Beispiel erstellt eine Tabelle und ruft dann eine gespeicherte Perl-Prozedur auf, um die Tabelle mit Daten zu füllen:

```
CREATE TABLE perlTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePerlExample'
NEW
FROM VALUE 'sub ServerSidePerlSub
{ $sa_perl_default_connection->do(
  "INSERT INTO perlTab SELECT table_id, table_name FROM SYS.SYSTAB" );
  $sa_perl_default_connection->do(
    "COMMIT" );
}'
ENVIRONMENT PERL;

CREATE PROCEDURE PerlPopulateTable()
EXTERNAL NAME '<file=ServerSidePerlExample> ServerSidePerlSub()'
LANGUAGE PERL;

CALL PerlPopulateTable();

// The following should return 2 identical rows
SELECT count(*) FROM perlTab
UNION ALL
SELECT count(*) FROM SYS.SYSTAB;
```

Weitere Hinweise und Beispiele zur Unterstützung für Perl in der Datenbank finden Sie unter den Beispielen, die sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\ExternalEnvironments\Perl` befinden.

Die externe PHP-Umgebung

SQL Anywhere stellt Unterstützung für gespeicherte PHP-Prozeduren und -Funktionen bereit. Eine gespeicherte PHP-Prozedur oder -Funktion verhält sich wie eine gespeicherte SQL-Prozedur oder Funktion, abgesehen davon, dass der Code für die Prozedur oder Funktion in PHP geschrieben ist und die Ausführung der Prozedur oder Funktion außerhalb des Datenbankservers stattfindet (d.h. innerhalb einer PHP-Programminstanz). Es gibt eine separate Instanz des PHP-Programms für jede Verbindung, die gespeicherte PHP-Prozeduren und -Funktionen verwendet. Dieses Verhalten unterscheidet sich deutlich von gespeicherten Java-Prozeduren und Funktionen. Bei Java gibt es eine Instanz der Java VM für jede Datenbank statt einer Instanz für jede Verbindung. Der andere Hauptunterschied zwischen PHP und Java besteht darin, dass gespeicherte PHP-Prozeduren keine Ergebnismengen zurückgeben, während gespeicherte Java-Prozeduren Ergebnismengen zurückgeben können. PHP gibt nur ein Objekt vom Typ `LONG VARCHAR` zurück, was die Ausgabe des PHP-Skripts darstellt.

Um PHP in der Datenbank verwenden zu können, müssen Sie den SQL Anywhere-PHP-Treiber und das SQL Anywhere PHP External Environment-Modul installieren und PHP so konfigurieren, dass PHP diese Komponenten finden und laden kann. Im folgenden Abschnitt wird beschrieben, wie Sie dazu vorgehen.

PHP External Environment-Modul unter Windows oder Unix installieren

Installieren Sie das externe Umgebungsmodul zur Unterstützung für PHP in der Datenbank.

Voraussetzungen

1. Eine Kopie von PHP muss auf dem Datenbankserver-Computer installiert sein und der SQL Anywhere-Datenbankserver muss in der Lage sein, die PHP-Programmdatei zu finden.
2. Der SQL Anywhere-PHP-Treiber (mit SQL Anywhere mitgeliefert) muss auf dem Datenbankserver-Computer installiert sein und PHP muss für seine Verwendung konfiguriert werden. Siehe „[PHP Client-Deployment](#)“ auf Seite 984.

Kontext und Bemerkungen

Zusätzlich zu den zwei obenstehenden Voraussetzungen muss der Datenbankadministrator auch das Modul "SQL Anywhere PHP External Environment" installieren. Im voraus kompilierte Module für mehrere Versionen von PHP sind im SQL Anywhere-Lieferumfang enthalten. Wenn Sie vorab aufgebaute Module installieren möchten, kopieren Sie das entsprechende Treibermodul in Ihr Verzeichnis für PHP-Erweiterungen (das in der Datei `php.ini` angegeben ist). Unter Unix können Sie auch eine Symbolverknüpfung verwenden. In den folgenden Schritten wird beschrieben, wie Sie dazu vorgehen.

Aufgabe

1. Suchen Sie die `php.ini`-Datei Ihrer PHP-Installation und öffnen Sie sie in einem Texteditor. Suchen Sie die Zeile, in der der Standort des **extension_dir**-Verzeichnisses angegeben wird. Wenn **extension_dir** nicht auf ein bestimmtes Verzeichnis gesetzt ist, sollten Sie es aus Gründen der Systemsicherheit so einstellen, dass es auf ein isoliertes Verzeichnis zeigt.
2. Kopieren Sie das gewünschte externe PHP-Umgebungsmodul aus dem SQL Anywhere-Installationsverzeichnis in das Verzeichnis für Ihre PHP-Installation oder Ihre PHP-Erweiterungen. Ändern Sie "x.y" so, dass der Wert der von Ihnen ausgewählten Version entspricht.

Verwenden Sie unter Windows einen Befehl wie den folgenden:

```
copy "%SQLANY16%\Bin32\php-5.x.y_sqlanywhere_extenv16.dll" php-ext-dir
```

Verwenden Sie unter Unix einen Befehl wie den folgenden:

```
cp $SQLANY16/bin32/php-5.x.y_sqlanywhere_extenv16.so php-ext-dir
```

In den oben genannten Beispielen ist *Verz_PHP-Erw* der Pfad zu dem Verzeichnis für Ihre PHP-Erweiterungen (z.B. `c:\php\ext`).

3. Sie können eine Zeile zum Abschnitt "Dynamic Extensions" der Datei `php.ini` hinzufügen, um das PHP-Modul der externen Umgebung automatisch zu laden. Ändern Sie "x.y" so, dass der Wert der von Ihnen ausgewählten Version entspricht.

Fügen Sie unter Windows die folgende Zeile hinzu:

```
extension=php-5.x.y_sqlanywhere_extenv16.dll
```

Fügen Sie unter Unix die folgende Zeile hinzu:

```
extension=php-5.x.y_sqlanywhere_extenv16.so
```

4. Speichern und schließen Sie die Datei *php.ini*.
5. Vergewissern Sie sich, dass Sie auch den SQL Anywhere PHP-Treiber vom SQL Anywhere-Installationsverzeichnis in Ihrem PHP-Erweiterungsverzeichnis installiert haben. Dieses Dateinamenpräfix folgt dem Muster **php-5.x.y_sqlanywhere**, wobei x und y die Versionsnummern sind. Diese sollten mit den Versionsnummern übereinstimmen, die Sie in Schritt 2 kopiert haben.

Ergebnisse

Das externe Umgebungsmodul wird installiert.

PHP aus der Datenbank verwenden

Die Unterstützung für PHP in der Datenbank steht nur für SQL Anywhere-Datenbanken der Version 11 oder später zur Verfügung. Wenn eine SQL Anywhere 10-Datenbank geladen ist und Sie versuchen, die Unterstützung für PHP in der Datenbank zu verwenden, wird ein Fehler zurückgegeben, der angibt, dass externe Umgebungen nicht unterstützt werden.

Um PHP in der Datenbank zu verwenden, muss der Datenbankserver in der Lage sein, die Position des PHP-Programms zu ermitteln und es zu starten. Sie können überprüfen, ob der Datenbankserver in der Lage ist, die Position des PHP-Programms zu ermitteln und es zu starten, indem Sie die folgende Anweisung ausführen:

```
START EXTERNAL ENVIRONMENT PHP;
```

Wenn eine Meldung des Inhalts angezeigt wird, dass das 'externe Programm' nicht gefunden wurde, dann liegt das Problem darin, dass der Datenbankserver nicht in der Lage ist, die Position des PHP-Programms zu ermitteln. In diesem Fall sollten Sie die ALTER EXTERNAL ENVIRONMENT-Anweisung ausführen, um explizit den Speicherort des PHP-Programms einschließlich des Programmnamens festzulegen, oder sicherstellen, dass die PATH-Umgebungsvariable das Verzeichnis umfasst, in dem sich das PHP-Programm befindet.

```
ALTER EXTERNAL ENVIRONMENT PHP  
LOCATION 'php-path';
```

Beispiel:

```
ALTER EXTERNAL ENVIRONMENT PHP  
LOCATION 'c:\\php\\php-5.4.8-win32\\php.exe';
```

Um die Standardeinstellung wiederherzustellen, führen Sie die folgende Anweisung aus:

```
ALTER EXTERNAL ENVIRONMENT PHP  
LOCATION 'php';
```

Wenn Sie eine Meldung des Inhalts sehen, dass der 'Haupt-Thread' nicht gefunden werden kann, überprüfen Sie Folgendes:

- Vergewissern Sie sich, dass die Module *php-5.x.y_sqlanywhere* und *php-5.x.y_sqlanywhere_extenv16* sich in dem Verzeichnis befinden, das **extension_dir** in der Datei *php.ini* angibt. Überprüfen Sie die im vorherigen Abschnitt beschriebenen Installationsschritte.
- Vergewissern Sie sich, dass die Module *php-5.x.y_sqlanywhere* und *php-5.x.y_sqlanywhere_extenv16* beide im Abschnitt "Dynamic Extensions" der Datei *php.ini* aufgeführt sind. Überprüfen Sie die im vorherigen Abschnitt beschriebenen Installationsschritte.
- Vergewissern Sie sich, dass die Position von *phpenv.php* ermittelt werden kann. Vergewissern Sie sich, dass sich der SQL Anywhere-Ordner *bin32* in Ihrem PATH befindet.
- Unter Windows muss gewährleistet sein, dass die 32-Bit-DLLs (*dbcapi.dll*, *dblib16.dll*, *dbicu16.dll*, *dbicudt16.dll*, *dbngen16.dll* und *dbextenv16.dll*) gefunden werden können. Vergewissern Sie sich, dass sich der SQL Anywhere-Ordner *bin32* in Ihrem PATH befindet.
- Unter Linux, Unix und Mac OS X muss gewährleistet sein, dass die 32-Bit-Shared Objects (*libdbcapi_r*, *libdblib16_r*, *libdbicu16_r*, *libdbicudt16*, *dbngen16.res* und *libdbextenv16_r*) gefunden werden können. Vergewissern Sie sich, dass sich der SQL Anywhere-Ordner *bin32* in Ihrem PATH befindet.
- Vergewissern Sie sich, dass die Umgebungsvariable **PHPRC** nicht gesetzt ist, oder vergewissern Sie sich, dass sie auf die Version von PHP zeigt, die Sie verwenden wollen.

Beachten Sie, dass die Anweisung **START EXTERNAL ENVIRONMENT PHP** nur benötigt wird, um zu überprüfen, ob der Datenbankserver in der Lage ist, PHP zu starten. Üblicherweise startet PHP automatisch, wenn Sie einen Aufruf einer gespeicherten PHP-Prozedur oder Funktion durchführen.

Die Anweisung **STOP EXTERNAL ENVIRONMENT PHP** ist auch nicht erforderlich, um eine Instanz von PHP zu stoppen, weil die Instanz mit dem Ende der Verbindung automatisch beendet wird. Wenn Sie allerdings die Arbeit mit PHP abgeschlossen haben und Ressourcen freigeben wollen, entfernt die Anweisung **STOP EXTERNAL ENVIRONMENT PHP** die PHP-Instanz für Ihre Verbindung.

Wenn Sie überprüft haben, dass der Datenbankserver das PHP-Programm starten kann, besteht der nächste Schritt darin, den erforderlichen PHP-Code in der Datenbank zu installieren. Sie erreichen dies, indem Sie die Anweisung **INSTALL** verwenden. Sie können beispielsweise die folgende Anweisung ausführen, um ein bestimmtes PHP-Skript in der Datenbank zu installieren.

```
INSTALL EXTERNAL OBJECT 'php-script'
NEW
FROM FILE 'php-file'
ENVIRONMENT PHP;
```

PHP-Code kann wie folgt aus einem Ausdruck extrahiert und installiert werden:

```
INSTALL EXTERNAL OBJECT 'php-script'
NEW
FROM VALUE 'php-statements'
ENVIRONMENT PHP;
```

PHP-Code kann wie folgt aus einer Variablen extrahiert und installiert werden:

```
CREATE VARIABLE PHPVariable LONG VARCHAR;
SET PHPVariable = 'php-statements';
INSTALL EXTERNAL OBJECT 'php-script'
NEW
FROM VALUE PHPVariable
ENVIRONMENT PHP;
```

Um PHP aus der Datenbank zu entfernen, verwenden Sie die Anweisung **REMOVE**:

```
REMOVE EXTERNAL OBJECT 'php-script';
```

Um bestehenden PHP-Code zu ändern, können Sie die UPDATE-Klausel der Anweisung INSTALL verwenden:

```
INSTALL EXTERNAL OBJECT 'php-script'
UPDATE
FROM FILE 'php-file'
ENVIRONMENT PHP;

INSTALL EXTERNAL OBJECT 'php-script'
UPDATE
FROM VALUE 'php-statements'
ENVIRONMENT PHP;

SET PHPVariable = 'php-statements';
INSTALL EXTERNAL OBJECT 'php-script'
UPDATE
FROM VALUE PHPVariable
ENVIRONMENT PHP;
```

Wenn der PHP-Code in der Datenbank installiert ist, können Sie anschließend die erforderlichen gespeicherten PHP-Prozeduren und -Funktionen erstellen. Wenn Sie gespeicherte PHP-Prozeduren und Funktionen erstellen, ist LANGUAGE immer PHP und die Zeichenfolge EXTERNAL NAME enthält die Informationen, die erforderlich sind, um die PHP-Subroutinen aufzurufen und OUT-Parameter zurückzugeben.

Die Argumente werden so an das PHP-Skript im \$argv-Array übergeben, wie PHP Argumente von der Befehlszeile annehmen würde (d.h. \$argv[1] ist das erste Argument). Um einen Ausgabeparameter festzulegen, ordnen Sie ihn dem entsprechenden \$argv-Element zu. Der Rückgabewert ist immer die Ausgabe des Skripts (als Typ LONG VARCHAR).

Eine gespeicherte PHP-Prozedur kann mit jedem Datentypsatz für Eingabe- oder Ausgabeargumente erstellt werden. Die Parameter werden allerdings für die Verwendung innerhalb des PHP-Skripts von bzw. in boolean-, integer-, double- oder string-Werte konvertiert. Der Rückgabewert ist immer ein Objekt vom Typ LONG VARCHAR. Es folgt ein einfaches PHP-Beispiel:

```
INSTALL EXTERNAL OBJECT 'SimplePHPExample'
NEW
FROM VALUE '<?php function SimplePHPFunction(
    $arg1, $arg2, $arg3, $arg4 )
{ return ($arg1 * 1000) +
    ($arg2 * 100) +
    ($arg3 * 10) +
    $arg4;
} ?>'
ENVIRONMENT PHP;

CREATE FUNCTION SimplePHPDemo(
    IN thousands INT,
    IN hundreds INT,
    IN tens INT,
    IN ones INT)
RETURNS LONG VARCHAR
EXTERNAL NAME '<file=SimplePHPExample> print SimplePHPFunction(
    $argv[1], $argv[2], $argv[3], $argv[4]);'
LANGUAGE PHP;
```

```
// The number 1234 should appear
SELECT SimplePHPDemo(1,2,3,4);
```

Bei PHP wird die Zeichenfolge EXTERNAL NAME in einer einzigen Zeile mit SQL-Code angegeben.

Um serverseitiges PHP zu verwenden, kann der PHP-Code die Standarddatenbankverbindung benutzen. Um ein Handle für die Datenbankverbindung zu erhalten, rufen Sie `sasql_pconnect` mit einem leeren Zeichenfolgenargument ("" oder "") auf. Das leere Zeichenfolgenargument teilt dem SQL Anywhere PHP-Treiber mit, die aktuelle externe Umgebungsverbindung zurückzugeben, anstatt eine neue zu öffnen. Das folgende Beispiel erstellt eine Tabelle und ruft dann eine gespeicherte PHP-Prozedur auf, um die Tabelle mit Daten zu füllen:

```
CREATE TABLE phpTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePHPExample'
NEW
FROM VALUE '<?php function ServerSidePHPSub() {
    $conn = sasql_pconnect( '' );
    sasql_query( $conn,
    "INSERT INTO phpTab
        SELECT table_id, table_name FROM SYS.SYSTAB" );
    sasql_commit( $conn );
} ?>'
ENVIRONMENT PHP;

CREATE PROCEDURE PHPPopulateTable()
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;

CALL PHPPopulateTable();

// The following should return 2 identical rows
SELECT count(*) FROM phpTab
UNION ALL
SELECT count(*) FROM SYS.SYSTAB;
```

Bei PHP wird die Zeichenfolge EXTERNAL NAME in einer einzigen Zeile mit SQL-Code angegeben. Im Beispiel oben werden die Apostrophe aus Gründen der syntaktischen Analyse in SQL verdoppelt. Befände sich der PHP-Quellcode in einer Datei, würden die Apostrophe nicht verdoppelt werden.

Um einen Fehler an den Datenbankserver zurückzugeben, lösen Sie eine PHP-Ausnahme aus. Das folgende Beispiel zeigt, wie Sie hierzu vorgehen.

```
CREATE TABLE phpTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePHPExample'
NEW
FROM VALUE '<?php function ServerSidePHPSub() {
    $conn = sasql_pconnect( '' );
    if( !sasql_query( $conn,
        "INSERT INTO phpTabNoExist
            SELECT table_id, table_name FROM SYS.SYSTAB" )
    ) throw new Exception(
        sasql_error( $conn ),
        sasql_errorcode( $conn )
    );
    sasql_commit( $conn );
} ?>'
ENVIRONMENT PHP;
```

```
CREATE PROCEDURE PHPPopulateTable()  
  EXTERNAL NAME  
    '<file=ServerSidePHPExample> ServerSidePHPSub()'  
  LANGUAGE PHP;  
  
CALL PHPPopulateTable();
```

Das obenstehende Beispiel sollte mit dem Fehler `SQLU_UNHANDLED_EXTENV_EXCEPTION` beendet werden, der angibt, dass die Tabelle `phpTabNoExist` nicht gefunden wurde.

Weitere Hinweise und Beispiele zur Unterstützung für PHP in der Datenbank finden Sie unter den Beispielen, die sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\ExternalEnvironments\PHP` befinden.

Perl/DBI-Unterstützung

DBD::SQLAnywhere ist der SQL Anywhere-Datenbanktreiber für DBI, eine Datenzugriffs-API für die Sprache Perl. Die DBI-API-Spezifikation definiert eine Gruppe von Funktionen, Variablen und Konventionen, die eine konsistente Datenbankschnittstelle unabhängig von der gerade benutzten Datenbank bereitstellen. Mit DBI und DBD::SQLAnywhere haben Ihre Perl-Skripten direkten Zugriff auf SQL Anywhere-Datenbankserver.

DBD::SQLAnywhere

DBD::SQLAnywhere ist ein Treiber für das DBI-Modul (Database Independent Interface for Perl), das von Tim Bunce geschrieben wurde. Nachdem Sie das DBI-Modul und DBD::SQLAnywhere installiert haben, können Sie aus Perl-Skripten auf SQL Anywhere-Datenbanken zugreifen und darin Daten ändern.

Der DBD::SQLAnywhere-Treiber ist threadsicher, wenn Sie Perl mit ithreads verwenden.

Anforderungen

Die DBD::SQLAnywhere-Schnittstelle benötigt die nachstehend angeführten Komponenten.

- Perl 5.6.0 oder später. Unter Windows ist mindestens ActivePerl 5.6.0 Build 616 erforderlich.
- DBI 1.34 oder später.
- Ein C Compiler. Unter Windows wird nur der Microsoft Visual C++-Compiler unterstützt.

DBD::SQLAnywhere unter Windows installieren

Installieren Sie die DBD::SQLAnywhere-Schnittstelle auf der unterstützten Windows-Plattform, um mit Perl auf SQL Anywhere-Datenbanken zugreifen zu können.

Voraussetzungen

- Installieren Sie ActivePerl ab Version 5.6.0. Sie können das ActivePerl-Installationsprogramm verwenden, um Perl zu installieren und Ihren Computer zu konfigurieren. Perl braucht nicht rekompiliert zu werden.
- Installieren Sie Microsoft Visual Studio und konfigurieren Sie die Umgebung.

Wenn Sie Ihre Umgebung nicht bei der Installation konfiguriert haben, müssen Sie die Umgebungsvariablen PATH, LIB und INCLUDE einrichten, bevor Sie fortfahren. Microsoft stellt für diesen Zweck eine Batchdatei bereit. Für 32-Bit-Builds wird die Batchdatei *vcvars32.bat* im Unterverzeichnis *vc\bin* der Visual Studio 2005- oder 2008-Installation bereitgestellt. Für 64-Bit-Builds müssen Sie nach einer 64-Bit-Version dieser Batchdatei suchen, z.B. *vcvarsamd64.bat*. Öffnen Sie eine Eingabeaufforderung und führen Sie die Batchdatei aus, bevor Sie fortfahren.

Weitere Hinweise zum Konfigurieren von einer 64-Bit-Visual C++-Umgebung finden Sie unter <http://msdn.microsoft.com/de-de/library/x4d2c09s.aspx>.

Aufgabe

1. An einer Eingabeaufforderung wechseln Sie ins Unterverzeichnis *bin* Ihres ActivePerl-Installationsverzeichnis.

Es wird dringend empfohlen, mit der systemeigenen Eingabeaufforderung zu arbeiten, da die nachstehend beschriebenen Schritte mit einer anderen Eingabeaufforderung möglicherweise nicht funktionieren.

2. Geben Sie im Perl Module Manager folgenden Befehl ein.

```
ppm query dbi
```

Wenn ppm nicht startet, prüfen Sie, ob Perl richtig installiert ist.

Dieser Befehl erzeugt zwei Zeilen Text nach dem nachstehend gezeigten Muster. In diesem Fall weist die Information darauf hin, dass ActivePerl Version 5.8.1 Build 807 läuft und DBI Version 1.38 installiert ist.

```
Querying target 1 (ActivePerl 5.8.1.807)
1. DBI [1.38] Database independent interface for Perl
```

Spätere Versionen von Perl zeigen möglicherweise stattdessen eine Tabelle ähnlich der folgenden an. In diesem Fall gibt die Information an, dass DBI Version 1.58 installiert ist.

Name	Version	Kurzbeschreibung	Bereich
DBI	1.58	Datenbankunabhängige Schnittstelle für Perl	Perl

Wenn DBI nicht installiert ist, müssen Sie die Installation vornehmen. Dafür geben Sie an der ppm-Eingabeaufforderung den nachstehenden Befehl ein.

```
ppm install dbi
```

3. Wechseln Sie an einer Eingabeaufforderung in das Unterverzeichnis *SDK\Perl* des SQL Anywhere-Installationsverzeichnis.
4. Geben Sie die nachstehenden Befehle ein, um DBD::SQLAnywhere zu erstellen und zu testen.

```
perl Makefile.PL
```

```
nmake
```

Wenn Sie aus einem bestimmten Grund neu beginnen müssen, können Sie den Befehl **nmake clean** ausführen, um teilweise bereits kompilierte Targets zu löschen.

5. Um DBD::SQLAnywhere zu testen, kopieren Sie die Beispieldatenbankdatei in das Verzeichnis *SDK\Perl* und führen Sie die Tests durch.

```
copy "%SQLANYSAMPL6%\demo.db" .
```

```
dbsrv16 demo
```

```
nmake test
```

Wenn die Tests nicht ausgeführt werden können, vergewissern Sie sich, dass das Unterverzeichnis *bin32* oder *bin64* der SQL Anywhere-Installation in der PATH-Umgebungsvariablen angegeben ist.

6. Um die Installation abzuschließen, führen Sie an derselben Eingabeaufforderung den folgenden Befehl aus.

```
nmake install
```

Ergebnisse

Das DBI Perl-Modul und die DBD::SQLAnywhere-Schnittstelle sind nun einsatzbereit.

DBD::SQLAnywhere unter Unix und Mac OS X installieren

Installieren Sie die DBD::SQLAnywhere-Schnittstelle auf der unterstützten Unix- oder Mac OS X-Plattform, um mit Perl auf SQL Anywhere-Datenbanken zugreifen zu können.

Voraussetzungen

Sie müssen mindestens ActivePerl 5.6.0 Build 616 oder später sowie einen C-Compiler installiert haben.

Aufgabe

1. Laden Sie die Quelldatei des DBI-Moduls unter www.cpan.org herunter.
2. Extrahieren Sie den Inhalt dieser Datei in ein neues Verzeichnis.
3. Wechseln Sie an einer Eingabeaufforderung in das neue Unterverzeichnis und führen Sie die folgenden Befehle aus, um das DBI-Modul zu erstellen.

```
perl Makefile.PL  
make
```

Wenn Sie aus einem bestimmten Grund neu beginnen müssen, können Sie den Befehl **make clean** ausführen, um teilweise bereits kompilierte Targets zu löschen.

4. Testen Sie das DBI-Modul mit dem nachstehenden Befehl.

```
make test
```

5. Um die Installation abzuschließen, führen Sie an derselben Eingabeaufforderung den folgenden Befehl aus.

```
make install
```

6. Stellen Sie sicher, dass die Umgebung für SQL Anywhere eingerichtet wurde.

Je nach der benutzten Shell geben Sie den entsprechenden Befehl ein, um das SQL Anywhere-Konfigurationsskript aus dem SQL Anywhere-Installationsverzeichnis aufzurufen:

Verwendete Shell	Verwendeter Befehl
sh, ksh oder bash	<code>. bin/sa_config.sh</code>
csh oder tcsh	<code>source bin/sa_config.csh</code>

7. An einer Shell-Eingabeaufforderung wechseln Sie ins Unterverzeichnis *sdk/perl* Ihres SQL Anywhere-Installationsverzeichnisses.
8. Geben Sie an einer Eingabeaufforderung die folgenden Befehle ein, um DBD::SQLAnywhere zu erstellen:

```
perl Makefile.PL  
  
make
```

Wenn Sie aus einem bestimmten Grund neu beginnen müssen, können Sie den Befehl **make clean** ausführen, um teilweise bereits kompilierte Targets zu löschen.

9. Um DBD::SQLAnywhere zu testen, kopieren Sie die Beispieldatenbankdatei in das Verzeichnis *sdk/perl* und führen Sie die Tests durch.

```
cp samples-dir/demo.db .  
  
dbsrv16 demo  
  
make test
```

Wenn die Tests nicht ausgeführt werden können, vergewissern Sie sich, dass das Unterverzeichnis *bin32* oder *bin64* der SQL Anywhere-Installation in der PATH-Umgebungsvariablen angegeben ist.

10. Um die Installation abzuschließen, führen Sie an derselben Eingabeaufforderung den folgenden Befehl aus.

```
make install
```

Ergebnisse

Das DBI Perl-Modul und die DBD::SQLAnywhere-Schnittstelle sind einsatzbereit.

Nächste Schritte

Optional können Sie die DBI-Quellstruktur löschen. Sie wird nicht mehr benötigt.

Perl-Skripten, die DBD::SQLAnywhere verwenden

In diesem Abschnitt finden Sie einen Überblick über die Erstellung von Perl-Skripten, die die DBD::SQLAnywhere-Schnittstelle verwenden. DBD::SQLAnywhere ist ein Treiber für das DBI-Modul. Die vollständige Dokumentation für das DBI-Modul finden Sie im Internet unter <http://dbi.perl.org>.

Das DBI-Modul

Um die DBD::SQLAnywhere-Schnittstelle aus einem Perl-Skript verwenden zu können, müssen Sie erst Perl informieren, dass Sie das DBI-Modul verwenden möchten. Dazu schreiben Sie die nachstehende Zeile an den Anfang der Datei.

```
use DBI;
```

Außerdem wird dringend empfohlen, dass Sie Perl im Modus "strict" ausführen. Diese Anweisung, die beispielsweise explizite Variablendefinitionen zwingend vorschreibt, kann dazu beitragen, dass Sie seltener auf Fehler treffen, die sich beispielsweise aus Tippfehlern ergeben.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
```

Falls erforderlich, lädt das DBI-Modul automatisch die DBD-Treiber, einschließlich DBD::SQLAnywhere.

Datenbankverbindungen mithilfe von Perl DBI öffnen und schließen

In der Regel öffnen Sie eine einzelne Verbindung zu einer Datenbank und führen dann alle erforderlichen Vorgänge durch, indem Sie eine Sequenz von SQL-Anweisungen ausführen. Öffnen Sie eine Verbindung mithilfe der Methode "connect". Der Rückgabewert ist ein Handle zur Datenbankverbindung, mit dem Sie die nachfolgenden Vorgänge in dieser Verbindung durchführen.

Die Parameter der Methode "connect" lauten wie folgt:

1. "DBI:SQLAnywhere:" und zusätzliche Verbindungsparameter, die durch Semikola getrennt werden.
2. Ein Benutzername. Wenn die Zeichenfolge nicht leer ist, wird ";UID=Wert" der Verbindungszeichenfolge angehängt.
3. Ein Kennwortwert. Wenn die Zeichenfolge nicht leer ist, wird ";PWD=Wert" der Verbindungszeichenfolge angehängt.
4. Ein Zeiger zu einem Hash von Standardwerten. Einstellungen wie AutoCommit, RaiseError und PrintError können auf diese Weise definiert werden.

Das nachstehende Codebeispiel öffnet und schließt eine Verbindung zur SQL Anywhere-Beispieldatenbank. Bevor Sie dieses Skript ausführen, müssen Sie den Datenbankserver und die Beispieldatenbank starten.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
```

```
my $pwd      = "sql";
my %defaults = (
    AutoCommit => 1, # Autocommit enabled.
    PrintError => 0  # Errors not automatically printed.
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$dbh->disconnect;
exit(0);
__END__
```

Optional können Sie den Wert des Benutzernamens oder des Kennworts an die Datenquellen-Zeichenfolge anhängen und nicht als getrennte Parameter übergeben. Wenn Sie dies tun, übergeben Sie eine leere Zeichenfolge für das entsprechende Argument. Beispiel: Das oben gezeigte Skript kann geändert werden, indem die Anweisung, die die Verbindung öffnet, durch die nachstehenden Anweisungen ersetzt wird:

```
$data_src .= ";UID=$uid";
$data_src .= ";PWD=$pwd";
my $dbh = DBI->connect($data_src, '', '', \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
```

Ergebnismengen mit Perl DBI abrufen

Nachdem Sie ein Handle für eine offene Verbindung erhalten haben, können Sie auf die Daten in der Datenbank zugreifen und sie ändern. Der einfachste Vorgang ist dabei, einige Zeilen abzurufen und auszudrucken.

SQL-Anweisungen, die Zeilenmengen zurückgeben, müssen vor ihrer Ausführung vorbereitet werden. Die prepare-Methode gibt ein Handle für diese Anweisung zurück. Mit diesem Handle führen Sie die Anweisung aus. Anschließend rufen Sie die Metadaten der Ergebnismenge und die Zeilen der Ergebnismenge ab.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
               FROM Customers
               ORDER BY GivenName, Surname";

my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
    my($sel, $dbh) = @_;
    my($row, $sth) = undef;
```

```

    $sth = $dbh->prepare($sel);
    $sth->execute;
    print "Fields:      $sth->{NUM_OF_FIELDS}\n";
    print "Params:      $sth->{NUM_OF_PARAMS}\n\n";
    print join("\t\t", @{$sth->{NAME}}), "\n\n";
    while($row = $sth->fetchrow_arrayref) {
        print join("\t\t", @$row), "\n";
    }
    $sth = undef;
}
__END__

```

Vorbereitete Anweisungen werden vom Datenbankserver nicht gelöscht, bis der Perl-Anweisungshandle entfernt wird. Sie entfernen einen Anweisungshandle, indem Sie die Variable erneut verwenden oder auf "undef" setzen. Durch den Aufruf der finish-Methode wird der Handle nicht gelöscht. Die finish-Methode sollte grundsätzlich nur dann aufgerufen werden, wenn eine Ergebnismenge nicht gelesen werden soll.

Um Handle-Leaks zu erkennen, begrenzt der SQL Anywhere-Datenbankserver standardmäßig die Anzahl von Cursors und vorbereiteten Anweisungen auf maximal 50 pro Verbindung. Der Ressourcenwächter generiert automatisch einen Fehler, wenn diese Grenzwerte überschritten werden. Wenn dieser Fehler auftritt, suchen Sie nach nicht entfernten Anweisungshandles. Verwenden Sie `prepare_cached` mit Bedacht, da die Anweisungshandles nicht entfernt werden.

Erforderlichenfalls können Sie diese Grenzwerte ändern, indem Sie die Optionen `max_cursor_count` und `max_statement_count` definieren.

Siehe auch

- „`max_cursor_count`-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „`max_statement_count`-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]

Mehrere Ergebnismengen mit Perl DBI verarbeiten

Die Methode zur Behandlung von mehreren Ergebnismengen aus einer Abfrage umfasst das Einbetten der Abrufschleife in einer anderen Schleife, der sich zwischen Ergebnismengen bewegt.

SQL-Anweisungen, die mehrere Ergebnismengen zurückgeben, müssen vor ihrer Ausführung vorbereitet werden. Die `prepare`-Methode gibt ein Handle für diese Anweisung zurück. Mit diesem Handle führen Sie die Anweisung aus. Anschließend rufen Sie die Metadaten der Ergebnismenge und die Zeilen der einzelnen Ergebnismenge ab.

```

#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
               FROM Customers
               ORDER BY GivenName, Surname;
               SELECT *
               FROM Departments
               ORDER BY DepartmentID";

```

```
my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
&db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
    my($sel, $dbh) = @_;
    my($row, $sth) = undef;
    $sth = $dbh->prepare($sel);
    $sth->execute;
    do {
        print "Fields:      $sth->{NUM_OF_FIELDS}\n";
        print "Params:      $sth->{NUM_OF_PARAMS}\n\n";
        print join("\t\t", @{$sth->{NAME}}), "\n\n";
        while($row = $sth->fetchrow_arrayref) {
            print join("\t\t", @$row), "\n";
        }
        print "---end of results---\n\n";
    } while (defined $sth->more_results);
    $sth = undef;
}
__END__
```

Zeilen mit Perl DBI einfügen

Zum Einfügen von Zeilen wird ein Handle für eine offene Verbindung benötigt. Die einfachste Methode besteht darin, eine parametrisierte INSERT-Anweisung zu verwenden, in der Fragezeichen als Platzhalter für Werte eingesetzt werden. Die Anweisung wird erst vorbereitet und dann pro neuer Zeile ausgeführt. Die neuen Zeilenwerte werden als Parameter an die execute-Methode übergeben.

Das folgende Beispielprogramm fügt zwei neue Kunden ein. Obwohl die Zeilenwerte als Literalzeichenfolgen erscheinen, können Sie die Werte aus einer Datei einlesen.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $ins_stmt = "INSERT INTO Customers (ID, GivenName, Surname,
                                     Street, City, State, Country, PostalCode,
                                     Phone, CompanyName)
               VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Can't connect to $data_src: $DBI::errstr\n";
&db_insert($ins_stmt, $dbh);
$dbh->commit;
$dbh->disconnect;
```

```
exit(0);

sub db_insert {
    my($ins, $dbh) = @_;
    my($sth) = undef;
    my @rows = (
        "801,Alex,Alt,5 Blue Ave,New York,NY,USA,10012,5185553434,BXM",
        "802,Zach,Zed,82 Fair St,New York,NY,USA,10033,5185552234,Zap"
    );
    $sth = $dbh->prepare($ins);
    my $row = undef;
    foreach $row ( @rows ) {
        my @values = split(/,/ , $row);
        $sth->execute(@values);
    }
}

__END__
```

Python-Unterstützung

Die SQL Anywhere-Python-Datenbankschnittstelle, sqlanydb, ist eine Datenzugriffs-API für die Python-Sprache. In diesem Abschnitt wird beschrieben, wie Sie SQL Anywhere mit Python verwenden.

sqlanydb

Die SQL Anywhere Python-Datenbankschnittstelle, sqlanydb, ist eine Datenzugriffs-API für die Python-Sprache. Die Python-Datenbank-API-Spezifikation definiert eine Reihe von Methoden, die eine konsistente Datenbankschnittstelle bereitstellt, die von der tatsächlich verwendeten Datenbank unabhängig ist. Die Verwendung des sqlanydb-Moduls ermöglicht Ihren Python-Skripten einen direkten Zugriff auf SQL Anywhere-Datenbankserver.

Das sqlanydb-Modul implementiert die Python-Datenbank-API-Spezifikation v2.0 von Marc-André Lemburg einschließlich Erweiterungen. Nachdem Sie das sqlanydb-Modul installiert haben, können Sie aus Python-Skripten auf SQL Anywhere-Datenbanken zugreifen und darin Daten ändern.

Weitere Hinweise zur Python Database API Specification v2.0 (in englischer Sprache) finden Sie unter <http://www.python.org/dev/peps/pep-0249/>.

Wenn Python mit Threads verwendet wird, ist das sqlanydb-Modul threadsicher.

Anforderungen

Für das sqlanydb-Modul sind folgende Komponenten erforderlich.

- Python ist erforderlich. Eine Liste der unterstützten Versionen finden Sie unter <http://www.sybase.com/detail?id=1068981>.
- Das ctypes-Modul ist erforderlich. Um zu testen, dass das ctypes-Modul vorhanden ist, öffnen Sie ein Eingabeaufforderungsfenster und führen Python aus.

An der Python-Eingabeaufforderung geben Sie die folgende Anweisung ein.

```
import ctypes
```

Wenn Sie eine Fehlermeldung sehen, ist ctypes nicht vorhanden. Im Folgenden finden Sie ein Beispiel.

```
>>> import ctypes
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named ctypes
```

Wenn ctypes in Ihrer Python-Installation nicht enthalten ist, installieren Sie das Modul. Installationen befinden sich bei den SourceForge.net-Dateien unter http://sourceforge.net/project/showfiles.php?group_id=71702.

Peak EasyInstall installiert auch ctypes. Sie können Peak EasyInstall von <http://peak.telecommunity.com/DevCenter/EasyInstall> herunterladen.

Installieren der Python-Unterstützung unter Windows

Sie können Python-Unterstützung unter Windows einrichten, indem Sie das geeignete Python-Setupskript aus dem Unterverzeichnis *SDK\Python* Ihrer SQL Anywhere-Installation ausführen.

Voraussetzungen

Stellen Sie sicher, dass Python und das ctypes-Modul installiert sind. Eine Liste der unterstützten Versionen von Python finden Sie unter <http://www.sybase.com/detail?id=1068981>.

Aufgabe

1. Wechseln Sie an einer Systemeingabeaufforderung in das Unterverzeichnis *SDK\Python* des SQL Anywhere-Installationsverzeichnis.
2. Führen Sie zur Installation von sqlanydb den folgenden Befehl aus.

```
python setup.py install
```

3. Kopieren Sie zum Testen von sqlanydb eine Kopie der Beispieldatenbankdatei in das aktuelle Verzeichnis und führen Sie einen Test durch.

```
newdemo  
dbsrv16 demo  
python Scripts\test.py
```

Das Testskript stellt eine Verbindung mit dem Datenbankserver her und führt eine SQL-Abfrage aus. Wenn der Test erfolgreich war, wird die Meldung `sqlanydb successfully installed` angezeigt.

Wenn die Tests nicht ausgeführt werden können, vergewissern Sie sich, dass das Unterverzeichnis *bin32* oder *bin64* der SQL Anywhere-Installation in der PATH-Umgebungsvariablen angegeben ist.

Ergebnisse

Das sqlanydb-Modul ist nun zur Verwendung bereit.

Installieren der Python-Unterstützung unter Unix und Mac OS X

Sie können Python-Unterstützung unter Windows einrichten, indem Sie das geeignete Python-Setupskript aus dem Unterverzeichnis *sdk/Python* Ihrer SQL Anywhere-Installation ausführen.

Voraussetzungen

Stellen Sie sicher, dass Python und das ctypes-Modul installiert sind. Eine Liste der unterstützten Versionen von Python finden Sie unter <http://www.sybase.com/detail?id=1068981>.

Aufgabe

1. Stellen Sie sicher, dass die Umgebung für SQL Anywhere eingerichtet wurde.

Abhängig von der verwendeten Shell müssen Sie den entsprechenden Befehl eingeben, um das SQL Anywhere-Konfigurationsskript aus dem SQL Anywhere-Installationsverzeichnis mit dem source-Befehl auszuführen (*bin64* kann anstelle von *bin32* benutzt werden, wenn Sie die 64-Bit-Software installiert haben):

Verwendete Shell	Verwendeter Befehl
sh, ksh oder bash	<code>. bin32/sa_config.sh</code>
csh oder tcsh	<code>source bin/sa_config.csh</code>

2. Wechseln Sie an einer Shell-Eingabeaufforderung ins Unterverzeichnis *sdk/python* des SQL Anywhere-Installationsverzeichnisses.
3. Geben Sie zur Installation von sqlanydb den folgenden Befehl ein.

```
python setup.py install
```

4. Kopieren Sie zum Testen von sqlanydb eine Kopie der Beispieldatenbankdatei in das aktuelle Verzeichnis und führen Sie einen Test durch.

```
newdemo
dbsrv16 demo
python scripts/test.py
```

Das Testskript stellt eine Verbindung mit dem Datenbankserver her und führt eine SQL-Abfrage aus. Wenn der Test erfolgreich war, wird die Meldung `sqlanydb successfully installed.` angezeigt.

Wenn der Test nicht ausgeführt werden kann, vergewissern Sie sich, dass sich das Unterverzeichnis *bin32* bzw. *bin64* der SQL Anywhere-Installation in Ihrem Pfad befindet.

Ergebnisse

Das sqlanydb-Modul ist nun zur Verwendung bereit.

Python-Skripten, die sqlanydb verwenden

In diesem Abschnitt finden Sie einen Überblick über die Erstellung von Python-Skripten, die die sqlanydb-Schnittstelle verwenden. Die vollständige Dokumentation für die API finden Sie im Internet unter <http://www.python.org/dev/peps/pep-0249/>.

Das sqlanydb-Modul

Wenn Sie das sqlanydb-Modul aus einem Python-Skript verwenden möchten, müssen Sie es zunächst laden, indem Sie die folgende Zeile am Anfang der Datei einfügen.

```
import sqlanydb
```

Datenbankverbindungen mithilfe von Python öffnen und schließen

In der Regel öffnen Sie eine einzelne Verbindung zu einer Datenbank und führen dann alle erforderlichen Vorgänge durch, indem Sie eine Sequenz von SQL-Anweisungen ausführen. Öffnen Sie eine Verbindung mithilfe der Methode "connect". Der Rückgabewert ist ein Handle zur Datenbankverbindung, mit dem Sie die nachfolgenden Vorgänge in dieser Verbindung durchführen.

Die Parameter der connect-Methode werden als Gruppe von durch Kommata getrennten Schlüsselwort=Wert-Paaren angegeben.

```
sqlanydb.connect( keyword=value, ...)
```

Im Folgenden sind einige häufig verwendete Verbindungsparameter aufgelistet.

- **DataSourceName="DSN"** Eine Kurzform für diesen Verbindungsparameter ist **DSN="DSN"**. Beispiel: DataSourceName="SQL Anywhere 16 Demo".
- **UserID="Benutzer-ID"** Eine Kurzform für diesen Verbindungsparameter ist **UID="Benutzer-ID"**. Beispiel: UserID="DBA".
- **Password="Kennwort"** Eine Kurzform für diesen Verbindungsparameter ist **PWD="Kennwort"**. Beispiel: Password="sql".
- **DatabaseFile="Datenbankdatei"** Eine Kurzform für diesen Verbindungsparameter ist **DBF="Datenbankdatei"**. Beispiel: DatabaseFile="demo.db".

Das nachstehende Codebeispiel öffnet und schließt eine Verbindung zur SQL Anywhere-Beispieldatenbank. Bevor Sie dieses Skript ausführen, müssen Sie den Datenbankserver und die Beispieldatenbank starten.

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( userid="DBA",
                        password="sql" )

# Close the connection
con.close()
```

Um den Datenbankserver nicht manuell starten zu müssen, können Sie eine für den Start des Servers konfigurierte Datenquelle verwenden. Das folgende Beispiel zeigt eine entsprechende Datenquelle.

```
import sqlanydb

# Create a connection object
```

```
con = sqlanydb.connect( DSN="SQL Anywhere 16 Demo" )

# Close the connection
con.close()
```

Siehe auch

- „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)]

Ergebnismengen mit Python abrufen

Nachdem Sie ein Handle für eine offene Verbindung erhalten haben, können Sie auf die Daten in der Datenbank zugreifen und sie ändern. Der einfachste Vorgang ist dabei, einige Zeilen abzurufen und auszudrucken.

Mit der cursor-Methode können Sie einen Cursor zum Öffnen einer Verbindung erstellen. Mit der execute-Methode können Sie eine Ergebnismenge erstellen. Mit der fetchall-Methode können Sie die Zeilen in dieser Ergebnismenge abrufen.

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA",
                        password="sql" )

cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

Zeilen mit Python einfügen

Die einfachste Möglichkeit, Zeilen in eine Tabelle einzufügen, ist die Verwendung einer INSERT-Anweisung ohne Parameter, bei der die Werte als Teil der SQL-Anweisung angegeben werden. Für jede neue Zeile wird eine neue Anweisung erstellt und ausgeführt. Wie im vorherigen Beispiel ist ein Cursor erforderlich, um SQL-Anweisungen auszuführen.

Das folgende Beispielprogramm fügt zwei neue Kunden in die Beispieldatenbank ein. Bevor die Verbindung getrennt wird, werden die Transaktionen in der Datenbank festgeschrieben.

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA", pwd="sql" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
        'USA','10012','5185553434','BXM'),
        (802,'Zach','Zed','82 Fair St','New York','NY',
        'USA','10033','5185552234','Zap'))

# Set up a SQL INSERT
parms = ("%s", " * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql % rows[0]
cursor.execute(sql % rows[0])
print sql % rows[1]
cursor.execute(sql % rows[1])
cursor.close()
con.commit()
con.close()
```

Eine alternative Methode bietet die Verwendung einer parametrisierten INSERT-Anweisung, in der Fragezeichen als Platzhalter für Werte eingesetzt werden. Die Methode `executemany` wird verwendet, um für jedes Mitglied der Zeilengruppe eine INSERT-Anweisung auszuführen. Die neuen Zeilenwerte werden als ein einziges Argument der `executemany`-Methode bereitgestellt.

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA", pwd="sql" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
        'USA','10012','5185553434','BXM'),
        (802,'Zach','Zed','82 Fair St','New York','NY',
        'USA','10033','5185552234','Zap'))

# Set up a parameterized SQL INSERT
parms = ("?", " * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql
cursor.executemany(sql, rows)
cursor.close()
con.commit()
con.close()
```

Wenngleich beide Beispiele gleichermaßen geeignete Verfahren zum Einfügen von Zeilendaten in eine Tabelle zu sein scheinen, ist das zweite Beispiel aus verschiedenen Gründen vorzuziehen. Wenn die Datenwerte durch Eingabeaufforderungen für die Dateneingabe erhalten werden, ist das erste Beispiel anfällig für das Eindringen von schädlichen Daten, möglicherweise auch von SQL-Anweisungen. Im ersten Beispiel wird für jede in die Tabelle einzufügende Zeile die `execute`-Methode aufgerufen. Im zweiten Beispiel wird die `executemany`-Methode nur einmal aufgerufen, und alle Zeilen werden gleichzeitig in die Tabelle eingefügt.

Datenbanktypkonvertierung

Um zu steuern, wie Datenbanktypen beim Abrufen von Ergebnissen vom Datenbankserver in Python-Objekte zugeordnet werden, können Sie Konvertierungs-Callbacks registrieren.

Callbacks werden mithilfe der auf Modulebene ausgeführten `register_converter`-Methode registriert. Diese Methode wird aufgerufen mit dem Datenbanktyp als erstem Parameter und der Konvertierungsfunktion als zweitem Parameter. Um beispielsweise anzufordern, dass sqlanydb Decimal-Objekte für Daten in einer als Typ `DT_DECIMAL` beschriebenen Spalte erstellt, würden Sie das folgende Beispiel verwenden:

```
import sqlanydb
import decimal

def convert_to_decimal(num):
    return decimal.Decimal(num)

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_decimal)
```

Konvertierer können für die folgenden Datenbanktypen registriert werden:

```
DT_DATE
DT_TIME
DT_TIMESTAMP
DT_VARCHAR
DT_FIXCHAR
DT_LONGVARCHAR
DT_DOUBLE
DT_FLOAT
DT_DECIMAL
DT_INT
DT_SMALLINT
DT_BINARY
DT_LONGBINARY
DT_TINYINT
DT_BIGINT
DT_UNSMALLINT
DT_UNSBIGINT
DT_BIT
```

Das folgende Beispiel zeigt, wie Sie Dezimalergebnisse in Ganzzahlen konvertieren, wobei Stellen nach dem Dezimalzeichen gekürzt werden. Der Gehaltsbetrag, der beim Ausführen der Anwendung angezeigt wird, ist ein Ganzzahlwert.

```
import sqlanydb

def convert_to_int(num):
    return int(float(num))

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_int)

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA",
                        password="sql" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees WHERE EmployeeID=105"
cursor.execute(sql)
```

```
# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

PHP-Unterstützung

SQL Anywhere-PHP-Unterstützung

Bei PHP (**PHP: Hypertext Preprocessor**) handelt es sich um eine Open Source-Skriptsprache. Sie kann zwar für diverse Programmierzwecke verwendet werden, war aber ursprünglich als programmiererfreundliche Sprache für Skripten gedacht, die in HTML-Dokumente eingebettet werden. Anders als bei JavaScript-Skripten, die oft auf dem Client ausgeführt werden, verarbeitet der Webserver die PHP-Skripten und sendet die resultierenden HTML-Seiten an den Client. Die Syntax von PHP beruht auf der Syntax anderer weit verbreiteter Sprachen wie etwa Java oder Perl.

Durch die Möglichkeit, Daten aus vielen weit verbreiteten Datenbanksystemen (wie beispielsweise SQL Anywhere) abzurufen, wird PHP zu einer anwendungsfreundlichen Sprache, mit der dynamische Webseiten erstellt werden können. Mit SQL Anywhere wird eine Erweiterung geliefert, die den Zugriff auf SQL Anywhere-Datenbanken aus PHP ermöglicht. Sie können diese Erweiterung und die Sprache PHP verwenden, um eigenständige Skripten zu schreiben und dynamische Webseiten zu erstellen, die mit Daten aus SQL Anywhere-Datenbanken arbeiten.

Die SQL Anywhere-PHP-Erweiterung bietet Ihnen eine native Methode für den Zugriff auf Ihre Datenbanken. Sie ist aufgrund seiner einfachen Struktur anderen PHP-Datenzugriffsmethoden vorzuziehen und trägt dazu bei, Ausfälle von Systemressourcen zu vermeiden, die bei der Verwendung anderer Methoden auftreten können.

Bereits kompilierte Versionen der PHP-Erweiterung stehen für Windows, Linux und Solaris zur Verfügung und befinden sich im Unterverzeichnis für Binärdateien Ihrer SQL Anywhere-Installation. Der Quellcode für die SQL Anywhere-PHP-Erweiterung befindet sich im Unterverzeichnis `sdk\php` Ihrer SQL Anywhere-Installation.

Weitere Hinweise und die neuesten SQL Anywhere-PHP-Treiber finden Sie unter <http://www.sybase.com/detail?id=1019698>.

Weitere Hinweise zur Installation der SQL Anywhere-PHP-Erweiterung finden Sie unter „[PHP Client-Deployment](#)“ auf Seite 984.

Testen der PHP-Erweiterung

Führen Sie eine schnellen Prüfung aus, um sich zu vergewissern, dass die PHP-Erweiterung von SQL Anywhere einwandfrei funktioniert.

Voraussetzungen

Alle erforderlichen PHP-Komponenten müssen auf Ihrem System installiert sein.

Aufgabe

1. Achten Sie darauf, dass das *bin32*-Unterverzeichnis Ihrer SQL Anywhere-Installation in Ihrem Pfad enthalten ist. Die DLL für die SQL Anywhere-PHP-Erweiterung erfordert, dass das *bin32*-Verzeichnis in Ihrem Pfad enthalten ist.
2. Führen Sie an einer Eingabeaufforderung den folgenden Befehl aus, um die SQL Anywhere-Beispieldatenbank zu starten.

```
dbsrv16 "%SQLANYAMP16%\demo.db"
```

Der Befehl startet einen Datenbankserver unter Verwendung der Beispieldatenbank.

3. An einer Eingabeaufforderung wechseln Sie ins Unterverzeichnis *SDK\PHP\Examples* Ihrer SQL Anywhere-Installation. Vergewissern Sie sich, dass das **php**-Programmverzeichnis in Ihrem Pfad enthalten ist. Geben Sie folgenden Befehl ein:

```
php test.php
```

Meldungen wie die folgende sollten angezeigt werden. Wenn der PHP-Befehl nicht erkannt wird, vergewissern Sie sich, dass PHP in Ihrem Pfad enthalten ist.

```
Installation successful
Using php-5.2.11_sqlanywhere.dll
Connected successfully
```

Wenn die SQL Anywhere-PHP-Erweiterung nicht geladen wird, können Sie den Befehl "php -i" verwenden, um nützliche Informationen zu Ihrem PHP-Setup zu erhalten. Suchen Sie nach **extension_dir** und **sqlanywhere** in der Ausgabe dieses Befehls.

4. Stoppen Sie abschließend den SQL Anywhere-Datenbankserver, indem Sie im Meldungsfenster des Datenbankservers auf **Herunterfahren** klicken.

Ergebnisse

Die Tests müssen erfolgreich verlaufen und zeigen, dass die PHP-Erweiterung von SQL Anywhere einwandfrei funktioniert.

Siehe auch

- „PHP-Testseiten erstellen und ausführen“ auf Seite 670

PHP-Testseiten erstellen und ausführen

Erstellen Sie einige Webseiten und führen Sie sie aus, um zu testen, ob PHP korrekt eingerichtet ist.

Voraussetzungen

Sie müssen PHP installieren. Hinweise zum Installieren von PHP finden Sie unter <http://us2.php.net/install>.

Kontext und Bemerkungen

Diese Prozedur gilt für alle Konfigurationen.

Aufgabe

1. Erstellen Sie eine Datei in Ihrem Root-Webinhaltsverzeichnis namens *info.php*.

Wenn Sie sich nicht sicher sind, welches Verzeichnis verwendet werden soll, prüfen Sie die Konfigurationsdatei Ihres Webservers. In Apache-Installationen wird das Inhaltsverzeichnis oft *htdocs* genannt. Wenn Sie Mac OS X verwenden, kann der Name des Webinhaltsverzeichnisses davon abhängen, welches Konto Sie benutzen:

- Wenn Sie der Systemadministrator auf einem Mac OS X-System sind, verwenden Sie */Library/WebServer/Documents*.
- Wenn Sie ein Mac OS X-Benutzer sind, legen Sie die Datei in */Users/Ihr_Benutzername/Sites/* an.

2. Fügen Sie folgenden Code in diese Datei ein:

```
<?php phpinfo(); ?>
```

Die PHP-Funktion `phpinfo` generiert eine Seite mit Informationen zur Systemkonfiguration. Damit wird bestätigt, dass Ihre Installation von PHP und Ihr Webserver einwandfrei zusammenarbeiten.

3. Kopieren Sie die Datei *connect.php* aus dem Verzeichnis *sdk\php\examples* in Ihr Stammverzeichnis für Webinhalte. Damit wird bestätigt, dass Ihre Installation von PHP und SQL Anywhere einwandfrei zusammenarbeiten.
4. Erstellen Sie in Ihrem Stammverzeichnis für Webinhalte eine Datei namens *sa_test.php* und fügen Sie folgenden Code in diese Datei ein.

```
<?php
$conn = sasql_connect( "UID=DBA;PWD=sql" );
$result = sasql_query( $conn, "SELECT * FROM Employees" );
sasql_result_all( $result );
sasql_free_result( $result );
sasql_disconnect( $conn );
?>
```

Die *sa_test*-Seite zeigt den Inhalt der Tabelle "Employees" an.

5. Starten Sie Ihren Webserver, wenn dies erforderlich ist.

Um beispielsweise den Apache-Webserver zu starten, führen Sie folgenden Befehl aus dem Unterverzeichnis *bin* Ihrer Apache-Installation aus:

```
apachectl start
```

6. Unter Linux oder Mac OS X setzen Sie die SQL Anywhere-Umgebungsvariablen über eines der mitgelieferten Skripten.

Je nach der benutzten Shell geben Sie den entsprechenden Befehl ein, um das SQL Anywhere-Konfigurationsskript aus dem SQL Anywhere-Installationsverzeichnis aufzurufen:

Verwendete Shell	Befehl
sh, ksh oder bash	<code>. /bin32/sa_config.sh</code>
csh oder tcsh	<code>source /bin32/sa_config.csh</code>

7. Führen Sie an einer Eingabeaufforderung den folgenden Befehl aus, um die SQL Anywhere-Beispieldatenbank zu starten (falls dies noch nicht geschehen ist):

```
dbsrv16 "%SQLANYAMP16%\demo.db"
```

8. Um zu testen, dass PHP und Ihr Webserver mit SQL Anywhere problemlos zusammenarbeiten, greifen Sie auf die Testseiten über einen Browser zu, der auf demselben Computer läuft wie der Server:

Testseite	Verwendete URL
<i>info.php</i>	<code>http://localhost/info.php</code>
<i>connect.php</i>	<code>http://localhost/connect.php</code>
<i>sa_test.php</i>	<code>http://localhost/sa_test.php</code>

Ergebnisse

Die info-Seite zeigt die Ausgabe aus dem `phpinfo()`-Aufruf an.

Die connect-Seite zeigt die Meldung `Connected successfully` an.

Die sa_test-Seite zeigt den Inhalt der Tabelle "Employees" an.

PHP-Skriptentwicklung

In diesem Abschnitt wird beschrieben, wie Sie PHP-Skripten schreiben, die die SQL Anywhere-PHP-Erweiterung für den Zugriff auf SQL Anywhere-Datenbanken verwenden.

Der Quellcode für diese und andere Beispiele befindet sich im Unterverzeichnis *SDK\PHP\Examples* Ihrer SQL Anywhere-Installation.

So stellen Sie die Verbindung mit einer Datenbank über PHP her

Um eine Verbindung mit einer Datenbank vorzunehmen, übergeben Sie eine SQL Anywhere-Standard-Verbindungszeichenfolge an den Datenbankserver als Parameter für die Funktion `sa_sql_connect`. Die Tags `<?php` und `>` weisen den Webserver an, PHP den Code ausführen zu lassen, der von ihnen eingeschlossen wird, und ihn mit der PHP-Ausgabe zu ersetzen.

Der Quellcode für dieses Beispiel ist in Ihrer SQL Anywhere-Installation in einer Datei namens *connect.php* enthalten.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    echo "Connection failed\n";
} else {
    echo "Connected successfully\n";
    sasql_close( $conn );
}?>
```

Dieses Skript versucht, eine Verbindung mit einer Datenbank auf einem lokalen Server herzustellen. Damit der Code erfolgreich ausgeführt werden kann, muss auf einem lokalen Server die SQL Anywhere-Beispieldatenbank oder eine Datenbank mit identischen Anmeldeinformationen gestartet werden.

Daten aus einer Datenbank mit PHP abrufen

In Webseiten werden PHP-Skripten oft verwendet, um die in einer Datenbank enthaltenen Informationen abzurufen und anzuzeigen. Die nachstehenden Beispiele zeigen einige nützliche Techniken.

Einfache Select-Abfrage

Der nachstehende PHP-Code zeigt eine einfache Methode, um die Ergebnismenge einer SELECT-Anweisung in eine Webseite einzufügen. In diesem Beispiel wird eine Verbindung mit der SQL Anywhere-Beispieldatenbank aufgenommen und eine Liste von Kunden zurückgegeben.

Dieser Code kann in eine Webseite eingebettet werden, wenn Ihr Webserver für PHP-Skripten konfiguriert ist.

Der Quellcode für dieses Beispiel ist in Ihrer SQL Anywhere-Installation in einer Datei namens *query.php* enthalten.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    echo "sasql_connect failed\n";
} else {
    echo "Connected successfully\n";
    # Execute a SELECT statement
    $result = sasql_query( $conn, "SELECT * FROM Customers" );
    if( ! $result ) {
        echo "sasql_query failed!";
    } else {
        echo "query completed successfully\n";
        # Generate HTML from the result set
        sasql_result_all( $result );
        sasql_free_result( $result );
    }
    sasql_close( $conn );
}
?>
```

Die Funktion `sasql_result_all` holt alle Zeilen der Ergebnismenge und generiert eine HTML-Ausgabetabelle, um sie anzuzeigen. Die Funktion `sasql_free_result` gibt die Ressourcen frei, die zum Speichern der Ergebnismenge verwendet wurden.

Nach Spaltennamen abrufen

Es kann sein, dass Sie nicht alle Daten aus einer Ergebnismenge anzeigen wollen oder die Daten in unterschiedlicher Weise aufbereitet werden sollen. Im folgenden Beispiel wird gezeigt, wie Sie größere Kontrolle über das Ausgabeformat der Ergebnismenge gewinnen können. PHP ermöglicht Ihnen die Anzeige von Daten in jeder beliebigen Menge und in jedem gewünschten Format.

Der Quellcode für dieses Beispiel ist in Ihrer SQL Anywhere-Installation in einer Datei namens *fetch.php* enthalten.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    die ("Connection failed");
} else {
    # Connected successfully.
}
# Execute a SELECT statement
$result = sasql_query( $conn, "SELECT * FROM Customers" );
if( ! $result ) {
    echo "sasql_query failed!";
    return 0;
} else {
    echo "query completed successfully\n";
}
# Retrieve meta information about the results
$num_cols = sasql_num_fields( $result );
$num_rows = sasql_num_rows( $result );
echo "Num of rows = $num_rows\n";
echo "Num of cols = $num_cols\n";
while( ($field = sasql_fetch_field( $result )) ) {
    echo "Field # : $field->id \n";
    echo "\tname      : $field->name \n";
    echo "\tlength   : $field->length \n";
    echo "\ttype      : $field->type \n";
}
# Fetch all the rows
$curr_row = 0;
while( ($row = sasql_fetch_row( $result )) ) {
    $curr_row++;
    $curr_col = 0;
    while( $curr_col < $num_cols ) {
        echo "$row[$curr_col]\t|";
        $curr_col++;
    }
    echo "\n";
}
# Clean up.
sasql_free_result( $result );
sasql_disconnect( $conn );
?>
```

Die Funktion `sasql_fetch_array` gibt eine einzelne Zeile aus der Tabelle zurück. Die Daten können nach Spaltennamen und Spaltenindizes abgerufen werden.

Die Funktion `sasql_fetch_assoc` gibt eine einzelne Zeile als ein assoziatives Array zurück. Die Daten können unter Verwendung der Spaltennamen als Indizes abgerufen werden. Im Folgenden finden Sie ein Beispiel.

Der Quellcode für dieses Beispiel ist in Ihrer SQL Anywhere-Installation in einer Datei namens *fetch_assoc.php* enthalten.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );

/* check connection */
if( sasql_errorcode() ) {
    printf("Connect failed: %s\n", sasql_error());
    exit();
}

$query = "SELECT Surname, Phone FROM Employees ORDER by EmployeeID";

if( $result = sasql_query($conn, $query) ) {

    /* fetch associative array */
    while( $row = sasql_fetch_assoc($result) ) {
        printf ("%s (%s)\n", $row["Surname"], $row["Phone"]);
    }

    /* free result set */
    sasql_free_result($result);
}

/* close connection */
sasql_close($conn);
?>
```

Zwei weitere ähnliche Methoden werden in der PHP-Schnittstelle bereitgestellt: *sasql_fetch_row* gibt eine Zeile zurück, die nur nach Spaltenindizes durchsucht werden kann, während *sasql_fetch_object* eine Zeile zurückgibt, die nur nach Spaltennamen durchsucht werden kann.

Ein Beispiel für die Funktion *sasql_fetch_object* finden Sie im Beispielskript *fetch_object.php*.

Verschachtelte Ergebnismengen

Wenn eine SELECT-Anweisung an die Datenbank geschickt wird, gibt das Programm eine Ergebnismenge zurück. Die Funktionen *sasql_fetch_row* und *sasql_fetch_array* rufen Daten aus den einzelnen Zeilen einer Ergebnismenge ab und geben jede Zeile als Spaltenarray zurück, in dem weitere Abfragen durchgeführt werden können.

Der Quellcode für dieses Beispiel ist in Ihrer SQL Anywhere-Installation in einer Datei namens *nested.php* enthalten.

```
<?php
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( $conn ) {
    // get the GROUPO user id
    $result = sasql_query( $conn,
        "SELECT user_id FROM SYS.SYSUSER " .
        "WHERE user_name='GROUPO' " );
    if( $result ) {
        $row = sasql_fetch_array( $result );
        $user = $row[0];
    } else {
        $user = 0;
    }
    // get the tables created by user GROUPO
```

```

$result = sasql_query( $conn,
    "SELECT table_id, table_name FROM SYS.SYSTABLE " .
    "WHERE creator = $user" );
if( $result ) {
    $num_rows = sasql_num_rows( $result );
    echo "Returned rows : $num_rows\n";
    while( $row = sasql_fetch_array( $result ) ) {
        echo "Table: $row[1]\n";
        $query = "SELECT table_id, column_name FROM SYS.SYSCOLUMN " .
            "WHERE table_id = '$row[table_id]';"
        $result2 = sasql_query( $conn, $query );
        if( $result2 ) {
            echo "Columns:";
            while( $detailed = sasql_fetch_array( $result2 ) ) {
                echo " $detailed[column_name]";
            }
            sasql_free_result( $result2 );
        }
        echo "\n\n";
    }
    sasql_free_result( $result );
}
sasql_disconnect( $conn );
}
?>

```

Im obigen Beispiel wählt die SQL-Anweisung die Tabellen-ID und den Namen für jede Tabelle in SYSTAB aus. Die Funktion `sasql_query` gibt ein Zeilenarray zurück. Das Skript durchläuft die Zeilen mithilfe der Funktion `sasql_fetch_array`, um die Zeilen aus einem Array abzufragen. Eine innere Iteration wandert durch die Spalten der einzelnen Zeile und gibt ihre Werte aus.

Webformulare

PHP kann Benutzereingaben aus einem Webformular entgegennehmen, als SQL-Abfrage an die Datenbank weitergeben und das zurückgegebene Ergebnis anzeigen. Das folgende Beispiel zeigt ein einfaches Webformular, das dem Benutzer die Möglichkeit gibt, die Beispieldatenbank mit SQL-Anweisungen abzufragen und die Ergebnisse in einer HTML-Tabelle auszugeben.

Der Quellcode für dieses Beispiel ist in Ihrer SQL Anywhere-Installation in einer Datei namens *webisql.php* enthalten.

```

<?php
echo "<HTML>\n";
echo "<body onload=\"document.getElementById('qbox').focus()\">\n";

$qname = $_POST[qname];
$qname = str_replace( "\\\"", "", $qname );
echo "<form method=post action=webisql.php>\n";
echo "<br>Query : <input id=qbox type=text size=80 name=qname value="
\"$qname\">\n";
echo "<input type=submit>\n";
echo "</form>\n";
echo "<HR><br>\n";

if( ! $qname ) {
    echo "No Current Query\n";
    return;
}

```

```

$conn = sasql_connect( "UID=DBA;PWD=sql" );

if( ! $conn ) {
    echo "sasql_connect failed\n";
} else {

    $qname = str_replace( "\\\"", "\"", $qname );
    $result = sasql_query( $conn, $qname );

    if( ! $result ) {
        echo "sasql_query failed!";
    } else {

        if( sasql_field_count( $conn ) > 0 ) {
            sasql_result_all( $result, "border=1" );
            sasql_free_result( $result );
        } else {
            echo "The statement <h3>$qname</h3> executed successfully!";
        }
    }

    sasql_close( $conn );
}

echo "</body>\n";
echo "</html>\n";
?>

```

Dieses Design kann erweitert werden, um komplexe Webformulare zu verarbeiten, indem benutzerdefinierte SQL-Abfragen basierend auf den vom Benutzer eingegebenen Werten formuliert werden.

BLOBs in PHP-Anwendungen

SQL Anywhere-Datenbanken können alle Arten von Daten als Binary Large Objects (BLOBs) speichern. Wenn diese Daten einen Datentyp haben, der von einem Webbrowser gelesen werden kann, können Sie sie mit einem PHP-Skript aus der Datenbank abrufen und in einer dynamisch generierten Seite anzeigen.

BLOB-Felder werden häufig zum Speichern von Daten verwendet, bei denen es sich nicht um Text handelt, wie z.B. Bilder in den Formaten GIF oder JPG. Viele Datentypen können an einen Webbrowser übergeben werden, ohne dass zusätzliche Software oder Datentypkonvertierungen notwendig wären. Das folgende Beispiel zeigt, wie ein Bild in die Datenbank eingefügt und dann zur Anzeige in einem Webbrowser wieder abgerufen wird.

Dieses Beispiel ähnelt dem Beispielcode in den Dateien *image-insert.php* und *image-retrieve.php* der SQL Anywhere-Installation. Die Beispiele veranschaulichen auch die Verwendung einer BLOB-Spalte zum Speichern von Bildern.

```

<?php
$conn = sasql_connect( "UID=DBA;PWD=sql" )
    or die("Cannot connect to database");
$create_table = "CREATE TABLE images (ID INTEGER PRIMARY KEY, img IMAGE)";
sasql_query( $conn, $create_table);
$insert = "INSERT INTO images VALUES (99,
xp_read_file('ianywhere_logo.gif'))";
sasql_query( $conn, $insert );
$query = "SELECT img FROM images WHERE ID = 99";

```

```
$result = sasql_query($conn, $query);
$data = sasql_fetch_row($result);
$img = $data[0];
header("Content-type: image/gif");
echo $img;
sasql_disconnect($conn);
?>
```

Um die Binärdaten aus der Datenbank direkt in einem Webbrowser einzulesen, muss das Skript den MIME-Typ der Daten über die Header-Funktion festlegen. In diesem Fall wird dem Browser mitgeteilt, dass ein GIF-Bild übertragen wird, damit er dieses richtig anzeigen kann.

Erstellen der SQL Anywhere-PHP-Erweiterung unter Unix und Mac OS X

Um PHP unter Unix und Mac OS X über die SQL Anywhere-PHP-Erweiterung mit SQL Anywhere zu verbinden, müssen Sie die Dateien der SQL Anywhere-PHP-Erweiterung der PHP-Quellenstruktur hinzufügen und danach PHP neu kompilieren.

SQL Anywhere-PHP-Erweiterungsdateien zur PHP-Quellenstruktur unter Unix oder Mac OS X hinzufügen

In diesem Thema werden die Schritte beschrieben, die erforderlich sind, um SQL Anywhere-PHP-Erweiterungsdateien zur PHP-Quellenstruktur hinzuzufügen.

Voraussetzungen

Nachfolgend wird eine Liste von Programmen gezeigt, die Sie auf Ihrem System benötigen, um die Nutzung der SQL Anywhere-PHP-Erweiterung unter Unix und Mac OS X zu ermöglichen:

- Ihre SQL Anywhere-Installation kann auf demselben Computer laufen wie der Apache-Webserver oder auf einem anderen Computer.
- Der Quellcode für die SQL Anywhere-PHP-Erweiterung, der unter http://download.sybase.com/anywhere/php/2.0.3/src/sasql_php.zip heruntergeladen werden kann.

Sie müssen auch **sqlpp** und *libdblib16.so* (Unix) oder *libdblib16.dylib* (Mac OS X) installiert haben. (Überprüfen Sie Ihr SQL Anywhere-Verzeichnis *lib32*.)

- Der PHP-Quellcode, der unter <http://www.php.net> heruntergeladen werden kann.
Eine Liste der unterstützten Versionen finden Sie unter <http://www.sybase.com/detail?id=1068981>.

- Der Apache-Webserver-Quellcode, der unter <http://httpd.apache.org> heruntergeladen werden kann.

Wenn Sie vorhaben, eine bereits erstellte Version von Apache zu verwenden, vergewissern Sie sich, dass Sie **apache** und **apache-devel** installiert haben.

- Wenn Sie vorhaben, die Unified ODBC PHP-Erweiterung zu verwenden, müssen Sie *libdbodbc16.so* (Unix) oder *libdbodbc16.dylib* (Mac OS X) installiert haben. (Überprüfen Sie Ihr SQL Anywhere-Verzeichnis *lib32*).

Die folgenden Binärdateien sollten von Ihrem Unix-Installationsmedium installiert werden, falls sie noch nicht installiert sind. Sie stehen als RPMs zur Verfügung:

- make
- automake
- autoconf
- libtool (glibtool für Mac OS X)
- makeinfo
- bison
- gcc
- cpp
- glibc-devel
- kernel-headers
- flex

Um bestimmte Installationsschritte ausführen zu können, müssen Sie dieselben Zugriffsprivilegien haben wie die Person, die PHP installiert hat. Die meisten Unix-basierten Systeme stellen einen **sudo**-Befehl bereit, der es Benutzern mit unzureichenden Berechtigungen ermöglicht, bestimmte Befehle als Benutzer mit den entsprechenden Berechtigungen auszuführen.

Kontext und Bemerkungen

Um bestimmte Installationsschritte ausführen zu können, müssen Sie dieselben Zugriffsprivilegien haben wie die Person, die PHP installiert hat. Die meisten Unix-basierten Systeme stellen einen **sudo**-Befehl bereit, der es Benutzern mit unzureichenden Berechtigungen ermöglicht, bestimmte Befehle als Benutzer mit den entsprechenden Berechtigungen auszuführen.

Aufgabe

1. Laden Sie den Quellcode für die SQL Anywhere-PHP-Erweiterung unter <http://www.sybase.com/detail?id=1019698> herunter. Suchen Sie nach dem Abschnitt mit dem Titel **Building the Driver from Source**.
2. Aus dem Verzeichnis, in dem Sie die SQL Anywhere-PHP-Erweiterung gespeichert haben, extrahieren Sie die Dateien in das *ext*-Unterverzeichnis der PHP-Quellenstruktur (wobei Mac OS X-Benutzer statt **tar** hier **gnutar** verwenden müssen):

```
$ tar -xzf sasql_php.zip -C PHP-source-directory/ext/
```

Das folgende Beispiel gilt für PHP Version 5.2.11. Sie müssen "php-5.2.11" in die Version von PHP ändern, die Sie verwenden.

```
$ tar -xzf sqlanywhere_php-1.0.8.tar.gz -C ~/php-5.2.11/ext
```

3. Weisen Sie PHP auf die Erweiterung hin:

```
$ cd PHP-source-directory/ext/sqlanywhere
$ touch *
$ cd ~/PHP-source-directory
$ ./buildconf
```

Das folgende Beispiel gilt für PHP Version 5.2.11. Sie müssen php-5.2.11 auf die Version von PHP ändern, die Sie verwenden.

```
$ cd ~/php-5.2.11/ext/sqlanywhere
$ touch *
$ cd ~/php-5.2.11
$ ./buildconf
```

4. Überprüfen Sie, ob PHP das Vorhandensein der Erweiterung erkennt.

```
$ ./configure -help | egrep sqlanywhere
```

Ergebnisse

Wenn es Ihnen gelungen ist, PHP auf das Vorhandensein der SQL Anywhere-Erweiterung hinzuweisen, sollten Sie den folgenden Text sehen:

```
--with-sqlanywhere=[DIR]
```

Protokollieren Sie im Fall eines Fehlschlagens die Ausgabe für diesen Befehl und veröffentlichen Sie sie im SQL Anywhere-Forum unter <http://sqlanywhere-forum.sybase.com>, um Unterstützung zu erhalten.

Apache und PHP kompilieren

PHP kann als gemeinsam genutztes Modul eines Webservers (z.B. Apache) oder als CGI-Programmdatei kompiliert werden. Wenn Sie einen Webserver verwenden, der von PHP nicht unterstützt wird, oder PHP-Skripten in einer Shell statt auf einer Webseite ausführen möchten, sollten Sie PHP als CGI-Programmdatei kompilieren. Wenn Sie hingegen PHP für die Ausführung mit Apache installieren möchten, kompilieren Sie PHP als Apache-Modul.

PHP als Apache-Modul kompilieren

Wenn Sie PHP für die Ausführung mit Apache installieren möchten, kompilieren Sie es als Apache-Modul.

Voraussetzungen

Konfigurieren Sie mit den folgenden Schritten Apache so, dass es gemeinsam genutzte Module erkennt.

Hinweis

Mac OS X wird mit einem vorinstallierten Apache-Webserver geliefert.

1. Konfigurieren Sie Apache so, dass es gemeinsam genutzte Module erkennt.

Führen Sie Befehle ähnlich den folgenden in dem Verzeichnis aus, in das Ihre Apache-Dateien extrahiert wurden.

```
$ cd Apache-source-directory
$ ./configure --enabled-shared=max --enable-module=most --prefix=/Apache-
installation-directory
```

Das folgende Beispiel gilt für Apache Version 2.2.9. Sie müssen **apache_2.2.9** auf die Version von Apache ändern, die Sie verwenden.

```
$ cd ~/apache_2.2.9
$ ./configure --enabled-shared=max --enable-module=most --prefix=/usr/
local/web/apache
```

2. Neukompilieren und installieren Sie die relevanten Komponenten:

```
$ make
$ make install
```

Nun sind Sie bereit, PHP für den Betrieb als Apache-Modul zu kompilieren.

Aufgabe

1. Stellen Sie sicher, dass die Umgebung für SQL Anywhere eingerichtet wurde.

Geben Sie abhängig von der verwendeten Shell den entsprechenden Befehl über das Verzeichnis ein, in dem SQL Anywhere installiert ist (standardmäßig */opt/sqlanywhere16*). Unter Mac OS X ist das Standardverzeichnis */Applications/SQLAnywhere16/System*.

Verwendete Shell	Befehl
sh, ksh, bash	<code>. ./bin32/sa_config.sh</code>
csh, tcsh	<code>source ./bin32/sa_config.csh</code>

2. Konfigurieren Sie PHP als Apache-Modul, um die SQL Anywhere-PHP-Erweiterung einzubeziehen.

Führen Sie die folgenden Befehle aus:

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere --with-apxs=/Apache-installation-
directory/bin/apxs
```

Das folgende Beispiel gilt für PHP Version 5.2.11. Sie müssen php-5.2.11 auf die Version von PHP ändern, die Sie verwenden.

```
$ cd ~/php-5.2.11
$ ./configure --with-sqlanywhere --with-apxs=/usr/local/web/apache/bin/
apxs
```

Das *configure*-Skript wird versuchen, die Version und die Position Ihrer SQL Anywhere-Installation zu ermitteln. In der Ausgabe des Befehls sollten ähnliche Zeilen wie die folgenden erscheinen:

```
checking for SQL Anywhere support... yes
checking SQL Anywhere install dir... /opt/sqlanywhere16
checking SQL Anywhere version... 16
```

3. Neukompilieren Sie die relevanten Komponenten:

```
$ make
```

4. Überprüfen Sie, dass die Bibliotheken korrekt verknüpft sind:

- Linux-Benutzer (das folgende Beispiel nimmt an, dass Sie PHP Version 5 verwenden):

```
ldd ../libs/libphp5.so
```

- Mac OS X-Benutzer:

Überprüfen Sie Ihre *httpd.conf*-Konfigurationsdatei, um zu ermitteln, wo sich *libphp5.so* auf Ihrem Computer befindet. Führen Sie die Überprüfung mit dem folgenden Befehl durch:

```
otool -L $LIBPHP5_DIR/libphp5.so
```

\$LIBPHP5_DIR ist das Verzeichnis, in dem sich *libphp5.so* befindet (entsprechend der Konfiguration Ihres Datenbankservers).

Dieser Befehl gibt eine Liste der Bibliotheken aus, die *libphp5.so* verwendet. Überprüfen Sie, ob sich *libdblib16.so* in der Liste befindet.

5. Installieren Sie die PHP-Binärdateien im *lib*-Verzeichnis von Apache.

```
$ make install
```

6. Führen Sie eine Verifizierung durch. PHP tut dies automatisch. Alles, was Sie durchführen müssen, ist sicherzustellen, dass Ihre *httpd.conf*-Konfigurationsdatei verifiziert ist und Apache die *.php*-Dateien als PHP-Skripten erkennt.

httpd.conf ist im *conf*-Unterverzeichnis des *Apache*-Verzeichnisses gespeichert:

```
$ cd Apache-installation-directory/conf
```

Beispiel:

```
$ cd /usr/local/web/apache/conf
```

Erstellen Sie eine Sicherungskopie von *httpd.conf*, bevor Sie sie bearbeiten (Sie können **pico** durch einen Texteditor Ihrer Wahl ersetzen):

```
$ cp httpd.conf httpd.conf.backup  
$ pico httpd.conf
```

Fügen Sie die folgenden Zeilen in *httpd.conf* ein hinzu oder entfernen Sie die Kommentarzeichen vor den entsprechenden Zeilen (die Zeilen befinden sich in der Datei nicht direkt untereinander):

```
LoadModule php5_module    libexec/libphp5.so  
AddModule mod_php5.c  
AddType application/x-httpd-php .php  
AddType application/x-httpd-php-source .phps
```

Hinweis

Unter Mac OS X sollten die letzten beiden Zeilen in die Datei *httpd_macosxserver.conf* eingefügt oder deren Kommentierung aufgehoben werden.

Die ersten beiden Zeilen verweisen Apache auf die Dateien, die für die Interpretation des PHP-Codes verwendet werden, während die anderen zwei Zeilen Dateitypen für Dateien deklarieren, deren Erweiterung *.php* oder *.phps* ist, damit Apache sie erkennen und entsprechend behandeln kann.

Ergebnisse

PHP wird erfolgreich als gemeinsam genutztes Modul kompiliert.

Siehe auch

- „PHP als CGI-Programmdatei kompilieren“ auf Seite 683
- „PHP-Testseiten erstellen und ausführen“ auf Seite 670

PHP als CGI-Programmdatei kompilieren

Wenn Sie einen Webserver verwenden, der von PHP nicht unterstützt wird, oder PHP-Skripten in einer Shell statt auf einer Webseite ausführen möchten, sollten Sie PHP als CGI-Programmdatei kompilieren.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Stellen Sie sicher, dass die Umgebung für SQL Anywhere eingerichtet wurde.

Geben Sie abhängig von der verwendeten Shell den entsprechenden Befehl über das Verzeichnis ein, in dem SQL Anywhere installiert ist (standardmäßig `/opt/sqlanywhere16`). Unter Mac OS X ist das Standardverzeichnis `/Applications/SQLAnywhere16/System`.

Verwendete Shell	Befehl
sh, ksh, bash	<code>. ./bin32/sa_config.sh</code>
csh, tcsh	<code>source ./bin32/sa_config.csh</code>

2. Konfigurieren Sie PHP als CGI-Programmdatei und mit der PHP-Erweiterung von SQL Anywhere.

Führen Sie den folgenden Befehl in dem Verzeichnis aus, in das Ihre PHP-Dateien extrahiert wurden:

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere
```

Das folgende Beispiel gilt für PHP Version 5.2.11. Sie müssen "php-5.2.11" in die Version von PHP ändern, die Sie verwenden.

```
$ cd ~/php-5.2.11
$ ./configure --with-sqlanywhere
```

Das Konfigurationsskript wird versuchen, die Version und die Position Ihrer SQL Anywhere-Installation zu ermitteln. Wenn Sie die Ausgabe dieses Befehls überprüfen, sollten Sie ähnliche Zeilen wie die folgenden sehen:

```
checking for SQL Anywhere support... yes
checking      SQL Anywhere install dir... /opt/sqlanywhere16
checking      SQL Anywhere version... 16
```

3. Kompilieren Sie die Programmdatei:

```
$ make
```

4. Installieren Sie die Komponenten.

```
$ make install
```

Ergebnisse

PHP wird erfolgreich als CGI-Programmdatei kompiliert.

Siehe auch

- „PHP als Apache-Modul kompilieren“ auf Seite 680
- „PHP-Testseiten erstellen und ausführen“ auf Seite 670

Überblick über die SQL Anywhere-PHP-API

Die PHP-API unterstützt die folgenden Funktionen:

Verbindungen

- „sasql_close“
- „sasql_connect“
- „sasql_disconnect“
- „sasql_error“
- „sasql_errorcode“
- „sasql_insert_id“
- „sasql_message“
- „sasql_pconnect“
- „sasql_set_option“

Abfragen

- „sasql_affected_rows“
- „sasql_next_result“
- „sasql_query“
- „sasql_real_query“
- „sasql_store_result“
- „sasql_use_result“

Ergebnismengen

- „sasql_data_seek“
- „sasql_fetch_array“
- „sasql_fetch_assoc“
- „sasql_fetch_field“
- „sasql_fetch_object“
- „sasql_fetch_row“
- „sasql_field_count“
- „sasql_free_result“
- „sasql_num_rows“
- „sasql_result_all“

Transaktionen

- „sasql_commit“
- „sasql_rollback“

Anweisungen

- „sasql_prepare“
- „sasql_stmt_affected_rows“
- „sasql_stmt_bind_param“
- „sasql_stmt_bind_param_ex“
- „sasql_stmt_bind_result“
- „sasql_stmt_close“
- „sasql_stmt_data_seek“
- „sasql_stmt_execute“
- „sasql_stmt_fetch“
- „sasql_stmt_field_count“
- „sasql_stmt_free_result“
- „sasql_stmt_insert_id“
- „sasql_stmt_next_result“
- „sasql_stmt_num_rows“
- „sasql_stmt_param_count“
- „sasql_stmt_reset“
- „sasql_stmt_result_metadata“
- „sasql_stmt_send_long_data“
- „sasql_stmt_store_result“

Verschiedenes

- „sasql_escape_string“
- „sasql_get_client_info“

sasql_affected_rows

Prototyp

int **sasql_affected_rows**(sasql_conn \$conn)

Beschreibung

Gibt die Anzahl der Zeilen zurück, die von der letzten SQL-Anweisung betroffen sind. Diese Funktion wird üblicherweise bei INSERT-, UPDATE- oder DELETE-Anweisungen verwendet. Bei SELECT-Anweisungen verwenden Sie die Funktion `sasql_num_rows`.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

Die Anzahl der betroffenen Zeilen

Siehe auch

- [„sasql_num_rows“](#)

sasql_commit

Prototyp

```
bool sasql_commit( sasql_conn $conn )
```

Beschreibung

Beendet eine Transaktion in der SQL Anywhere-Datenbank und schreibt während der Transaktion erfolgte Änderungen dauerhaft fest. Nur sinnvoll, wenn die Option `auto_commit` auf Off gesetzt ist.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_rollback“](#)
- [„sasql_set_option“](#)
- [„sasql_pconnect“](#)
- [„sasql_disconnect“](#)

sasql_close

Prototyp

```
bool sasql_close( sasql_conn $conn )
```

Beschreibung

Schließt eine vorher geöffnete Datenbankverbindung.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

sasql_connect

Prototyp

```
sasql_conn sasql_connect( string $con_str )
```

Beschreibung

Stellt eine Verbindung zu einer SQL Anywhere-Datenbank her.

Parameter

\$con_str Eine Verbindungszeichenfolge, die von SQL Anywhere erkannt wird.

Rückgabe

Eine aktive SQL Anywhere-Verbindungsressource bei Erfolg, eine Fehlermeldung und 0 bei einem Fehlschlag.

Siehe auch

- „sasql_pconnect“
- „sasql_disconnect“
- „Verbindungsparameter“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Datenbankverbindungen“ [[SQL Anywhere Server - Datenbankadministration](#)]

sasql_data_seek

Prototyp

```
bool sasql_data_seek( sasql_result $result, int row_num )
```

Beschreibung

Setzt den Cursor in Zeile *row_num* im *\$result*, das mit *sasql_query* geöffnet wurde.

Parameter

\$result Die Ergebnisressource, die von der Funktion *sasql_query* zurückgegeben wird.

row_num Ein ganzzahliger Wert, der die neue Position des Cursors innerhalb der Ergebnisressource repräsentiert. Sie können z.B. 0 angeben, um den Cursor in die erste Zeile der Ergebnismenge zu verschieben, oder 5, um ihn in die sechste Zeile zu verschieben. Negative Zahlen repräsentieren Zeilen relativ zum Ende der Ergebnismenge. Der Wert -1 verschiebt den Cursor z.B. in die letzte Zeile der Ergebnismenge und -2 verschiebt ihn in die vorletzte Zeile.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „[sasql_fetch_field](#)“
- „[sasql_fetch_array](#)“
- „[sasql_fetch_assoc](#)“
- „[sasql_fetch_row](#)“
- „[sasql_fetch_object](#)“
- „[sasql_query](#)“

sasql_disconnect

Prototyp

```
bool sasql_disconnect( sasql_conn $conn )
```

Beschreibung

Schließt eine Verbindung, die mit `sasql_connect` oder `sasql_pconnect` geöffnet wurde.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „[sasql_connect](#)“
- „[sasql_pconnect](#)“

sasql_error

Prototyp

```
string sasql_error( [ sasql_conn $conn ] )
```

Beschreibung

Gibt den Fehlertext der zuletzt ausgeführten SQL Anywhere PHP-Funktion zurück. Fehlermeldungen werden pro Verbindung gespeichert. Wenn keine `$conn` angegeben wurde, gibt `sasql_error` die letzte Fehlermeldung zurück, bei der keine Verbindung verfügbar war. Wenn Sie z.B. `sasql_connect` aufrufen und die Verbindung fehlschlägt, rufen Sie `sasql_error` ohne Parameter für `$conn` auf, um die Fehlermeldung zu erhalten. Wenn Sie den entsprechenden SQL Anywhere-Fehlercodewert abrufen möchten, verwenden Sie die `sasql_errorcode`-Funktion.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

Eine Zeichenfolge, die den Fehler beschreibt. Eine Liste der Fehlermeldungen finden Sie unter [„SQL Anywhere - Fehlermeldungen“](#) [*Fehlermeldungen*].

Siehe auch

- [„sasql_errorcode“](#)
- [„sasql_sqlstate“](#)
- [„sasql_set_option“](#)
- [„sasql_stmt_errno“](#)
- [„sasql_stmt_error“](#)

sasql_errorcode

Prototyp

```
int sasql_errorcode( [ sasql_conn $conn ] )
```

Beschreibung

Gibt den Fehlercode der zuletzt ausgeführten SQL Anywhere PHP-Funktion zurück. Fehlercodes werden pro Verbindung gespeichert. Wenn keine *\$conn* angegeben wurde, gibt `sasql_errorcode` den letzten Fehlercode zurück, bei der keine Verbindung verfügbar war. Wenn Sie z.B. `sasql_connect` aufrufen und die Verbindung fehlschlägt, rufen Sie `sasql_errorcode` ohne Parameter für *\$conn* auf, um den Fehlercode zu erhalten. Wenn Sie die entsprechende Fehlermeldung abrufen möchten, verwenden Sie die `sasql_error`-Funktion.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

Eine Ganzzahl, die einen SQL Anywhere-Fehlercode repräsentiert. Der Fehlercode 0 bedeutet, die Funktion war erfolgreich. Ein positiver Fehlercode zeigt einen erfolgreichen Vorgang mit Warnungen an. Ein negativer Fehlercode gibt einen Fehler an. Eine Liste der Fehlercodes finden Sie unter [„SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE“](#) [*Fehlermeldungen*].

Siehe auch

- [„sasql_connect“](#)
- [„sasql_pconnect“](#)
- [„sasql_error“](#)
- [„sasql_sqlstate“](#)
- [„sasql_set_option“](#)
- [„sasql_stmt_errno“](#)
- [„sasql_stmt_error“](#)

sasql_escape_string

Prototyp

string **sasql_escape_string**(sasql_conn \$conn, string \$str)

Beschreibung

Setzt Escapezeichen bei allen Sonderzeichen in der Zeichenfolge. Die Sonderzeichen, bei denen Escapezeichen gesetzt werden, sind \r, \n, ', ", ;, \ und das NULL-Zeichen. Diese Funktion ist ein Alias von sasql_real_escape_string.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$str Die Zeichenfolge, in der Escapezeichen gesetzt werden sollen.

Rückgabe

Die Zeichenfolge mit Escapezeichen.

Siehe auch

- „sasql_real_escape_string“
- „sasql_connect“

sasql_fetch_array

Prototyp

array **sasql_fetch_array**(sasql_result \$result [, int \$result_type])

Beschreibung

Ruft eine Zeile aus der Ergebnismenge ab. Diese Zeile wird als Array zurückgegeben, das durch die Spaltennamen oder die Spaltenindizes indiziert werden kann.

Parameter

\$result Die Ergebnisressource, die von der Funktion sasql_query zurückgegeben wird.

\$result_type Dieser optionale Parameter ist eine Konstante, die angibt, welcher Typ von Array von den aktuellen Zeilendaten erzeugt werden soll. Die möglichen Werte für diesen Parameter sind die Konstanten SASQL_ASSOC, SASQL_NUM oder SASQL_BOTH. Standardmäßig wird SASQL_BOTH verwendet.

Durch die Verwendung der Konstante SASQL_ASSOC verhält sich diese Funktion genauso wie die Funktion sasql_fetch_assoc, während sich SASQL_NUM genauso wie die Funktion sasql_fetch_row verhält. Die letzte Option SASQL_BOTH erstellt ein einzelnes Array mit den Attributen von beiden.

Rückgabe

Ein Array, das eine Zeile aus der Ergebnismenge repräsentiert, sonst FALSE, wenn keine Zeilen verfügbar sind.

Siehe auch

- [„sasql_data_seek“](#)
- [„sasql_fetch_assoc“](#)
- [„sasql_fetch_field“](#)
- [„sasql_fetch_row“](#)
- [„sasql_fetch_object“](#)

sasql_fetch_assoc

Prototyp

array **sasql_fetch_assoc**(sasql_result *\$result*)

Beschreibung

Ruft eine Zeile aus der Ergebnismenge als assoziatives Array ab.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

Rückgabe

Ein assoziatives Array von Zeichenfolgen, das die abgerufene Zeile in der Ergebnismenge darstellt, wobei jeder Schlüssel im Array den Namen einer der Spalten in der Ergebnismenge darstellt, oder FALSE, wenn es keine weiteren Zeilen in der Ergebnismenge gibt.

Siehe auch

- [„sasql_data_seek“](#)
- [„sasql_fetch_field“](#)
- [„sasql_fetch_field“](#)
- [„sasql_fetch_row“](#)
- [„sasql_fetch_object“](#)

sasql_fetch_field

Prototyp

object **sasql_fetch_field**(sasql_result *\$result* [, int *\$field_offset*])

Beschreibung

Gibt ein Objekt zurück, das Informationen über eine bestimmte Spalte enthält.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

\$field_offset Eine Ganzzahl, die die Spalte/das Feld darstellt, über die/das Sie Daten abrufen wollen. Spalten sind nullbasiert. Um die erste Spalte abzufragen, geben Sie den Wert 0 an. Wird dieser Parameter nicht verwendet, wird das nächste Feldobjekt zurückgegeben.

Rückgabe

Ein Objekt mit den folgenden Eigenschaften:

- **id** Enthält die Nummer des Felds.
- **name** Enthält den Namen des Felds.
- **numeric** Gibt an, ob das Feld ein numerischer Wert ist.
- **length** Gibt die native Speichergröße des Felds zurück.
- **type** Gibt den Typ des Felds zurück.
- **native_type** Gibt den nativen Typ des Felds zurück. Dies sind Werte wie DT_FIXCHAR, DT_DECIMAL oder DT_DATE. Siehe „Datentypen in Embedded SQL“ auf Seite 482.
- **precision** Gibt die numerische Gesamtstellenzahl des Felds zurück. Diese Eigenschaft wird nur bei Feldern mit native_type ist gleich DT_DECIMAL gesetzt.
- **scale** Gibt die Anzahl der Dezimalstellen des Felds zurück. Diese Eigenschaft wird nur bei Feldern mit native_type ist gleich DT_DECIMAL gesetzt.

Siehe auch

- „[sasql_data_seek](#)“
- „[sasql_fetch_array](#)“
- „[sasql_fetch_assoc](#)“
- „[sasql_fetch_row](#)“
- „[sasql_fetch_object](#)“

sasql_fetch_object

Prototyp

object **sasql_fetch_object**(sasql_result *\$result*)

Beschreibung

Ruft eine Zeile aus der Ergebnismenge als Objekt ab.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

Rückgabe

Ein Objekt, das die abgerufene Zeile in der Ergebnismenge darstellt, wobei jeder Eigenschaftsname einem der Ergebnismengen-Spaltennamen entspricht, oder FALSE, wenn es keine weiteren Zeilen in der Ergebnismenge gibt.

Siehe auch

- [„sasql_data_seek“](#)
- [„sasql_fetch_field“](#)
- [„sasql_fetch_array“](#)
- [„sasql_fetch_assoc“](#)
- [„sasql_fetch_row“](#)

sasql_fetch_row

Prototyp

array **sasql_fetch_row**(sasql_result *\$result*)

Beschreibung

Ruft eine Zeile aus der Ergebnismenge ab. Diese Zeile wird als Array zurückgegeben, das nur durch die Spaltenindizes indiziert werden kann.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

Rückgabe

Ein Array, das eine Zeile aus der Ergebnismenge repräsentiert, sonst FALSE, wenn keine Zeilen verfügbar sind.

Siehe auch

- [„sasql_data_seek“](#)
- [„sasql_fetch_field“](#)
- [„sasql_fetch_array“](#)
- [„sasql_fetch_assoc“](#)
- [„sasql_fetch_object“](#)

sasql_field_count

Prototyp

int **sasql_field_count**(sasql_conn *\$conn*)

Beschreibung

Gibt die Anzahl der Spalten (Felder) zurück, die das letzte Ergebnis enthält.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

Eine positive Anzahl von Spalten, oder FALSE, wenn *\$conn* nicht gültig ist.

sasql_field_seek

Prototyp

bool **sasql_field_seek**(sasql_result \$result, int \$field_offset)

Beschreibung

Setzt den Feldcursor an den angegebenen Ausgangspunkt. Der nächste Aufruf von `sasql_fetch_field` ruft die Felddefinition der Spalte auf, die diesem Ausgangspunkt zugeordnet ist.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

\$field_offset Eine Ganzzahl, die die Spalte/das Feld darstellt, über die/das Sie Daten abrufen wollen. Spalten sind nullbasiert. Um die erste Spalte abzufragen, geben Sie den Wert 0 an. Wird dieser Parameter nicht verwendet, wird das nächste Feldobjekt zurückgegeben.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

sasql_free_result

Prototyp

bool **sasql_free_result**(sasql_result \$result)

Beschreibung

Gibt Datenbankressourcen frei, die mit einer von `sasql_query` zurückgegebenen Ergebnisressource verbunden sind.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_query“](#)

sasql_get_client_info

Prototyp

string **sasql_get_client_info**()

Beschreibung

Gibt die Versionsinformationen des Clients zurück.

Parameter

Keine

Rückgabe

Eine Zeichenfolge, die die SQL Anywhere-Clientsoftware-Version darstellt. Die Zeichenfolge hat die Form X.Y.Z.W, wobei X die Hauptversionsnummer, Y die Unterversionsnummer, Z die Patch-Nummer und W die Build-Nummer ist (z.B. 10.0.1.3616).

sasql_insert_id

Prototyp

```
int sasql_insert_id( sasql_conn $conn )
```

Beschreibung

Gibt den letzten Wert zurück, der in eine IDENTITY-Spalte oder eine DEFAULT AUTOINCREMENT-Spalte eingegeben wurde, oder Null, wenn die letzte Einfügung in einer Tabelle erfolgt ist, die keine IDENTITY- oder DEFAULT AUTOINCREMENT-Spalte enthält.

Die Funktion sasql_insert_id wird aus Gründen der Kompatibilität mit MySQL-Datenbanken bereitgestellt.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

Die für eine autoincrement-Spalte von einer vorhergehenden INSERT-Anweisung generierte ID oder Null, wenn der letzte Einfügevorgang keine Auswirkungen auf eine autoincrement-Spalte hatte. Die Funktion kann FALSE zurückgeben, wenn \$conn nicht gültig ist.

sasql_message

Prototyp

```
bool sasql_message( sasql_conn $conn, string $message )
```

Beschreibung

Schreibt eine Meldung in das Meldungsfenster des Datenbankservers.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$message Eine Meldung, die in das Meldungsfenster des Datenbankservers geschrieben werden soll.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

sasql_multi_query

Prototyp

```
bool sasql_multi_query( sasql_conn $conn, string $sql_str )
```

Beschreibung

Bereitet eine oder mehrere SQL-Abfragen vor, die durch *\$sql_str* angegeben sind, und führt sie aus, indem die gelieferte Verbindungsressource verwendet wird. Die einzelnen Abfragen sind voneinander durch Semikolons getrennt.

Das erste Abfrageergebnis kann unter Verwendung von `sasql_store_result` oder `sasql_use_result` gespeichert bzw. abgerufen werden. `sasql_field_count` kann auch verwendet werden um zu überprüfen, ob die Abfrage eine Ergebnismenge zurückgibt.

Alle nachfolgenden Abfrageergebnisse können unter Verwendung von `sasql_next_result` und `sasql_use_result/sasql_store_result` verarbeitet werden.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$sql_str Eine oder mehr SQL-Anweisungen, durch Semikolons getrennt.

Weitere Hinweise über SQL-Anweisungen finden Sie unter „SQL-Anweisungen“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „[sasql_store_result](#)“
- „[sasql_use_result](#)“
- „[sasql_field_count](#)“

sasql_next_result

Prototyp

```
bool sasql_next_result( sasql_conn $conn )
```

Beschreibung

Bereitet die nächste Ergebnismenge von der letzten Abfrage vor, die auf *\$conn* ausgeführt wurde.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

FALSE, wenn es keine weitere Ergebnismenge zum Abrufen gibt. TRUE, wenn es eine weitere Ergebnismenge zum Abrufen gibt. Rufen Sie `sasql_use_result` oder `sasql_store_result` auf, um die nächste Ergebnismenge abzurufen.

Siehe auch

- [„sasql_use_result“](#)
- [„sasql_store_result“](#)

sasql_num_fields

Prototyp

```
int sasql_num_fields( sasql_result $result )
```

Beschreibung

Gibt die Anzahl der Felder zurück, die eine Zeile in *\$result* enthält.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

Rückgabe

Gibt die Anzahl der Felder in der angegebenen Ergebnismenge zurück.

Siehe auch

- [„sasql_num_rows“](#)
- [„sasql_query“](#)

sasql_num_rows

Prototyp

```
int sasql_num_rows( sasql_result $result )
```

Beschreibung

Gibt die Anzahl der Zeilen zurück, die *\$result* enthält.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

Rückgabe

Eine positive Zahl, wenn die Anzahl der Zeilen exakt ist, oder eine negative Zahl, wenn es sich um eine Schätzung handelt. Um die exakte Anzahl von Zeilen zu erhalten, muss die Datenbankoption `row_counts` in der Datenbank auf permanent oder für die Verbindung auf temporär gesetzt sein. Siehe [„sasql_set_option“](#) auf Seite 702.

Siehe auch

- „sasql_num_fields“
- „sasql_query“

sasql_pconnect

Prototyp

`sasql_conn` **sasql_pconnect**(string *\$con_str*)

Beschreibung

Stellt eine dauerhafte Verbindung zu einer SQL Anywhere-Datenbank her. Aufgrund der Apache-Prozedur für die Erstellung von Child-Prozessen können Sie die Performance verbessern, wenn Sie `sasql_pconnect` anstelle von `sasql_connect` verwenden. Dauerhafte Verbindungen können die Performance ähnlich wie Verbindungspools verbessern. Wenn Ihr Datenbankserver nur über eine begrenzte Anzahl von Verbindungen verfügt (so ist zum Beispiel der Personal Server auf 10 gleichzeitige Verbindungen beschränkt), sollten Sie bei der Verwendung dauerhafter Verbindungen vorsichtig sein. Dauerhafte Verbindungen können an jeden Child-Prozess angehängt werden, und wenn Sie in Apache mehr Child-Prozesse als verfügbare Verbindungen laufen haben, kommt es zu Verbindungsfehlern.

Parameter

`$con_str` Eine Verbindungszeichenfolge, die von SQL Anywhere erkannt wird.

Rückgabe

Eine aktive beständige SQL Anywhere-Verbindungsressource bei Erfolg, eine Fehlermeldung und 0 bei einem Fehlschlag.

Siehe auch

- „sasql_connect“
- „sasql_disconnect“
- „Verbindungsparameter“ [*SQL Anywhere Server - Datenbankadministration*]
- „Datenbankverbindungen“ [*SQL Anywhere Server - Datenbankadministration*]

sasql_prepare

Prototyp

`sasql_stmt` **sasql_prepare**(`sasql_conn` *\$conn*, string *\$sql_str*)

Beschreibung

Bereitet die angegebene SQL-Zeichenfolge vor.

Parameter

`$conn` Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$sql_str Die vorzubereitende SQL-Anweisung. Die Zeichenfolge kann Parametermarkierungen enthalten, indem Fragezeichen an den entsprechenden Positionen eingebettet werden.

Weitere Hinweise über SQL-Anweisungen finden Sie unter „SQL-Anweisungen“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Rückgabe

Ein Anweisungsobjekt oder FALSE bei Fehlschlag.

Siehe auch

- „sasql_stmt_param_count“
- „sasql_stmt_bind_param“
- „sasql_stmt_bind_param_ex“
- „sasql_prepare“
- „sasql_stmt_execute“
- „sasql_connect“
- „sasql_pconnect“

sasql_query

Prototyp

mixed **sasql_query**(sasql_conn \$conn, string \$sql_str [, int \$result_mode])

Beschreibung

Bereitet die SQL-Abfrage *\$sql_str* für die mit `sasql_connect` oder `sasql_pconnect` eingerichtete und durch *\$conn* identifizierte Verbindung vor und führt sie aus.

Die `sasql_query`-Funktion ist gleichwertig mit dem Aufrufen von zwei Funktionen: `sasql_real_query` und entweder `sasql_store_result` oder `sasql_use_result`.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$sql_str Eine SQL-Anweisung, die von SQL Anywhere unterstützt wird.

\$result_mode Entweder `SASQL_USE_RESULT` oder `SASQL_STORE_RESULT` (Standardwert).

Weitere Hinweise über SQL-Anweisungen finden Sie unter „SQL-Anweisungen“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Rückgabe

FALSE bei Fehlschlag; TRUE bei Erfolg für INSERT, UPDATE, DELETE, CREATE; `sasql_result` für SELECT.

Siehe auch

- „[sasql_real_query](#)“
- „[sasql_free_result](#)“
- „[sasql_fetch_array](#)“
- „[sasql_fetch_field](#)“
- „[sasql_fetch_object](#)“
- „[sasql_fetch_row](#)“

sasql_real_escape_string

Prototyp

string **sasql_real_escape_string**(sasql_conn \$conn, string \$str)

Beschreibung

Setzt Escapezeichen bei allen Sonderzeichen in der Zeichenfolge. Die Sonderzeichen, bei denen Escapezeichen gesetzt werden, sind \r, \n, ', ", ;, \ und das NULL-Zeichen.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$str Die Zeichenfolge, in der Escapezeichen gesetzt werden sollen.

Rückgabe

Die Zeichenfolge mit Escapezeichen oder FALSE bei einem Fehler.

Siehe auch

- „[sasql_escape_string](#)“
- „[sasql_connect](#)“

sasql_real_query

Prototyp

bool **sasql_real_query**(sasql_conn \$conn, string \$sql_str)

Beschreibung

Führt eine Abfrage in der Datenbank unter Verwendung der angegebenen Verbindungsressource durch. Das Abfragenergebnis kann unter Verwendung von `sasql_store_result` oder `sasql_use_result` gespeichert bzw. abgerufen werden. Die `sasql_field_count`-Funktion kann verwendet werden um zu überprüfen, ob die Abfrage eine Ergebnismenge zurückgibt.

Die `sasql_query`-Funktion ist gleichwertig mit dem Aufrufen dieser Funktion zusammen mit der Funktion `sasql_store_result` oder `sasql_use_result`.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$sql_str Eine SQL-Anweisung, die von SQL Anywhere unterstützt wird.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „[sasql_query](#)“
- „[sasql_store_result](#)“
- „[sasql_use_result](#)“
- „[sasql_field_count](#)“

sasql_result_all

Prototyp

```
bool sasql_result_all( resource $result
[, $html_table_format_string
[, $html_table_header_format_string
[, $html_table_row_format_string
[, $html_table_cell_format_string
]]])
```

Beschreibung

Ruft alle Ergebnisse von *\$result* ab und generiert eine HTML-Ausgabetablelle mit einer optionalen Formatierungszeichenfolge.

Parameter

\$result Die Ergebnisressource, die von der Funktion `sasql_query` zurückgegeben wird.

\$html_table_format_string Eine Formatzeichenfolge, die auf HTML-Tabellen angewendet werden kann. Beispiel: "**Border=1; Cellpadding=5**". Dieser spezielle Wert erstellt keine HTML-Tabelle. Dies ist nützlich, wenn Sie Ihre Spaltennamen oder Skripten anpassen möchten. Wenn Sie keinen expliziten Wert für diesen Parameter angeben möchten, verwenden Sie den Parameterwert NULL.

\$html_table_header_format_string Eine Formatzeichenfolge, die auf Spaltentitel von HTML-Tabellen angewendet werden kann. Beispiel: "**bgcolor=#FF9533**". Dieser spezielle Wert erstellt keine HTML-Tabelle. Dies ist nützlich, wenn Sie Ihre Spaltennamen oder Skripten anpassen möchten. Wenn Sie keinen expliziten Wert für diesen Parameter angeben möchten, verwenden Sie den Parameterwert NULL.

\$html_table_cell_format_string Eine Formatzeichenfolge, die auf Zeilen in HTML-Tabellen angewendet werden kann. Beispiel: "**onclick='alert('this')'**". Wenn Sie verschiedene, sich abwechselnde Formate wünschen, verwenden Sie das Spezial-Token `><`. Die linke Seite des Tokens gibt an, welches Format für ungerade Zeilen verwendet werden soll, und die rechte Seite des Tokens für das Format gerader Zeilen. Wenn Sie dieses Token nicht in Ihre Formatzeichenfolge positionieren, haben alle Zeilen dasselbe Format. Wenn Sie keinen expliziten Wert für diesen Parameter angeben wollen, verwenden Sie den Parameterwert NULL.

\$html_table_cell_format_string Eine Formatzeichenfolge, die auf Zellen in HTML-Tabellenzeilen angewendet werden kann. Zum Beispiel, "**onclick='alert('this')'**". Wenn Sie keinen expliziten Wert für diesen Parameter angeben wollen, verwenden Sie den Parameterwert NULL.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_query“](#)

sasql_rollback

Prototyp

bool **sasql_rollback**(sasql_conn \$conn)

Beschreibung

Beendet eine Transaktion in der SQL Anywhere-Datenbank und verwirft alle während der Transaktion erfolgten Änderungen. Diese Funktion ist nur sinnvoll, wenn die Option auto_commit auf Off gesetzt ist.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_commit“](#)
- [„sasql_set_option“](#)

sasql_set_option

Prototyp

bool **sasql_set_option**(sasql_conn \$conn, string \$option, mixed \$value)

Beschreibung

Legt den Wert der angegebenen Option für die angegebene Verbindung fest. Sie können den Wert für die folgenden Optionen festlegen:

Name	Beschreibung	Standardwert
auto_commit	Wenn diese Option auf on gesetzt ist, führt der Datenbankserver nach jeder Anweisung einen Festschreibevorgang aus.	on

Name	Beschreibung	Standardwert
row_counts	Wenn diese Option den Wert FALSE hat, gibt die Funktion <code>sasql_num_rows</code> eine Schätzung der Anzahl der betroffenen Zeilen zurück. Wenn Sie die genaue Anzahl abrufen möchten, setzen Sie diese Option auf TRUE.	FALSE
verbose_errors	Wenn diese Option auf TRUE gesetzt ist, gibt der PHP-Treiber ausführliche Fehlermeldungen zurück. Wenn diese Option auf FALSE gesetzt ist, müssen Sie die Funktion <code>sasql_error</code> bzw. <code>sasql_errorcode</code> aufrufen, um weitere Informationen zu erhalten.	TRUE

Sie können den Standardwert für eine Option ändern, indem Sie folgende Zeile in die *php.ini*-Datei einfügen. In diesem Beispiel ist für die Option `auto_commit` der Standardwert festgelegt.

```
sqlanywhere.auto_commit=0
```

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

\$option Der Name der Option, die Sie einrichten wollen

\$value Der neue Optionswert

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „[sasql_commit](#)“
- „[sasql_error](#)“
- „[sasql_errorcode](#)“
- „[sasql_num_rows](#)“
- „[sasql_rollback](#)“

sasql_stmt_affected_rows

Prototyp

```
int sasql_stmt_affected_rows( sasql_stmt $stmt )
```

Beschreibung

Gibt die Anzahl der Zeilen zurück, die von der Anweisung betroffen sind

Parameter

\$stmt Eine Anweisungsressource, die von `sasql_stmt_execute` ausgeführt wurde.

Rückgabe

Die Anzahl der betroffenen Zeilen oder FALSE bei Fehlschlag.

Siehe auch

- „[sasql_stmt_execute](#)“

sasql_stmt_bind_param

Prototyp

```
bool sasql_stmt_bind_param( sasql_stmt $stmt, string $types, mixed &$var_1 [, mixed &$var_2 .. ] )
```

Beschreibung

Bindet PHP-Variable an Anweisungsparameter.

Parameter

\$stmt Eine vorbereitete Anweisungsressource, die von der Funktion `sasql_prepare` zurückgegeben wurde.

\$types Eine Zeichenfolge, die ein oder mehr Zeichen enthält, die die Typen der entsprechenden Bindung festlegen. Dies kann einer der folgenden Werte sein: **s** für string, **i** für integer, **d** für double, **b** für blobs. Die Länge der \$types-Zeichenfolge muss der Anzahl der Parameter entsprechen, die dem \$types-Parameter nachfolgen (\$var_1, \$var_2, ...). Die Anzahl der Zeichen sollte auch der Anzahl der Parametermarkierungen (Fragezeichen) in der vorbereiteten Anweisung entsprechen.

\$var_n Die Variablenreferenzen.

Rückgabe

TRUE, wenn das Binden der Variablen erfolgreich war, ansonsten FALSE.

Siehe auch

- „[sasql_prepare](#)“
- „[sasql_stmt_param_count](#)“
- „[sasql_stmt_bind_param_ex](#)“
- „[sasql_stmt_execute](#)“

sasql_stmt_bind_param_ex

Prototyp

```
bool sasql_stmt_bind_param_ex( sasql_stmt $stmt, int $param_number, mixed &$var, string $type [, bool $is_null [, int $direction ] ] )
```

Beschreibung

Bindet eine PHP-Variable an einen Anweisungsparameter.

Parameter

\$stmt Eine vorbereitete Anweisungsressource, die von der Funktion `sasql_prepare` zurückgegeben wurde.

\$param_number Die Parameternummer. Dies sollte eine Zahl zwischen 0 und `(sasql_stmt_param_count($stmt) - 1)` sein.

\$var Eine PHP-Variable. Nur Referenzen zu PHP-Variablen sind zulässig.

\$type Typ der Variablen. Dies kann einer der folgenden Werte sein: **s** für string, **i** für integer, **d** für double, **b** für blobs.

\$is_null Ob der Wert der Variablen NULL ist oder nicht.

\$direction Kann `SASQL_D_INPUT`, `SASQL_D_OUTPUT` oder `SASQL_INPUT_OUTPUT` sein.

Rückgabe

TRUE, wenn das Binden der Variablen erfolgreich war, ansonsten FALSE.

Siehe auch

- „`sasql_prepare`“
- „`sasql_stmt_param_count`“
- „`sasql_stmt_bind_param`“
- „`sasql_stmt_execute`“

sasql_stmt_bind_result

Prototyp

```
bool sasql_stmt_bind_result( sasql_stmt $stmt, mixed &$var1 [, mixed &$var2 .. ] )
```

Beschreibung

Bindet eine oder mehrere PHP-Variablen an Ergebnisspalten einer ausgeführten Anweisung und gibt eine Ergebnismenge zurück.

Parameter

\$stmt Eine Anweisungsressource, die von `sasql_stmt_execute` ausgeführt wurde.

\$var1 Referenzen zu PHP-Variablen, die an Ergebnismengenspalten gebunden werden, die von `sasql_stmt_fetch` zurückgegeben werden.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „`sasql_stmt_execute`“
- „`sasql_stmt_fetch`“

sasql_stmt_close

Prototyp

bool **sasql_stmt_close**(sasql_stmt \$stmt)

Beschreibung

Schließt die angegebene Anweisungsressource und gibt ihr zugeordnete Ressourcen frei. Diese Funktion gibt auch Ergebnisobjekte frei, die von sasql_stmt_result_metadata zurückgegeben wurden.

Parameter

\$stmt Eine vorbereitete Anweisungsressource, die von der Funktion sasql_prepare zurückgegeben wurde.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehlschlag.

Siehe auch

- „sasql_stmt_result_metadata“
- „sasql_prepare“

sasql_stmt_data_seek

Prototyp

bool **sasql_stmt_data_seek**(sasql_stmt \$stmt, int \$offset)

Beschreibung

Diese Funktion führt einen Suchlauf an den angegebenen Ausgangspunkt in der Ergebnismenge durch.

Parameter

\$stmt Eine Anweisungsressource.

\$offset Der Ausgangspunkt in der Ergebnismenge. Dies ist eine Zahl zwischen 0 und (sasql_stmt_num_rows(\$stmt) - 1).

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehlschlag.

Siehe auch

- „sasql_stmt_num_rows“

sasql_stmt_errno

Prototyp

int **sasql_stmt_errno**(sasql_stmt \$stmt)

Beschreibung

Gibt den Fehlercode für die zuletzt ausgeführte Anweisungsfunktion unter Verwendung der angegebenen Anweisungsressource zurück.

Parameter

\$stmt Eine vorbereitete Anweisungsressource, die von der Funktion `sasql_prepare` zurückgegeben wurde.

Rückgabe

Ein Ganzzahlfehlercode. Eine Liste der Fehlercodes finden Sie unter „[SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE](#)“ [*Fehlermeldungen*].

Siehe auch

- „`sasql_stmt_error`“
- „`sasql_error`“
- „`sasql_errorcode`“
- „`sasql_prepare`“
- „`sasql_stmt_result_metadata`“

sasql_stmt_error

Prototyp

string `sasql_stmt_error`(sasql_stmt \$stmt)

Beschreibung

Gibt den Fehlertext für die zuletzt ausgeführte Anweisungsfunktion unter Verwendung der angegebenen Anweisungsressource zurück.

Parameter

\$stmt Eine vorbereitete Anweisungsressource, die von der Funktion `sasql_prepare` zurückgegeben wurde.

Rückgabe

Eine Zeichenfolge, die den Fehler beschreibt. Eine Liste der Fehlermeldungen finden Sie unter „[SQL Anywhere - Fehlermeldungen](#)“ [*Fehlermeldungen*].

Siehe auch

- „`sasql_stmt_erno`“
- „`sasql_error`“
- „`sasql_errorcode`“
- „`sasql_prepare`“
- „`sasql_stmt_result_metadata`“

sasql_stmt_execute

Prototyp

bool **sasql_stmt_execute**(sasql_stmt \$stmt)

Beschreibung

Führt die vorbereitete Anweisung aus. `sasql_stmt_result_metadata` kann verwendet werden um zu überprüfen, ob die Anweisung eine Ergebnismenge zurückgibt.

Parameter

\$stmt Eine vorbereitete Anweisungsressource, die von der Funktion `sasql_prepare` zurückgegeben wurde. Variable sollten vor dem Aufruf von `execute` gebunden sein.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehlschlag.

Siehe auch

- [„sasql_prepare“](#)
- [„sasql_stmt_param_count“](#)
- [„sasql_stmt_bind_param“](#)
- [„sasql_stmt_bind_param_ex“](#)
- [„sasql_stmt_result_metadata“](#)
- [„sasql_stmt_bind_result“](#)

sasql_stmt_fetch

Prototyp

bool **sasql_stmt_fetch**(sasql_stmt \$stmt)

Beschreibung

Diese Funktion ruft eine Zeile aus dem Ergebnis der Anweisung ab und platziert die Spalten in die Variablen, die mit `sasql_stmt_bind_result` gebunden wurden.

Parameter

\$stmt Eine Anweisungsressource.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_stmt_bind_result“](#)

sasql_stmt_field_count

Prototyp

int **sasql_stmt_field_count**(sasql_stmt \$stmt)

Beschreibung

Diese Funktion gibt die Anzahl der Spalten in der Ergebnismenge der Anweisung zurück.

Parameter

\$stmt Eine Anweisungsressource.

Rückgabe

Die Anzahl der Spalten im Ergebnis der Anweisung. Wenn die Anweisung kein Ergebnis zurückgibt, wird 0 zurückgegeben.

Siehe auch

- [„sasql_stmt_result_metadata“](#)

sasql_stmt_free_result

Prototyp

bool **sasql_stmt_free_result**(sasql_stmt \$stmt)

Beschreibung

Diese Funktion gibt die im Cache abgelegte Ergebnismenge der Anweisung frei.

Parameter

\$stmt Eine Anweisungsressource, die unter Verwendung von sasql_stmt_execute ausgeführt wurde.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_stmt_execute“](#)
- [„sasql_stmt_store_result“](#)

sasql_stmt_insert_id

Prototyp

int **sasql_stmt_insert_id**(sasql_stmt \$stmt)

Beschreibung

Gibt den letzten Wert zurück, der in eine IDENTITY-Spalte oder eine DEFAULT AUTOINCREMENT-Spalte eingegeben wurde, oder Null, wenn die letzte Einfügung in einer Tabelle erfolgt ist, die keine IDENTITY- oder DEFAULT AUTOINCREMENT-Spalte enthält.

Parameter

\$stmt Eine Anweisungsressource, die von `sasql_stmt_execute` ausgeführt wurde.

Rückgabe

Die ID, die für eine IDENTITY-Spalte oder eine DEFAULT AUTOINCREMENT-Spalte von einer vorherigen INSERT-Anweisung generiert wurde, oder 0, wenn sich die letzte Einfügung nicht auf eine IDENTITY- bzw. DEFAULT AUTOINCREMENT-Spalte ausgewirkt hat. Die Funktion kann FALSE (0) zurückgeben, wenn \$stmt nicht gültig ist.

Siehe auch

- [„sasql_stmt_execute“](#)

sasql_stmt_next_result

Prototyp

```
bool sasql_stmt_next_result( sasql_stmt $stmt )
```

Beschreibung

Diese Funktion geht zum nächsten Ergebnis der Anweisung. Wenn es eine weitere Ergebnismenge gibt, werden die derzeitigen Ergebnisse im Cache verworfen und das zugeordnete Ergebnismengenobjekt gelöscht (wie von `sasql_stmt_result_metadata` zurückgegeben).

Parameter

\$stmt Eine Anweisungsressource.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehlschlag.

Siehe auch

- [„sasql_stmt_result_metadata“](#)

sasql_stmt_num_rows

Prototyp

```
int sasql_stmt_num_rows( sasql_stmt $stmt )
```

Beschreibung

Gibt die Anzahl der Zeilen in der Ergebnismenge zurück. Die tatsächliche Zeilenanzahl in der Ergebnismenge kann nur ermittelt werden, nachdem die Funktion `sasql_stmt_store_result` aufgerufen wird, um die gesamte Ergebnismenge zu puffern. Wenn die Funktion `sasql_stmt_store_result` nicht aufgerufen wurde, wird 0 zurückgegeben.

Parameter

\$stmt Eine Anweisungsressource, die von `sasql_stmt_execute` ausgeführt und für die `sasql_stmt_store_result` aufgerufen wurde.

Rückgabe

Die Anzahl der im Ergebnis verfügbaren Zeilen, oder 0 bei Fehlschlag.

Siehe auch

- „[sasql_stmt_execute](#)“
- „[sasql_stmt_store_result](#)“

sasql_stmt_param_count

Prototyp

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

Beschreibung

Gibt die Anzahl der Parameter in der gelieferten vorbereiteten Anweisungsressource zurück.

Parameter

\$stmt Eine Anweisungsressource, die von der Funktion `sasql_prepare` zurückgegeben wird.

Rückgabe

Die Anzahl der Parameter oder FALSE bei Fehlschlag.

Siehe auch

- „[sasql_prepare](#)“
- „[sasql_stmt_bind_param](#)“
- „[sasql_stmt_bind_param_ex](#)“

sasql_stmt_reset

Prototyp

```
bool sasql_stmt_reset( sasql_stmt $stmt )
```

Beschreibung

Diese Funktion setzt das *\$stmt*-Objekt auf den Status unmittelbar nach dem DESCRIBE zurück. Gebundene Variablen werden entbunden und mit *sasql_stmt_send_long_data* gesendete Daten werden gelöscht.

Parameter

\$stmt Eine Anweisungsressource.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- „[sasql_stmt_send_long_data](#)“

sasql_stmt_result_metadata

Prototyp

```
sasql_result sasql_stmt_result_metadata( sasql_stmt $stmt )
```

Beschreibung

Gibt ein Ergebnismengenobjekt für die gelieferte Anweisung zurück.

Parameter

\$stmt Eine vorbereitete und ausgeführte Anweisungsressource.

Rückgabe

sasql_result-Objekt oder FALSE, wenn die Anweisung keine Ergebnisse zurückgibt.

sasql_stmt_send_long_data

Prototyp

```
bool sasql_stmt_send_long_data( sasql_stmt $stmt, int $param_number, string $data )
```

Beschreibung

Ermöglicht es dem Benutzer, Parameterdaten in Abschnitten zu senden. Der Benutzer muss zuerst *sasql_stmt_bind_param* oder *sasql_stmt_bind_param_ex* aufrufen, bevor er Daten senden kann. Der bind-Parameter muss vom Typ string oder blob sein. Ein wiederholter Aufruf dieser Funktion hängt Daten an vorher gesendete Daten an.

Parameter

\$stmt Eine Anweisungsressource, die unter Verwendung von *sasql_prepare* vorbereitet wurde.

\$param_number Die Parameternummer. Dies muss eine Zahl zwischen 0 und (*sasql_stmt_param_count(\$stmt)* - 1) sein.

\$data Die zu sendenden Daten.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_stmt_bind_param“](#)
- [„sasql_stmt_bind_param_ex“](#)
- [„sasql_prepare“](#)
- [„sasql_stmt_param_count“](#)

sasql_stmt_store_result

Prototyp

```
bool sasql_stmt_store_result( sasql_stmt $stmt )
```

Beschreibung

Diese Funktion ermöglicht es dem Client, die gesamte Ergebnismenge der Anweisung im Cache abzulegen. Sie können die Funktion `sasql_stmt_free_result` verwenden, um das im Cache abgelegte Ergebnis freizugeben.

Parameter

\$stmt Eine Anweisungsressource, die unter Verwendung von `sasql_stmt_execute` ausgeführt wurde.

Rückgabe

TRUE bei Erfolg oder FALSE bei Fehler.

Siehe auch

- [„sasql_stmt_free_result“](#)
- [„sasql_stmt_execute“](#)

sasql_store_result

Prototyp

```
sasql_result sasql_store_result( sasql_conn $conn )
```

Beschreibung

Überträgt die Ergebnismenge der letzten Abfrage auf der Datenbankverbindung `$conn` für die Verwendung mit der Funktion `sasql_data_seek`.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

FALSE, wenn die Abfrage kein Ergebnisobjekt zurückgibt, oder ein Ergebnismengenobjekt, das alle Zeilen des Ergebnisses enthält. Das Ergebnis wird im Cache des Client abgelegt.

Siehe auch

- „[sasql_data_seek](#)“
- „[sasql_stmt_execute](#)“

sasql_sqlstate

Prototyp

```
string sasql_sqlstate( sasql_conn $conn )
```

Beschreibung

Gibt die aktuellste SQLSTATE-Zeichenfolge zurück. SQLSTATE gibt an, ob die zuletzt ausgeführte SQL-Anweisung einen Erfolg, einen Fehler oder eine Warnung bewirkte. SQLSTATE-Codes bestehen aus fünf Zeichen, wobei "00000" kein Fehler bedeutet. Die Werte werden durch den ISO/ANSI SQL-Standard festgelegt.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

Gibt eine Zeichenfolge von fünf Zeichen zurück, die den aktuellen SQLSTATE-Code enthält. Das Ergebnis "00000" bedeutet, dass kein Fehler aufgetreten ist. Eine Liste der SQLSTATE-Codes finden Sie unter „[SQL Anywhere-Fehlermeldungen - sortiert nach SQLSTATE](#)“ [[Fehlermeldungen](#)].

Siehe auch

- „[sasql_error](#)“
- „[sasql_errorcode](#)“

sasql_use_result

Prototyp

```
sasql_result sasql_use_result( sasql_conn $conn )
```

Beschreibung

Initiiert einen Ergebnismengenabruf für die letzte Abfrage, die auf der Verbindung ausgeführt wurde.

Parameter

\$conn Die Verbindungsressource, die von einer Verbindungsfunktion zurückgegeben wird.

Rückgabe

FALSE, wenn die Abfrage kein Ergebnisobjekt zurückgibt, oder ein Ergebnismengenobjekt. Das Ergebnis wird nicht im Cache des Client abgelegt.

Siehe auch

- [„sasql_data_seek“](#)
- [„sasql_stmt_execute“](#)

Ruby-Unterstützung

SQL Anywhere-Ruby-API-Unterstützung

Es gibt drei verschiedene Ruby-APIs, die von SQL Anywhere unterstützt werden. Erstens wird die SQL Anywhere Ruby-API angeboten. Diese API bietet einen Ruby-Wrapper für die Schnittstelle, die von der SQL Anywhere C-API exponiert wird. Zweitens wird Unterstützung für ActiveRecord, einen objektrelationalen Mapper angeboten, der Teil des Webentwicklungs-Frameworks Ruby on Rails ist. Und drittens wird Unterstützung für Ruby DBI angeboten. SQL Anywhere stellt einen Ruby-Datenbanktreiber (DBD) bereit, der mit DBI verwendet werden kann.

Im SQL Anywhere für Ruby-Projekt sind drei verschiedene Pakete verfügbar. Die einfachste Möglichkeit, eines dieser Pakete zu installieren, ist es, **RubyGems** zu verwenden. Sie können RubyGems unter <http://rubyforge.org/projects/rubygems/> herunterladen.

Die Startseite für das SQL Anywhere Ruby-Projekt befindet sich unter <http://sqlanywhere.rubyforge.org/>.

Nativer SQL Anywhere Ruby-Treiber

sqlanywhere Dieses Paket enthält einen Low-Level-Treiber, der es Ruby ermöglicht, mit SQL Anywhere-Datenbanken über eine Schnittstelle zu verbinden. Dieses Paket bietet einen Ruby-Wrapper auf die Schnittstelle, die von der SQL Anywhere C-API bereitgestellt wird. Dieses Paket ist in C geschrieben und als Source oder als vorkompilierte RubyGems für Windows und Linux verfügbar. Wenn Sie **RubyGems** installiert haben, kann dieses Paket durch Ausführen des folgenden Befehls bezogen werden:

```
gem install sqlanywhere
```

Dieses Paket ist Voraussetzung für die anderen SQL Anywhere-Ruby-Pakete.

Rails

Rails ist ein Webentwicklungs-Framework, das in der Ruby-Sprache geschrieben ist. Seine Stärke liegt in der Entwicklung von Webanwendungen. Es wird dringend empfohlen, dass Sie sich mit der Ruby-Programmiersprache vertraut machen, bevor Sie mit der Rails-Entwicklung beginnen. Weitere Hinweise finden Sie unter <http://www.rubyonrails.org/>.

SQL Anywhere ActiveRecord-Adapter

activerecord-sqlanywhere-adapter Dieses Paket ist ein Adapter, der es ActiveRecord ermöglicht, mit SQL Anywhere zu kommunizieren. ActiveRecord ist ein objektrelationaler Mapper, der Teil des Webentwicklungs-Frameworks "Ruby on Rails" ist. Dieses Paket ist in reinem Ruby geschrieben und im Source- oder im Gem-Format verfügbar. Dieser Adapter verwendet das **sqlanywhere**-Gem und hat entsprechende Abhängigkeiten. Wenn Sie **RubyGems** installiert haben, können dieses Paket und seine Abhängigkeiten durch Ausführen des folgenden Befehls installiert werden:

```
gem install activerecord-sqlanywhere-adapter
```

SQL Anywhere Ruby/DBI-Treiber

dbi Dieses Paket ist ein DBI-Treiber für Ruby. Wenn Sie RubyGems installiert haben, können dieses Paket und seine Abhängigkeiten durch Ausführen des folgenden Befehls installiert werden:

```
gem install dbi
```

dbd-sqlanywhere Dieses Paket ist ein Treiber, der es Ruby/DBI ermöglicht, mit SQL Anywhere zu kommunizieren. Ruby/DBI ist eine generische Datenbankschnittstelle, die dem populären DBI-Modul von Perl ähnelt. Dieses Paket ist in reinem Ruby geschrieben und im Source- oder im Gem-Format verfügbar. Dieser Treiber verwendet das sqlanywhere-Gem und hat entsprechende Abhängigkeiten. Wenn Sie RubyGems installiert haben, können dieses Paket und seine Abhängigkeiten durch Ausführen des folgenden Befehls installiert werden:

```
gem install dbd-sqlanywhere
```

Für Rückmeldungen zu diesen Paketen können Sie die Mailing-Liste sqlanywhere-users@rubyforge.com verwenden.

Siehe auch

- Ruby on Rails: <http://www.rubyonrails.org/>
- „SQL Anywhere-Ruby-API-Referenz“
- „Ruby-DBI-Treiber für SQL Anywhere“
- Ruby-Programmiersprache: <http://www.ruby-lang.org>
- RubyForge / Ruby Central: <http://rubyforge.org/>
- Ruby/DBI - Direkte Datenbankzugriffsschicht für Ruby <http://ruby-dbi.rubyforge.org/>
- „Rails-Unterstützung in SQL Anywhere konfigurieren“
- Download-Quelle: <http://rubyforge.org/projects/sqlanywhere>
- RDocs: <http://sqlanywhere.rubyforge.org/sqlanywhere/>
- RDocs: <http://sqlanywhere.rubyforge.org/dbd-sqlanywhere>
- RDocs: <http://sqlanywhere.rubyforge.org/activerecord-sqlanywhere-adapter>

Rails-Unterstützung in SQL Anywhere konfigurieren

Sie können die Unterstützung für Ruby on Rails in SQL Anywhere installieren.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Installieren Sie RubyGems. Es vereinfacht die Installation der Ruby-Pakete. Über die Download-Seite Ruby on Rails wird Ihnen die korrekte Version für die Installation gezeigt. Weitere Hinweise finden Sie unter <http://www.rubyonrails.org/>.
2. Installieren Sie den Ruby-Interpreter auf Ihrem System. Auf der Ruby on Rails-Website finden Sie Empfehlungen, welche Version Sie installieren sollten. Weitere Hinweise finden Sie unter <http://www.rubyonrails.org/>.

3. Installieren Sie Ruby Rails und die davon abhängigen Komponenten, indem Sie den folgenden Befehl ausführen:

```
gem install rails
```

4. Installieren Sie das Ruby Development Kit (**DevKit**). Laden Sie das **DevKit** von <http://rubyinstaller.org/downloads/> herunter und befolgen Sie Anweisungen unter <http://github.com/oneclick/rubyinstaller/wiki/Development-Kit>.

5. Installieren Sie die ActiveRecord-Unterstützung von SQL Anywhere (activerecord-sqlanywhere-adapter), indem Sie den folgenden Befehl ausführen:

```
gem install activerecord-sqlanywhere-adapter
```

6. Fügen Sie SQL Anywhere zu den von Rails unterstützten Datenbank-Managementsystemen hinzu. Zum Zeitpunkt der Dokumentationserstellung war die Version Rails 3.1.3 aktuell.

- a. Konfigurieren Sie eine Datenbank, indem Sie eine Datei namens *sqlanywhere.yml* im Rails-Verzeichnis *configs/databases* erstellen. Wenn Sie Ruby im Verzeichnis *\Ruby* installiert haben und die Version 3.1.3 von Rails installiert ist, lautet der Pfad zu dieser Datei *\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\rails\app\templates\config\databases*. Diese Datei muss Folgendes enthalten:

```
#
# SQL Anywhere database configuration
#
# This configuration file defines the patten used for
# database filenames. If your application is called "blog",
# then the database names will be blog_development,
# blog_test, blog_production. The specified username and
# password should permit DBA access to the database.
#

development:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_development
  username: DBA
  password: sql

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_test
  username: DBA
  password: sql

production:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_production
  username: DBA
  password: sql
```

Die Datei *sqlanywhere.yml* bietet eine Vorlage für die Erstellung von *database.yml*-Dateien in Rails-Projekten. Die folgenden Datenbankoptionen können angegeben werden:

- **adapter** (erforderlich, kein Standardwert). Diese Option muss auf **sqlanywhere** eingestellt werden, damit der SQL Anywhere ActiveRecord-Adapter verwendet werden kann.
 - **database** (erforderlich, kein Standardwert). Diese Option entspricht DatabaseName in einer Verbindungszeichenfolge.
 - **server** (optional, standardmäßig Option **database**). Diese Option entspricht ServerName in einer Verbindungszeichenfolge.
 - **username** (optional, standardmäßig 'DBA'). Diese Option entspricht UserID in einer Verbindungszeichenfolge.
 - **password** (optional, standardmäßig 'sql'). Diese Option entspricht Password in einer Verbindungszeichenfolge.
 - **encoding** (optional, standardmäßig der Betriebssystem-Zeichensatz der Datenbank). Diese Option entspricht CharSet in einer Verbindungszeichenfolge.
 - **commlinks** (optional). Diese Option entspricht CommLinks in einer Verbindungszeichenfolge.
 - **connection_name** (optional). Diese Option entspricht ConnectionName in einer Verbindungszeichenfolge.
- b. Aktualisieren Sie die Rails-Datei *app_base.rb*. Unter den gleichen Annahmen wie im vorherigen Schritt befindet sich diese Datei im Pfad `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\app_base.rb`. Bearbeiten Sie die Datei *app_base.rb* und suchen Sie die folgende Zeile:

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase ibm_db  
sqlserver )
```

Fügen Sie **sqlanywhere** folgendermaßen zur Liste hinzu:

```
DATABASES = %w( sqlanywhere mysql oracle postgresql sqlite3 frontbase  
ibm_db sqlserver )
```

7. Arbeiten Sie die praktische Einführung auf der Ruby on Rails-Website (http://guides.rails.info/getting_started.html) durch und verwenden Sie dabei die folgenden SQL Anywhere-spezifischen Hinweise:
- In der praktischen Einführung erfahren Sie, welchen Befehl Sie verwenden, um das **blog**-Projekt zu initialisieren. Hier ist der Befehl, um das **blog**-Projekt für die Verwendung mit SQL Anywhere zu initialisieren:

```
rails new blog -d sqlanywhere
```

- Wechseln Sie nach dem Erstellen der **blog**-Anwendung zu deren Ordner, um direkt in dieser Anwendung weiterzuarbeiten:

```
cd blog
```

- Bearbeiten Sie die Datei *gemfile*, um eine **gem**-Direktive für den SQL Anywhere ActiveRecord-Adapter einzubeziehen. Fügen Sie die neue Direktive nach der unten angegebenen Zeile hinzu:

```
gem 'sqlanywhere'  
gem 'activerecord-sqlanywhere-adapter'
```

- Die Datei `config/database.yml` referenziert Entwicklungs-, Test- und Produktionsdatenbank. Statt einen **rake**-Befehl zum Erstellen der Datenbanken zu verwenden, wie in der praktischen Einführung angegeben, wechseln Sie in das Verzeichnis `db` des Projekts und erstellen Sie folgendermaßen drei Datenbanken.

```
cd db
dbinit -dba DBA,sql blog_development
dbinit -dba DBA,sql blog_test
dbinit -dba DBA,sql blog_production
cd ..
```

Ergebnisse

Sie haben die Rails-Unterstützung in SQL Anywhere konfiguriert.

Nächste Schritte

Starten Sie folgendermaßen den Datenbankserver und die drei Datenbanken.

```
dbsrv16 -n blog blog_development.db blog_production.db blog_test.db
```

Der Datenbankservername in der Befehlszeile (*blog*) muss zum Namen passen, der durch die Tags **server:** in der Datei `database.yml` festgelegt ist. Die Vorlagendatei `sqlanywhere.yml` wird so konfiguriert, dass gewährleistet ist, dass der Datenbankservername in allen generierten `database.yml`-Dateien mit dem Projektnamen übereinstimmt.

Siehe auch

- <http://www.rubyonrails.org/>

Ruby-DBI-Treiber für SQL Anywhere

Dieser Abschnitt bietet eine Übersicht darüber, wie Sie Ruby-Anwendungen schreiben, die den SQL Anywhere DBI-Treiber verwenden. Die vollständige Dokumentation für das DBI-Modul finden Sie im Internet unter <http://ruby-dbi.rubyforge.org/>.

DBI-Modul laden

Um die DBD:SQLAnywhere-Schnittstelle aus einer Ruby-Anwendung verwenden zu können, müssen Sie zuerst Ruby informieren, dass Sie das Ruby-DBI-Modul verwenden möchten. Dazu schreiben Sie die nachstehende Zeile an den Anfang der Ruby-Quelldatei.

```
require 'dbi'
```

Das DBI-Modul lädt, falls erforderlich, automatisch die Schnittstelle für den SQL Anywhere-Datenbanktreiber (DBD).

Verbindung öffnen und schließen

In der Regel öffnen Sie eine einzelne Verbindung zu einer Datenbank und führen dann alle erforderlichen Vorgänge durch, indem Sie eine Sequenz von SQL-Anweisungen ausführen. Öffnen Sie eine Verbindung mithilfe der `connect`-Funktion. Der Rückgabewert ist ein Handle zur Datenbankverbindung, mit dem Sie die nachfolgenden Vorgänge in dieser Verbindung durchführen.

Der Aufruf der connect-Funktion hat die allgemeine Form:

```
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password, options)
```

- **Servername** Dies ist der Name des Datenbankservers, zu dem Sie eine Verbindung herstellen wollen. Alternativ können Sie auch eine Verbindungszeichenfolge im Format "Option1=Wert;Option2=Wert;..." angeben.
- **Benutzer-ID** Dies ist eine gültige Benutzer-ID. Wenn die Zeichenfolge nicht leer ist, wird ";UID=Wert" der Verbindungszeichenfolge angehängt.
- **Kennwort** Dies ist das entsprechende Kennwort für die Benutzer-ID. Wenn die Zeichenfolge nicht leer ist, wird ";PWD=Wert" der Verbindungszeichenfolge angehängt.
- **Optionen** Dies ist ein Hash mit zusätzlichen Verbindungsparametern wie z.B. DatabaseName, DatabaseFile und ConnectionName. Diese werden der Verbindungszeichenfolge im Format "Option=Wert1;Option=Wert2;..." angehängt.

Um die connect-Funktion zu veranschaulichen, starten Sie den Datenbankserver und die Beispieldatenbank, bevor Sie die Ruby Beispielskripten ausführen.

```
dbsrv16 -n myserver "%SQLANYSAMPl6%\demo.db"
```

Das nachstehende Codebeispiel öffnet und schließt eine Verbindung zur SQL Anywhere-Beispieldatenbank. Die Zeichenfolge "myserver" im Beispiel unten ist der Servername.

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', 'DBA', 'sql') do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

Optional können Sie auch eine Verbindungszeichenfolge statt des Servernamens angeben. Das obenstehende Skript kann beispielsweise geändert werden, indem Sie den ersten Parameter in der connect-Funktion folgendermaßen ändern:

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:SERVER=myserver;DBN=demo', 'DBA', 'sql') do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

Die Benutzer-ID und das Kennwort können nicht in der Verbindungszeichenfolge angegeben werden. Ruby DBI füllt automatisch den Benutzernamen und das Kennwort mit Standardwerten, wenn diese Argumente ausgelassen werden. Daher sollten Sie in Ihrer Verbindungszeichenfolge niemals einen UID- oder PWD-Verbindungsparameter angeben. Wenn Sie es dennoch tun, wird eine Ausnahmebedingung generiert.

Das folgende Beispiel zeigt, wie zusätzliche Verbindungsparameter an die connect-Funktion als Hash aus Schlüsselwort/Wert-Paaren übergeben werden können.

```

require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', 'DBA', 'sql',
  { :ConnectionName => "RubyDemo",
    :DatabaseFile => "demo.db",
    :DatabaseName => "demo" }
) do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end

```

Daten auswählen

Nachdem Sie ein Handle für eine offene Verbindung erhalten haben, können Sie auf die Daten in der Datenbank zugreifen und sie ändern. Der einfachste Vorgang ist dabei, einige Zeilen abzurufen und auszudrucken.

Zuerst muss eine SQL-Anweisung ausgeführt werden. Wenn die Anweisung eine Ergebnismenge zurückgibt, verwenden Sie das resultierende Anweisungs-Handle, um Metainformationen über die Ergebnismenge und die Zeilen der Ergebnismenge abzurufen. Das folgende Beispiel bezieht die Spaltennamen aus den Metadaten und zeigt die Spaltennamen und die Werte für jede abgerufene Zeile an.

```

require 'dbi'

def db_query( dbh, sql )
  sth = dbh.execute(sql)
  print "# of Fields:  #{sth.column_names.size}\n"
  sth.fetch do |row|
    print "\n"
    sth.column_info.each_with_index do |info, i|
      unless info["type_name"] == "LONG VARBINARY"
        print "#{info["name"]}={#{row[i]}}\n"
      end
    end
  end
  sth.finish
end

begin
  dbh = DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql')
  db_query(dbh, "SELECT * FROM Products")
rescue DBI::DatabaseError => e
  puts "An error occurred"
  puts "Error code: #{e.err}"
  puts "Error message: #{e.errstr}"
  puts "Error SQLSTATE: #{e.state}"
ensure
  dbh.disconnect if dbh
end

```

Die ersten paar Zeilen der Ausgabe, die angezeigt werden, finden Sie untenstehend.

```

# of Fields:  8

ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28

```

```
UnitPrice=9.00

ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

Es ist wichtig, "finish" aufzurufen, um das Anweisungs-Handle freizugeben, wenn Sie fertig sind. Wenn Sie dies nicht tun, erhalten Sie möglicherweise einen Fehler wie den folgenden:

```
Resource governor for 'prepared statements' exceeded
```

Um Handle-Leaks zu erkennen, begrenzt der SQL Anywhere-Datenbankserver standardmäßig die Anzahl von Cursors und vorbereiteten Anweisungen auf maximal 50 pro Verbindung. Der Ressourcenwächter generiert automatisch einen Fehler, wenn diese Grenzwerte überschritten werden. Wenn dieser Fehler auftritt, suchen Sie nach nicht entfernten Anweisungshandles. Verwenden Sie `prepare_cached` mit Bedacht, da die Anweisungshandles nicht entfernt werden.

Erforderlichenfalls können Sie diese Grenzwerte ändern, indem Sie die Optionen `max_cursor_count` und `max_statement_count` definieren.

Zeilen einfügen

Zum Einfügen von Zeilen wird ein Handle für eine offene Verbindung benötigt. Die einfachste Methode, Zeilen einzufügen, besteht darin, eine parametrisierte INSERT-Anweisung zu verwenden, in der Fragezeichen als Platzhalter für Werte eingesetzt werden. Die Anweisung wird erst vorbereitet und dann pro neuer Zeile ausgeführt. Die neuen Zeilenwerte werden als Parameter an die `execute`-Methode übergeben.

```
require 'dbi'

def db_query( dbh, sql )
  sth = dbh.execute(sql)
  print "# of Fields:  #{sth.column_names.size}\n"
  sth.fetch do |row|
    print "\n"
    sth.column_info.each_with_index do |info, i|
      unless info["type_name"] == "LONG VARBINARY"
        print "#{info["name"]}#{row[i]}\n"
      end
    end
  end
  sth.finish
end

def db_insert( dbh, rows )
  sql = "INSERT INTO Customers (ID, GivenName, Surname,
    Street, City, State, Country, PostalCode,
    Phone, CompanyName)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
  sth = dbh.prepare(sql);
  rows.each do |row|
    sth.execute(row[0],row[1],row[2],row[3],row[4],
      row[5],row[6],row[7],row[8],row[9])
  end
end
```

```

begin
  dbh = DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql')
  rows = [
    [801, 'Alex', 'Alt', '5 Blue Ave', 'New York', 'NY', 'USA',
     '10012', '5185553434', 'BXM'],
    [802, 'Zach', 'Zed', '82 Fair St', 'New York', 'NY', 'USA',
     '10033', '5185552234', 'Zap']
  ]
  db_insert(dbh, rows)
  dbh.commit
  db_query(dbh, "SELECT * FROM Customers WHERE ID > 800")
rescue DBI::DatabaseError => e
  puts "An error occurred"
  puts "Error code: #{e.err}"
  puts "Error message: #{e.errstr}"
  puts "Error SQLSTATE: #{e.state}"
ensure
  dbh.disconnect if dbh
end

```

Siehe auch

- „max_cursor_count-Option“ [\[SQL Anywhere Server - Datenbankadministration\]](#)
- „max_statement_count-Option“ [\[SQL Anywhere Server - Datenbankadministration\]](#)

SQL Anywhere-Ruby-API-Referenz

SQL Anywhere stellt eine Low-Level-Schnittstelle zur SQL Anywhere C-API bereit. Die API, die in den folgenden Abschnitten beschrieben wird, ermöglicht die schnelle Entwicklung von SQL-Anwendungen. Um die Mächtigkeit der Ruby-Anwendungsentwicklung zu veranschaulichen, betrachten Sie das folgende Ruby-Beispielprogramm. Es lädt die SQL Anywhere-Ruby-Erweiterung, stellt eine Verbindung zur Beispieldatenbank her, listet die Spaltennamen aus der Products-Tabelle auf, trennt die Verbindung und beendet.

```

begin
  require 'rubygems'
  gem 'sqlanywhere'
  unless defined? SQLAnywhere
    require 'sqlanywhere'
  end
end

api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
conn = api.sqlany_new_connection()
api.sqlany_connect( conn, "DSN=SQL Anywhere 16 Demo" )
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Products" )
num_rows = api.sqlany_num_rows( stmt )
num_rows.times {
  api.sqlany_fetch_next( stmt )
  num_cols = api.sqlany_num_cols( stmt )
  for col in 1..num_cols do
    info = api.sqlany_get_column_info( stmt, col - 1 )
    unless info[3]==1 # Don't do binary
      rc, value = api.sqlany_get_column( stmt, col - 1 )
      print "#{info[2]}=#{value}\n"
    end
  end
end
end

```

```
        print "\n"
    }
    api.sqlany_free_stmt( stmt )
    api.sqlany_disconnect(conn)
    api.sqlany_free_connection(conn)
    api.sqlany_fini()
    SQLAnywhere::API.sqlany_finalize_interface( api )
```

Die ersten zwei Zeilen der Ergebnismengenausgabe dieses Ruby-Programms werden unten angezeigt.

```
ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00
```

```
ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

Die folgenden Abschnitte beschreiben die einzelnen unterstützten Funktionen.

sqlany_affected_rows

Gibt die Anzahl der Zeilen zurück, die von der Ausführung der vorbereiteten Anweisung betroffen sind.

Syntax

```
sqlany_affected_rows ( $stmt )
```

Parameter

- **\$stmt** Eine Anweisung, die vorbereitet war und erfolgreich ausgeführt wurde, und in der keine Ergebnismenge zurückgegeben wurde. Beispiel: Eine INSERT-, UPDATE- oder DELETE-Anweisung wurde ausgeführt.

Rückgabe

Gibt einen skalaren Wert zurück, der die Anzahl der betroffenen Zeilen angibt; oder -1 bei einem Fehler.

Siehe auch

- [„sqlany_execute-Funktion“](#) auf Seite 731

Beispiel

```
affected = api.sqlany_affected( stmt )
```

sqlany_bind_param-Funktion

Bindet einen benutzerdefinierten Puffer als Parameter an die vorbereitete Anweisung.

Syntax

```
sqlany_bind_param ( $stmt, $index, $param )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch die erfolgreiche Ausführung von `sqlany_prepare` zurückgegeben wird.
- **\$index** Der Index des Parameters. Die Zahl muss zwischen 0 und `sqlany_num_params()` - 1 liegen.
- **\$param** Ein gefülltes Bindungsobjekt, das von `sqlany_describe_bind_param` abgerufen wird.

Rückgabe

Gibt einen skalaren Wert zurück: bei Erfolg 1, sonst 0.

Siehe auch

- „[sqlany_describe_bind_param-Funktion](#)“ auf Seite 729

Beispiel

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_clear_error-Funktion

Löscht den letzten gespeicherten Fehlercode.

Syntax

```
sqlany_clear_error ( $conn )
```

Parameter

- **\$conn** Ein Verbindungsobjekt, das von `sqlany_new_connection` zurückgegeben wird.

Rückgabe

Gibt NIL zurück.

Siehe auch

- „[sqlany_new_connection-Funktion](#)“ auf Seite 739

Beispiel

```
api.sqlany_clear_error( conn )
```

sqlany_client_version-Funktion

Gibt die aktuelle Clientversion zurück.

Syntax

```
sqlany_client_version ( )
```

Rückgabe

Gibt einen skalaren Wert zurück, der die Clientversion-Zeichenfolge angibt.

Beispiel

```
buffer = api.sqlany_client_version()
```

sqlany_commit-Funktion

Schreibt die aktuelle Transaktion fest.

Syntax

```
sqlany_commit ( $conn )
```

Parameter

- **\$conn** Das Verbindungsobjekt, auf dem der Festschreibungsvorgang durchgeführt werden soll.

Rückgabe

Gibt einen skalaren Wert zurück: bei Erfolg 1, sonst 0.

Siehe auch

- „[sqlany_rollback-Funktion](#)“ auf Seite 743

Beispiel

```
rc = api.sqlany_commit( conn )
```

sqlany_connect-Funktion

Erstellt eine Verbindung zu einem SQL Anywhere-Datenbankserver, indem das angegebene Verbindungsobjekt und die Verbindungszeichenfolge verwendet werden.

Syntax

```
sqlany_connect ( $conn, $str )
```

Parameter

- **\$conn** Das Verbindungsobjekt, das durch `sqlany_new_connection` erstellt wurde.
- **\$str** Eine SQL Anywhere-Verbindungszeichenfolge.

Rückgabe

Gibt einen skalaren Wert zurück, der 1 ist, wenn die Verbindung erfolgreich hergestellt wurde, oder 0, wenn dies nicht der Fall war. Verwenden Sie `sqlany_error`, um den Fehlercode und die Fehlermeldung abzurufen.

Siehe auch

- „[sqlany_new_connection-Funktion](#)“ auf Seite 739
- „[sqlany_error-Funktion](#)“ auf Seite 730
- „[Verbindungsparameter](#)“ [*SQL Anywhere Server - Datenbankadministration*]
- „[Datenbankverbindungen](#)“ [*SQL Anywhere Server - Datenbankadministration*]

Beispiel

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Connection status = #{status}\n"
```

sqlany_describe_bind_param-Funktion

Beschreibt die Bindungsparameter einer vorbereiteten Anweisung.

Syntax

```
sqlany_describe_bind_param ( $stmt, $index )
```

Parameter

- **\$stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare` vorbereitet wurde.
- **\$index** Der Index des Parameters. Die Zahl muss zwischen 0 und `sqlany_num_params()` - 1 liegen.

Rückgabe

Gibt ein 2-Elemente-Array zurück, das 1 bei Erfolg und sonst 0 als erstes Element und einen beschriebenen Parameter als zweites Element enthält.

Bemerkungen

Diese Funktion ermöglicht es dem Aufrufer, Informationen über vorbereitete Anweisungsparameter festzulegen. Der Typ der vorbereiteten Anweisung (gespeicherte Prozedur oder DML) bestimmt die Menge der bereitgestellten Informationen. Die Richtung der Parameter (Eingabe, Ausgabe oder Eingabe/Ausgabe) wird immer übergeben.

Siehe auch

- „[sqlany_bind_param-Funktion](#)“ auf Seite 726
- „[sqlany_prepare-Funktion](#)“ auf Seite 742

Beispiel

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_disconnect-Funktion

Trennt eine SQL Anywhere-Verbindung. Alle nicht festgeschriebenen Transaktionen werden zurückgesetzt.

Syntax

```
sqlany_disconnect ( $conn )
```

Parameter

- **\$conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect` hergestellten Verbindung.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 ist und sonst 0.

Siehe auch

- „[sqlany_connect-Funktion](#)“ auf Seite 728
- „[sqlany_new_connection-Funktion](#)“ auf Seite 739

Beispiel

```
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}\n"
```

sqlany_error-Funktion

Gibt den letzten Fehlercode und -meldung zurück, die im Verbindungsobjekt gespeichert sind.

Syntax

```
sqlany_error ( $conn )
```

Parameter

- **\$conn** Ein Verbindungsobjekt, das von `sqlany_new_connection` zurückgegeben wird.

Rückgabe

Gibt ein 2-Elemente-Array zurück, das den SQL-Fehlercode als erstes Element und eine Fehlermeldung-Zeichenfolge als zweites Element enthält.

Beim Fehlercode sind positive Werte Warnungen und negative Werte Fehler, während 0 Erfolg bedeutet.

Siehe auch

- „[sqlany_connect-Funktion](#)“ auf Seite 728
- „[SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE](#)“ [[Fehlermeldungen](#)]

Beispiel

```
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

sqlany_execute-Funktion

Führt eine vorbereitete Anweisung aus.

Syntax

```
sqlany_execute ( $stmt )
```

Parameter

- **\$stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare` vorbereitet wurde.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 ist und sonst 0.

Bemerkungen

Sie können `sqlany_num_cols` verwenden, um zu überprüfen, ob die Anweisung eine Ergebnismenge zurückgegeben hat.

Siehe auch

- „[sqlany_prepare-Funktion](#)“ auf Seite 742

Beispiel

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_execute_direct-Funktion

Führt die SQL-Anweisung aus, die durch das Zeichenfolgenargument angegeben ist.

Syntax

```
sqlany_execute_direct ( $conn, $sql )
```

Parameter

- **\$conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect` hergestellten Verbindung.
- **\$sql** Eine SQL-Zeichenfolge. Die SQL-Zeichenfolge sollte keine Parameter wie z.B. `?` enthalten.

Rückgabe

Gibt ein Anweisungsobjekt oder NIL bei Fehler zurück.

Bemerkungen

Verwenden Sie diese Funktion, wenn Sie eine Anweisung in einem Schritt vorbereiten und ausführen möchten. Verwenden Sie diese Funktion nicht, um eine SQL-Anweisung mit Parametern auszuführen.

Siehe auch

- „`sqlany_fetch_absolute`-Funktion“ auf Seite 733
- „`sqlany_fetch_next`-Funktion“ auf Seite 733
- „`sqlany_num_cols`-Funktion“ auf Seite 740
- „`sqlany_get_column`-Funktion“ auf Seite 736

Beispiel

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname    = api.sqlany_get_column( stmt, 2 )
rc, givenName  = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_execute_immediate-Funktion

Führt die angegebene SQL-Anweisung unmittelbar aus, ohne eine Ergebnismenge zurückzugeben. Dies ist nützlich bei Anweisungen, die keine Ergebnismengen zurückgeben.

Syntax

```
sqlany_execute_immediate ( $conn, $sql )
```

Parameter

- **\$conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect` hergestellten Verbindung.
- **\$sql** Eine SQL-Zeichenfolge. Die SQL-Zeichenfolge sollte keine Parameter wie z.B. `?` enthalten.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 ist und sonst 0.

Siehe auch

- „[sqlany_error-Funktion](#)“ auf Seite 730

Beispiel

```
rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= 50" )
```

sqlany_fetch_absolute-Funktion

Verschiebt die aktuelle Zeile in der Ergebnismenge an die angegebene Zeilennummer und ruft anschließend die Daten in dieser Zeile ab.

Syntax

```
sqlany_fetch_absolute ( $stmt, $row_num )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.
- **\$row_num** Die abzurufende Zeilennummer. Die erste Zeile ist 1, die letzte Zeile ist -1.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 ist und sonst 0.

Siehe auch

- „[sqlany_error-Funktion](#)“ auf Seite 730
- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731
- „[sqlany_fetch_next-Funktion](#)“ auf Seite 733

Beispiel

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_absolute( stmt, 2 )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_fetch_next-Funktion

Gibt die nächste Zeile in der Ergebnismenge zurück. Diese Funktion verschiebt zuerst den Zeilenzeiger nach vorne und ruft anschließend die Daten in der neuen Zeile ab.

Syntax

```
sqlany_fetch_next ( $stmt )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 ist und sonst 0.

Siehe auch

- [„sqlany_error-Funktion“ auf Seite 730](#)
- [„sqlany_execute-Funktion“ auf Seite 731](#)
- [„sqlany_execute_direct-Funktion“ auf Seite 731](#)
- [„sqlany_fetch_absolute-Funktion“ auf Seite 733](#)

Beispiel

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname    = api.sqlany_get_column( stmt, 2 )
rc, givenName  = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_fini-Funktion

Gibt durch die API zugewiesene Ressourcen frei.

Syntax

```
sqlany_fini ( )
```

Rückgabe

Gibt NIL zurück.

Siehe auch

- [„sqlany_init-Funktion“ auf Seite 739](#)

Beispiel

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
```

```
api.sqlany_fini()  
  
# Close the interface  
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_connection-Funktion

Gibt die Ressourcen frei, die mit einem Verbindungsobjekt verbunden sind.

Syntax

```
sqlany_free_connection ( $conn )
```

Parameter

- **\$conn** Ein Verbindungsobjekt, das durch `sqlany_new_connection` erstellt wurde.

Rückgabe

Gibt NIL zurück.

Siehe auch

- „[sqlany_new_connection-Funktion](#)“ auf Seite 739

Beispiel

```
# Disconnect from the database  
api.sqlany_disconnect( conn )  
  
# Free the connection resources  
api.sqlany_free_connection( conn )  
  
# Free resources the api object uses  
api.sqlany_fini()  
  
# Close the interface  
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_stmt-Funktion

Gibt Ressourcen frei, die mit einem Anweisungsobjekt verbunden sind.

Syntax

```
sqlany_free_stmt ( $stmt )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch die erfolgreiche Ausführung von `sqlany_prepare` oder `sqlany_execute_direct` zurückgegeben wird.

Rückgabe

Gibt NIL zurück.

Siehe auch

- „[sqlany_prepare-Funktion](#)“ auf Seite 742
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731

Beispiel

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
rc = api.sqlany_free_stmt( stmt )
```

sqlany_get_bind_param_info-Funktion

Ruft Informationen über die Parameter ab, die mit `sqlany_bind_param` gebunden wurden.

Syntax

```
sqlany_get_bind_param_info ( $stmt, $index )
```

Parameter

- **\$stmt** Eine Anweisung, die erfolgreich mit `sqlany_prepare` vorbereitet wurde.
- **\$index** Der Index des Parameters. Die Zahl muss zwischen 0 und `sqlany_num_params()` - 1 liegen.

Rückgabe

Gibt ein 2-Elemente-Array zurück, das 1 bei Erfolg und sonst 0 als erstes Element und einen beschriebenen Parameter als zweites Element enthält.

Siehe auch

- „[sqlany_bind_param-Funktion](#)“ auf Seite 726
- „[sqlany_describe_bind_param-Funktion](#)“ auf Seite 729
- „[sqlany_prepare-Funktion](#)“ auf Seite 742

Beispiel

```
# Get information on first parameter (0)
rc, param_info = api.sqlany_get_bind_param_info( stmt, 0 )
print "Param_info direction = ", param_info.get_direction(), "\n"
print "Param_info output = ", param_info.get_output(), "\n"
```

sqlany_get_column-Funktion

Gibt den Wert zurück, der aus der angegebenen Spalte abgerufen wurde.

Syntax

```
sqlany_get_column ( $stmt, $col_index )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.
- **\$col_index** Die Nummer der abzurufenden Spalte. Eine Spaltennummer zwischen 0 und `sqlany_num_cols() - 1`.

Rückgabe

Gibt ein 2-Elemente-Array zurück, das 1 bei Erfolg und sonst 0 als erstes Element und den Spaltenwert als zweites Element enthält.

Siehe auch

- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731
- „[sqlany_fetch_absolute-Funktion](#)“ auf Seite 733
- „[sqlany_fetch_next-Funktion](#)“ auf Seite 733

Beispiel

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname    = api.sqlany_get_column( stmt, 2 )
rc, givenName  = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_get_column_info-Funktion

Ruft die Spalteninformationen für die angegebene Ergebnismengenspalte ab.

Syntax

```
sqlany_get_column_info ( $stmt, $col_index )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.
- **\$col_index** Die Spaltennummer zwischen 0 und `(sqlany_num_cols() - 1)`.

Rückgabe

Gibt ein 9-Elemente-Array von Informationen zurück, das eine Spalte in der Ergebnismenge beschreibt. Das erste Element enthält 1 bei Erfolg und 0 bei Fehler. Die Array-Elemente werden in der folgenden Tabelle beschrieben.

Elementnummer	Typ	Beschreibung
0	Ganzzahl	1 bei Erfolg oder 0 bei Fehlschlag.
1	Ganzzahl	Spaltenindex (0 bis <code>sqlany_num_cols()</code> - 1).
2	Zeichenfolge	Spaltenname.
3	Ganzzahl	Spaltentyp. Siehe „ Spaltentypen “ auf Seite 744.
4	Ganzzahl	Nativer Spaltentyp. Siehe „ Native Spaltentypen “ auf Seite 744.
5	Ganzzahl	Gesamtstellenzahl der Spalte (bei numerischen Typen)
6	Ganzzahl	Dezimalstellen der Spalte (bei numerischen Typen)
7	Ganzzahl	Spaltengröße
8	Ganzzahl	Nullwertfähigkeit der Spalte (1=nullwertfähig, 0=nicht nullwertfähig)

Siehe auch

- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731
- „[sqlany_prepare-Funktion](#)“ auf Seite 742

Beispiel

```
# Get column info for first column (0)
rc, col_num, col_name, col_type, col_native_type, col_precision, col_scale,
  col_size, col_nullable = api.sqlany_get_column_info( stmt, 0 )
```

sqlany_get_next_result-Funktion

Geht bei einer Abfrage mit mehreren Ergebnismengen zur nächsten Ergebnismenge.

Syntax

```
sqlany_get_next_result ( $stmt )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 ist und sonst 0.

Siehe auch

- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731

Beispiel

```
stmt = api.sqlany_prepare(conn, "call two_results()" )
rc = api.sqlany_execute( stmt )
# Fetch from first result set
rc = api.sqlany_fetch_absolute( stmt, 3 )
# Go to next result set
rc = api.sqlany_get_next_result( stmt )
# Fetch from second result set
rc = api.sqlany_fetch_absolute( stmt, 2 )
```

sqlany_init-Funktion

Initialisiert die Schnittstelle.

Syntax

```
sqlany_init ( )
```

Rückgabe

Gibt ein 2-Elemente-Array zurück, das bei Erfolg 1 oder bei Fehler 0 als erstes Element und die Ruby-Schnittstellenversion als zweites Element enthält.

Siehe auch

- „[sqlany_fini-Funktion](#)“ auf Seite 734

Beispiel

```
# Load the SQLAnywhere gem
begin
  require 'rubygems'
  gem 'sqlanywhere'
  unless defined? SQLAnywhere
    require 'sqlanywhere'
  end
end

# Create an interface
api = SQLAnywhere::SQLAnywhereInterface.new()
# Initialize the interface (loads the DLL/SO)
SQLAnywhere::API.sqlany_initialize_interface( api )
# Initialize our api object
api.sqlany_init()
```

sqlany_new_connection-Funktion

Erstellt ein Verbindungsobjekt.

Syntax

```
sqlany_new_connection ( )
```

Rückgabe

Gibt einen skalaren Wert zurück, der ein Verbindungsobjekt ist.

Bemerkungen

Ein Verbindungsobjekt muss erstellt werden, bevor eine Datenbankverbindung hergestellt wird. Fehler können aus dem Verbindungsobjekt abgerufen werden. Nur jeweils eine Anforderung kann auf einer Verbindung abgearbeitet werden.

Siehe auch

- „[sqlany_connect-Funktion](#)“ auf Seite 728
- „[sqlany_disconnect-Funktion](#)“ auf Seite 730

Beispiel

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Status=#{status}\n"
```

sqlany_num_cols-Funktion

Gibt die Anzahl der Spalten in der Ergebnismenge zurück.

Syntax

```
sqlany_num_cols ( $stmt )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.

Rückgabe

Gibt einen skalaren Wert zurück, der die Anzahl der Spalten in der Ergebnismenge ist, oder -1 bei Fehler.

Siehe auch

- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731
- „[sqlany_prepare-Funktion](#)“ auf Seite 742

Beispiel

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of result set columns
num_cols = api.sqlany_num_cols( stmt )
```

sqlany_num_params-Funktion

Gibt die Anzahl der Parameter zurück, die bei einer vorbereiteten Anweisung erwartet werden.

Syntax

```
sqlany_num_params ( $stmt )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch die erfolgreiche Ausführung von `sqlany_prepare` zurückgegeben wird.

Rückgabe

Gibt einen skalaren Wert zurück, der die Anzahl der Parameter in einer vorbereiteten Anweisung darstellt, oder -1 bei Fehler.

Siehe auch

- „[sqlany_prepare-Funktion](#)“ auf Seite 742

Beispiel

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts  
    SET Contacts.ID = Contacts.ID + 1000  
    WHERE Contacts.ID >= ?" )  
num_params = api.sqlany_num_params( stmt )
```

sqlany_num_rows-Funktion

Gibt die Anzahl der Zeilen in der Ergebnismenge zurück.

Syntax

```
sqlany_num_rows ( $stmt )
```

Parameter

- **\$stmt** Ein Anweisungsobjekt, das durch `sqlany_execute` oder `sqlany_execute_direct` ausgeführt wurde.

Rückgabe

Gibt einen skalaren Wert zurück, der die Anzahl der Zeilen in der Ergebnismenge darstellt. Wenn die Anzahl der Zeilen eine Schätzung ist, ist die zurückgegebene Anzahl negativ und die Schätzung der absolute Wert der zurückgegebenen Ganzzahl. Der zurückgegebene Wert ist positiv, wenn die Anzahl der Zeilen exakt ist.

Bemerkungen

Standardmäßig gibt diese Funktion nur eine Schätzung zurück. Um eine exakte Zählung zurückzugeben, setzen Sie die `ROW_COUNTS`-Option bei der Verbindung. Weitere Hinweise finden Sie unter „[row_counts-Option](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

Eine Zählung der Zeilenanzahl in einer Ergebnismenge kann bei einer Anweisung, die mehrere Ergebnismengen zurückgibt, nur für die erste Ergebnismenge zurückgegeben werden. Wenn `sqlany_get_next_result` verwendet wird, um zur nächsten Ergebnismenge zu gehen, gibt `sqlany_num_rows` dennoch nur die Zeilenanzahl in der ersten Ergebnismenge zurück.

Siehe auch

- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_execute_direct-Funktion](#)“ auf Seite 731

Beispiel

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of rows in result set
num_rows = api.sqlany_num_rows( stmt )
```

sqlany_prepare-Funktion

Bereitet die angegebene SQL-Zeichenfolge vor.

Syntax

```
sqlany_prepare ( $conn, $sql )
```

Parameter

- **\$conn** Ein Verbindungsobjekt mit einer unter Verwendung von `sqlany_connect` hergestellten Verbindung.
- **\$sql** Die vorzubereitende SQL-Anweisung..

Rückgabe

Gibt einen skalaren Wert zurück, der das Anweisungsobjekt ist, oder NIL bei Fehler.

Bemerkungen

Die Anweisung, die mit dem Anweisungsobjekt verbunden ist, wird durch `sqlany_execute` ausgeführt. Sie können `sqlany_free_stmt` verwenden, um die Ressourcen freizugeben, die mit dem Anweisungsobjekt verbunden sind.

Siehe auch

- „[sqlany_execute-Funktion](#)“ auf Seite 731
- „[sqlany_free_stmt-Funktion](#)“ auf Seite 735
- „[sqlany_num_params-Funktion](#)“ auf Seite 741
- „[sqlany_describe_bind_param-Funktion](#)“ auf Seite 729
- „[sqlany_bind_param-Funktion](#)“ auf Seite 726

Beispiel

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
```

```
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_rollback-Funktion

Setzt die aktuelle Transaktion zurück.

Syntax

```
sqlany_rollback ( $conn )
```

Parameter

- **\$conn** Das Verbindungsobjekt, auf dem der Rollback-Vorgang durchgeführt werden soll.

Rückgabe

Gibt einen skalaren Wert zurück, der bei Erfolg 1 und bei Fehler 0 ist.

Siehe auch

- „[sqlany_commit-Funktion](#)“ auf Seite 728

Beispiel

```
rc = api.sqlany_rollback( conn )
```

sqlany_sqlstate-Funktion

Ruft den aktuellen SQLSTATE ab.

Syntax

```
sqlany_sqlstate ( $conn )
```

Parameter

- **\$conn** Ein Verbindungsobjekt, das von `sqlany_new_connection` zurückgegeben wird.

Rückgabe

Gibt einen skalaren Wert zurück, der den aktuellen fünfstelligen SQLSTATE darstellt.

Siehe auch

- „[sqlany_error-Funktion](#)“ auf Seite 730
- „[SQL Anywhere-Fehlermeldungen - sortiert nach SQLSTATE](#)“ [[Fehlermeldungen](#)]

Beispiel

```
sql_state = api.sqlany_sqlstate( conn )
```

Spaltentypen

Die folgende Ruby-Klasse legt die Spaltentypen fest, die von einigen SQL Anywhere Ruby-Funktionen zurückgegeben werden.

```
class Types
  A_INVALID_TYPE = 0
  A_BINARY       = 1
  A_STRING       = 2
  A_DOUBLE       = 3
  A_VAL64        = 4
  A_UVAL64       = 5
  A_VAL32        = 6
  A_UVAL32       = 7
  A_VAL16        = 8
  A_UVAL16       = 9
  A_VAL8         = 10
  A_UVAL8        = 11
end
```

Native Spaltentypen

Die folgende Tabelle legt die nativen Spaltentypen fest, die von einigen SQL Anywhere-Funktionen zurückgegeben werden.

Wert des nativen Typs	Nativer Typ
384	DT_DATE
388	DT_TIME
390	DT_TIMESTAMP_STRUCT
392	DT_TIMESTAMP
448	DT_VARCHAR
452	DT_FIXCHAR
456	DT_LONGVARCHAR
460	DT_STRING
480	DT_DOUBLE
482	DT_FLOAT
484	DT_DECIMAL
496	DT_INT
500	DT_SMALLINT

Wert des nativen Typs	Nativer Typ
524	DT_BINARY
528	DT_LONGBINARY
600	DT_VARIABLE
604	DT_TINYINT
608	DT_BIGINT
612	DT_UNSINT
616	DT_UNSSMALLINT
620	DT_UNSBIGINT
624	DT_BIT
628	DT_NSTRING
632	DT_NFIXCHAR
636	DT_NVARCHAR
640	DT_LONGNVARCHAR

Sybase Open Client-Unterstützung

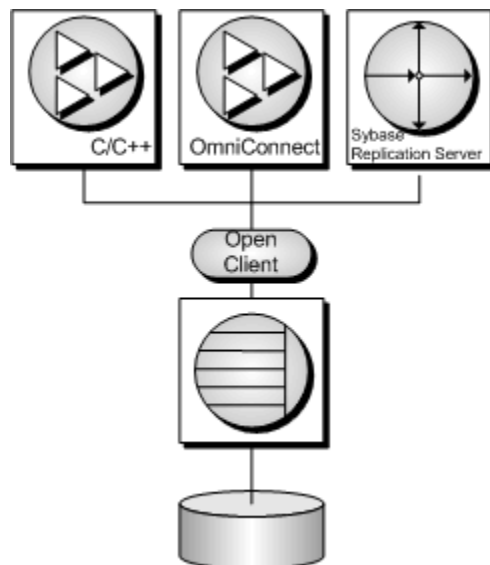
Sybase Open Client bietet Kundenanwendungen, Produkten von Drittprovidern und anderen Sybase-Produkten die erforderlichen Schnittstellen zur Kommunikation mit SQL Anywhere und anderen Open Servern.

Einsatzbereich des Open Clients

Der Einsatz der Open Client-Schnittstelle kommt in Betracht, wenn Sie Kompatibilität mit Adaptive Server Enterprise ermöglichen möchten oder wenn Sie andere Sybase-Produkte verwenden, die die Open Client-Schnittstelle unterstützen, zum Beispiel Replication Server.

Open Client-Anwendungen

Sie können Anwendungen in C oder C++ entwickeln und dann mithilfe der Open Client-API eine Verbindung mit SQL Anywhere herstellen. Andere Sybase-Anwendungen, z.B. OmniConnect, verwenden ebenfalls Open Client. Die Open Client-API wird auch von Sybase Adaptive Server Enterprise unterstützt.



Siehe auch

- „SQL Anywhere als Open Server“ [[SQL Anywhere Server - Datenbankadministration](#)]

Die Open Client-Architektur

Hinweis

In diesem Abschnitt wird die Sybase Open Client-Programmierschnittstelle für SQL Anywhere beschrieben. Die primäre Quelle für Informationen zur Sybase Open Client-Anwendungsentwicklung ist die Sybase Open Client-Dokumentation. In diesem Abschnitt werden spezifische Funktionen für SQL Anywhere beschrieben; er ist jedoch keine umfassende Anleitung für die Anwendungsprogrammierung mit Sybase Open Client.

Sybase Open Client besitzt zwei Komponenten, nämlich die Programmierschnittstellen und die Netzwerkdienste.

DB-Library und Client-Library

Sybase Open Client bietet zwei Kern-Programmierschnittstellen für das Schreiben von Clientanwendungen, nämlich die DB-Library und die Client-Library.

Die Open Client DB-Library bietet Unterstützung für ältere Open Client-Anwendungen und ist eine von der Client-Library vollständig getrennte Programmierschnittstelle. Die DB-Library ist dokumentiert im *Open Client Client-Library/C Reference Manual* (in englischer Sprache), das zum Lieferumfang des Produkts Sybase Open Client gehört.

Client-Library-Programme sind auch von der CS-Library abhängig, die Routinen sowohl für den Einsatz in Client-Library- als auch Server-Library-Anwendungen bereitstellt. Client-Library-Anwendungen können auch Routinen der Bulk-Library benutzen, um die schnelle Datenübertragung zu unterstützen.

Die CS-Library und die Bulk-Library sind in Sybase Open Client enthalten. Dieses Produkt kann separat erworben werden.

Netzwerkdienste

Open Client-Netzwerkdienste enthalten die Sybase Net-Library, die die Unterstützung für bestimmte Netzwerkprotokolle wie TCP/IP und DECnet bereithält. Die Net-Library-Schnittstelle ist für Anwendungsentwickler unsichtbar. Auf manchen Plattformen benötigt eine Anwendung unter Umständen einen anderen Net-Library-Treiber für unterschiedliche System-Netzwerkkonfigurationen. Je nach Ihrer Hostplattform wird der Net-Library-Treiber entweder durch die Konfiguration der Sybase-Produkte des Systems bestimmt, oder durch die Kompilierung und Verknüpfung Ihrer Programme.

Hinweise zur Treiberkonfiguration finden Sie in der Dokumentation *Open Client/Server Configuration Guide*.

Hinweise zum Aufbau von Client-Library-Programmen finden Sie in der Dokumentation *Open Client/Server Programmer's Supplement*.

Was Sie für die Entwicklung von Open Client-Anwendungen brauchen

Um Open Client-Anwendungen auszuführen, müssen Sie Sybase Open Client-Komponenten auf dem Computer, auf dem die Anwendung laufen soll, installieren und konfigurieren. Diese Komponenten sind eventuell bereits als Teil Ihrer Installation anderer Sybase Produkte vorhanden; andernfalls können Sie diese Bibliotheken wahlweise mit SQL Anywhere entsprechend den Bedingungen Ihres Lizenzvertrags installieren.

Open Client-Anwendungen benötigen keine Open Client-Komponenten auf dem Computer, auf dem der Datenbankserver ausgeführt wird.

Um Open Client-Anwendungen zu erstellen, benötigen Sie die Entwicklungsversion von Open Client. Diese ist bei Sybase erhältlich.

Standardmäßig werden SQL Anywhere-Datenbanken ohne Berücksichtigung von Groß- und Kleinschreibung erstellt, während SQL Enterprise-Datenbanken unter Berücksichtigung von Groß- und Kleinschreibung erstellt werden.

Siehe auch

- „SQL Anywhere als Open Server“ [[SQL Anywhere Server - Datenbankadministration](#)]

Open Client-Datentypzuordnungen

Open Client hat seine eigenen internen Datentypen, die in einigen Details von den Datentypen abweichen, die in SQL Anywhere zur Verfügung stehen. Aus diesem Grunde ordnet SQL Anywhere intern einige Datentypen von Open Client-Anwendungen den in SQL Anywhere zur Verfügung stehenden Datentypen zu.

Um Open Client-Anwendungen zu erstellen, benötigen Sie die Entwicklungsversion von Open Client. Damit Open Client-Anwendungen verwendet werden können, müssen die Open Client-Laufzeitprogramme auf dem Computer, auf dem die Anwendung laufen soll, installiert und konfiguriert sein.

Der SQL Anywhere-Server erfordert keine externe Kommunikations-Laufzeitumgebung für die Unterstützung von Open Client-Anwendungen.

Jeder Open Client-Datentyp wird dem entsprechenden SQL Anywhere-Datentyp zugeordnet. Es werden alle Open Client-Datentypen unterstützt.

SQL Anywhere-Datentypen ohne direkte Entsprechung in Open Client

Die folgende Tabelle zeigt die Zuordnung von Datentypen, die in SQL Anywhere unterstützt werden, und die keine direkte Entsprechung in Open Client haben.

SQL Anywhere-Datentyp	Open Client-Datentyp
unsigned short	int
unsigned int	bigint
unsigned bigint	numeric(20,0)
string	varchar
timestamp	datetime

Bereichseinschränkungen bei der Zuordnung von Open Client-Datentypen

Einige Datentypen haben in SQL Anywhere andere Wertebereiche als in Open Client. In solchen Fällen kann es zu Überlauf Fehlern während der Abfrage oder des Einfügens von Daten kommen.

Die folgende Tabelle zeigt Open Client-Anwendungsdatentypen, die zwar SQL Anywhere-Datentypen zugeordnet werden können, aber nur mit einigen Einschränkungen im Wertebereich möglicher Werte.

Normalerweise wird der Open Client-Datentyp einem SQL Anywhere-Datentyp zugeordnet, der einen größeren Bereich möglicher Werte hat. Deshalb ist es möglich, einen Wert an SQL Anywhere zu übergeben, der zwar akzeptiert und in einer Datenbank gespeichert wird, der aber zu groß ist, um von einer Open Client-Anwendung abgerufen zu werden.

Datentyp	Untere Open Client-Grenze	Obere Open Client-Grenze	Untere SQL Anywhere-Grenze	Obere SQL Anywhere-Grenze
MONEY	-922 377 203 685 477.5808	922 377 203 685 477.5807	-999 999 999 999 999,9999	999 999 999 999 999,9999
SMALLMONEY	-214 748.3648	214 748.3647	-999 999,9999	-999 999,9999
DATETIME [1]	1. Januar 1753	31. Dezember 9999	1. Januar 0001	31. Dezember 9999
SMALLDATETIME	1. Januar 1900	6. Juni 2079	1. Januar 0001	31. Dezember 9999

[1] Für frühere Versionen als OpenClient 15.5; sonst wird der gesamten Bereich der Datumsangaben von 0001-01-01 bis 9999-12-31 unterstützt.

Beispiel

Die Open Client-Datentypen MONEY und SMALLMONEY umfassen zum Beispiel nicht den gesamten numerischen Bereich der zugrunde liegenden SQL Anywhere-Implementierung. Daher kann in einer

Spalte in SQL Anywhere ein Wert enthalten sein, der die Grenzwerte des Open Client-Datentyps MONEY überschreitet. Ruft der Client einen solchen unzulässigen Wert mit SQL Anywhere ab, wird eine Fehlermeldung erzeugt.

Zeitstempel

Bei Zeitstempelwerten, die in SQL Anywhere eingefügt oder daraus abgefragt werden, ist der Datumsteil auf Datumsangaben ab dem 1. Januar 1753 beschränkt und die Zeitversion auf eine Genauigkeit von 1/300 Sekunde, wenn der Client die Version Open Client 15.1 oder früher verwendet. Wenn jedoch der Client die Version Open Client 15.5 oder später verwendet wird, gilt keine Einschränkung für die Zeitstempelwerte.

SQL in Open Client-Anwendungen

Dieser Abschnitt bietet eine kurze Einführung in die Verwendung von SQL in Open Client-Anwendungen, mit einem Schwerpunkt auf Themen, die für SQL Anywhere spezifisch sind.

Eine vollständige Beschreibung finden Sie in der Open Client-Dokumentation unter <http://www.sybase.com/products/databasemanagement/openserver>.

Siehe auch

- „Anwendungsentwicklung mit SQL“ auf Seite 1

Ausführung von Open Client-SQL-Anweisungen

Sie senden SQL-Anweisungen an einen Datenbankserver, indem Sie sie in Client Library-Funktionsaufrufe einbeziehen. Die folgenden beiden Aufrufe führen zum Beispiel eine DELETE-Anweisung aus:

```
ret = ct_command(cmd, CS_LANG_CMD,
                  "DELETE FROM Employees
                  WHERE EmployeeID=105"
                  CS_NULLTERM,
                  CS_UNUSED);
ret = ct_send(cmd);
```

Vorbereitete Open Client-Anweisungen

Die Funktion `ct_dynamic` dient zur Verwaltung von vorbereiteten Anweisungen. Diese Funktion hat den Parameter *Typ*, der die Aktion beschreibt, die Sie ausführen.

Führen Sie die folgenden Aufgaben aus, um eine vorbereitete Anweisung in Open Client zu verwenden:

1. Bereiten Sie die Anweisung mit der Funktion `ct_dynamic` mit `CS_PREPARE` als *Typ*-Parameter vor.
2. Legen Sie die Anweisungsparameter mit `ct_param` fest.
3. Führen Sie die Anweisung mit `ct_dynamic` mit `CS_EXECUTE` als *Typ*-Parameter aus.

4. Geben Sie die mit der Anweisung verbundenen Ressourcen frei, indem Sie `ct_dynamic` mit `CS_DEALLOC` als *Typ*-Parameter verwenden.

Weitere Hinweise zur Verwendung von vorbereiteten Anweisungen in Open Client finden Sie in der Open Client-Dokumentation.

Cursor-Verwaltung in Open Client

Die Funktion `ct_cursor` wird für die Verwaltung von Cursors verwendet. Diese Funktion hat den Parameter *Typ*, der die Aktion beschreibt, die Sie ausführen.

Unterstützte Cursortypen

Einige der Cursortypen, die SQL Anywhere unterstützt, stehen in der Open Clientschnittstelle nicht zur Verfügung. Sie können über Open Client weder Scroll-Cursor, noch dynamische Scroll-Cursor, noch insensitive Cursor benutzen.

Eindeutigkeit und Aktualisierbarkeit sind zwei Eigenschaften von Cursors. Cursor können eindeutig sein (unabhängig davon, ob sie von der Anwendung benutzt wird oder nicht, hat jede Zeile einen Primärschlüssel oder eine Eindeutigkeitsinformation) oder nicht eindeutig sein. Cursor können schreibgeschützt oder aktualisierbar sein. Ist ein Cursor aktualisierbar und nicht eindeutig, kann die Performance beeinträchtigt werden, da in diesem Fall keine Zeilen vorab abgerufen werden können. Dies ist unabhängig von der `CS_CURSOR_ROWS`-Einstellung.

Schritte bei der Verwendung von Cursor

Im Gegensatz zu anderen Schnittstellen wie Embedded SQL ordnet Open Client einem Cursor eine SQL Anweisung als Zeichenfolge zu. Embedded SQL bereitet zuerst eine Anweisung vor, dann wird der Cursor mithilfe des Statement-Handles deklariert.

Führen Sie die folgenden Aufgaben aus, um Cursor in Open Client zu verwenden:

1. Um einen Cursor in Open Client zu deklarieren, verwenden Sie `ct_cursor` mit `CS_CURSOR_DECLARE` als *Typ*-Parameter.
2. Nachdem Sie einen Cursor deklariert haben, können Sie steuern, wieviele Zeilen vorab vom Client abgerufen werden, und zwar jedesmal, wenn eine Zeile mit `ct_cursor` mit `CS_CURSOR_ROWS` als *Typ*-Parameter vom Server abgerufen wird.

Durch Speichern der vorab abgerufenen Zeilen reduziert sich die Anzahl der Serveraufrufe. Dies verbessert den Gesamtdurchsatz und die Durchlaufzeit. Vorab abgerufene Zeilen werden nicht sofort an die Anwendung übergeben, sondern werden verwendungsbereit in einem clientseitigen Puffer zwischengespeichert.

Die Einstellung der Datenbankoption `prefetch` steuert den Prefetch-Vorgang für andere Schnittstellen. Sie wird von Open Client-Verbindungen nicht beachtet. Die Einstellung `CS_CURSOR_ROWS` wird bei nicht eindeutigen, aktualisierbaren Cursor nicht beachtet.

3. Sie öffnen einen Cursor in Open Client mit `ct_cursor` mit `CS_CURSOR_OPEN` als *Typ*-Parameter.

4. Verwenden Sie `ct_fetch` zum Abrufen einer Zeile in der Anwendung.
5. Um einen Cursor zu schließen, verwenden Sie `ct_cursor` mit `CS_CURSOR_CLOSE`.
6. In Open Client müssen Sie außerdem die Ressourcen freigeben, die an den Cursor gebunden waren. Verwenden Sie dafür `ct_cursor` mit `CS_CURSOR_DEALLOC`. Sie können auch `CS_CURSOR_CLOSE` mit dem zusätzlichen Parameter `CS_DEALLOC` benutzen, um diese Vorgänge in einem Schritt auszuführen.

Zeilenänderung durch einen Cursor in Open Client

Mit Open Client können Sie Zeilen in einem Cursor löschen oder aktualisieren, sofern der Cursor für eine einzelne Tabelle gilt. Der Benutzer muss die Berechtigung zur Aktualisierung der Tabelle haben und der Cursor muss für Aktualisierung markiert sein.

Statt einen Abruf durchzuführen, können Sie die aktuelle Zeile im Cursor mit `ct_cursor` und `CS_CURSOR_DELETE` löschen oder mit `ct_cursor` und `CS_CURSOR_UPDATE` aktualisieren.

Sie können in Open Client-Anwendungen mit einem Cursor keine Zeilen einfügen.

Open Client-Ergebnismengen

Open Client geht mit Ergebnismengen anders um als andere SQL Anywhere-Schnittstellen.

In Embedded SQL und ODBC **beschreiben** Sie eine Abfrage oder eine gespeicherte Prozedur, um die richtige Anzahl und Typen der Variablen zu setzen, die die Ergebnisse aufnehmen sollen. Die Beschreibung wird in der Anweisung selbst gegeben.

In Open Client brauchen Sie eine Anweisung nicht zu beschreiben. stattdessen kann jede Zeile, die vom Server zurückgegeben wird, eine Beschreibung Ihrer Inhalte enthalten. Falls Sie Anweisungen mit `ct_command` und `ct_send` ausführen, können Sie mit der Funktion `ct_results` alle Aspekte der zurückgegebenen Zeilen behandeln.

Falls Sie nicht nach dieser Zeile-für-Zeile-Methode vorgehen möchten, können Sie `ct_dynamic` verwenden, um eine SQL Anweisung vorzubereiten, und `ct_describe`, um die Ergebnismenge zu beschreiben. Dies entspricht mehr dem Beschreiben von SQL-Anweisungen bei anderen Schnittstellen.

Bekannte Open Client-Einschränkungen von SQL Anywhere

Sie können mit der Open Clientschnittstelle eine SQL Anywhere-Datenbank nahezu genau so verwenden, wie eine Adaptive Server Enterprise-Datenbank. Es gibt allerdings einige Einschränkungen, unter anderem folgende:

- SQL Anywhere unterstützt den Adaptive Server Enterprise Commit Service nicht.

- Die **Fähigkeiten** einer Client/Server-Verbindung bestimmen, welche Clientanforderungen und Serverantworten für diese Verbindung zulässig sind. Folgende Funktionen werden nicht unterstützt:
 - CS_CSR_ABS
 - CS_CSR_FIRST
 - CS_CSR_LAST
 - CS_CSR_PREV
 - CS_CSR_REL
 - CS_DATA_BOUNDARY
 - CS_DATA_SENSITIVITY
 - CS_OPT_FORMATONLY
 - CS_PROTO_DYNPROC
 - CS_REG_NOTIF
 - CS_REQ_BCP
- Sicherheitsoptionen, wie z.B. SSL, werden nicht unterstützt. Kennwortverschlüsselung wird allerdings unterstützt.
- Open Client-Anwendungen können über TCP/IP keine Verbindung zu SQL Anywhere herstellen.
Weitere Hinweise zu Funktionen finden Sie in der Dokumentation *Open Server Server-Library C Reference Manual (in englischer Sprache)*.
- Wenn Sie CS_DATAFMT mit CS_DESCRIBE_INPUT verwenden, wird der Datentyp einer Spalte nicht zurückgegeben, wenn eine parametrisierte Variable als Eingabe an SQL Anywhere gesendet wird.

HTTP-Webdienste

SQL Anywhere als HTTP-Webserver

SQL Anywhere enthält einen integrierten HTTP-Webserver, mit dem Sie Online-Webdienste in SQL Anywhere-Datenbanken erstellen können. SQL Anywhere-Webserver unterstützen HTTP und SOAP über von Webbrowser und Clientanwendungen gesendete HTTP-Anforderungen. Die Webserver-Performance ist optimiert, da Webdienste in der Datenbank eingebettet sind.

SQL Anywhere-Webdienste stellen Clientanwendungen mit einer Alternative zu traditionellen Schnittstellen bereit, wie etwa JDBC und ODBC. Sie können einfach bereitgestellt werden, da keine weiteren Komponenten erforderlich sind. Es kann darauf von in einer Vielzahl an Sprachen (einschließlich Skriptsprachen wie Perl und Python) geschriebenen Clientanwendungen auf mehreren Plattformen zugegriffen werden.

Zusätzlich zur Bereitstellung von Webdiensten über einen HTTP-Webserver, kann SQL Anywhere auch als SOAP- oder HTTP-Clientanwendung fungieren, um auf Standardwebdienste zuzugreifen, die über das Internet und andere SQL Anywhere-HTTP-Webserver verfügbar sind.

Siehe auch

- „Zugriff auf Webdienste mithilfe von Webclients“ auf Seite 794

Kurzeinführung zur Verwendung von SQL Anywhere als HTTP-Webserver

Dieser Abschnitt zeigt die Vorgehensweise zum Starten eines SQL Anywhere-HTTP-Webservers, Erstellen eines Webdienstes und Zugreifen darauf von einem Webbrowser. SQL Anywhere-Webdienstfunktionen, wie etwa Unterstützung für SOAP über HTTP und die Entwicklung von Anwendungen werden nicht in vollem Umfang erläutert. Über den Umfang dieses Handbuchs hinaus sind viele SQL Anywhere-Webdienstfunktionen verfügbar.

Führen Sie die folgenden Aufgaben aus, um einen SQL Anywhere-HTTP-Webserver und einen HTTP-Webdienst zu erstellen:

1. Starten Sie den SQL Anywhere-HTTP-Webserver während des Ladens einer SQL Anywhere-Datenbank.

Führen Sie den folgenden Befehl an einer Eingabeaufforderung aus:

```
dbsrv16 -xs http(port=8082) "%SQLANYAMP16%\demo.db"
```

Hinweis

Starten Sie mit dem Befehl `dbsrv16` einen Datenbankserver, auf den über ein Netzwerk zugegriffen werden kann. Verwenden Sie den Befehl `dbeng16`, um einen Personal Datenbankserver zu starten, auf den nur durch den lokalen Host zugegriffen werden kann.

Die Option `-xs http(port=8082)` weist den Server an, auf Port 8082 auf HTTP-Anforderungen zu warten. Verwenden Sie eine andere Portnummer, wenn auf Port 8082 bereits ein Webserver ausgeführt wird.

2. Verwenden Sie die `CREATE SERVICE`-Anweisung zum Erstellen eines Webdienstes, der auf eingehende Webbrowser-Anforderungen antwortet.
 - a. Verbinden Sie sich über Interactive SQL mit der Datenbank *demo.db*, indem Sie den folgenden Befehl ausführen:

```
dbisql -c "dbf=%SQLANYAMP16%\demo.db;uid=DBA;pwd=sql"
```

- b. Erstellen Sie einen neuen Webdienst in der Datenbank.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE SERVICE SampleWebService
  TYPE 'web-service-type-clause'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT 'Hello world!';
```

Ersetzen Sie *Webdienst-TYPE-Klausel* durch den gewünschten Webdiensttyp. Die **TYPE-Klausel HTML** wird für die Webbrowser-Kompatibilität empfohlen. Andere allgemeine TYPE-Klauseln für HTTP-Webdienste sind **XML**, **RAW** und **JSON**.

Die `CREATE SERVICE`-Anweisung erstellt den Webdienst **SampleWebService**, der die Ergebnismenge der `SELECT`-Anweisung zurückgibt. In diesem Beispiel gibt die Anweisung die Zeichenfolge "Hello world!" zurück.

Die `AUTHORIZATION OFF`-Klausel gibt an, dass für den Zugriff auf den Webdienst keine Autorisierung erforderlich ist.

Die `USER DBA`-Anweisung zeigt an, dass die Dienstanweisung unter dem DBA-Loginnamen ausgeführt werden sollte.

Die `AS SELECT`-Klausel lässt zu, dass der Dienst eine Auswahl in einer Tabelle oder Funktion trifft oder Daten direkt anzeigt. Verwenden Sie die `AS CALL`-Klausel als Alternative für den Aufruf einer gespeicherten Prozedur.

3. Rufen Sie Webdienst in einem Webbrowser auf.

Öffnen Sie auf dem Computer, auf dem der SQL Anywhere-HTTP-Webserver ausgeführt wird, einen Webbrowser wie Internet Explorer oder Firefox und gehen Sie zu folgender URL:

```
http://localhost:8082/demo/SampleWebService
```

Diese URL leitet Ihren Webbrowser auf den HTTP-Webserver auf Port 8082 weiter.

SampleWebService gibt "Hello world" aus. Die Ergebnismenge wird im von der *Webdienst-TYPE-Klausel* aus Schritt 2 festgelegten Format angezeigt. Weitere Hinweise zur Anzeige von Ergebnismengen in Webbrowsern finden Sie unter „[Webdiensttypen](#)“ auf Seite 760.

Andere Ressourcen für Beispiele

Beispiele befinden sich im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\http`.

Weitere Beispiele sind möglicherweise auf CodeXchange unter <http://www.sybase.com/developer/codexchange> verfügbar.

Siehe auch

- „HTTP-Webserver starten“ auf Seite 757
- „Webdiensttypen“ auf Seite 760
- „CREATE SERVICE-Anweisung [HTTP-Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

HTTP-Webserver starten

Der SQL Anywhere-HTTP-Webserver wird automatisch gestartet, wenn Sie den Datenbankserver mit der Serveroption `-xs` starten. Mithilfe dieser Option können Sie die folgenden Aufgaben ausführen:

- Aktivieren eines Webdienstprotokolls zum Warten auf Webdienstanforderungen
- Konfigurieren von Netzwerkprotokolloptionen, wie etwa Serverport, Protokollierung, Kriterien für die Zeitüberschreitung und maximal zulässige Anforderungsgröße

Die generelle Syntax des Befehls in der Befehlszeile ist wie folgt:

```
dbsrv16 -xs protocol-type(protocol-options) your-database-name.db
```

Ersetzen Sie *protocol-type* und *protocol-options* durch eines der folgenden unterstützten Protokolle und entsprechende Protokolloptionen:

- **HTTP** Verwenden Sie dieses Protokoll zum Warten auf HTTP-Verbindungen. Hier sehen Sie ein Beispiel.

```
dbsrv16 -xs HTTP(PORT=8082) services.db
```

- **HTTPS** Verwenden Sie dieses Protokoll zum Warten auf HTTPS-Verbindungen. SSL-Version 3.0 und TLS-Version 1.0/1.1 werden unterstützt. Hier sehen Sie ein Beispiel.

```
dbsrv16 -xs "HTTPS(FIPS=N;PORT=8082;IDENTITY=C:\Users\Public\Documents\SQL Anywhere 16\Samples\Certificates\rsaserver.id;IDENTITY_PASSWORD=test)" services.db
```

Hinweis

Für die verschiedenen unterstützten Protokolle stehen Netzwerkprotokolloptionen zur Verfügung. Mit diesen Optionen können Sie das Protokollverhalten steuern. Diese Optionen können außerdem beim Starten des Datenbankservers an der Befehlszeile konfiguriert werden.

Siehe auch

- „Datenbankserveroption -xs “ [\[SQL Anywhere Server - Datenbankadministration\]](#)
- „Konfiguration von Netzwerkprotokolloptionen“ auf Seite 758
- „Definition: Webdienste“ auf Seite 760

Konfiguration von Netzwerkprotokolloptionen

Netzwerkprotokolloptionen sind optionale Einstellungen, mit denen ein bestimmtes Webdienstprotokoll gesteuert werden kann. Diese Einstellungen werden an der Befehlszeile konfiguriert, wenn Sie den Datenbankserver mit der Serveroption -xs starten. Siehe „Datenbankserveroption -xs “ [\[SQL Anywhere Server - Datenbankadministration\]](#).

Beispielsweise konfiguriert folgende Befehlszeile einen HTTPS-Listener, wobei die Netzwerkprotokolloptionen PORT, FIPS, Identity und identity_password angegeben sind:

```
dbsrv16 -xs https(PORT=544;FIPS=YES;
  IDENTITY=certificate.id;IDENTITY_PASSWORD=password) your-database-name.db
```

Dieser Befehl startet einen Datenbankserver, der das HTTPS-Webdienstprotokoll für die Datenbank *Datenbankname.db* startet. Die Netzwerkprotokolloptionen geben an, dass der Webserver folgende Aufgaben ausführen soll:

- Hören Sie Port 544 ab statt des HTTPS-Standardports (443).
- Aktivieren der FIPS-bestätigten Sicherheitsalgorithmen für die Verschlüsselung der Kommunikation
- Lokalisiert die angegebene Identitätsdatei, *certificate.id*, die ein öffentliches Zertifikat und den zugehörigen privaten Schlüssel enthält.
- Validiert den privaten Schlüssel anhand des angegebenen Identitätskennworts, **password**.

Die folgende Liste zeigt die häufig für Webdienstprotokolle verwendeten Netzwerkprotokolloptionen:

Netzwerkprotokolloption	Verfügbare Webdienstprotokolle	Beschreibung
„DatabaseName-Protokolloption (DBN)“ [SQL Anywhere Server - Datenbankadministration]	HTTP, HTTPS	Gibt den Namen der Datenbank an, die benutzt werden soll, wenn Webanforderungen verarbeitet werden, bzw. benutzt die Schlüsselwörter REQUIRED oder AUTO, um festzulegen, ob Datenbanknamen als Teil der URL-Adresse erforderlich sind.
„FIPS-Protokolloption“ [SQL Anywhere Server - Datenbankadministration]	HTTPS	Aktiviert FIPS-bestätigte Sicherheitsalgorithmen zum Verschlüsseln von Datenbankdateien, Kommunikationen für die Client/Server-Kommunikation und Webdiensten.
„Identity-Protokolloption“ [SQL Anywhere Server - Datenbankadministration]	HTTPS	Gibt den Namen einer Identitätsdatei zur Verwendung in Verbindung mit sicheren HTTPS-Verbindungen an.

Netzwerkprotokolloption	Verfügbare Webdienstprotokolle	Beschreibung
„identity_password-Protokolloption“ [<i>SQL Anywhere Server - Datenbank-administration</i>]	HTTPS	Gibt das Kennwort für das Verschlüsselungszertifikat an.
„LocalOnly-Protokolloption (LO)“ [<i>SQL Anywhere Server - Datenbank-administration</i>]	HTTP, HTTPS	Ermöglicht es, für einen Client festzulegen, dass nur mit einem Server auf dem lokalen Computer eine Verbindung hergestellt wird, falls einer vorhanden ist
„LogFile-Protokolloption (LOG)“ [<i>SQL Anywhere Server - Datenbank-administration</i>]	HTTP, HTTPS	Legt den Namen der Datei fest, in die der Datenbankserver Informationen über Webanforderungen schreiben soll
„LogFormat-Protokolloption (LF)“ [<i>SQL Anywhere Server - Datenbank-administration</i>]	HTTP, HTTPS	Steuert das Format von Meldungen, die in die Logdatei geschrieben werden, in die der Datenbankserver Informationen über Webanforderungen schreibt, und gibt an, welche Felder in den Meldungen angezeigt werden.
„LogOptions-Protokolloption (LOPT)“ [<i>SQL Anywhere Server - Datenbank-administration</i>]	HTTP, HTTPS	Gibt die Typen der Meldungen an, die in der Logdatei aufgezeichnet werden, in die der Datenbankserver Informationen über Webanforderungen schreibt.
„ServerPort-Protokolloption (PORT)“ [<i>SQL Anywhere Server - Datenbank-administration</i>]	HTTP, HTTPS	Gibt den Port an, den der Datenbankserver abhört.

Siehe auch

- „Netzwerkprotokolloptionen“ [*SQL Anywhere Server - Datenbankadministration*]

Mehrere HTTP-Webserver starten

Eine Konfiguration mit mehreren HTTP-Webservern gibt Ihnen die Möglichkeit, Webdienste in mehreren Datenbanken zu erstellen und sie als Teil einer einzigen Website auszuweisen. Sie können mehrere HTTP-Webserver starten, indem Sie mehrere Instanzen der Datenbankserveroption -xs verwenden. Diese Aufgabe wird durch die Angabe einer eindeutigen Portnummer für jeden HTTP-Webserver ausgeführt.

Beispiel

In diesem Beispiel startet die folgende Befehlszeileneingabe zwei HTTP-Webdienste, einen für *Meine-erste-Datenbank.db* und einen zweiten für *Meine-zweite-Datenbank.db*:

```
dbsrv16 -xs http(port=80;dbn=your-first-database),http(port=8800;dbn=your-second-database)
your-first-database.db your-second-database.db
```

Siehe auch

- „DatabaseName-Protokolloption (DBN)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Datenbankserveroption -xs“ [[SQL Anywhere Server - Datenbankadministration](#)]

Definition: Webdienste

Unter Webdiensten versteht man Software, die den Datenverkehr und die Zusammenarbeit zwischen Systemen unterstützt. Durch Webdienste wird ein Teil der Geschäftslogik über das Internet verfügbar gemacht. URLs werden beim Verwalten von Webdiensten auf einem HTTP-Webserver für Clients verfügbar. Die bei der Angabe einer URL angewendeten Konventionen legen fest, wie der Server mit Webclients kommunizieren soll.

Die Webdienstverwaltung umfasst die folgenden Aufgaben:

- Wählen der zu verwaltenden Webdiensttypen
- Erstellen und Verwalten dieser Webdienste

Webdienste können in einer SQL Anywhere-Datenbank erstellt und gespeichert werden.

Webdiensttypen

Wenn ein Webbrowser oder eine Clientanwendung eine Webdienstanforderung an einen SQL Anywhere-Webdienst stellt, wird die Anforderung verarbeitet und in der Antwort eine Ergebnismenge zurückgegeben. SQL Anywhere unterstützt mehrere Webdiensttypen, mit denen Sie das Format der Ergebnismenge und die Art und Weise, wie Ergebnismengen zurückgegeben werden, steuern können. Sie legen den Webservertyp mit der TYPE-Klausel der CREATE SERVICE- bzw. der ALTER SERVICE-Anweisung nach Auswahl eines entsprechenden Webdiensttyps fest.

Die folgenden Webdiensttypen werden unterstützt:

- **HTML** Die Ergebnismenge einer Anweisung, Funktion oder Prozedur wird automatisch in ein HTML-Dokument formatiert, das eine Tabelle enthält. Webbrowser zeigen den Hauptteil des HTML-Dokuments an.
- **XML** Die Ergebnismenge einer Anweisung, Funktion oder Prozedur wird als XML-Dokument zurückgegeben. Nicht in XML formatierte Ergebnismengen werden automatisch in XML formatiert. Webbrowser zeigen den reinen XML-Code einschließlich Tags und Attributen an.

Die XML-Formatierung entspricht der Verwendung der FOR XML RAW-Klausel in einer SELECT-Anweisung, wie in der folgenden beispielhaften SQL-Anweisung zu sehen:

```
SELECT * FROM table-name FOR XML RAW
```

- **RAW** Die Ergebnismenge einer Anweisung, Funktion oder Prozedur wird ohne automatische Formatierung zurückgegeben.

Dieser Dienstyp bietet die größte Kontrolle über die Ergebnismenge. Jedoch, müssen Sie die Antwort generieren, indem Sie das erforderliche Markup (HTML, XML) ausdrücklich innerhalb der gespeicherten Prozedur schreiben. Sie können mit der SA_SET_HTTP_HEADER-Systemprozedur den HTTP Content-Type-Header zur Festlegung des MIME-Typs einstellen. So wird die Ergebnismenge in Webbrowsern korrekt angezeigt.

Ein Beispiel für eine gespeicherte Prozedur, die mit dem RAW-Webdiensttyp funktioniert, finden Sie unter „[So passen Sie Webseiten an](#)“ auf Seite 771.

- **JSON** Die Ergebnismenge einer Anweisung, Funktion oder Prozedur wird in JSON (JavaScript Object Notation) zurückgegeben. JavaScript Object Notation (JSON) ist ein sprachenunabhängiges, textbasiertes Datenaustauschformat, das für die Serialisierung von JavaScript-Daten entwickelt wurde. JSON stellt vier Basistypen dar: Zeichenfolgen, Zahlen, boolesche Werte und NULL. Außerdem stellt JSON zwei strukturierte Typen dar: Objekte und Arrays. Weitere Hinweise zu JSON finden Sie unter <http://www.json.org>

Dieser Dienst wird von AJAX für HTTP-Aufrufe von Webanwendungen verwendet. Ein Beispiel für den JSON-Typ finden Sie in `%SQLANYSAMPLE%\SQLAnywhere\HTTP\json_sample.sql`.

- **SOAP** Die Ergebnismenge einer Anweisung, Funktion oder Prozedur wird als SOAP-Antwort zurückgegeben. SOAP-Dienste bieten einen gemeinsamen Datenaustausch, um verschiedenen Clientanwendungen, die SOAP unterstützen, Datenzugriff zu bieten. SOAP-Anforderungs- und Antwortrahmen werden als XML-Daten unter Verwendung von HTTP (SOAP über HTTP) transportiert. Eine Anforderung an einen SOAP-Dienst muss eine gültige SOAP-Anforderung, und nicht nur eine einfache HTTP-Anforderung sein. Die Ausgabe von SOAP-Diensten kann angepasst werden, indem das FORMAT- und das DATATYPE-Attribut der CREATE bzw. ALTER SERVICE-Anweisung verwendet werden.
- **DISH** Ein DISH-Dienst (Determine SOAP Handler) ist ein SQL Anywhere SOAP-Endpunkt. Der DISH-Dienst exponiert das WSDL-Dokument (Web Services Description Language), das alle SOAP-Vorgänge (SQL Anywhere SOAP-Dienste) beschreibt, die darüber zugänglich sind. Ein SOAP-Client-Toolkit erstellt die Clientanwendung mit Schnittstellen basierend auf der WSDL. Die SOAP-Clientanwendung leitet alle SOAP-Anforderungen an den SOAP-Endpunkt (den SQL Anywhere DISH-Dienst) weiter.

Beispiel

Das folgende Beispiel zeigt die Erstellung eines allgemeinen HTTP-Webdienstes, der den Dienstyp **RAW** verwendet:

```
CREATE PROCEDURE sp_echotext(str LONG VARCHAR)
BEGIN
    CALL sa_set_http_header( 'Content-Type', 'text/plain' );
    SELECT str;
END;

CREATE SERVICE SampleWebService
```

```
TYPE 'RAW'  
AUTHORIZATION OFF  
USER DBA  
AS CALL sp_echoText ( :str );
```

Siehe auch

- „Webdienste erstellen oder ändern“ auf Seite 762
- „Webdienst-Systemprozeduren“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „HTTP- und SOAP-Anforderungsstrukturen“ auf Seite 830
- „CREATE SERVICE-Anweisung [HTTP-Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „DISH-Dienste erstellen“ auf Seite 764
- „SOAP über HTTP-Dienste erstellen“ auf Seite 763

Webdienstwartung

Die Webdienstverwaltung umfasst die folgenden Aufgaben:

- **Webdienste erstellen oder ändern** Erstellen oder ändern Sie Webdienste, um Webanwendungen, die eine Webbrowser-Schnittstelle unterstützen, zur Verfügung zu stellen und Datenaustausch über das Web mit REST- und SOAP-Methoden zu ermöglichen. Ein Beispiel für die Erstellung eines allgemeinen HTTP-Webdienstes finden Sie unter „[Kurzeinführung zur Verwendung von SQL Anywhere als HTTP-Webserver](#)“ auf Seite 755.
- **Webdienste löschen** Das Löschen eines Webdienstes führt dazu, dass die darauf folgenden Anforderungen für diesen Dienst eine HTTP-Statusmeldung des Typs 404 Not Found zurückgeben. Alle offenen Anforderungen werden unabhängig davon verarbeitet, ob sie beabsichtigt oder unbeabsichtigt gesendet wurden, sofern ein **root**-Webdienst vorhanden ist.
- **Webdienste kommentieren** Das Kommentieren ist optional und erlaubt Ihnen Dokumentation für Ihre Webdienste bereitzustellen.
- **root-Webdienst erstellen und anpassen** Sie können einen **root** Webdienst für die Verarbeitung von HTTP-Anforderungen, die mit keiner anderen Anforderung an Webdienste übereinstimmen, erstellen.
- **Webdienste aktivieren und deaktivieren** Ein deaktivierter Webdienst gibt eine HTTP-Statusmeldung des Typs 404 Not Found aus. Die METHOD-Klausel legt die HTTP-Methoden fest, die für einen bestimmten Webdienst aufgerufen werden können. Siehe „[CREATE SERVICE-Anweisung \[HTTP-Webdienst\]](#)“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Webdienste erstellen oder ändern

Das Erstellen oder Ändern eines Webdienstes erfordert die Verwendung der CREATE SERVICE- bzw. der ALTER SERVICE-Anweisung. In diesem Abschnitt wird die Ausführung dieser Anweisungen mit Interactive SQL für die Erstellung von unterschiedliche Arten von Webdiensten gezeigt. Bei den Beispielen in diesem Abschnitt wird davon ausgegangen, dass Sie mit Interactive SQL und unter Verwendung des folgenden Befehls eine Verbindung mit einer SQL Anywhere-Datenbank, *Meine-SQL-Anywhere-Datenbank.db*, hergestellt haben:

```
dbisql -c "dbf=your-sql-anywhere-database.db;uid=your-userid;pwd=your-
password"
```

HTTP-Webdienste erstellen

HTTP-Webdienste sind als HTML, XML oder RAW klassifiziert. Alle HTTP-Webdienste können mit derselben CREATE SERVICE- oder ALTER SERVICE-Anweisungssyntax erstellt bzw. geändert werden.

Beispiel

Führen Sie die folgende Anweisung in Interactive SQL aus, um auf dem HTTP-Webserver einen allgemeinen Beispiel-HTTP-Webdienst zu erstellen:

```
CREATE SERVICE SampleWebService
  TYPE 'web-service-type-clause'
  URL OFF
  USER DBA
  AUTHORIZATION OFF
  AS sql-statement;
```

Die CREATE SERVICE-Anweisung erstellt einen neuen Webdienst mit dem Namen **SampleWebService** und gibt die Ergebnismenge von *SQL-Anweisung* zurück. Sie können *SQL-Anweisung* entweder durch eine SELECT-Anweisung für die direkte Auswahl von Daten aus einer Tabelle oder Ansicht oder durch eine CALL-Anweisung für den Aufruf einer gespeicherten Prozedur in der Datenbank ersetzen. Weitere Hinweise zum Erstellen von gespeicherten Funktionen und Prozeduren für Ihre Webdienste finden Sie unter [„Webdienstanwendungen auf einem HTTP-Webserver entwickeln“ auf Seite 770](#).

Ersetzen Sie *Webdienst-TYPE-Klausel* durch den gewünschten Webdiensttyp. Gültige Klauseln für HTTP-Webdienste sind beispielsweise **HTML**, **XML**, **RAW** und **JSON**.

Sie können die generierte Ergebnismenge für den Dienst **SampleWebService** anzeigen, indem Sie über einen Webbrowser auf den Dienst zugreifen.

Siehe auch

- [„Webdiensttypen“ auf Seite 760](#)
- [„CREATE SERVICE-Anweisung \[HTTP-Webdienst\]“ \[SQL Anywhere Server - SQL-Referenzhandbuch\]](#)
- [„ALTER SERVICE-Anweisung \[HTTP-Webdienst\]“ \[SQL Anywhere Server - SQL-Referenzhandbuch\]](#)
- [„Auf einem SQL Anywhere-HTTP-Webserver navigieren“ auf Seite 790](#)

SOAP über HTTP-Dienste erstellen

SOAP ist ein Datenaustauschstandard, der von vielen Entwicklungsumgebungen unterstützt wird. SOAP-Nutzdaten bestehen aus einem XML-Dokument, das auch als SOAP-Envelope bezeichnet wird. Ein SOAP-Anforderungsrahmen enthält den SOAP-Vorgang (einen SOAP-Dienst) und gibt alle erforderlichen Parameter an. Der SOAP-Dienst analysiert den Anforderungsrahmen zum Abrufen der Parameter und ruft eine gespeicherte Prozedur oder Funktion auf, wie dies jeder andere Dienst ebenfalls tut. Die Darstellungsschicht des SOAP-Dienstes sendet die Ergebnismenge per Datenstrom in einem SOAP-Envelope mit vordefiniertem Format an den Client zurück. Das Format hängt von den Angaben in

der WSDL des DISH-Dienstes ab. Weitere Hinweise zu SOAP-Standards finden Sie unter <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

Standardmäßig werden Parameter und Ergebnisdaten von SOAP-Diensten als XmlSchema-Zeichenfolgeparameter geschrieben. DATATYPE ON legt fest, dass für Eingabeparameter und Antwortdaten die entsprechenden Datentypen verwendet werden sollen. Die Angabe von DATATYPE führt zur entsprechenden Änderung der WSDL-Spezifikation, sodass Client-SOAP-Toolkits Schnittstellen mit dem entsprechenden Parameter- und Antwortobjekttyp generieren. Weitere Hinweise zur Formatierung von Datentypen für Clientanwendungen finden Sie unter „[Variable, auf die aus Ergebnismengen zugegriffen wird](#)“ auf Seite 814.

Die FORMAT-Klausel wird für bestimmte SOAP-Toolkits mit unterschiedlichen Funktionen verwendet. DNET bietet Microsoft .NET Clientanwendungen zum Verarbeiten eine SOAP-Dienstantwort als System.Data.DataSet-Objekt. CONCRETE exponiert eine allgemeinere Struktur, mit der eine objektorientierte Anwendung, wie etwa .NET oder Java, Antwortobjekte generieren kann, die Zeilen und Spalten verpacken. XML gibt die gesamte Antwort in Form eines XML-Dokuments zurück und exponiert sie dabei als Zeichenfolge. Clients können die Daten mit einem XML-Parser weiter verarbeiten. Die FORMAT-Klausel der CREATE SERVICE-Anweisung unterstützt mehrere Client Anwendungstypen. Ein Beispiel für andere von SOAP-Diensten unterstützte Formate finden Sie unter „[Beispiele für HTTP-Webdienste](#)“ auf Seite 834.

Hinweis

Die DATATYPE-Klausel gilt nur für SOAP-Dienste (in HTML werden keine Datentypen angegeben). Die FORMAT-Klausel kann entweder für einen SOAP- oder einen DISH-Dienst angegeben werden. Eine FORMAT-Spezifikation für den SOAP-Dienst überschreibt diejenige des DISH-Dienstes.

Beispiel

Führen Sie in Interactive SQL die folgende Anweisung aus, um einen SOAP über HTTP-Dienst zu erstellen:

```
CREATE SERVICE SampleSOAPService
  TYPE 'SOAP'
  DATATYPE ON
  FORMAT 'CONCRETE'
  USER DBA
  AUTHORIZATION OFF
  AS sql-statement;
```

Siehe auch

- „[Webdiensttypen](#)“ auf Seite 760
- „[CREATE SERVICE-Anweisung \[SOAP-Webdienst\]](#)“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „[ALTER SERVICE-Anweisung \[SOAP-Webdienst\]](#)“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „[DISH-Dienste erstellen](#)“ auf Seite 764

DISH-Dienste erstellen

In SQL Anywhere können Sie DISH-Dienste erstellen, die als SOAP-Endpunkte für Gruppen von SOAP-Diensten agieren. DISH-Dienste erstellen auch automatisch WSDL-Dokumente (Web Services

Description Language), mit denen SOAP-Client-Toolkits die notwendigen Schnittstellen für den Datenaustausch mit den durch die WSDL beschriebenen SOAP-Diensten generieren können. SOAP-Dienste können ohne Wartung der DISH-Dienste hinzugefügt und entfernt werden, weil der aktuelle Arbeitssatz von SOAP über HTTP-Diensten immer bereitgestellt wird.

Beispiel

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus, um Beispiel-SOAP- und -DISH-Dienste auf dem HTTP-Webserver zu erstellen:

```
CREATE SERVICE "Samples/TestSoapOp"
  TYPE 'SOAP'
  DATATYPE ON
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT( ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR )
BEGIN
  SELECT i, f, s;
END;

CREATE SERVICE "dnet_endpoint"
  TYPE 'DISH'
  GROUP "Samples"
  FORMAT 'DNET';
```

Die erste CREATE SERVICE-Anweisung erstellt einen neuen SOAP-Dienst mit dem Namen **Samples/TestSoapOp**.

Die zweite CREATE SERVICE-Anweisung erstellt einen neuen DISH-Dienst mit dem Namen **dnet_endpoint**. Der **Samples**-Teil der GROUP-Klausel kennzeichnet die Gruppe der bereitzustellenden SOAP-Dienste. Sie können das vom DISH-Dienst generierte WSDL-Dokument anzeigen. Wenn Sie Ihren SQL Anywhere-Webserver auf einem Computer ausführen, können Sie auf den Dienst mit der URL `http://localhost:port-number/dnet_endpoint` zugreifen, wobei *port-number* für die Portnummer steht, auf der der Server läuft.

In diesem Beispiel enthält der SOAP-Dienst keine FORMAT-Klausel zur Angabe eines SOAP-Antwortformats. Aus diesem Grund wird das SOAP-Antwortformat vom DISH-Dienst bestimmt, wobei dieser nicht die FORMAT-Klausel des SOAP-Dienstes überschreibt. Mit dieser Funktion können Sie homogene DISH-Dienste erstellen, wobei jeder DISH-Endpunkt SOAP-Clients verschiedene Funktionen bereitstellen kann.

Homogene DISH-Dienste erstellen

Wenn SOAP-Dienstdefinitionen die Spezifikation der FORMAT-Klausel dem DISH-Dienst überlassen, kann ein Satz von SOAP-Diensten innerhalb eines DISH-Dienstes gruppiert werden, der das Format definiert. Mehrere DISH-Dienste können dann dieselbe Gruppe von SOAP-Diensten mit unterschiedlichen FORMAT-Spezifikationen bereitstellen. Wenn Sie das **TestSoapOp**-Beispiel ausweiten, können Sie mit folgender SQL-Anweisung einen anderen DISH-Dienst mit dem Namen **java_endpoint** erstellen:

```
CREATE SERVICE "java_endpoint"
  TYPE 'DISH'
```

```
GROUP "Samples"  
FORMAT 'CONCRETE' ;
```

In diesem Beispiel erhält der SOAP-Client ein Antwortobjekt mit dem Namen **TestSoapOp_Dataset**, wenn er eine Webdienstanforderung für den Vorgang **TestSoapOp** über den DISH-Dienst **java_endpoint** erstellt. Die WSDL kann zum Vergleich der Unterschiede zwischen **dnet_endpoint** und **java_endpoint** untersucht werden. Wenn Sie so vorgehen, kann ein SOAP-Endpunkt, der die Anforderungen eines bestimmten SOAP-Client-Toolkits erfüllt, schnell konstruiert werden.

Siehe auch

- „Webdiensttypen“ auf Seite 760
- „Auf einem SQL Anywhere-HTTP-Webserver navigieren“ auf Seite 790
- „Beispiele für HTTP-Webdienste“ auf Seite 834
- „CREATE SERVICE-Anweisung [SOAP-Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „ALTER SERVICE-Anweisung [SOAP-Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „SOAP über HTTP-Dienste erstellen“ auf Seite 763

Webdienste löschen

Das Löschen eines Webdienstes führt dazu, dass die darauf folgenden Anforderungen für diesen Dienst eine HTTP-Statusmeldung des Typs 404 **Not Found** zurückgeben. Alle offenen Anforderungen werden unabhängig davon verarbeitet, ob sie beabsichtigt oder unbeabsichtigt gesendet wurden, sofern ein **root**-Webdienst vorhanden ist.

Beispiel

Führen Sie die folgende SQL-Anweisung aus, um einen Webdienst namens **SampleWebService** zu löschen:

```
DROP SERVICE SampleWebService;
```

Siehe auch

- „DROP SERVICE-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „root-Webdienste erstellen und anpassen“ auf Seite 767

Webdienste kommentieren

Die Dokumentation eines Webdienstes erfordert die Verwendung der **COMMENT ON SERVICE**-Anweisung. Ein Kommentar kann entfernt werden, indem die Einstellung der *Anweisung*-Klausel auf **NULL** gesetzt wird.

Beispiel

Führen Sie beispielsweise folgende SQL-Anweisung aus, um einen neuen Kommentar zu einem Webdienst mit dem Namen **SampleWebService** zu erstellen:

```
COMMENT ON SERVICE SampleWebService  
IS "This is a comment on my web service.";
```

Siehe auch

- „COMMENT-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

root-Webdienste erstellen und anpassen

Eine HTTP-Clientanforderung, die mit keiner Webdienstanforderung übereinstimmt, wird vom **root**-Webdienst verarbeitet, wenn ein **root**-Webdienst definiert ist.

Der **root**-Webdienst bietet Ihnen eine einfache und flexible Methode für die Verarbeitung von beliebigen HTTP-Anforderungen, deren URLs zum Zeitpunkt der Erstellung Ihrer Anwendung nicht unbedingt bekannt sind, und für die Verarbeitung von nicht erkannten Anforderungen.

Beispiel

Dieses Beispiel zeigt die Verwendung eines **root**-Webdienstes, der in einer Tabelle innerhalb der Datenbank gespeichert ist, um für Webbrowser und HTTP-Clients Inhalte bereitzustellen. Es wird davon ausgegangen, dass Sie einen lokalen HTTP-Webserver auf einer einzelnen Datenbank gestartet haben und Port 80 überwacht wird. Alle Skripte werden auf dem Webserver ausgeführt.

Stellen Sie über Interactive SQL eine Verbindung zum Datenbankserver her und führen Sie folgende SQL-Anweisung zur Erstellung eines **root**-Webdienstes aus, der die vom Client bereitgestellte Hostvariable URL an eine Prozedur mit dem Namen **PageContent** weitergibt:

```
CREATE SERVICE root
  TYPE 'RAW'
  AUTHORIZATION OFF
  SECURE OFF
  URL ON
  USER DBA
  AS CALL PageContent(:url);
```

Der URL ON-Teil gibt an dass die vollständige Pfadkomponente durch eine HTTP-Variable mit dem Namen URL zugriffsfähig gemacht wird.

Führen Sie folgende SQL-Anweisung zum Erstellen einer Tabelle zum Speichern von Seiteninhalten aus. In diesem Beispiel wird der Seiteninhalt durch ihre URL, ihren MIME-Typ und den Inhalt selbst definiert.

```
CREATE TABLE Page_Content (
  url          VARCHAR(1024) NOT NULL PRIMARY KEY,
  content_type VARCHAR(128)  NOT NULL,
  image        LONG VARCHAR NOT NULL
);
```

Führen Sie folgende SQL-Anweisung aus, um die Tabelle mit Daten zu füllen. In diesem Beispiel soll der für den HTTP-Client bereitzustellende Inhalt bei Anforderung der index.html-Seite definiert werden.

```
INSERT INTO Page_Content
VALUES(
  'index.html',
  'text/html',
  '<html><body><h1>Hello World</h1></body></html>'
);
COMMIT;
```

Führen Sie die folgenden SQL-Anweisungen zur Implementierung der Prozedur **PageContent** aus, die die vom **root**-Webdienst übergebene URL-Hostvariable akzeptiert:

```
CREATE PROCEDURE PageContent(IN @url LONG VARCHAR)
RESULT ( html_doc LONG VARCHAR )
BEGIN
    DECLARE @status CHAR(3);
    DECLARE @type VARCHAR(128);
    DECLARE @image LONG VARCHAR;

    SELECT content_type, image INTO @type, @image
    FROM Page_Content
    WHERE url = @url;

    IF @image is NULL THEN
        SET @status = '404';
        SET @type = 'text/html';
        SET @image = '<html><body><h1>404 - Page Not Found</h1>'
            || '<p>There is no content located at the URL "'
            || html_encode( @url ) || '" on this server.<p>'
            || '</body></html>';
    ELSE
        SET @status = '200';
    END IF;
    CALL sa_set_http_header( '@HttpStatus', @status );
    CALL sa_set_http_header( 'Content-Type', @type );
    SELECT @image;
END;
```

Der **root**-Webdienst ruft die Prozedur **PageContent** auf, wenn eine Anforderung an den HTTP-Server mit keiner anderen definierten Webdienst-URL übereinstimmt. Die Prozedur prüft, ob die vom Client gelieferte URL mit einer URL in der Tabelle **Page_Content** übereinstimmt. Die SELECT-Anweisung sendet eine Antwort an den Client. Wenn die vom Client bereitgestellte URL in der Tabelle nicht gefunden wurde, wird eine generische HTML-Seite des Typs 404 - Page Not Found erstellt und an den Client gesendet.

Einige Browser antworten beim Status 404 mit ihrer eigenen Seite, sodass nicht gewährleistet ist, dass die generische Seite angezeigt wird.

In der Fehlermeldung wird die HTML_ENCODE-Funktion dazu verwendet, die Sonderzeichen in der vom Client bereitgestellten URL zu kodieren.

Der @HttpStatus-Header wird zum Festlegen des mit der Anforderung zurückgegebenen Statuscode verwendet. Ein Status von 404 weist darauf hin, dass die Seite nicht gefunden wurde. Ein Status von 200 bedeutet, dass die Seite OK ist. Der 'Content-Type'-Header wird zum Festlegen des mit der Anforderung zurückgegebenen Inhaltstyps verwendet. In diesem Beispiel lautet der Inhaltstyp (MIME-Typ) der Seite index.html "text/html". Weitere Hinweise finden Sie unter „sa_set_http_header-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Siehe auch

- „So passen Sie Webseiten an“ auf Seite 771
- „Auf einem SQL Anywhere-HTTP-Webserver navigieren“ auf Seite 790
- „HTML_ENCODE-Funktion [Verschiedene]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „CREATE SERVICE-Anweisung [HTTP-Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Webdienst-SQL-Anweisungen

Die folgenden SQL-Anweisungen stehen für die Webdienst-Entwicklung zur Verfügung:

Webserver-bezogene SQL-Anweisungen	Beschreibung
„CREATE SERVICE-Anweisung [HTTP-Webdienst]“ <i>[SQL Anywhere Server - SQL-Referenzhandbuch]</i>	Erstellt einen neuen HTTP-Webdienst.
„CREATE SERVICE-Anweisung [SOAP-Webdienst]“ <i>[SQL Anywhere Server - SQL-Referenzhandbuch]</i>	Erstellt einen neuen SOAP über HTTP- oder DISH-Dienst.
„ALTER SERVICE-Anweisung [HTTP-Webdienst]“ <i>[SQL Anywhere Server - SQL-Referenzhandbuch]</i>	Ändert einen vorhandenen HTTP-Webdienst.
„ALTER SERVICE-Anweisung [SOAP-Webdienst]“ <i>[SQL Anywhere Server - SQL-Referenzhandbuch]</i>	Ändert einen vorhandenen HTTP über SOAP- oder DISH-Dienst.
„COMMENT-Anweisung“ <i>[SQL Anywhere Server - SQL-Referenzhandbuch]</i>	<p>Speichert einen Kommentar für ein Datenbankobjekt in den Systemtabellen.</p> <p>Verwenden Sie die folgende Syntax zum Kommentieren eines Webdienstes:</p> <pre>COMMENT ON SERVICE 'web-service-name' IS 'your comments'</pre>
„DROP SERVICE-Anweisung“ <i>[SQL Anywhere Server - SQL-Referenzhandbuch]</i>	Löscht einen Webdienst.

Verbindungspooling für Webdienste

Jede Datenbank, die Webdienste bereitstellt, hat Zugriff auf einen Pool von Datenbankverbindungen. Der Pool ist nach Benutzername gruppiert, sodass alle unter einer bestimmten USER-Klausel definierten Dienste sich in derselben Verbindungspoolgruppe befinden.

Eine Dienstanforderung, die eine Abfrage zum ersten Mal ausführt, muss eine Optimierungsphase zur Einrichtung eines Ausführungsplans durchlaufen. Wenn der Plan im Cache abgelegt und wiederverwendet werden kann, kann die Optimierungsphase für die nachfolgenden Ausführungen übersprungen werden. HTTP-Verbindungspooling nutzt für Datenbankverbindungen die im Cache abgelegten Pläne, indem es sie, wo immer möglich, wiederverwendet. Jeder Dienst verwaltet zur Optimierung der Wiederverwendung seine eigene Liste von Verbindungen. Bei Spitzenlasten kann es dazu kommen, dass ein Dienst die am wenigsten verwendeten Verbindungen aus derselben Benutzerverbindungsgruppe "stiehlt".

Mit der Zeit kann eine bestimmte Verbindung möglicherweise im Cache abgelegte Pläne erwerben, die die Performance für die Ausführung einer Reihe von Diensten optimieren können. Weitere Hinweise zu den Einstellungen für im Cache abgelegte Pläne finden Sie unter „[max_plans_cached-Option](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

In einem Verbindungspool versucht eine HTTP-Anforderung für einen bestimmten Dienst eine Datenbankverbindung aus einem Pool zu erwerben. Anders als bei HTTP-Sitzungen werden gepoolte Verbindungen durch Zurücksetzen der Verbindungsumgebung saniert, d. h. durch Zurücksetzen der Verbindungsbereichsvariablen und der temporären Tabellen.

Innerhalb des HTTP-Verbindungspools gepoolte Datenbankverbindungen werden in Zusammenhang mit der Lizenzierung nicht als verwendete Verbindungen gezählt. Verbindungen werden als lizenzierte Verbindungen gezählt, wenn sie aus einem Pool erworben werden. Ein Status des Typs 503 *Service Temporarily Unavailable* wird zurückgegeben, wenn bei einer HTTP-Anforderung die Lizenz einschränkungen überschritten werden, während eine Verbindung aus dem Pool erworben wird.

Webdienste können nur einen Verbindungspool verwenden, wenn sie mit `AUTHORIZATION OFF` definiert sind. Weitere Hinweise finden Sie unter „[CREATE SERVICE-Anweisung](#) [[HTTP-Webdienst](#)]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)] und „[CREATE SERVICE-Anweisung](#) [[SOAP-Webdienst](#)]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)].

Eine Datenbankverbindung in einem Pool wird nicht aktualisiert, wenn Änderungen an Datenbank- und Verbindungsoptionen vorgenommen werden.

Siehe auch

- [http_connection_pool_basesize-Verbindungseigenschaft](#) [[SQL Anywhere Server - Datenbankadministration](#)]
- [http_connection_pool_timeout-Verbindungseigenschaft](#) [[SQL Anywhere Server - Datenbankadministration](#)]
- „[Verbindungseigenschaften von Webdiensten](#)“ auf Seite 789

Webdienstanwendungen auf einem HTTP-Webserver entwickeln

Dieser Abschnitt bietet einen Überblick über die Erstellung und Anpassung von Webseiten. Außerdem wird darin die Entwicklung von gespeicherten Prozeduren für Ihren HTTP-Webserver erklärt. Es wird hierbei davon ausgegangen, dass Sie über Kenntnisse zum Starten von SQL Anywhere-HTTP-Webservern und Erstellen von Webdiensten, die gespeicherte Prozeduren aufrufen, verfügen.

Detailliertere Beispiele für Webdienstanwendungen finden Sie im Verzeichnis %SQLANYSAMPI6%\SQLAnywhere\HTTP.

Siehe auch

- „HTTP-Webserver starten“ auf Seite 757
- „Definition: Webdienste“ auf Seite 760

So passen Sie Webseiten an

Sie müssen zuerst das Format des vom HTTP-Webserver aufgerufenen Webdienstes berücksichtigen, um Ihre Webseiten anpassen zu können. Beispielsweise sind Webseiten in HTML formatiert, wenn der Webdienst den Typ **HTML** angibt.

Der Webdiensttyp **RAW** benötigt den größten Aufwand beim Anpassen, da er erfordert, dass Webdienstprozeduren und -funktionen Code für das erforderliche Markup (z.B. HTML oder XML) enthalten. Bei Verwendung des Typs **RAW** müssen folgende Aufgaben durchgeführt werden, um Webseiten anzupassen:

- Einstellen des HTTP Content-Type-Headerfelds auf den entsprechenden MIME-Typ (etwa text/html) in der aufgerufenen gespeicherten Prozedur.
- Anwenden des entsprechenden Markup für den MIME-Typ bei der Generierung einer Webseitenausgabe ausgehend von der abgerufenen gespeicherten Prozedur.

Beispiel

Das folgende Beispiel zeigt die Vorgehensweise zum Erstellen eines neuen Webdienstes, für den der Typ **RAW** angegeben ist:

```
CREATE SERVICE WebServiceName
  TYPE 'RAW'
  AUTHORIZATION OFF
  URL ON
  USER DBA
  AS CALL HomePage( :url );
```

In diesem Beispiel ruft der Webdienst die gespeicherte Prozedur **HomePage** auf, die erforderlich ist zum Definieren eines einzelnen URL-Parameters, der die PATH-Komponenten der URL erhält.

Content-Type-Headerfeld definieren

Verwenden Sie die Systemprozedur sa_set_http_header für die Definition des HTTP Content-Type-Headers, um sicherzustellen, dass der Inhalt korrekt in Webbrowsern dargestellt wird.

Das folgende Beispiel zeigt, wie mithilfe der Systemprozedur sa_set_http_header die Webseitenausgabe in HTML unter Verwendung des MIME-Typs text/html formatiert wird:

```
CREATE PROCEDURE HomePage (IN url LONG VARCHAR)
  RESULT (html_doc XML)
  BEGIN
    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    -- Your SQL code goes here.
```

END . . .

Tag-Konventionen des MIME-Typs anwenden

Sie müssen in Ihrer gespeicherten Prozedur die Tag-Konventionen des durch den Content-Type-Header angegebenen MIME-Typs anwenden. SQL Anywhere verfügt über mehrere Funktionen zum Erstellen von Tags.

Das folgende Beispiel zeigt die Verwendung der Funktionen XMLCONCAT und XMLELEMENT zum Generieren von HTML-Inhalt. Es wird hierbei davon ausgegangen, dass die Systemprozedur `sa_set_http_header` zum Einstellen des MIME-Typs `text/html` für den Content-Type-Headers verwendet wird:

```
XMLCONCAT(  
  CAST('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' AS XML),  
  XMLELEMENT(  
    'HTML',  
    XMLELEMENT(  
      'HEAD',  
      XMLELEMENT('TITLE', 'My Home Page')  
    ),  
    XMLELEMENT(  
      'BODY',  
      XMLELEMENT('H1', 'My home on the web'),  
      XMLELEMENT('P', 'Thank you for visiting my web site!')  
    )  
  )  
)
```

Da für Elementinhalte immer Escapezeichen verwendet werden (es sei denn, der Datentyp ist XML), verwendet das obige Beispiel die CAST-Funktion. Andernfalls werden für Sonderzeichen immer Escapezeichen verwendet (z.B. `<` für `<`).

Ein ausführliches Beispiel eines **RAW**-Webdienstes, der eine benutzerdefinierte Webseite generiert, finden Sie unter „[root-Webdienste erstellen und anpassen](#)“ auf Seite 767.

Siehe auch

- „Webdiensttypen“ auf Seite 760
- „`sa_set_http_header`-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Webdienstfunktionen“ auf Seite 788
- „XMLCONCAT-Funktion [Zeichenfolge]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „XMLELEMENT-Funktion [Zeichenfolge]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Auf einem SQL Anywhere-HTTP-Webserver navigieren“ auf Seite 790
- „Funktionen“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Zugriff auf vom Client bereitgestellte HTTP-Variablen und -Header

Auf Variablen und Header in einer Anforderung eines HTTP-Clients kann mit einer der folgenden Methoden zugegriffen werden:

- Deklaration der Webdienst-Anweisung, sodass sie als Host-Parameter eines Aufrufs einer gespeicherten Funktion oder Prozedur übergeben werden

- Aufruf der Funktionen HTTP_VARIABLE, NEXT_HTTP_VARIABLE, HTTP_HEADER und NEXT_HTTP_HEADER in einer gespeicherten Funktion oder Prozedur.

Siehe auch

- [„An Webdienste übergebene Variable“ auf Seite 811](#)

Zugriff auf HTTP-Variablen mithilfe von Hostparametern

Sie können vom Client bereitgestellte Variablen referenzieren, wenn Sie sie als Host-Parameter eines Funktions- oder Prozeduraufrufs übergeben.

Beispiel

Das folgende Beispiel zeigt die Vorgehensweise für den Zugriff auf die in einem Webdienst mit dem Namen **ShowTable** verwendeten Host-Parameter:

```
CREATE SERVICE ShowTable
  TYPE 'RAW'
  AUTHORIZATION ON
  AS CALL ShowTable( :user_name, :table_name );

CREATE PROCEDURE ShowTable(IN username VARCHAR(128), IN tblname VARCHAR(128))
BEGIN
  -- write SQL code utilizing the username and tblname variables here.
END;
```

Dienst-Host-Parameter werden in der Deklarationsreihenfolge der Prozedurparameter zugeordnet. Im oben gezeigten Beispiel werden die Host-Parameter **user_name** und **table_name** den Parametern **username** und **tblname** zugeordnet.

Weitere Hinweise zum Übergeben der URL als Host-Parameter finden Sie unter [„Auf einem SQL Anywhere-HTTP-Webserver navigieren“ auf Seite 790](#).

Zugriff auf HTTP-Variablen und -Header mithilfe von Webdienstfunktionen

Die Funktionen HTTP_VARIABLE, NEXT_HTTP_VARIABLE, HTTP_HEADER und NEXT_HTTP_HEADER können für Iterationen durch die vom Client bereitgestellten Variablen und Header verwendet werden.

Mit HTTP_VARIABLE und HTTP_NEXT_VARIABLE auf Variablen zugreifen

Sie können mit den Funktionen NEXT_HTTP_VARIABLE und HTTP_VARIABLE Iterationen durch alle vom Client bereitgestellten Variablen durchführen.

Mit der Funktion HTTP_VARIABLE können Sie den Wert einer Variablen abrufen.

Die Funktion NEXT_HTTP_VARIABLE ermöglicht Ihnen die Iteration durch alle vom Client gesendeten Variablen. Übergeben Sie den Wert NULL beim ersten Aufruf der Funktion, um den ersten Variablennamen abzurufen. Verwenden Sie den zurückgegeben Variablenname als Parameter für einen Aufruf der Funktion HTTP_VARIABLE, um ihren Wert abzurufen. Durch die Übergabe des vorigen Variablennamens an den Aufruf von NEXT_HTTP_VARIABLE wird der nächste Variablenname abgerufen. NULL wird zurückgegeben, wenn der letzte Variablenname übergeben wird.

Iterationen durch die Variablennamen stellen sicher, dass jeder Variablenname genau einmal zurückgegeben wird. Die Reihenfolge der Variablennamen ist jedoch möglicherweise nicht dieselbe wie die Reihenfolge, in der sie in der Clientanforderung auftreten.

Das folgende Beispiel zeigt die Verwendung der Funktion HTTP_VARIABLE zum Abrufen von Werten aus Parametern, die in einer auf den **ShowDetail**-Dienst zugreifenden Clientanforderung bereitgestellt werden:

```
CREATE SERVICE ShowDetail
  TYPE 'HTML'
  URL PATH OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowDetail();

CREATE PROCEDURE ShowDetail()
BEGIN
  DECLARE v_customer_id LONG VARCHAR;
  DECLARE v_product_id LONG VARCHAR;
  SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
  SET v_product_id = HTTP_VARIABLE( 'product_id' );
  CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

Das folgende Beispiel zeigt das Abrufen dreier Attribute aus Headerfeldwerten in Zusammenhang mit der **image** Variable:

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

Die Übergabe einer Ganzzahl als zweiten Parameter ermöglicht Ihnen den Abruf zusätzlicher Werte. Mit dem dritten Parameter können Sie die Werte des Headerfeldes aus mehrteiligen Anforderungen abrufen. Geben Sie den Namen eines Headerfeldes ein, um seinen Wert abzurufen.

Mit HTTP_HEADER und NEXT_HTTP_HEADER auf Header zugreifen

HTTP-Anforderungsheader können unter Verwendung der Funktionen NEXT_HTTP_HEADER und HTTP_HEADER von Anforderungen bezogen werden.

Die Funktion HTTP_HEADER gibt den Wert des angegebenen HTTP-Headerfelds zurück.

Die Funktion NEXT_HTTP_HEADER führt eine Iteration durch die HTTP-Header durch und gibt den nächsten HTTP-Headernamen zurück. Wird diese Funktion mit NULL aufgerufen, gibt sie den Namen des ersten Headers zurück. Nachfolgende Header werden abgerufen, indem der Funktion der Name des vorigen Headers übergeben wird. NULL wird zurückgegeben, wenn der letzte Headernamen aufgerufen wird.

Die folgende Tabelle enthält eine Liste der gängigen HTTP-Anforderungsheader und ihrer gängigen Werte:

Headername	Headerwert
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language	en-us
Accept-Charset	utf-8, iso-8859-5;q=0.8
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/4.0 (kompatibel; MSIE 7.0; Windows NT 5.2; WOW64; SV1; .NET CLR 2.0.50727)
Host	localhost:8080
Connection	Keep-Alive

Die folgende Tabelle enthält eine Liste der Spezial-Header und ihrer gängigen Werte:

Headername	Headerwert
@HttpMethod	GET
@HttpURI	/demo/ShowHTTPHeaders
@HttpVersion	HTTP/1.1
@HttpQueryString	id=-123&version=109&lang=en

Sie können mit dem Spezial-Header @HttpStatus den Statuscode der gerade verarbeiteten Anforderung einstellen.

Das folgende Beispiel zeigt das Formatieren von Headernamen und -werten als HTML-Tabelle.

Den Webdienst **ShowHTTPHeaders** erstellen:

```
CREATE SERVICE ShowHTTPHeaders
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL HTTPHeaderExample();
```

Erstellen Sie eine **HTTPHeaderExample**-Prozedur, die mithilfe der Funktion NEXT_HTTP_HEADER den Namen des Headers abrufen, und verwenden Sie anschließend die Funktion HTTP_HEADER, um dessen Wert abzurufen:

```
CREATE PROCEDURE HTTPHeaderExample()
  RESULT ( html_string LONG VARCHAR )
  BEGIN
    declare header_name LONG VARCHAR;
```

```

declare header_value LONG VARCHAR;
declare header_query LONG VARCHAR;
declare table_rows XML;
set header_name = NULL;
set table_rows = NULL;
header_loop:
LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
        LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    SET header_query = HTTP_HEADER( '@HttpQueryString' );
    -- Format header name and value into an HTML table row
    SET table_rows = table_rows ||
        XMLELEMENT( name "tr",
            XMLATTRIBUTES( 'left' AS "align",
                            'top' AS "valign" ),
            XMLELEMENT( name "td", header_name ),
            XMLELEMENT( name "td", header_value ),
            XMLELEMENT( name "td", header_query ) );

END LOOP;
SELECT XMLELEMENT( name "table",
    XMLATTRIBUTES( ' ' AS "BORDER",
                    '10' AS "CELLPADDING",
                    '0' AS "CELLSPACING" ),
    XMLELEMENT( name "th",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        'Header Name' ),
    XMLELEMENT( name "th",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        'Header Value' ),
    XMLELEMENT( name "th",
        XMLATTRIBUTES( 'left' AS "align",
                        'top' AS "valign" ),
        'HTTP Query String' ),
    table_rows );

END;
```

Greifen Sie in einem Webbrowser auf **ShowHTTPHeaders** zu und lassen Sie sich die Anforderungsheader in einer HTML-Tabelle angeordnet anzeigen.

Siehe auch

- „An Webdienste übergebene Variable“ auf Seite 811
- „HTTP_VARIABLE-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_HTTP_VARIABLE-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_HEADER-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_HTTP_HEADER-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Zugriff auf vom Client bereitgestellte SOAP-Anforderungsheader

Mit einer Kombination der Funktionen NEXT_SOAP_HEADER und SOAP_HEADER können Sie Header in SOAP-Anforderungen erhalten. Die Funktion NEXT_SOAP_HEADER durchläuft die SOAP-

Header, die in einem SOAP-Anforderungsrahmen enthalten sind, und gibt den nächsten SOAP-Headernamen zurück. Sie mit NULL aufzurufen führt dazu, dass sie den Namen des ersten Headers zurückgibt. Nachfolgende Header werden abgerufen, indem der Funktion NEXT_SOAP_HEADER der Name des vorigen Headers übergeben wird. Diese Funktion gibt NULL zurück, wenn sie mit dem Namen des letzten Headers aufgerufen wird.

Das folgende Beispiel zeigt, wie der SOAP-Header abgerufen wird:

```
SET hd_key = NEXT_SOAP_HEADER( hd_key );
IF hd_key IS NULL THEN
  -- no more header entries
  LEAVE header_loop;
END IF;
```

Wenn diese Funktion mehrfach aufgerufen wird, werden alle Headerbereiche genau einmal zurückgegeben, jedoch nicht unbedingt in der Reihenfolge, in der sie in der SOAP-Anforderung angeordnet sind.

Die Funktion SOAP_HEADER gibt den Wert des angegebenen SOAP-Header-Felds zurück oder NULL, wenn sie nicht von einem SOAP-Dienst aufgerufen wurde. Sie wird verwendet, wenn eine SOAP-Anforderung über einen Webdienst verarbeitet wird. Wenn ein Header für einen angegebenen Feldnamen nicht existiert, ist der Rückgabewert NULL.

Das Beispiel sucht nach einem SOAP-Header namens "Authentication". Wenn dieser Header gefunden wird, werden der Wert für den gesamten SOAP-Header sowie die Werte der Attribute **@namespace** und **mustUnderstand** extrahiert. Der Wert des SOAP-Headers könnte z.B. wie die folgende XML-Zeichenfolge aussehen:

```
<Authentication xmlns="CustomerOrderURN" mustUnderstand="1">
  <userName pwd="none">
    <first>John</first>
    <last>Smith</last>
  </userName>
</Authentication>
```

Für diesen Header hat das Attribut **@namespace** den Wert **CustomerOrderURN**.

Das Attribut **mustUnderstand** hat in diesem Beispiel den Wert 1.

Diese XML-Zeichenfolge wird mit der Funktion OPENXML syntaktisch analysiert. Hierzu wird eine XPath-Zeichenfolge mit dem Wert **/*:Authentication/*:userName** verwendet.

```
SELECT * FROM OPENXML( hd_entry, xpath )
WITH ( pwd LONG VARCHAR '@*:pwd',
       first_name LONG VARCHAR '/*:first/text()',
       last_name LONG VARCHAR '/*:last/text()' );
```

Unter Verwendung des oben dargestellten SOAP-Headerwerts erstellt die SELECT-Anweisung wie folgt eine Ergebnismenge:

pwd	first_name	last_name
none	John	Smith

Für diese Ergebnismenge wird ein Cursor deklariert und die drei Spaltenwerte werden in drei Variablen abgerufen. Sie haben nun alle gewünschten Informationen, die an den Webdienst übergeben wurden.

Beispiel

Das folgende Beispiel zeigt, wie ein Webserver SOAP-Anforderungen mit Parametern und SOAP-Header verarbeiten kann. Das Beispiel implementiert einen SOAP-Vorgang mit dem Namen **addItem**, der zwei Parameter benötigt: **amount** vom Typ int und **item** vom Typ string. Die Prozedur **sp_addItems** verarbeitet einen SOAP-Header zur Authentifizierung, indem sie den Vor- und den Nachnamen des Benutzers extrahiert. Die Werte werden dafür verwendet, den Validierungsheader einer SOAP-Antwort mit der Systemprozedur **sa_set_soap_header** mit Daten zu füllen. Die Antwort ist ein Ergebnis bestehend aus drei Spalten: **quantity**, **item** und **status** mit den Datentypen INT, LONG VARCHAR und LONG VARCHAR.

```
// create the SOAP service
CREATE SERVICE addItemS
  TYPE 'SOAP'
  FORMAT 'CONCRETE'
  AUTHORIZATION OFF
  USER DBA
  AS CALL sp_addItems( :amount, :item );

// create SOAP endpoint for related services
CREATE SERVICE itemStore
  TYPE 'DISH'
  AUTHORIZATION OFF
  USER DBA;

// create the procedure that will process the SOAP requests for the addItemS
service
CREATE PROCEDURE sp_addItems(count INT, item LONG VARCHAR)
RESULT(quantity INT, item LONG VARCHAR, status LONG VARCHAR)
BEGIN
  DECLARE hd_key LONG VARCHAR;
  DECLARE hd_entry LONG VARCHAR;
  DECLARE pwd LONG VARCHAR;
  DECLARE first_name LONG VARCHAR;
  DECLARE last_name LONG VARCHAR;
  DECLARE xpath LONG VARCHAR;
  DECLARE authinfo LONG VARCHAR;
  DECLARE namespace LONG VARCHAR;
  DECLARE mustUnderstand LONG VARCHAR;

header_loop:
  LOOP
    SET hd_key = next_soap_header( hd_key );
    IF hd_key IS NULL THEN
      // no more header entries.
      leave header_loop;
    END IF;
    IF hd_key = 'Authentication' THEN
      SET hd_entry = soap_header( hd_key );
      SET xpath = '/*: ' || hd_key || '/*:userName';
      SET namespace = soap_header( hd_key, 1, '@namespace' );
      SET mustUnderstand = soap_header( hd_key, 1, 'mustUnderstand' );
      BEGIN
        // parse for the pieces that you are interested in
        DECLARE crsr CURSOR FOR SELECT * FROM
          OPENXML( hd_entry, xpath )
          WITH ( pwd LONG VARCHAR '@*:pwd',
```

```

                                first_name LONG VARCHAR '*:first/text()',
                                last_name LONG VARCHAR '*:last/text()');
                                OPEN crsr;
                                FETCH crsr INTO pwd, first_name, last_name;
                                CLOSE crsr;
                                END;
                                // build a response header, based on the pieces from the request
header
                                SET authinfo = XMLELEMENT( 'Validation',
                                XMLATTRIBUTES(
                                    namespace as xmlns,
                                    mustUnderstand as mustUnderstand ),
                                XMLELEMENT( 'first', first_name ),
                                XMLELEMENT( 'last', last_name ) );
                                CALL sa_set_soap_header( 'authinfo', authinfo);
                                END IF;
                                END LOOP header_loop;
                                // code to validate user/session and check item goes here...
                                SELECT count, item, 'available';
                                END;

```

Weitere Hinweise zur clientseitigen Implementierung dieses Beispiels finden Sie unter „[Header der SOAP-Anforderung verwalten](#)“ auf Seite 805.

HTTP-Sitzungsverwaltung auf einem HTTP-Server

Eine Webanwendung kann Sitzungen auf unterschiedliche Arten unterstützen. Ausgeblendete Felder in HTML-Formularen können dazu verwendet werden, Client/Server-Daten über mehrere Anforderungen hinweg beizubehalten. Alternativ dazu können Sie mithilfe von Web 2.0-Methoden, wie etwa AJAX-aktiviertes clientseitiges JavaScript, asynchrone HTTP-Anforderungen basierend auf dem Clientstatus erstellen. SQL Anywhere bietet diese zusätzlichen Funktionen zum Aufrechterhalten einer Datenbankverbindung für die exklusive Verwendung durch sitzungsbasierte HTTP-Anforderungen.

Auf alle innerhalb der HTTP-Sitzung erstellten und geänderten Verbindungsbereichsvariablen und temporäre Tabellen kann von nachfolgenden HTTP-Anforderungen, die die entsprechende SessionID angeben, zugegriffen werden. Die SessionID kann durch eine GET- oder POST HTTP-Anforderungsmethode oder innerhalb eines HTTP-Cookie-Headers angegeben werden. Beim Empfang einer HTTP-Anforderung gemeinsam mit einer SessionID-Variablen prüft der Server sein Sitzungs-Repository auf einen damit übereinstimmenden Kontext. Wenn er eine Sitzung mit übereinstimmenden Daten findet, nutzt der Server seine Datenbankverbindung zum Verarbeiten der Anforderung. Wenn sich die Sitzung in Verwendung befindet, stellt sie die HTTP-Anforderung in die Warteschlange und aktiviert sie, wenn die Sitzung freigegeben wird.

Die Methode `sa_set_http_option` kann zum Erstellen, Löschen und Ändern von Sitzungs-IDs verwendet werden.

HTTP-Sitzungen erfordern eine spezielle Behandlung für die Verwaltung von Sitzungskriterien. Nur eine Datenbankverbindung ist für die Verwendung durch eine bestimmte SessionID vorhanden, sodass aufeinander folgende Clientanforderungen für die betreffende SessionID vom Server serialisiert werden. Für eine SessionID können bis zu 16 Anforderungen in die Warteschlange gestellt werden. Nachfolgende Anforderungen für die angegebene SessionID werden mit einem Status des Typs 503 `Service Unavailable` zurückgewiesen, wenn die Sitzungswarteschlange bereits die zulässigen 16 Anforderungen enthält.

Bei der ersten Erstellung einer SessionID wird die SessionID sofort vom System registriert. Nachfolgende Anforderungen, die die SessionID ändern oder löschen, werden nur angewandt, wenn die angegebene HTTP-Anforderung beendet ist. Diese Methode führt zu einem konsistenten Verhalten, wenn die Verarbeitung von Anforderungen zu einer Zurücksetzung führt, oder wenn die Anwendung die SessionID löscht und zurücksetzt.

Die aktuelle Sitzung wird gelöscht und durch die laufende Sitzung ersetzt, wenn eine HTTP-Anforderung die SessionID ändert. Die Datenbankverbindung, die von der Sitzung in den Cache geschrieben wird, wird in den neuen Sitzungskontext verschoben, und alle Statusdaten, wie die temporären Tabellen und erstellten Variablen, werden beibehalten.

Ein vollständiges Beispiel für die Verwendung von HTTP-Sitzungen finden Sie in `%SQLANYSAMPI6%\SQLAnywhere\HTTP\session.sql`.

Hinweis

Veraltete Sitzungen sollten gelöscht werden und es sollte ein sinnvoller Timeout eingestellt werden, damit die Anzahl ausstehender Verbindungen möglichst gering gehalten wird, denn jede Verbindung der Clientanwendung nimmt einen Lizenzplatz in Anspruch. Verbindungen, die HTTP-Sitzungen zugeordnet sind, behalten ihren Zugriff auf die Serverdatenbank für die Dauer der Verbindung bei.

Weitere Hinweise zur Lizenzierung in SQL Anywhere finden Sie unter <http://www.sybase.com/detail?id=1056242>.

Siehe auch

- „sa_set_http_option-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_set_http_header-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CONNECTION_PROPERTY-Funktion [System]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Liste der Verbindungseigenschaften“ [*SQL Anywhere Server - Datenbankadministration*]

HTTP-Sitzungen erstellen

Sitzungen können unter Verwendung der SessionID-Option in der Systemprozedur sa_set_http_option erstellt werden. Eine Sitzungs-ID kann durch eine beliebige Zeichenfolge die nicht NULL ist, definiert sein.

Die Verwaltung des Sitzungsstatus wird durch URLs und Cookies unterstützt. Auf HTTP-Sitzungen kann über HTTP-Cookies, über die URL einer GET-Anforderung oder vom Hauptteil einer POST-Anforderung (x-www-form-urlencoded) aus zugegriffen werden. Folgende URL nutzt beispielsweise die Datenbankverbindung **XYZ**, wenn sie ausgeführt wird:

```
http://localhost/sa_svc?SESSIONID=XYZ
```

Die Anforderung wird als standardmäßige sitzungsfreie Anforderung verarbeitet, wenn keine Datenbankverbindung **XYZ** vorhanden ist.

Beispiel

Der folgende Code veranschaulicht die Erstellung eines **RAW**-Webdienstes, der Sitzungen erstellt und löscht. Eine Verbindungsbereichsvariable mit namens **request_count** wird bei jeder HTTP-Anforderung mit einer gültigen SessionID inkrementiert.

```
CREATE SERVICE mysession
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL mysession_proc();

CREATE PROCEDURE mysession_proc()
BEGIN
  DECLARE body LONG VARCHAR;
  DECLARE hostname LONG VARCHAR;
  DECLARE svcname LONG VARCHAR;
  DECLARE sesid LONG VARCHAR;

  CALL sa_set_http_header ( 'Content-Type', 'text/html' );
  SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
  SELECT CONNECTION_PROPERTY('HttpServiceName') INTO svcname;
  SELECT HTTP_HEADER( 'Host' ) INTO hostname;
  IF HTTP_VARIABLE('delete') IS NOT NULL THEN
    CALL sa_set_http_option( 'SessionID', NULL );
    SET body = '<html><body>Deleted ' || sesid
              || '</BR><a href="http://' || hostname || '/' || svcname ||
'">Start Again</a>';
    SELECT body;
  END IF;
  IF sesid = '' THEN
    SET sesid = set_session_url();
    CREATE VARIABLE request_count INT;
    SET request_count = 0;

    SET body = '<html><body> Created session ID ' || sesid
              || '</br><a href="http://' || hostname || '/' || svcname
              || '?SessionID=' || sesid || '"> Enter into Session</a>';
  ELSE
    SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
    SET request_count = request_count + 1;
    SET body = '<html><body>Session ' || sesid || '</br>'
              || 'created ' || CONNECTION_PROPERTY('SessionCreateTime') || '</
br>'
              || 'last access ' || CONNECTION_PROPERTY('SessionLastTime') ||
'</br>'
              || 'connection ID ' || CONNECTION_PROPERTY('Number') || '</br>'
              || '<h3>REQUEST COUNT is ' || request_count || '</h3><hr></br>'
              || '<a href="http://' || hostname || '/' || svcname
              || '?SessionID=' || sesid || '">Enter into Session</a></br>'
              || '<a href="http://' || hostname || '/' || svcname
              || '?SessionID=' || sesid || '&delete">Delete Session</a>';
  END IF;

  SELECT body;
END;
```

Siehe auch

- „Die URL für die Verwaltung einer Sitzung verwenden“ auf Seite 782

Die URL für die Verwaltung einer Sitzung verwenden

In einem URL-basierten Sitzungsstatusverwaltungs-System stellen die Clientanwendung oder der Webbrowser die Sitzungs-ID in einer URL bereit.

Beispiel

Das folgende Beispiel zeigt die Erstellung einer eindeutigen Sitzungs-ID innerhalb einer SQL-Funktion eines HTTP-Webserver, wobei Sitzungs-IDs nur durch eine URL bereitgestellt werden können:

```
CREATE FUNCTION set_session_url()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
        CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND, tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

Die SessionID wird als eine leere Zeichenfolge dargestellt, wenn session_id für die Verbindung nicht definiert ist, und es wird eine sitzungslose Verbindung hergestellt.

Die Systemprozedur sa_set_http_option gibt einen Fehler zurück, wenn die session_id einer anderen HTTP-Anforderung gehört.

Siehe auch

- „sa_set_http_option-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „sa_set_http_header-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „CONNECTION_PROPERTY-Funktion [System]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Liste der Verbindungseigenschaften“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Inaktive HTTP-Sitzungen erkennen“ auf Seite 783
- „HTTP-Sitzungen löschen oder Sitzungs-IDs ändern“ auf Seite 784
- „Fehlercodereferenz für Webdienste“ auf Seite 832

Cookies für die Verwaltung einer Sitzung verwenden

In einem Cookie-basierten Sitzungsstatusverwaltungs-System stellen die Clientanwendung oder der Webbrowser die Sitzungs-ID in einem HTTP-Cookie-Header anstatt in einer URL bereit. Die Cookie-Sitzungsverwaltung wird in Verbindung mit dem HTTP-Antwortheader 'Set-Cookie' der Systemprozedur sa_set_http_header unterstützt.

Hinweis

Sie können sich nicht auf die Cookie-Zustandsverwaltung verlassen, wenn in der Clientanwendung oder im Webbrowser Cookies deaktiviert werden können. Unterstützung für URL- sowie Cookie-Statusverwaltung wird empfohlen. Wenn Sitzungs-IDs sowohl durch die URL als auch durch ein Cookie bereitgestellt werden, wird die durch die URL bereitgestellte Sitzungs-ID verwendet.

Beispiel

Das folgende Beispiel zeigt die Erstellung einer eindeutigen Sitzungs-ID innerhalb einer SQL-Funktion eines HTTP-Webserver, wobei Sitzungs-IDs durch eine URL oder ein Cookie bereitgestellt werden können:

```
CREATE FUNCTION set_session_cookie()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
        CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND, tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    CALL sa_set_http_header( 'Set-Cookie',
        'sessionid=' || session_id || ';' ||
        'max-age=60;' ||
        'path=/session;' );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

Siehe auch

- „sa_set_http_option-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „sa_set_http_header-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „CONNECTION_PROPERTY-Funktion [System]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Liste der Verbindungseigenschaften“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Inaktive HTTP-Sitzungen erkennen“ auf Seite 783
- „HTTP-Sitzungen löschen oder Sitzungs-IDs ändern“ auf Seite 784

Inaktive HTTP-Sitzungen erkennen

Die Verbindungseigenschaften SessionCreateTime und SessionLastTime können dazu verwendet werden, festzustellen, ob die aktuelle Verbindung in einen Sitzungskontext eingebunden ist. Die HTTP-Anforderung wird nicht innerhalb eines Sitzungskontextes ausgeführt, wenn eine der Abfragen der Verbindungseigenschaft eine leere Zeichenfolge zurückgibt.

Die Verbindungseigenschaft SessionCreateTime stellt eine Metrik für den Zeitpunkt der Erstellung einer bestimmten Sitzung bereit. Sie wird erstmalig definiert, wenn die Systemprozedur sa_set_http_option zur Herstellung der SessionID aufgerufen wird.

Die Verbindungseigenschaft SessionLastTime stellt den Zeitpunkt bereit, zu dem die zuletzt verarbeitete Sitzungsanforderung die Datenbankverbindung bei Beenden der vorherigen Anforderung freigegeben hat. Bei der Erstellung der Sitzung wird eine leere Zeichenfolge zurückgegeben, bis die Erstellieranforderung die Verbindung freigibt.

Hinweis

Sie können die Dauer des Sitzungs-Timeouts mit der Option http_session_timeout anpassen.

Beispiel

Das folgende Beispiel zeigt die Sitzungserkennung unter Verwendung der Verbindungseigenschaften SessionCreateTime und SessionLastTime:

```
SELECT CONNECTION_PROPERTY( 'sessioncreatetime' ) INTO ses_create;  
SELECT CONNECTION_PROPERTY( 'sessionlasttime' ) INTO ses_last;
```

Siehe auch

- „CONNECTION_PROPERTY-Funktion [System]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Liste der Verbindungseigenschaften“ [*SQL Anywhere Server - Datenbankadministration*]
- „HTTP-Sitzungen erstellen“ auf Seite 780
- „HTTP-Sitzungen löschen oder Sitzungs-IDs ändern“ auf Seite 784
- „http_session_timeout“ [*SQL Anywhere Server - Datenbankadministration*]

HTTP-Sitzungen löschen oder Sitzungs-IDs ändern

Das explizite Löschen einer Datenbankverbindung, die innerhalb eines Sitzungskontexts im Cache gehalten wird, führt dazu, dass die Sitzung gelöscht wird. Diese Art der Löschung der Sitzung ist ein Abbruchvorgang. Von der Sitzungswarteschlange freigegebene Anforderungen haben einen Abbruchstatus. Diese Aktion stellt sicher, dass alle noch anstehenden Anforderungen der Sitzung beendet werden. Auf ähnliche Weise führt auch das Herunterfahren des Servers oder der Datenbank zum Abbruch aller Datenbankverbindungen.

Eine Sitzung kann gelöscht werden, indem die SessionID-Option in der sa_set_http_option-Systemprozedur auf NULL oder eine leere Zeichenfolge gesetzt wird.

Der folgende Code kann für das Löschen von Sitzungen verwendet werden:

```
CALL sa_set_http_option( 'SessionID', null );
```

Wenn eine HTTP-Sitzung gelöscht oder die SessionID geändert wird, werden alle ausstehenden HTTP-Anforderungen in der Sitzungswarteschlange freigegeben und ihre Ausführung außerhalb eines Sitzungskontextes zugelassen. Die ausstehenden Anforderungen verwenden nicht dieselbe Datenbankverbindung wieder.

Eine Sitzungs-ID kann nicht auf eine bereits vorhandene Sitzungs-ID eingestellt werden. Ausstehende Anforderungen, die sich auf die alte SessionID beziehen, werden zur Ausführung als sitzungslose Anforderungen freigegeben, wenn sich die SessionID geändert hat. Nachfolgende Anforderungen, die sich auf die neue SessionID beziehen, verwenden die durch die alte SessionID instanzierte Datenbank wieder.

Die folgenden Bedingungen gelten beim Löschen oder Ändern einer HTTP-Sitzung:

- Das Verhalten ist unterschiedlich, je nachdem, ob die aktuelle Anforderung eine Sitzung geerbt hat, wodurch eine zu einer Sitzung gehörende Datenbankverbindung erworben wurde, oder ob eine sitzungslose Anforderung eine neue Sitzung instanziiert hat. Wenn die Anforderung als sitzungslose Anforderung begonnen hat, findet sofort eine Sitzungslöschung oder -erstellung statt. Wenn die Anforderung eine Sitzung geerbt hat, erfolgt eine Änderung im Sitzungsstatus, wie etwa das Löschen der Sitzung oder das Ändern der SessionID erst nach Beenden der Anforderung und der Festschreibung

ihrer Änderungen. Der Unterschied im Verhalten bezieht sich auf die Verarbeitung von Anomalien, die auftreten können, wenn der Client gleichzeitige Anforderungen stellt und dabei die gleiche SessionID verwendet.

- Das Ändern der Sitzungs-ID in den SessionID-Wert der aktuellen Sitzung (ohne laufende Sitzung) ist kein Fehler und hat keine substanzielle Auswirkung.
- Das Ändern der Sitzungs-ID in den SessionID-Wert, der von einer anderen HTTP-Anforderung verwendet wird, führt zu einem Fehler.
- Das Ändern einer Sitzung, wenn eine Änderung bereits wartet, führt dazu, dass die laufende Sitzung gelöscht und eine neue laufende Sitzung erstellt wird. Die laufende Sitzung wird erst dann aktiviert, wenn die Anforderung erfolgreich beendet wurde.
- Ein Ändern einer Sitzung mit einer laufenden Sitzung zurück in ihre ursprüngliche SessionID führt dazu, dass die laufende Sitzung ohne jegliche Änderung an der aktuellen Sitzung gelöscht wird.

Siehe auch

- „[CONNECTION_PROPERTY-Funktion \[System\]](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „[Liste der Verbindungseigenschaften](#)“ [*SQL Anywhere Server - Datenbankadministration*]
- „[sa_set_http_option-Systemprozedur](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „[Inaktive HTTP-Sitzungen erkennen](#)“ auf Seite 783

HTTP-Sitzungsadministration

Eine von einer HTTP-Anforderung erstellte Sitzung wird immer sofort instanziiert, sodass alle nachfolgenden HTTP-Anforderungen, die diesen Sitzungskontext erfordern, von der Sitzung in eine Warteschlange gestellt werden.

In diesem Beispiel kann ein lokaler Hostclient auf die Sitzung mit der angegebenen Sitzungs-ID, session_63315422814117, zugreifen, die innerhalb der Datenbank, *DB-Name*, die ihrerseits den Dienst session_service mit der folgenden URL ausführt, sobald die Sitzung auf dem Server mit der Prozedur sa_set_http_option erstellt wurde.

```
http://localhost/dbname/session_service?sessionid=session_63315422814117
```

Eine Webanwendung kann eine Möglichkeit zur Verfolgung der Nutzung der aktiven Sitzung auf dem HTTP-Webserver erfordern. Sitzungsdaten können mit dem Funktionsaufruf NEXT_CONNECTION gefunden werden, der die aktiven Datenbankverbindungen durchläuft und sitzungsbezogene Eigenschaften, wie etwa SessionID, überprüft.

Die folgenden SQL-Anweisungen zeigen die Verfolgung einer aktiven Sitzung:

```
CREATE VARIABLE conn_id LONG VARCHAR;
CREATE VARIABLE the_sessionID LONG VARCHAR;
SELECT NEXT_CONNECTION( NULL, NULL ) INTO conn_id;
conn_loop:
  LOOP
    IF conn_id IS NULL THEN
      LEAVE conn_loop;
    END IF;
```

```
SELECT CONNECTION_PROPERTY( 'SessionID', conn_id )
    INTO the_sessionID;
IF the_sessionID != '' THEN
    PRINT 'conn_id = %1!', SessionID = %2!', conn_id, the_sessionID;
ELSE
    PRINT 'conn_id = %1!', conn_id;
END IF;
SELECT NEXT_CONNECTION( conn_id, NULL ) INTO conn_id;
END LOOP conn_loop;
PRINT '\n';
```

Im Meldungsfenster des Datenbankservers sehen Sie nun Daten, die der folgenden Ausgabe ähneln:

```
conn_id = 30
conn_id = 29, SessionID = session_63315442223323
conn_id = 28, SessionID = session_63315442220088
conn_id = 25, SessionID = session_63315441867629
```

Das explizite Löschen einer Verbindung, die einer Sitzung gehört, führt dazu, dass die Verbindung geschlossen und die Sitzung gelöscht wird. Wenn die zu löschende Sitzung aktuell aktiv ist und eine HTTP-Anforderung bedient, wird die Anforderung für die Löschung markiert und sie erhält ein Abbruchsignal zum Beenden der Anforderung. Wenn die Anforderung beendet wird, wird die Sitzung gelöscht und die Verbindung geschlossen. Das Löschen einer Sitzung kann dazu führen, dass ausstehende Anforderungen in der Warteschlange der betreffenden Sitzung in eine neue Warteschlange gestellt werden.

Falls die Verbindung aktuell nicht aktiv ist, wird die Sitzung für die Löschung markiert und an den Anfang der Sitzungs-Timeoutwarteschlange gestellt. Die Sitzung und die Verbindung werden beim nächsten Timeoutzyklus gelöscht (normalerweise innerhalb von 5 Sekunden). Für die Löschung markierte Sitzungen können von keiner neuen HTTP-Anforderung verwendet werden.

Alle Sitzungen gehen verloren, wenn die Datenbank gestoppt wird.

Siehe auch

- „sa_set_http_option-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „HTTP-Sitzungen erstellen“ auf Seite 780
- „HTTP-Sitzungen löschen oder Sitzungs-IDs ändern“ auf Seite 784
- „NEXT_CONNECTION-Funktion [System]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Fehlercodes in Verbindung mit HTTP-Sitzungen

Der Fehler 503 `Service Unavailable` wird ausgegeben, wenn eine neue Anforderung versucht, auf eine Sitzung zuzugreifen, für die bereits mehr als 16 Anforderungen ausstehen, oder wenn bei der Erstellung der Warteschlange ein Fehler aufgetreten ist.

Der Fehler 403 `Forbidden` wird ausgegeben, wenn die Client-IP-Adresse oder der Hostname nicht mit dem entsprechenden Wert des Erstellers der Sitzung übereinstimmt.

Eine Anforderung, die eine nicht vorhandene Sitzung angibt, generiert nicht automatisch einen Fehler. Die Webanwendung muss diese Bedingung erkennen (durch die Überprüfung der Verbindungseigenschaften `SessionID`, `SessionCreateTime` oder `SessionLastTime`) und die geeignete Aktion durchführen.

Siehe auch

- „Fehlercodereferenz für Webdienste“ auf Seite 832

Hinweise zur Zeichensatzkonvertierung

Die Zeichensatzkonvertierung wird bei ausgehenden Ergebnismengen des Typs text standardmäßig automatisch durchgeführt. Ergebnismengen anderer Typen, wie z.B. Binärobjekte, sind davon nicht betroffen. Der Zeichensatz der Anforderung wird in den Zeichensatz des HTTP-Webserver, und die Ergebnismenge in den Zeichensatz der Clientanwendung konvertiert. Der Server verwendet den ersten in der Anforderung aufgeführten geeigneten Zeichensatz, wenn mehrere Zeichensätze aufgelistet sind.

Die Zeichensatzkonvertierung kann durch Festlegen der HTTP-Option 'CharsetConversion' Option der Systemprozedur sa_set_http_option aktiviert oder deaktiviert.

Das folgende Beispiel veranschaulicht die Deaktivierung der automatischen Zeichensatzkonvertierung:

```
CALL sa_set_http_option('CharsetConversion', 'OFF');
```

Sie können mit der Option "AcceptCharset" der Systemprozedur sa_set_http_option die Voreinstellung für die Zeichensatzkodierung festlegen, wenn die Zeichensatzkonvertierung aktiviert ist.

Das folgende Beispiel veranschaulicht die Festlegung der Voreinstellung der Zeichensatzkodierung für den Webdienst auf ISO-8859-5, wenn unterstützt. Andernfalls lautet die Einstellung UTF-8:

```
CALL sa_set_http_option('AcceptCharset', 'iso-8859-5, utf-8');
```

Zeichensätze werden von der Servervoreinstellung priorisiert, aber die Auswahl berücksichtigt auch die Accept-Charset-Kriterien des Clients. Der vom Client bevorzugte Zeichensatz, der auch von dieser Option festgelegt wird, wird verwendet.

Siehe auch

- „sa_set_http_option-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Hinweise zu Cross-Site-Scripting

Wenn Sie Ihre Webanwendung entwickeln, müssen Sie dafür sorgen, dass sie nicht für Cross-Site-Scripting anfällig ist. Eine solche Sicherheitslücke tritt auf, wenn ein Angreifer versucht, ein Skript in Ihre Webseite einzufügen.

Es wird dringend empfohlen, dass Anwendungsentwickler und Datenbankadministratoren ihren Webanwendungscode auf mögliche Sicherheitslücken überprüfen, bevor er in die Produktionsumgebung übernommen wird. Das Open Web Application Security Project (<https://www.owasp.org>) enthält weitere Hinweise zum Sichern Ihrer Webanwendung.

Hier sehen Sie ein Beispiel für eine Sicherheitslücke durch Cross-Site-Scripting. Wenn Ihre Webanwendung die Seitenumleitung unterstützt, muss sie vor dem Umleiten die URL validieren. Wenn Ihre Webanwendung beispielsweise eine Login-Funktion unterstützt, die den Benutzer auf die Ausgangsseite zurückleitet, muss sie überprüfen, ob die Seite, auf die der Benutzer umgeleitet wird, den Benutzer von der Website wegleitet. Das folgende Beispiel zeigt eine mögliche bösartige Umleitung.

<https://www.mysite.com/login?referer=http://www.badsite.com/index.html>

Webdienst-Systemprozeduren

Folgende Systemprozeduren werden für Webdienste verwendet:

- „sa_http_header_info-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_http_php_page-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_http_php_page_interpreted-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_http_variable_info-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_set_http_header-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_set_http_option-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „sa_set_soap_header-Systemprozedur“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Siehe auch

- „Webdienstfunktionen“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „SQL Anywhere als HTTP-Webserver“ auf Seite 755
- „Datenbankserveroption -xs“ [*SQL Anywhere Server - Datenbankadministration*]

Webdienstfunktionen

Webdienstfunktionen sind bei der Verarbeitung von HTTP- und SOAP-Anforderungen innerhalb von Webdiensten hilfreich.

Folgende Funktionen sind verfügbar:

- „HTML_DECODE-Funktion [Verschiedene]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTML_ENCODE-Funktion [Verschiedene]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_BODY-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_DECODE-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_ENCODE-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_HEADER-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_RESPONSE_HEADER-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „HTTP_VARIABLE-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_HTTP_HEADER-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_HTTP_RESPONSE_HEADER-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_HTTP_VARIABLE-Funktion [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_SOAP_HEADER-Funktion [SOAP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „SOAP_HEADER-Funktion [SOAP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Es stehen auch viele Systemprozeduren für Webdienste zur Verfügung.

Siehe auch

- „SQL Anywhere als HTTP-Webserver“ auf Seite 755
- „Datenbankserveroption -xs“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Webdienst-Systemprozeduren“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Verbindungseigenschaften von Webdiensten

Verbindungseigenschaften von Webdiensten können Datenbankeigenschaften sein, auf die mit der Funktion CONNECTION_PROPERTY zugegriffen werden kann.

Verwenden Sie folgende Syntax zum Speichern eines Werts einer Verbindungseigenschaft vom HTTP-Server in einer lokalen Variablen in einer SQL-Funktion oder -Prozedur:

```
SELECT CONNECTION_PROPERTY('connection-property-name') INTO variable_name;
```

Im Folgenden finden Sie eine Liste der nützlichen Verbindungseigenschaften für HTTP-Anforderungen in der Laufzeitumgebung, die allgemein für Webdienstanwendungen verwendet werden:

- **HttpServiceName** Gibt den Ursprung des Dienstnamens bei einer Web-Anwendung zurück.
- **AuthType** Gibt den Typ der Authentifizierung zurück, der bei der Verbindungsherstellung verwendet wird.
- **ServerPort** Gibt die TCP/IP-Portnummer des Datenbankservers zurück, oder "0".
- **ClientNodeAddress** Gibt den Knoten für den Client in einer Client/Server-Verbindung zurück.
- **ServerNodeAddress** Gibt den Knoten für den Server in einer Client/Server-Verbindung zurück.
- **BytesReceived** Gibt die Anzahl von Bytes zurück, die während der Client/Server-Kommunikation empfangen wurden.

Siehe auch

- „Liste der Verbindungseigenschaften“ [[SQL Anywhere Server - Datenbankadministration](#)]
- HttpServiceName-Verbindungseigenschaft [[SQL Anywhere Server - Datenbankadministration](#)]
- AuthType-Verbindungseigenschaft [[SQL Anywhere Server - Datenbankadministration](#)]
- ServerPort-Verbindungseigenschaft [[SQL Anywhere Server - Datenbankadministration](#)]
- ClientNodeAddress-Verbindungseigenschaft [[SQL Anywhere Server - Datenbankadministration](#)]
- ServerNodeAddress-Verbindungseigenschaft [[SQL Anywhere Server - Datenbankadministration](#)]
- BytesReceived-Verbindungseigenschaft [[SQL Anywhere Server - Datenbankadministration](#)]

Webdienstoptionen

Webdienstoptionen steuern verschiedene Aspekte des Verhaltens von HTTP-Servern.

Verwenden Sie die folgende Syntax zum Erstellen einer öffentlichen Option auf einem HTTP-Server:

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

Im Folgenden finden Sie eine Liste von Optionen, die in Verbindung mit HTTP-Servern häufig für die Anwendungskonfiguration verwendet werden:

- **http_connection_pool_basesize** Gibt die nominale Schwellenwertgröße von Datenbankverbindungen an.
- **http_connection_pool_timeout** Gibt die maximale Dauer an, über die eine nicht verwendete Verbindung im Verbindungspool aufrecht erhalten werden kann.
- **http_session_timeout** Gibt den Standardwert für den Timeout in Minuten an, über den die HTTP-Sitzung bei Inaktivität bestehen bleibt.
- **request_timeout** Steuert die maximale Zeit, die eine einzelne Anforderung benötigen darf.
- **webservice_namespace_host** Gibt den Hostnamen an, der als der XML-Namespaces in der Spezifikation für DISH-Dienste verwendet werden soll.

Siehe auch

- „Datenbankoptionen“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „http_connection_pool_basesize-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „http_connection_pool_timeout-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „http_session_timeout“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „request_timeout-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „webservice_namespace_host-Option“ [[SQL Anywhere Server - Datenbankadministration](#)]

Auf einem SQL Anywhere-HTTP-Webserver navigieren

Verfügbare URL-Namen werden dadurch definiert, wie Ihre Webdienste benannt und konzipiert wurden. Jeder Webdienst stellt ihren eigenen Webinhalt bereit. Dieser Inhalt wird in der Regel durch benutzerdefinierte Funktionen und Prozeduren in Ihrer Datenbank generiert, aber der Inhalt kann auch mit einer URL generiert werden, die eine SQL-Anweisung angibt. Alternativ dazu oder in Verbindung damit können Sie auch den **root**-Webdienst definieren, der alle nicht von einem eigens dafür vorgesehenen Webdienst verarbeiteten HTTP-Anforderungen verarbeitet. Der **root**-Webdienst prüft normalerweise die URL und die Header der Anforderung, um zu entscheiden, wie die Verarbeitung der Anforderung erfolgen soll.

URLs geben in eindeutiger Weise Ressourcen an, wie etwa HTML-Inhalt, der über HTTP- oder sichere HTTPS-Anforderungen zur Verfügung steht. In diesem Abschnitt wird erklärt, wie Sie die URL-Syntax in Ihrem Webbrowser formatieren müssen, damit Sie auf die auf Ihrem SQL Anywhere-HTTP-Webserver definierten Webdienste zugreifen können.

Hinweis

Die Informationen in diesem Abschnitt gelten für HTTP-Webserver, die allgemeine HTTP-Webdiensttypen, wie etwa RAW-, XML- sowie HTML- und DISH-Dienste verwenden. SOAP-Anforderungen können nicht über einen Browser gesendet werden. JSON-Dienste geben Ergebnismengen für die Verarbeitung durch Webdienstanwendungen, die AJAX verwenden, zurück.

Syntax

```
{http | https} ://host-name [:port-number] [/ dbn] /service-name [/ path-name | ?url-query]
```

Parameter

- **host-name und port-number** Bestimmt den Standort des Webserver und (optional) die Portnummer, wenn diese nicht mit der standardmäßigen HTTP- oder HTTPS-Portnummer übereinstimmt. Der *Hostname* kann die IP-Adresse des Computers sein, auf dem der Webserver ausgeführt wird. Die *Portnummer* muss mit der beim Start des Webserver verwendeten Portnummer übereinstimmen.

Siehe „Datenbankserveroption -xs“ [[SQL Anywhere Server - Datenbankadministration](#)] und „ServerPort-Protokolloption (PORT)“ [[SQL Anywhere Server - Datenbankadministration](#)].

- **dbn** Gibt den Namen einer Datenbank an. Diese Datenbank muss auf dem Webserver laufen und Webdienste enthalten.

Sie müssen *dbn* nicht unbedingt angeben, wenn der Webserver nur eine Datenbank ausführt oder wenn der Datenbankname für den angegebenen HTTP/HTTPS-Listener der Protokolloption angegeben wurde.

Siehe „Datenbankserveroption -xs“ [[SQL Anywhere Server - Datenbankadministration](#)] und „DatabaseName-Protokolloption (DBN)“ [[SQL Anywhere Server - Datenbankadministration](#)].

- **service-name** Gibt den Namen des Webdienstes an, auf den der Zugriff erfolgen soll. Dieser Webdienst muss in der durch *dbn* angegebenen Datenbank vorhanden sein. Schrägstriche (/) sind beim Erstellen oder Ändern eines Webdienstes zulässig. Sie können sie daher als Teil des Dienstnamens *service-name* verwenden. SQL Anywhere stimmt den Rest des URLs mit den festgelegten Diensten ab.

Die Clientanforderung wird verarbeitet, wenn kein *service-name* angegeben und der **root**-Webdienst definiert ist. Ein Fehler des Typs 404 Not Found wird zurückgegeben, wenn der Server keinen geeigneten Dienst für die Verarbeitung der Anforderung ermitteln kann. Wenn der **root**-Webdienst vorhanden ist und die Anforderung basierend auf den URL-Kriterien nicht verarbeiten kann, ist er darüber hinaus auch dafür zuständig den Fehler 404 Not Found zu generieren.

- **path-name** Nach dem Auflösen des Dienstnamens kann auf den verbleibenden durch Schrägstriche getrennten Pfad über eine Webdienstprozedur zugegriffen werden. Wenn der Dienst mit der Einstellung URL ON erstellt wurde, kann auf den gesamten Pfad mit einer eigenen HTTP-Variablen mit dem Namen **URL** zugegriffen werden. Wenn der Dienst mit der Einstellung URL ELEMENTS erstellt wurde, kann auf jedes Pfadelement mit den eigens dafür vorgesehenen HTTP-Variablen **URL1** bis **URL10** zugegriffen werden.

Pfadelementvariablen können innerhalb der Parameterdeklaration der Dienstanweisungsdefinition als Hostvariable definiert werden. Alternativ dazu bzw. darüber hinaus kann auf HTTP-Variablen auch über einen Aufruf der Funktion HTTP_VARIABLE aus einer gespeicherten Prozedur heraus zugegriffen werden.

Das folgende Beispiel veranschaulicht die zum Erstellen eines Webdienstes verwendete SQL-Anweisung, wobei die URL-Klausel auf ELEMENTS gesetzt ist:

```
CREATE SERVICE TestWebService
  TYPE 'HTML'
  URL ELEMENTS
  AUTHORIZATION OFF
  USER DBA
  AS CALL TestProcedure ( :url1, :url2 );
```

Dieser **TestWebService**-Webdienst ruft eine Prozedur auf, die explizit auf die Hostvariablen **url1** und **url2** verweist.

Sie können mithilfe der folgenden URL auf diesen Webdienst zugreifen, wobei davon ausgegangen wird, dass **TestWebService** in der Datenbank **demo** von **localhost** aus über den Standardport ausgeführt wird:

```
http://localhost/demo/TestWebService/Assignment1/Assignment2/Assignment3
```

Diese URL greift auf **TestWebService** zu, der **TestProcedure** ausführt und den Wert für **Assignment1** der Variablen **url1** sowie den Wert für **Assignment2** der Variablen **url2** zuweist. Optional kann **TestProcedure** unter Verwendung der Funktion **HTTP_VARIABLE** auf andere Pfadelemente zugreifen. Beispiel: Der Funktionsaufruf **HTTP_VARIABLE('url3')** gibt **Assignment3** zurück.

Weitere Hinweise zu URL ELEMENTS und URL-basierten Variablen finden Sie unter „[CREATE SERVICE-Anweisung \[HTTP-Webdienst\]](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*].

- **url-query** Eine HTTP GET-Anforderung kann auf einen Pfad mit einer Abfragekomponente, die HTTP-Variablen festlegt, folgen. Auf ähnliche Weise kann der Hauptteil einer POST-Anforderung mit einem standardmäßigen application/x-www-form-urlencoded-Content-Type HTTP-Variablen im Anforderungshauptteil übergeben. In beiden Fällen werden die HTTP-Variablen als Name/Wert-Paare übergeben, wobei der Variablenname und der zugehörige Wert durch ein Gleichheitszeichen voneinander getrennt sind. Variablen werden durch ein kaufmännisches Und gekennzeichnet.

HTTP-Variablen können in der Parameterliste der Dienstanweisung explizit als Hostvariablen deklariert werden. Alternativ dazu kann mit der Funktion **HTTP_VARIABLE** aus der gespeicherten Prozedur der Dienstanweisung heraus auf sie zugegriffen werden.

Die folgende SQL-Anweisung erstellt beispielsweise einen Webdienst, der zwei Hostvariablen erfordert. Hostvariablen werden durch einen vorangestellten Doppelpunkt (:) gekennzeichnet.

```
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

Unter der Annahme, dass **ShowSalesOrderDetail** in der Demodatenbank von **localhost** aus über den Standardport ausgeführt wird, können Sie mithilfe der folgenden URL auf den Webdienst zugreifen:

```
http://localhost/demo/ShowSalesOrderDetail?customer_id=101&product_id=300
```

Diese URL greift auf **ShowSalesOrderDetail** zu und weist **customer_id** einen Wert von 101 und **product_id** einen Wert von 300 zu. Die sich ergebende Ausgabe wird in Ihrem Webbrowser im HTML-Format angezeigt.

Bemerkungen

Der Webbrowser fordert zur Eingabe eines Benutzernamens und Kennworts auf, wenn dies für die Herstellung einer Verbindung mit dem Server erforderlich ist. Der Browser verschlüsselt dann die Benutzereingabe mit base64 innerhalb eines Authorization-Anforderungsheaders und sendet die Anforderung weiter.

Wenn Ihre URL-Klausel für den Webdienst die Einstellung ON oder ELEMENTS hat, können die URL-Syntaxeigenschaften von *path-name* und *url-query* gleichzeitig verwendet werden, sodass auf den Webdienst mit einer von mehreren verschiedenen Formatierungsoptionen zugegriffen werden kann. Wenn Sie diese Syntaxeigenschaften gleichzeitig verwenden, muss zuerst das *decimal*-Format, gefolgt vom *url-query*-Format verwendet werden.

Im folgenden Beispiel erstellt diese SQL-Anweisung einen Webdienst, in dem die URL-Klausel auf ON gesetzt ist und der die Variable **url** definiert:

```
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :product_id, :url );
```

Nachstehend finden Sie ein Beispiel für eine Liste von zulässigen URLs, die der Variablen **url** einen Wert von 101 und der Variablen **product_id** einen Wert von 300 zuweisen:

- `http://localhost:80/demo/ShowSalesOrderDetail2/101?product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2?url=101&product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2?product_id=300&url=101`

Wenn ein Hostvariablenname im Kontext von *path-name* und *url-query* mehr als einmal zugewiesen wird, hat die zuletzt vorgenommene Zuweisung jeweils immer Vorrang. Zum Beispiel weisen die folgenden Beispiel-URLs **url** einen Wert von 101 und **product_id** einen Wert von 300 zu:

- `http://localhost:80/demo/ShowSalesOrderDetail2/302?url=101&product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2/String?product_id=300&url=101`

Siehe auch

- „CREATE SERVICE-Anweisung [HTTP-Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Zugriff auf vom Client bereitgestellte HTTP-Variablen und -Header“ auf Seite 772
- „HTTP_VARIABLE-Funktion [Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „root-Webdienste erstellen und anpassen“ auf Seite 767

Beispiel

Die folgende URL-Syntax wird für den Zugriff auf einen Webdienst mit dem Namen **gallery_image** verwendet, der in einer Datenbank mit dem Namen **demo** auf einem lokalen HTTP-Server über den

Standardport und unter der Annahme, dass der Dienst **gallery_image** mit **URL ON** definiert ist, ausgeführt:

```
http://localhost/demo/gallery_image/sunset.jpg
```

Die URL scheint von einem herkömmlichen Webserver eine Grafikdatei in einem Verzeichnis anzufordern, aber sie greift auf den Dienst **gallery_image** zu, wobei **sunset.jpg** als Eingabeparameter für einen HTTP-Webserver angegeben ist.

Die folgende SQL-Anweisung zeigt, wie der gallery_image-Dienst auf dem HTTP-Server so definiert wurde, dass er angemessen auf dieses Verhalten reagieren kann:

```
CREATE SERVICE gallery_image
  TYPE 'RAW'
  URL ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL gallery_image ( :url );
```

Der Dienst **gallery_image** ruft eine Prozedur mit demselben Namen auf und gibt die vom Client bereitgestellte URL weiter. Eine Beispielimplementierung einer **gallery_image**-Prozedur, auf die durch diese Webdienstdefinitionen zugegriffen werden kann, finden Sie in *%SQLANYSAMPI6%\SQLAnywhere\HTTP\gallery.sql*.

Zugriff auf Webdienste mithilfe von Webclients

SQL Anywhere kann als Webclient für den Zugriff auf Webdienste eingesetzt werden, die von einem SQL Anywhere-Webserver oder einem Webserver anderer Hersteller wie etwa Apache oder IIS gehostet werden.

Zusätzlich zur Verwendung von SQL Anywhere als Webclient bieten SQL Anywhere-Webdienste Clientanwendungen eine Alternative zu herkömmlichen Schnittstellen wie JDBC und ODBC. Sie können einfach bereitgestellt werden, da keine weiteren Komponenten erforderlich sind. Es kann darauf von in einer Vielzahl an Sprachen (einschließlich Skriptsprachen wie Perl und Python) geschriebenen Clientanwendungen von mehreren Plattformen zugegriffen werden.

Kurzeinführung zur Verwendung von SQL Anywhere als Webclient

Dieser Abschnitt veranschaulicht die Verwendung von SQL Anywhere als Webclientanwendung, über die eine Verbindung zu einem SQL Anywhere-HTTP-Server hergestellt sowie auf einen allgemeinen HTTP-Webdienst zugegriffen werden kann. Sie veranschaulicht nicht den vollen Umfang der Fähigkeiten des SQL Anywhere-Webclients. Es sind viele SQL Anywhere-Webclient-Funktionen verfügbar, die unter diesem Thema nicht alle behandelt werden können.

Sie können SQL Anywhere-Webclientanwendungen entwickeln, die eine Verbindung mit jedem beliebigen Online-Webservertyp herstellen, aber in diesem Abschnitt wird davon ausgegangen, dass Sie einen lokalen SQL Anywhere-HTTP-Server an Port 8082 gestartet haben und eine Verbindung mit einem

Webdienst namens **SampleHTMLService** herstellen möchten, der mit den folgenden SQL-Anweisungen erstellt wurde:

```
CREATE SERVICE SampleHTMLService
  TYPE 'HTML'
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
  RESULT(ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR)
  BEGIN
    SELECT i, f, s;
  END;
```

Führen Sie die folgenden Aufgaben aus, um eine SQL Anywhere-Webclientanwendung zu erstellen:

1. Führen Sie den folgenden Befehl aus, um eine SQL Anywhere-Clientdatenbank zu erstellen, falls noch keine existiert:

```
dbinit -dba DBA,sql client-database-name
```

Ersetzen Sie *Clientdatenbank-Name* durch einen neuen Namen für Ihre Clientdatenbank.

2. Führen Sie den folgenden Befehl aus, um die Clientdatenbank zu starten:

```
dbsrv16 client-database-name.db
```

3. Führen Sie den folgenden Befehl aus, um eine Verbindung zur Clientdatenbank mithilfe von Interactive SQL herzustellen:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=client-database-name"
```

4. Erstellen Sie eine neue Clientprozedur, die eine Verbindung zum Webdienst **SampleHTMLService** mithilfe der folgenden SQL-Anweisung herstellen:

```
CREATE PROCEDURE client_post(f REAL, i INTEGER, s VARCHAR(16), x
  VARCHAR(16))
  URL 'http://localhost:8082/SampleHTMLService'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SATest';
```

5. Führen Sie die folgende SQL-Anweisung aus, um die Clientprozedur aufzurufen und eine HTTP-Anforderung an den Webserver zu senden:

```
CALL client_post(3.14, 9, 's varchar', 'x varchar');
```

client_post erstellt eine HTTP POST-Anforderung ähnlich der folgenden Ausgabe:

```
POST /SampleHTMLService HTTP/1.0
ASA-Id: eal746b01cd0472eb4f0729948db60a2
User-Agent: SATest
Accept-Charset: windows-1252, UTF-8, *
Date: Wed, 9 Jun 2010 21:55:01 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=windows-1252
Content-Length: 58
```

```
&f=3.1400001049041748&i=9&s=s%20varchar&x=x%20varchar
```

Der Webdienst **SampleHTMLService** auf dem Webserver extrahiert die Parameterwerte für **i**, **f** und **s** aus der POST-Anforderung und übergibt sie als Parameter an die Prozedur **sp_echo**. Parameterwert **x** wird ignoriert. Die Prozedur **sp_echo** erstellt eine Ergebnismenge, die an den Webdienst zurückgegeben wird. Eine Vereinbarung der Parameternamen zwischen dem Client und dem Webserver ist für die richtige Zuordnung von entscheidender Bedeutung.

Der Webdienst erstellt die Antwort, die an den Client zurückgesendet wird. Die in Interactive SQL angezeigte Ausgabe sollte ähnlich wie die folgende Ausgabe sein:

Attribut	Wert
Status	HTTP/1.1 200 OK
Body	<pre><html> <head> <title>/SampleHTMLService</title></head> <body> <h3>/SampleHTMLService</h3> <table border=1> <tr class="header"><th>ret_i</th> <th>ret_f</th> <th>ret_s</th> </tr> <tr><td>9</td><td>3.1400001049041748</td><td>s varchar</td></pre>
Date	Wed, 09 Jun 2010 21:55:01 GMT
Connection	close
Expires	Wed, 09 Jun 2010 21:55:01 GMT
Content-Type	text/html; charset=windows-1252
Server	SQLAnywhere/16.0

Siehe auch

- „Kurzeinführung zur Verwendung von SQL Anywhere als HTTP-Webserver“ auf Seite 755
- „Webdienstanwendungen auf einem HTTP-Webserver entwickeln“ auf Seite 770
- „CREATE PROCEDURE-Anweisung [Webdienste]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Kurzeinführung für den Zugriff auf einen SQL Anywhere-HTTP-Webserver

Dieser Abschnitt veranschaulicht, wie Sie mithilfe von zwei verschiedenen Typen von Clientanwendungen, Python und C#, auf einen SQL Anywhere-HTTP-Webserver zugreifen können. Sie

veranschaulicht nicht den vollen Umfang der Fähigkeiten der SQL Anywhere-Webdienstanwendung. Es sind viele SQL Anywhere-Webservice-Funktionen verfügbar, die unter diesem Thema nicht alle behandelt werden können.

Sie können SQL Anywhere-Webclientanwendungen entwickeln, die eine Verbindung mit jedem beliebigen Online-Webservertyp herstellen, aber in diesem Handbuch wird davon ausgegangen, dass Sie einen lokalen SQL Anywhere-HTTP-Server an Port 8082 gestartet haben und eine Verbindung mit einem Webdienst namens **SampleXMLService** herstellen möchten, der mit den folgenden SQL-Anweisungen erstellt wurde:

```
CREATE SERVICE SampleXMLService
  TYPE 'XML'
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo2(:i, :f, :s);

CREATE PROCEDURE sp_echo2(i INTEGER, f NUMERIC(6,2), s LONG VARCHAR )
RESULT( ret_i INTEGER, ret_f NUMERIC(6,2), ret_s LONG VARCHAR )
BEGIN
  SELECT i, f, s;
END;
```

Führen Sie die folgenden Aufgaben für den Zugriff auf einen XML-Webdienst mithilfe von C# oder Python aus:

1. Erstellen Sie eine Prozedur, die eine Verbindung mit einem Webdienst auf einem HTTP-Server herstellt.

Schreiben Sie Code, der auf den Webdienst **SampleXMLService** zugreift.

- Für C# verwenden Sie den folgenden Code:

```
using System;
using System.Xml;

public class WebClient
{
    static void Main(string[] args)
    {
        XmlTextReader reader = new XmlTextReader(
            "http://localhost:8082/SampleXMLService?i=5&f=3.14&s=hello");
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element:
                    if (reader.Name == "row")
                    {
                        Console.Write(reader.GetAttribute("ret_i") +
                            " ");
                        Console.Write(reader.GetAttribute("ret_s") +
                            " ");
                        Console.WriteLine(reader.GetAttribute("ret_f"));
                    }
                    break;
            }
        }
    }
}
```

```

        reader.Close();
    }
}

```

Speichern Sie den Code in einer Datei mit dem Namen *DocHandler.cs*.

Um das Programm zu kompilieren, führen Sie den folgenden Befehl an der Eingabeaufforderung aus:

```
csc /out:DocHandler.exe DocHandler.cs
```

- Für Python verwenden Sie den folgenden Code:

```

import xml.sax

class DocHandler( xml.sax.ContentHandler ):
    def startElement( self, name, attrs ):
        if name == 'row':
            table_int = attrs.getValue( 'ret_i' )
            table_string = attrs.getValue( 'ret_s' )
            table_numeric = attrs.getValue( 'ret_f' )
            print('%s %s %s' % ( table_int, table_string,
table_numeric ))

parser = xml.sax.make_parser()
parser.setContentHandler( DocHandler() )
parser.parse('http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello')

```

Speichern Sie den Code in einer Datei mit dem Namen *DocHandler.py*.

2. Führen Sie Vorgänge in der vom HTTP-Server gesendeten Ergebnismenge aus.

- Für C# führen Sie den folgenden Befehl aus:

```
DocHandler
```

- Für Python führen Sie den folgenden Befehl aus:

```
python DocHandler.py
```

Die Anwendung zeigt die folgende Ausgabe an:

```
5 hello 3.14
```

Siehe auch

- [„Webdienstanwendungen auf einem HTTP-Webserver entwickeln“ auf Seite 770](#)
- [„Kurzeinführung zur Verwendung von SQL Anywhere als HTTP-Webserver“ auf Seite 755](#)

Webclient-Anwendungsentwicklung

SQL Anywhere-Datenbanken können als Webclientanwendungen für den Zugriff auf Webdienste, die von SQL Anywhere oder auf Webservern anderer Hersteller gehostet werden, agieren. SQL Anywhere-Webclientanwendungen werden erstellt, indem gespeicherte Prozeduren und Funktionen mithilfe von Konfigurationsklauseln geschrieben werden, wie etwa die URL-Klausel, die den Webdienst-Zielendpunkt angibt. Webclient-Prozeduren verfügen über keinen Hauptteil, werden jedoch ansonsten wie jede andere

gespeicherte Prozedur benutzt. Wenn eine Webclient-Prozedur aufgerufen wird, erstellt sie eine abgehende HTTP- oder SOAP-Anforderung. Eine Webclient-Prozedur kann nur abgehende HTTP-Anforderungen an sich selbst stellen; sie kann keinen Localhost-SQL Anywhere-Webdienst auf derselben Datenbank aufrufen.

Detailliertere Beispiele für Webdienstanwendungen finden Sie im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\HTTP`.

Siehe auch

- „SQL-Anweisungen für Webclients“ auf Seite 811

Anforderungen und Empfehlungen für Webclient-Funktionen und -Prozeduren

Webdienstclient-Prozeduren und -Funktionen erfordern die Definition einer URL-Klausel zur Angabe des Webdienst-Endpunkts. Eine Webdienstclient-Prozedur oder -Funktion verfügt über spezielle Klauseln für die Konfiguration, wird aber ansonsten wie jede andere gespeicherte Prozedur oder Funktion verwendet.

Sie können mit den Anweisungen `CREATE PROCEDURE` und `CREATE FUNCTION` Webclient-Funktionen und -Prozeduren zum Senden von SOAP- oder HTTP-Anforderungen an einen Webserver erstellen.

In der folgenden Liste werden die Anforderungen und Empfehlungen für die Erstellung oder Änderung von Webclient-Funktionen und -Prozeduren aufgeführt. Sie können die folgenden Informationen beim Erstellen oder Ändern einer Webclient-Funktion oder Prozedur eingeben:

- Die URL-Klausel für den Webdienst-Endpunkt, die eine absolute URL erfordert. (Erforderlich)
- Die TYPE-Klausel, um festzulegen, ob es sich um eine HTTP- oder SOAP über HTTP-Anforderung handelt. (Empfohlen)
- Ports, die für die Clientanwendung verfügbar sind. (Optional)
- Die HEADER-Klausel für die Angabe von HTTP-Anforderungsheadern. (Optional)
- Die SOAPHEADER-Klausel für die Angabe von SOAP-Headerkriterien innerhalb des SOAP-Anforderungsrahmens. (Optional. Nur für SOAP-Anforderungen)
- Der Namespace-URI. (Nur für SOAP-Anforderungen)

Webclient-URL-Klausel

Sie müssen die Adresse des Webdienst-Endpunkts angeben, damit Ihre Webclient-Funktion oder -Prozedur darauf zugreifen kann. Die URL-Klausel der Anweisungen `CREATE PROCEDURE` und `CREATE FUNCTION` stellt die Webdienst-URL bereit, auf die Sie zugreifen möchten.

HTTP-Dienst-URL angeben

Durch die Angabe eines HTTP-Schemas in der URL-Klausel konfigurieren Sie die Prozedur oder Funktion für nicht sichere Kommunikation unter Verwendung eines HTTP-Protokolls.

Die folgende Anweisung zeigt, wie Sie eine Prozedur erstellen, die Anforderungen an einen Webdienst namens **SampleHTMLService** sendet, der sich in einer Datenbank namens **dbname** befindet, die von einem HTTP-Webserver unter **localhost** auf dem Port 8082 gehostet wird:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
  URL 'http://localhost:8082/dbname/SampleHTMLService'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SATest';
```

Der Datenbankname ist nur erforderlich, wenn der HTTP-Server mehr als eine Datenbank hostet. Sie können **localhost** durch den Hostnamen oder die IP-Adresse des HTTP-Servers ersetzen.

Weitere Hinweise zum Starten eines HTTP-Servers mit diesen Beispielspezifikationen finden Sie unter [„HTTP-Webserver starten“ auf Seite 757](#).

HTTPS-Dienst-URL angeben

Durch die Angabe eines HTTPS-Schemas in der URL-Klausel konfigurieren Sie die Prozedur oder Funktion für eine sichere Kommunikation über Secure Socket Layer (SSL).

Ihre Webclientanwendung benötigt Zugriff auf ein RSA-Serverzertifikat oder das zum Signieren des Serverzertifikats verwendete Zertifikat, um eine sichere HTTPS-Anforderung auszugeben. Das Zertifikat ist für die Clientprozedur erforderlich, um den Server zu authentifizieren und so Mittelsmannangriffe zu verhindern.

Verwenden Sie die CERTIFICATE-Klausel der Anweisungen CREATE PROCEDURE und CREATE FUNCTION zur Authentifizierung des Servers und Einrichtung eines sicheren Datenkanals. Sie können entweder das Zertifikat in eine Datei platzieren und den Dateinamen angeben oder das gesamte Zertifikat als Zeichenfolgewart bereitstellen. Beides zusammen ist nicht möglich.

Die folgende Anweisung zeigt, wie eine Prozedur zum Senden von Anforderungen an einen Webdienst namens **SecureHTMLService** erstellt wird, der sich in einer Datenbank namens **dbname** auf einem HTTPS-Server unter **localhost** auf dem Port 8082 befindet:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
  URL 'HTTPS://localhost:8082/dbname/SecureHTMLService'
  CERTIFICATE 'file=C:\\Users\\Public\\Documents\\SQL Anywhere 16\\Samples\\
\\Certificates\\rsaroot.crt'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SATest';
```

Die CERTIFICATE-Klausel in diesem Beispiel gibt an, dass sich das RSA-Serverzertifikat in der Datei *C:\\Users\\Public\\Documents\\SQL Anywhere 16\\Samples\\Certificates\\rsaroot.crt* befindet.

Hinweis

Die Angabe von HTTPS_FIPS zwingt das System, die FIPS-Bibliotheken zu verwenden. Wenn HTTPS_FIPS angegeben wird, aber keine FIPS-Bibliotheken vorhanden sind, werden stattdessen Nicht-FIPS-Bibliotheken verwendet.

Weitere Hinweise zum Starten eines HTTP-Webserver mit diesen Beispielspezifikationen finden Sie unter „[HTTP-Webserver starten](#)“ auf Seite 757.

Proxy-Server-URL angeben

Einige Anforderungen müssen über einen Proxy-Server gesendet werden. Verwenden Sie die PROXY-Klausel der Anweisungen CREATE PROCEDURE und CREATE FUNCTION, um die Proxy-Server-URL anzugeben und Anforderungen an diese URL umzuleiten. Der Proxy-Server leitet die Anforderung an ihr Ziel weiter, empfängt die Antwort und leitet die Antwort an SQL Anywhere weiter.

Siehe auch

- „[SQL-Anweisungen für Webclients](#)“ auf Seite 811

Webdienst-Anforderungstypen

Sie können den Typ der an den Webserver gesendeten Clientanforderungen angeben, wenn Sie eine Webclient-Funktion oder -Prozedur erstellen. Die TYPE-Klausel der Anweisungen CREATE PROCEDURE und CREATE FUNCTION formatiert Anforderungen vor dem Versenden an den Webserver.

HTTP-Anforderungsformat angeben

Webclient-Funktionen und -Prozeduren senden HTTP-Anforderungen, wenn das angegebene Format in der TYPE-Klausel mit einem **HTTP**-Präfix beginnt.

Führen Sie beispielsweise die folgende SQL-Anweisung in der Webclient-Datenbank aus, um eine HTTP-Prozedur namens **PostOperation** zu erstellen, die HTTP-Anforderungen an die angegebene URL sendet:

```
CREATE PROCEDURE PostOperation(a INTEGER, b CHAR(128))
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  TYPE 'HTTP:POST';
```

In diesem Beispiel werden Anforderungen als HTTP: POST-Anforderungen formatiert, wobei eine Anforderung der folgendem Art erstellt wird:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.1403
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:02:49 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=windows-1252
Content-Length: 12

a=123&b=data
```

SOAP-Anforderungsformat angeben

Webclient-Funktionen und -Prozeduren senden HTTP-Anforderungen, wenn das angegebene Format in der TYPE-Klausel mit einem **SOAP**-Präfix beginnt.

Führen Sie z.B. die folgende Anweisung in der Webclient-Datenbank aus, um eine SOAP-Prozedur namens **SoapOperation** zu erstellen, die SOAP-Anforderungen an die angegebene URL sendet:

```
CREATE PROCEDURE SoapOperation(intVariable INTEGER, charVariable CHAR(128))
  URL 'HTTP://localhost:8082/dbname/SampleSoapService'
  TYPE 'SOAP:DOC';
```

In diesem Beispiel wird beim Aufrufen dieser Prozedur eine SOAP:DOC-Anforderung an die URL gesendet, die eine Anforderung der folgenden Art erzeugen würde:

```
POST /dbname/SampleSoapService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.1403
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:05:13 GMT
Host: localhost:8082
Connection: close
Content-Type: text/xml; charset=windows-1252
Content-Length: 428
SOAPAction: "HTTP://localhost:8082/SoapOperation"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Der Prozedurname erscheint im **<m:SoapOperation>**-Tag im Hauptteil. Die beiden Parameter der Prozedur **intVariable** und **charVariable** werden zu **<m:intVariable>** bzw. zu **<m:charVariable>**.

Der Name der gespeicherten Prozedur wird standardmäßig als SOAP-Vorgangsname verwendet, wenn eine SOAP-Anforderung erstellt wird. Parameternamen treten als Tag-Namen des SOAP-Envelope auf. Sie müssen diese Namen bei der Definition einer gespeicherten SOAP-Prozedur richtig referenzieren, da der Server diese Namen in der SOAP-Anforderung erwartet. Die SET-Klausel kann verwendet werden, um einen alternativen SOAP-Vorgangsname für die Prozedur anzugeben. Abgesehen von den einfachsten Fällen (beispielsweise gibt ein SOAP-RPC-Aufruf einen einzelnen Zeichenfolgewert zurück) wird empfohlen, Funktionsdefinitionen anstelle von Prozeduren zu benutzen. Eine SOAP-Funktion gibt den vollständigen SOAP-Antwortrahmen zurück, der mit OPENXML analysiert werden kann.

Siehe auch

- „SQL-Anweisungen für Webclients“ auf Seite 811
- „OPENXML-Operator“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „An Webdienste übergebene Variable“ auf Seite 811
- „HTTP- und SOAP-Anforderungsstrukturen“ auf Seite 830

Webclient-Ports

Manchmal muss angegeben werden, welche Ports beim Herstellen einer Serververbindung durch eine Firewall verwendet werden sollen. Sie können mit der CLIENTPORT-Klausel der Anweisungen CREATE PROCEDURE und CREATE FUNCTION die Portnummern angeben, über die die

Clientanwendung mithilfe von TCP/IP kommuniziert. Es wird empfohlen, dass Sie diese Funktion nur verwenden, wenn Ihre Firewall den Zugriff auf einen bestimmten Portbereich beschränkt.

Führen Sie z.B. die folgende SQL-Anweisung in der Webclient-Datenbank aus, um eine Prozedur namens **SomeOperation** zu erstellen, die Anforderungen an die angegebene URL unter Verwendung eines Ports im Bereich von 5050 bis 5060 bzw. von Port 5070 sendet:

```
CREATE PROCEDURE SomeOperation()  
  URL 'HTTP://localhost:8082/dbname/SampleWebService'  
  CLIENTPORT '5050-5060,5070';
```

Es wird empfohlen, dass Sie möglichst einen Bereich von Portnummern angeben. Es wird nur jeweils eine Verbindung aufrecht erhalten, wenn Sie eine einzelne Portnummer angeben. Die Clientanwendung versucht, auf alle angegebenen Portnummern zuzugreifen, bis sie eine Nummer findet, die sie verwenden kann. Nach dem Schließen der Verbindung wird eine Timeoutperiode von mehreren Minuten initiiert und daher kann keine neue Verbindung mit demselben Server und Port hergestellt werden.

Diese Funktion ähnelt dem Einstellen der ClientPort-Netzwerkprotokolloption.

Siehe auch

- „SQL-Anweisungen für Webclients“ auf Seite 811
- „ClientPort-Protokolloption (CPORT) (nur clientseitig)“ [*SQL Anywhere Server - Datenbankadministration*]

HTTP-Anforderungsheader verwalten

HTTP-Anforderungsheader können hinzugefügt, geändert oder entfernt werden, indem Sie die HEADER-Klausel der Anweisungen CREATE PROCEDURE und CREATE FUNCTION verwenden. Sie unterdrücken einen HTTP-Anforderungsheader durch die Referenzierung des Namens. Sie fügen einen Wert für den HTTP-Anforderungsheader hinzu, indem Sie nach dem Headernamen einen Doppelpunkt gefolgt vom Wert einfügen. Header-Wertangaben sind optional.

Führen Sie z.B. die folgende SQL-Anweisung in der Webclient-Datenbank aus, um eine Prozedur namens **SomeOperation2** zu erstellen, die Anforderungen an die angegebene URL sendet und die HTTP-Anforderungsheader einschränkt:

```
CREATE PROCEDURE SomeOperation2()  
  URL 'HTTP://localhost:8082/dbname/SampleWebService'  
  TYPE 'HTTP:GET'  
  HEADER 'SOAPAction\nDate\nFrom:\nCustomAlias:John Doe';
```

In diesem Beispiel wird der **Date**-Header, der von SQL Anywhere automatisch erstellt wird, unterdrückt. Der **From**-Header wird aufgenommen, ihm wird jedoch kein Wert zugewiesen. Ein neuer Header namens **CustomAlias** wird in die HTTP-Anforderung aufgenommen und erhält den Wert **John Doe**. Die GET-Anforderung sieht wie folgt aus:

```
GET /dbname/SampleWebService HTTP/1.0  
ASA-Id: e88a416e24154682bf81694feaf03052  
User-Agent: SQLAnywhere/16.0.0.1403  
Accept-Charset: windows-1252, UTF-8, *  
From:  
Host: localhost:8082
```

```
Connection: close
CustomAlias: John Doe
```

Das Umbrechen langer Headerwerte wird unterstützt, sofern ein oder mehrere Leerzeichen dem \n unmittelbar folgen.

Das folgende Beispiel zeigt die Unterstützung langer Headerwerte:

```
CREATE PROCEDURE SomeOperation3()
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  TYPE 'HTTP:POST'
  HEADER 'heading1: This long value\n is really long for a header.\n
        heading2:shortvalue';
```

Die POST-Anforderung sieht wie folgt aus:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.1403
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:26:04 GMT
heading1: This long value is really long for a header.
heading2:shortvalue
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=windows-1252
Content-Length: 0
```

Hinweis
Sie müssen den **SOAPAction**-HTTP-Anforderungsheader in dem angegebenen SOAP-Dienst-URI gemäß der WSDL einstellen, wenn Sie eine SOAP-Funktion oder -Prozedur erstellen.

Automatisch generierte HTTP-Anforderungsheader

Das Ändern automatisch generierter Header kann zu unerwarteten Ergebnissen führen. Die folgenden HTTP-Anforderungsheader sollten nur mit Vorsicht geändert werden:

HTTP-Header	Beschreibung
Accept-Charset	Wird immer automatisch generiert. Ändern oder Löschen dieses Headers kann zu unerwarteten Fehlern bei der Datenkonvertierung führen.
ASA-Id	Wird immer automatisch generiert. Dieser Header stellt sicher, dass die Clientanwendung keine Verbindung mit sich selbst erstellt und verhindert ein Deadlock.
Authorization	Automatisch generiert, wenn die URL Anmeldeinformationen enthält. Ändern oder Löschen dieses Headers kann zum Fehlschlagen der Anforderung führen. Es wird lediglich die BASIC-Autorisierung unterstützt. Benutzer- und Kennwortdaten sollten nur bei Verbindungen über HTTPS einbezogen werden.
Connection	"Connection: close", wird immer automatisch generiert. Clientanwendungen unterstützen keine dauerhaften Verbindungen. Die Verbindung kann bei Änderung hängen.

HTTP-Header	Beschreibung
Host	Wird immer automatisch generiert. HTTP/1.1-Server müssen mit 400 Bad Request antworten, wenn ein HTTP/1.1-Client keinen Host-Header angibt.
Transfer-Encoding	Automatisch generiert, wenn eine Anforderung im CHUNK-Modus gesendet wird. Das Entfernen dieses Headers oder das Löschen des Werts "chunked" führt zu einem Fehler, wenn der Client den CHUNK-Modus verwendet.
Content-Length	Automatisch generiert, wenn eine Anforderung nicht im CHUNK-Modus gesendet wird. Dieser Header wird benötigt, um dem Server die Inhaltslänge des Hauptteils mitzuteilen. Wenn die Inhaltslänge falsch angegeben wird, kann sich die Verbindung aufhängen oder Daten können verloren gehen.

Siehe auch

- „SQL-Anweisungen für Webclients“ auf Seite 811

Header der SOAP-Anforderung verwalten

Bei dem Header einer SOAP-Anforderung handelt es sich um ein XML-Fragment innerhalb des SOAP-Envelope. Während der SOAP-Vorgang und seine Parametern auch als RPC (Remote Procedure Call) betrachtet werden können, kann der Header einer SOAP-Anforderung verwendet werden, um Metadaten in einer spezifischen Anforderung oder Antwort zu übertragen. Die Header einer SOAP-Anforderung übertragen Metadaten wie etwa Autorisierungs- oder Sitzungskriterien.

Der Wert einer SOAPHEADER-Klausel muss ein gültiges XML-Fragment sein, das einem Headereintrag der SOAP-Anforderung entspricht. Es können mehrere Headereinträge für eine SOAP-Anforderung angegeben werden. Die gespeicherte Prozedur oder Funktion injiziert automatisch die Headereinträge der SOAP-Anforderung innerhalb eines SOAP-Headerelements (**SOAP-ENV:Header**). SOAPHEADER-Werte geben SOAP-Header an, die als statische Konstante deklariert oder dynamisch mithilfe des Parameterersatzungsmechanismus festgelegt werden können. Nachstehend wird ein Fragment aus einer SOAP-Anforderung gezeigt. Es enthält zwei XML-Header namens **Authentication** und **Session**.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="CustomerOrderURN">
      <userName pwd="none" mustUnderstand="1">
        <first>John</first>
        <last>Smith</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
```

```
<m:charVariable>data</m:charVariable>
</m:SoapOperation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP-Antwort-Header (zurückgegeben vom SOAP-Aufruf) werden für Prozeduren und Funktionen unterschiedlich verarbeitet. Bei Verwendung einer Funktion, was die flexibelste und empfohlene Methode darstellt, wird der gesamte SOAP-Antwortrahmen empfangen. Der Antwort-Envelope kann dann mit dem OPENXML-Operator verarbeitet werden, um SOAP-Header und SOAP-Hauptteildaten zu extrahieren. Bei Verwendung einer Prozedur können SOAP-Antwort-Header nur durch einen Ersetzungsparameter extrahiert werden, der einer IN- oder INOUT-Variable zugeordnet ist. In einer SOAP-Prozedur sind maximal ein IN- oder INOUT-Parameter zulässig.

Eine Webdienstfunktion muss den SOAP-Antwortrahmen syntaktisch analysieren, um die Header-Einträge zu erhalten.

Beispiele

Die folgenden Beispiele veranschaulichen das Erstellen von SOAP-Prozeduren und -Funktionen, die Parameter und SOAP-Header senden. Wrapper-Prozeduren werden verwendet, um die Aufrufe für Webdienstprozeduren zu füllen und die Antworten zu verarbeiten. Die **soapAddItemProc**-Prozedur veranschaulicht die Verwendung einer SOAP-Webdienstprozedur, die **soapAddItemFunc**-Funktion veranschaulicht die Verwendung einer SOAP-Webdienstfunktion und die **httpAddItemFunc**-Funktion zeigt, wie die SOAP-Daten an eine HTTP-Webdienstprozedur übergeben werden können.

Das folgende Beispiel zeigt eine SOAP-Clientprozedur, die Substitutionsparameter verwendet, um SOAP-Header zu senden. Ein einzelner INOUT-Parameter wird verwendet, um SOAP-Header zu empfangen. Eine als Wrapper fungierende gespeicherte Prozedur **addItemProcWrapper**, die **soapAddItemProc** aufruft, zeigt, wie SOAP-Header einschließlich Parametern gesendet und empfangen werden.

```
CREATE PROCEDURE soapAddItemProc(amount INT, item LONG VARCHAR,
    INOUT inoutheader LONG VARCHAR, IN inheader LONG VARCHAR)
URL 'http://localhost:8082/itemStore'
SET 'SOAP( OP=addItems )'
TYPE 'SOAP:DOC'
SOAPHEADER '!inoutheader!inheader';

CREATE PROCEDURE addItemProcWrapper(amount INT, item LONG VARCHAR,
    first_name LONG VARCHAR, last_name LONG VARCHAR)
BEGIN
    DECLARE io_header LONG VARCHAR;      // inout (write/read) soap header
    DECLARE resxml LONG VARCHAR;
    DECLARE soap_header_sent LONG VARCHAR;
    DECLARE i_header LONG VARCHAR;      // in (write) only soap header
    DECLARE err int;
    DECLARE crsr CURSOR FOR
        CALL soapAddItemProc( amount, item, io_header, i_header );

    SET io_header = XMLELEMENT( 'Authentication',
        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
        XMLELEMENT('userName', XMLATTRIBUTES(
            'none' as pwd,
            '1' as mustUnderstand ),
            XMLELEMENT( 'first', first_name ),
            XMLELEMENT( 'last', last_name ) ) );
    SET i_header = '<Session xmlns="SomeSession">123456789</Session>';
```

```

SET soap_header_sent = io_header || i_header;
OPEN crsr;
FETCH crsr INTO resxml, err;
CLOSE crsr;

SELECT resxml, err, soap_header_sent, io_header AS soap_header_received;
END;

/* example call to addItemProcWrapper */
CALL addItemProcWrapper( 5, 'shirt', 'John', 'Smith' );

```

Das folgende Beispiel zeigt eine SOAP-Clientfunktion, die Substitutionsparameter verwendet, um SOAP-Header zu senden. Ein kompletter SOAP-Antwortrahmen wird zurückgegeben. SOAP-Header können mit dem OPENXML-Operator syntaktisch analysiert werden. Eine als Wrapper fungierende Funktion **addItemFuncWrapper**, die **soapAddItemFunc** aufruft, zeigt, wie SOAP-Header einschließlich Parametern gesendet und empfangen werden. Außerdem zeigt sie, wie die Antwort mit dem OPENXML-Operator verarbeitet wird.

```

CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR,
    IN inheader1 LONG VARCHAR, IN inheader2 LONG VARCHAR )
RETURNS XML
URL 'http://localhost:8082/itemStore'
SET 'SOAP(OP=addItems)'
TYPE 'SOAP:DOC'
SOAPHEADER '!inheader1!inheader2';

CREATE PROCEDURE addItemFuncWrapper(amount INT, item LONG VARCHAR,
    first_name LONG VARCHAR, last_name LONG VARCHAR )
BEGIN
    DECLARE i_header1 LONG VARCHAR;
    DECLARE i_header2 LONG VARCHAR;
    DECLARE res LONG VARCHAR;
    DECLARE ns LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE header_entry LONG VARCHAR;
    DECLARE localname LONG VARCHAR;
    DECLARE namespaceuri LONG VARCHAR;
    DECLARE r_quantity int;
    DECLARE r_item LONG VARCHAR;
    DECLARE r_status LONG VARCHAR;

    SET i_header1 = XMLELEMENT( 'Authentication',
        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
        XMLELEMENT('userName', XMLATTRIBUTES(
            'none' as pwd,
            '1' as mustUnderstand ),
            XMLELEMENT( 'first', first_name ),
            XMLELEMENT( 'last', last_name ) ) );
    SET i_header2 = '<Session xmlns="SessionURN">123456789</Session>';

    SET res = soapAddItemFunc( amount, item, i_header1, i_header2 );

    SET ns = '<ns xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" '
        || 'xmlns:mp="urn:ianywhere-com:sa-xpath-metaprop" '
        || 'xmlns:customer="CustomerOrderURN" '
        || 'xmlns:session="SessionURN" '
        || 'xmlns:tns="http://localhost:8082"></ns>';

    // Process headers...
    SET xpath = '//SOAP-ENV:Header/*';
    BEGIN
        DECLARE crsr CURSOR FOR SELECT * FROM

```

```

OPENXML( res, xpath, 1, ns )
    WITH ( "header_entry" LONG VARCHAR '@mp:xmltext',
           "localname" LONG VARCHAR '@mp:localname',
           "namespaceuri" LONG VARCHAR
 '@mp:namespaceuri' );
    OPEN crsr;
    FETCH crsr INTO "header_entry", "localname", "namespaceuri";
    CLOSE crsr;
END;

// Process body...
SET xpath = '//tns:row';
BEGIN
    DECLARE crsr1 CURSOR FOR SELECT * FROM
        OPENXML( res, xpath, 1, ns )
            WITH ( "r_quantity" INT 'tns:quantity/text()',
                  "r_item" LONG VARCHAR 'tns:item/text()',
                  "r_status" LONG VARCHAR 'tns:status/
text()' );
    OPEN crsr1;
    FETCH crsr1 INTO "r_quantity", "r_item", "r_status";
    CLOSE crsr1;
END;

SELECT r_item, r_quantity, r_status, header_entry, localname,
namespaceuri;
END;

/* example call to addItemFuncWrapper */
CALL addItemFuncWrapper( 6, 'shorts', 'Jack', 'Smith' );

```

Das folgende Beispiel zeigt, wie ein HTTP:POST als Transport für eine ganze SOAP-Datenladung verwendet werden kann. Anstatt eine Webdienst-Client-SOAP-Prozedur zu erstellen, erzeugt diese Methode eine Webdienst-HTTP-Prozedur, die die SOAP-Daten transportiert. Eine Wrapper-Prozedur **addItemHttpWrapper** ruft **httpAddItemFunc** auf, um die Verwendung der POST-Funktion zu zeigen. Damit wird gezeigt, wie SOAP-Header einschließlich Parametern gesendet und empfangen werden und wie die Antwort akzeptiert wird.

```

CREATE FUNCTION httpAddItemFunc(soapPayload XML)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    TYPE 'HTTP:POST:text/xml'
    HEADER 'SOAPAction: "http://localhost:8082/addItems"';

CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
    RESULT(response XML)
    BEGIN
        DECLARE payload XML;
        DECLARE response XML;

        SET payload =
        '<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>

```

```

    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>';

  SET response = httpAddItemFunc( payload );
  /* process response as demonstrated in addItemFuncWrapper */
  SELECT response;
END;

/* example call to addItemHttpWrapper */
CALL addItemHttpWrapper( 7, 'socks' );

```

Weitere Hinweise zur serverseitigen Implementierung dieses Beispiels finden Sie unter [„Zugriff auf vom Client bereitgestellte SOAP-Anforderungsheader“ auf Seite 776](#).

Einschränkungen

- Serverseitige SOAP-Dienste können aktuell keine SOAP-Header-Anforderungen für die Ein- und Ausgabe definieren. Aus diesem Grund sind keine SOAP-Header-Metadaten in der WSDL-Ausgabe eines DISH-Dienstes verfügbar. SOAP-Client-Toolkits können nicht automatisch SOAP-Header-Schnittstellen für einen SQL Anywhere SOAP-Dienstendpunkt generieren.
- SOAP-Header-Fehler werden nicht unterstützt.

Anforderungen an den SOAP-Namespace-URI

Der Namespace-URI gibt den XML-Namespace an, aus dem der SOAP-Anforderungsrahmen für den angegebenen SOAP-Vorgang erstellt wird. Die Domänenkomponente der URL-Klausel wird verwendet, wenn der Namespace-URI nicht definiert ist.

Der SOAP-Prozessor auf dem Server verwendet diesen URI, um die Namen der verschiedenen Entitäten im Nachrichtenhauptteil der Anforderung zu interpretieren. Die NAMESPACE-Klausel der Anweisungen CREATE PROCEDURE und CREATE FUNCTION gibt den Namespace-URI an.

Möglicherweise müssen Sie einen Namespace-URI angeben, damit Prozeduraufrufe gelingen. Diese Informationen werden normalerweise in der öffentlichen Webserver-Dokumentation erläutert, Sie können jedoch den erforderlichen Namespace-URI auch über WSDL vom Webserver erhalten. Sie können WSDL durch Zugriff auf den DISH-Dienst generieren, wenn Sie versuchen, mit einem SQL Anywhere-Webserver zu kommunizieren.

Im Allgemeinen kann der NAMESPACE aus dem **targetNamespace**-Attribut, das am Beginn des WSDL-Dokuments innerhalb des **wSDL:definition** Elements angegeben wird, kopiert werden. Gehen Sie mit großer Sorgfalt vor, wenn Sie nachgestellte '/' einbeziehen, denn sie sind signifikant. Prüfen Sie außerdem, ob der angegebene SOAP-Vorgang ein **soapAction**-Attribut enthält. Es sollte dem **SOAPAction**-HTTP-Header entsprechen, der wie in den folgenden Absätzen erläutert generiert wird.

Die NAMESPACE-Klausel erfüllt zwei Funktionen. Sie gibt den Namespace für den Hauptteil des SOAP-Envelope an. Falls für die Prozedur TYPE 'SOAP:DOC' angegeben wurde, wird sie außerdem als Domänenkomponente des **SOAPAction**-HTTP-Headers benutzt.

Das folgende Beispiel veranschaulicht die Verwendung der NAMESPACE-Klausel:

```

CREATE FUNCTION an_operation(a_parameter LONG VARCHAR)
  RETURNS LONG VARCHAR
  URL 'http://wsdl.domain.com/fictitious.asmx'

```

```
TYPE 'SOAP:DOC'  
NAMESPACE 'http://wsdl.domain.com/'
```

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
SELECT an_operation('a_value');
```

Die Anweisung erstellt eine SOAP-Anforderung ähnlich wie die folgende Ausgabe:

```
POST /fictitious.asmx HTTP/1.0  
SOAPAction: "http://wsdl.domain.com/an_operation"  
Host: wsdl.domain.com  
Content-Type: text/xml  
Content-Length: 387  
Connection: close  
  
<?xml version="1.0"?>  
<SOAP-ENV:Envelope  
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:m="http://wsdl.domain.com/">  
  <SOAP-ENV:Body>  
    <m:an_operation>  
      <m:a_parameter>a_value</m:a_parameter>  
    </m:an_operation>  
  </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Der Namespace für das Präfix 'm' ist auf **http://wsdl.domain.com/** gesetzt und der **SOAPAction**-HTTP-Header gibt eine vollständig qualifizierte URL für den SOAP-Vorgang an.

Der nachgestellte Schrägstrich ist keine Voraussetzung für die richtige Funktion von SQL Anywhere. Er kann jedoch zu einem Antwortfehler führen, der schwer zu diagnostizieren ist. Der **SOAPAction**-HTTP-Header wird unabhängig von dem nachgestellten Schrägstrich richtig generiert.

Wenn kein NAMESPACE angegeben ist, wird die Domänenkomponente der URL-Klausel als Namespace für den SOAP-Hauptteil verwendet. Wenn für die Prozedur TYPE 'SOAP:DOC' angegeben ist, wird er außerdem verwendet, um die HTTP-Header für HTTP-**SOAPAction** zu generieren. Wenn im oben genannten Beispiel die NAMESPACE-Klausel nicht angegeben ist, dann wird **http://wsdl.domain.com** als Namespace verwendet. Der feine Unterschied liegt darin, dass kein nachgestellter Schrägstrich '/' vorhanden ist. In jeder anderen Beziehung ist die SOAP-Anforderung, einschließlich des **SOAPAction**-HTTP-Headers, mit dem oben genannten Beispiel identisch.

Die NAMESPACE-Klausel dient zur Festlegung des Namespace für den SOAP-Hauptteil, wie oben für SOAP:DOC beschrieben. Der **SOAPAction**-HTTP-Header wird jedoch mit einem leeren Wert generiert: SOAPAction: ""

Bei Verwendung des SOAP:DOC-Anforderungstyps wird der Namespace auch benutzt, um den **SOAPAction**-HTTP-Header zu erstellen.

Siehe auch

- „Webdienst-Anforderungstypen“ auf Seite 801
- „SQL-Anweisungen für Webclients“ auf Seite 811
- „DISH-Dienste erstellen“ auf Seite 764

SQL-Anweisungen für Webclients

Die folgenden SQL-Anweisungen stehen für die Webclient-Entwicklung zur Verfügung:

Webclient-bezogene SQL-Anweisungen	Beschreibung
„CREATE FUNCTION-Anweisung [Webdienst]“ [<i>SQL Anywhere Server - SQL-Referenzhandbuch</i>]	Erstellt eine Webclient-Funktion, die eine HTTP- oder SOAP-über-HTTP-Anforderung durchführt.
„ALTER FUNCTION-Anweisung“ [SQL Anywhere Server - <i>SQL-Referenzhandbuch</i>]	Ändert eine Funktion.
„CREATE PROCEDURE-Anweisung [Webdienste]“ [<i>SQL Anywhere Server - SQL-Referenzhandbuch</i>]	Erstellt eine benutzerdefinierte Webclient-Prozedur, die HTTP- oder SOAP-Anforderungen an einen HTTP-Server sendet.
„ALTER PROCEDURE-Anweisung“ [SQL Anywhere Server - <i>SQL-Referenzhandbuch</i>]	Ändert eine Prozedur.

An Webdienste übergebene Variable

Variablen können auf unterschiedliche Arten an einen Webdienst, abhängig von dessen Typ, übergeben werden.

Webclient-Anwendungen können Variablen an allgemeine HTTP-Webdienste auf eine der folgenden Arten übergeben:

- Im Suffix der URL
- Im Hauptteil einer HTTP-Anforderung

Variablen können dem **SOAP**-Diensttyp als Teil eines Standard-SOAP-Envelope übergeben werden.

Siehe auch

- „Webdienststypen“ auf Seite 760

In der URL an Webdienste übergebene Variable

Der HTTP-Webserver kann Variablen verwalten, die in der URL durch Webbrowser übergeben werden. Diese Variablen können anhand folgender Konventionen ausgedrückt werden:

- Sie können an das Ende der URL angefügt werden, wobei die einzelnen Parameterwerte durch einen Schrägstrich (/) getrennt werden, wie im folgenden Beispiel:

```
http://localhost/database-name/param1/param2/param3
```

- Sie können explizit in einer URL-Parameterliste definiert werden, wie im folgenden Beispiel:

```
http://localhost/database-name/?arg1=param1&arg2=param2&arg3=param3
```

- Eine Kombination aus URL-Anhang und Parameterlistendefinition ist ebenfalls möglich, wie im folgenden Beispiel:

```
http://localhost/database-name/param4/param5?  
arg1=param1&arg2=param2&arg3=param3
```

Wie der Webserver die URL interpretiert, hängt von der Definition der Webdienst-URL-Klausel ab. Weitere Hinweise zur Angabe der URL-Klausel und der Interpretation der URL-Parameter durch den Server finden Sie unter „Auf einem SQL Anywhere-HTTP-Webserver navigieren“ auf Seite 790.

Siehe auch

- „CREATE SERVICE-Anweisung [HTTP-Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Zugriff auf vom Client bereitgestellte HTTP-Variablen und -Header“ auf Seite 772
- „root-Webdienste erstellen und anpassen“ auf Seite 767

Im Hauptteil der HTTP-Anforderungen übergebene Variable

Sie können Variablen im Hauptteil einer HTTP-Anforderung durch die Angabe von HTTP:POST in der TYPE-Klausel einer WebClient-Funktion oder -Prozedur übergeben.

Standardmäßig verwendet TYPE HTTP:POST den Mime-Typ "application/x-www-form-urlencoded". Alle Parameter werden URL-kodiert und im Hauptteil der Anforderung übergeben. Wenn ein Medientyp zur Verfügung gestellt wird, kann der Content-Type-Header der Anforderung automatisch an den angegebenen Medientyp angepasst und ein einziger Parameterwert im Hauptteil der Anforderung hochgeladen werden.

Beispiel

Im folgenden Beispiel wird davon ausgegangen, dass ein Webdienst namens **XMLService** auf einem **localhost**-Webserver vorhanden ist. Richten Sie eine SQL Anywhere-Clientdatenbank ein, verbinden Sie sie mithilfe von Interactive SQL und führen Sie die folgende SQL-Anweisung aus:

```
CREATE PROCEDURE SendXMLContent(xmlcode LONG VARCHAR)  
  URL 'http://localhost/XMLService'  
  TYPE 'HTTP:POST:text/xml';
```

Die Anweisung erstellt eine Prozedur, mit der eine Variable im Hauptteil einer HTTP-Anforderung im text/xml-Format gesendet werden kann.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus, um eine HTTP-Anforderung an den **XMLService**-Webdienst zu senden:

```
CALL SendXMLContent('<title>Hello World!</title>');
```

Der Prozeduraufruf weist dem **xmlcode**-Parameter einen Wert zu und sendet ihn an den Webdienst.

Siehe auch

- „Zugriff auf vom Client bereitgestellte HTTP-Variablen und -Header“ auf Seite 772
- „CREATE PROCEDURE-Anweisung [Webdienste]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

In SOAP-Rahmen übergebene Variable

Sie können Variablen in einem SOAP-Envelope übergeben, indem Sie einen SOAP-Vorgang mit der SET SOAP-Option einer Webclient-Funktion oder -Prozedur einrichten.

Der folgende Code veranschaulicht, wie Sie einen SOAP-Vorgang in einer Webclient-Funktion einrichten:

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR)
  RETURNS XML
  URL 'http://localhost:8082/itemStore'
  SET 'SOAP(OP=addItems)'
  TYPE 'SOAP:DOC';
```

In diesem Beispiel enthält der **addItems**-Vorgang die **amount**- und **item**-Werte, die als Parameter an die **soapAddItemFunc**-Funktion übergeben werden.

Sie können eine Anforderung senden, indem Sie das folgende Beispielskript ausführen:

```
SELECT soapAddItemFunc(5, 'shirt');
```

Ein Aufruf des **soapAddItemFunc**-Funktionsaufrufs generiert einen SOAP-Envelope, der ähnlich wie der folgende aufgebaut ist:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>5</m:amount>
      <m:item>shirt</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Als Alternative zur vorherigen Vorgehensweise können Sie eigene SOAP-Daten erstellen und in einem HTTP-Wrapper an den Server senden.

Variablen für SOAP-Dienste müssen in eine Standard-SOAP-Anforderung eingefügt werden. Auf andere Art angegebene Werte werden ignoriert.

Der folgende Code veranschaulicht die Erstellung einer HTTP-Wrapper-Prozedur, die einen angepassten SOAP-Envelope aufbaut:

```
CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
END;
```

Der folgende Beispielcode veranschaulicht die Webclient-Funktion, die zum Senden der Anforderung verwendet wird:

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
RETURNS XML
URL 'http://localhost:8082/itemStore'
TYPE 'HTTP:POST:text/xml'
HEADER 'SOAPAction: "http://localhost:8082/addItems"';
```

Sie können eine Anforderung senden, indem Sie das folgende Beispielskript ausführen:

```
CALL addItemHttpWrapper( 7, 'socks' );
```

Variable, auf die aus Ergebnismengen zugegriffen wird

Webdienstclientaufrufe können mit gespeicherten Funktionen oder Prozeduren durchgeführt werden. Wenn sie mit einer Funktion durchgeführt werden, muss der Rückgabotyp ein Zeichendatentyp wie CHAR, VARCHAR oder LONG VARCHAR sein. Der Hauptteil der HTTP-Antwort ist der zurückgegebene Wert. Es werden keine Headerinformationen aufgenommen. Zusätzliche Informationen über die Anforderung, einschließlich von HTTP-Statusinformationen, werden von Prozeduren zurückgegeben. Daher sind Prozeduren vorzuziehen, wenn Zugriff auf zusätzliche Informationen gewünscht wird.

SOAP-Prozeduren

Die Antwort von einer SOAP-Funktion ist ein XML-Dokument, das die SOAP-Antwort enthält.

SOAP-Antworten sind strukturierte XML-Dokumente. Deshalb versucht SQL Anywhere standardmäßig, diese Informationen auszuwerten und eine sinnvollere Ergebnismenge zu erstellen. Jeder Tag der obersten Ebene innerhalb des zurückgegebenen Antwortdokuments wird extrahiert und als Spaltenname verwendet. Der Inhalt unter den einzelnen Tags in der Verzweigung wird als Zeilenwert für diese Spalte verwendet.

Zum Beispiel würde SQL Anywhere anhand der folgenden SOAP-Antwort die angezeigte Datenmenge erstellen:

```
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ElizaResponse xmlns:SOAPSDK4="SoapInterop">
      <Eliza>Hi, I'm Eliza. Nice to meet you.</Eliza>
    </ElizaResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Eliza
Hi, I'm Eliza. Nice to meet you.

In diesem Beispiel ist das Antwortdokument durch die <ElizaResponse>-Tags begrenzt, die innerhalb der <SOAP-ENV:Body>-Tags auftreten.

Ergebnismengen haben so viele Spalten, wie es Tags der obersten Ebene gibt. Diese Ergebnismenge hat nur eine Spalte, weil es nur einen Tag der obersten Ebene in der SOAP-Antwort gibt. Dieser einzige Tag der obersten Ebene, Eliza, wird zum Namen der Spalte.

XML-Verarbeitungseinrichtungen

Auf Informationen in XML-Ergebnismengen, einschließlich der SOAP-Antworten, kann mit der OPENXML-Prozedur zugegriffen werden.

Das folgende Beispiel verwendet die OPENXML-Prozedur, um Teile einer SOAP-Antwort zu extrahieren. Im Beispiel wird ein Webdienst verwendet, um den Inhalt der SYSWEBSERVICE-Tabelle als SOAP-Dienst zu exponieren:

```
CREATE SERVICE get_webservices
  TYPE 'SOAP'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT * FROM SYSWEBSERVICE;
```

Die folgende Webclient-Funktion, die in einer zweiten SQL Anywhere-Datenbank erstellt werden muss, gibt einen Aufruf an diesen Webdienst aus. Der Rückgabewert dieser Funktion ist das gesamte SOAP-Antwortdokument. Die Antwort ist im .NET **DataSet**-Format, da **DNET** das Standard-SOAP-Dienstformat ist.

```
CREATE FUNCTION get_webservices()
  RETURNS LONG VARCHAR
```

```
URL 'HTTP://localhost/get_webservices'  
TYPE 'SOAP:DOC';
```

Die folgende Anweisung zeigt, wie Sie mit der OPENXML-Prozedur zwei Spalten der Ergebnismenge extrahieren können. Die Spalten **service_name** und **secure_required** geben an, welche SOAP-Dienste sicher sind und wo HTTPS erforderlich ist.

```
SELECT *  
FROM OPENXML( get_webservices(), '//row' )  
WITH ( "Name" CHAR(128) 'service_name',  
      "Secure?" CHAR(1) 'secure_required' );
```

Diese Anweisung funktioniert, indem die Abkömmlinge des **row**-Knotens ausgewählt werden. Die WITH-Klausel erstellt die Ergebnismenge, die auf den beiden entscheidenden Elementen basiert. Unter der Annahme, dass nur der **get_webservices**-Webdienst existiert, gibt diese Funktion die folgende Ergebnismenge zurück:

Name	Secure?
get_webservices	N

Siehe auch

- „Variable, auf die aus Ergebnismengen zugegriffen wird“ auf Seite 814
- „XML in der Datenbank“ [[SQL Anywhere Server - SQL-Benutzerhandbuch](#)]
- „OPENXML-Operator“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Ergebnismengenabruf aus einem Webdienst

Webdienstprozeduren vom Typ HTTP geben alle Informationen zu einer Antwort in einer zweispaltigen Ergebnismenge zurück. Diese Ergebnismenge enthält den Antwortstatus, Headerinformationen und den Hauptteil. Die erste Spalte heißt **Attribute** und die zweite **Value**. Beide sind vom Datentyp LONG VARCHAR.

Die Ergebnismenge hat eine Zeile für jedes Antwort-Headerfeld sowie auch eine Zeile für die HTTP-Statuszeile (Status-Attribut) und eine Zeile für den Antwort-Hauptteil (Body-Attribut).

Das folgende Beispiel zeigt eine typische Antwort:

Attribut	Wert
Status	HTTP /1.0 200 OK
Body	<!DOCTYPE HTML ... ><HTML> ... </HTML>
Content-Type	text/html
Server	GWS/2.1
Content-Length	2234

Attribut	Wert
Date	Mon, 18 Oct 2004, 16:00:00 GMT

Erstellen Sie die folgende gespeicherte Webdienstprozedur, um sie als Beispiel zu verwenden.

```
CREATE OR REPLACE PROCEDURE SybaseWebPage()
URL 'http://www.sybase.com/mobilize'
TYPE 'HTTP';
```

Führen Sie die folgende SELECT-Abfrage aus, um die Antwort vom Webdienst als Ergebnismenge anzufordern.

```
SELECT * FROM SybaseWebPage()
WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

Da die Webdienstprozedur die Form der Ergebnismenge nicht beschreibt, ist die WITH-Klausel erforderlich, um eine temporäre Ansicht zu definieren.

Die Ergebnisse einer Abfrage können in einer Tabelle gespeichert werden. Führen Sie die folgende SQL-Anweisung aus, um eine Tabelle für die Werte der Ergebnismenge zu erstellen.

```
CREATE TABLE StoredResults(
Attribute LONG VARCHAR,
Value LONG VARCHAR
);
```

Die Ergebnismenge kann folgendermaßen in die Tabelle **StoredResults** eingefügt werden:

```
INSERT INTO StoredResults
SELECT * FROM SybaseWebPage()
WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

Sie können Klauseln gemäß der üblichen Syntax der SELECT-Anweisung hinzufügen. Wenn Sie z.B. nur eine bestimmte Zeile der Ergebnismenge auswählen möchten, können Sie eine WHERE-Klausel hinzufügen, um die Ergebnisse der SELECT-Anweisung auf eine einzige Zeile zu beschränken.

```
SELECT * FROM SybaseWebPage()
WITH (Attribute LONG VARCHAR, Value LONG VARCHAR)
WHERE Attribute = 'Status';
```

Diese SELECT-Anweisung ruft nur die Statusinformationen aus der Ergebnismenge ab. Sie kann daher verwendet werden um zu verifizieren, ob der Aufruf erfolgreich war.

Siehe auch

- „SQL-Anweisungen für Webclients“ auf Seite 811
- „Variable, auf die aus Ergebnismengen zugegriffen wird“ auf Seite 814

SOAP-Datentypen

Standardmäßig ist die XML-Kodierung von Parametereingaben eine Zeichenfolge und die Ergebnismengenausgabe für SOAP-Dienst-Formate enthält keine Informationen, die speziell den Datentyp der Ergebnismengenspalten beschreiben. Die Datentypen für Parameter aller Formate sind

Zeichenfolgen. Im DNET-Format haben alle Spalten innerhalb des Schemaabschnitts der Antwort den Zeichenfolgen-Datentyp. Die Formate CONCRETE und XML enthalten keine Datentypinformation in der Antwort. Dieses Standardverhalten kann mit der DATATYPE-Klausel geändert werden.

SQL Anywhere ermöglicht die Angabe von Datentypen mit der DATATYPE-Klausel. Datentypinformationen können in der XML-Kodierung der Parametereingabe und der Ergebnismengenausgabe bzw. -antworten für alle SOAP-Dienstformate enthalten sein. Dies vereinfacht die Parameterübergabe von SOAP-Toolkits, da es nicht erforderlich ist, dass der Clientcode Parameter explizit in Zeichenfolgen umwandelt. Eine Ganzzahl kann beispielsweise als "int" übergeben werden. XML-kodierte Datentypen ermöglichen es einem SOAP-Toolkit, die Daten syntaktisch zu analysieren und sie in den entsprechenden Datentyp umzuwandeln.

Wenn ausschließlich Zeichenfolgen-Datentypen verwendet werden, muss die Anwendung den Datentyp jeder Spalte der Ergebnismenge implizit kennen. Dies ist nicht erforderlich, wenn die Angabe von Datentypen vom Webserver verlangt wird. Um zu steuern, ob Datentypinformationen einbezogen werden, kann bei der Definition des Webdiensts die DATATYPE-Klausel angegeben werden.

Das folgende Beispiel zeigt eine Webdienstdefinition, die die Datentypfestlegung für die Ergebnismengenantwort enthält.

```
CREATE SERVICE "SASoapTest/EmployeeList"
  TYPE 'SOAP'
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA
  DATATYPE OUT
  AS SELECT * FROM Employees;
```

In diesem Beispiel werden Datentypinformationen nur für Ergebnismengenantworten angefordert, da dieser Dienst keine Parameter hat.

Die Datentypfestlegung ist für alle SQL Anywhere-Webdienste anwendbar, die mit dem Typ 'SOAP' definiert sind.

Datentypfestlegung für Eingabeparameter

Die Festlegung von Datentypen für Eingabeparameter wird dadurch unterstützt, dass die Datentypen der Parameter einfach in der vom DISH-Dienst generierten WSDL-Sprache als ihre tatsächlichen Datentypen dargestellt werden.

Eine typische Zeichenfolgen-Parameterdefinition (oder ein Parameter ohne Datentypangabe) sieht folgendermaßen aus:

```
<s:element minOccurs="0" maxOccurs="1" name="a_varchar" nillable="true"
  type="s:string" />
```

Der Zeichenfolgenparameter kann erscheinen oder auch nicht.

Für einen Parameter, für den ein Datentyp festgelegt wurde, wie etwa eine Ganzzahl, muss der Parameter zwingend erscheinen. Im Folgenden finden Sie ein Beispiel.

```
<s:element minOccurs="1" maxOccurs="1" name="an_int" nillable="false"
  type="s:int" />
```

Datentypfestlegung für Ausgabeparameter

Alle SQL Anywhere-Webdienste vom Typ 'SOAP' können Datentypinformationen in die Antwortdaten einbeziehen. Die Datentypen werden im Spaltenelement der Zeilenmenge in Form von Attributen angegeben.

Das folgende Beispiel zeigt eine typisierte SimpleDataSet-Antwort eines SOAP FORMAT 'CONCRETE'-Webdienstes.

```
<SOAP-ENV:Body>
  <tns:test_types_concrete_onResponse>
    <tns:test_types_concrete_onResult xsi:type='tns:SimpleDataset'>
      <tns:rowset>
        <tns:row>
          <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
          <tns:i xsi:type="xsd:int">99</tns:i>
          <tns:ii xsi:type="xsd:long">99999999</tns:ii>
          <tns:f xsi:type="xsd:float">3.25</tns:f>
          <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
          <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
          <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
        </tns:row>
      </tns:rowset>
    </tns:test_types_concrete_onResult>
    <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_concrete_onResponse>
</SOAP-ENV:Body>
```

Das folgende Beispiel zeigt eine Antwort eines SOAP FORMAT 'XML'-Webdienstes, in der die XML-Daten als Zeichenfolge zurückgegeben werden. Die innere Zeilengruppe umfasst kodierten XML-Code und wird hier aus Gründen der Lesbarkeit in dekodierter Form dargestellt.

```
<SOAP-ENV:Body>
  <tns:test_types_XML_onResponse>
    <tns:test_types_XML_onResult xsi:type='xsd:string'>
      <tns:rowset
        xmlns:tns="http://localhost/satest/dish"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <tns:row>
          <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
          <tns:i xsi:type="xsd:int">99</tns:i>
          <tns:ii xsi:type="xsd:long">99999999</tns:ii>
          <tns:f xsi:type="xsd:float">3.25</tns:f>
          <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
          <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
          <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
        </tns:row>
      </tns:rowset>
    </tns:test_types_XML_onResult>
    <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_XML_onResponse>
</SOAP-ENV:Body>
```

Der Namespace für die Elemente und das XML-Schema liefert zusätzlich zu den Datentypinformationen auch alle Informationen, die für die Nachbearbeitung durch einen XML-Parser erforderlich sind. Wenn in der Ergebnismenge keine Datentypinformationen vorhanden sind (DATATYPE OFF oder IN), werden die xsi:type- und die XML-Schema-Namespace-Deklaration weggelassen.

Das folgende Beispiel zeigt einen Webdienst mit dem SOAP FORMAT 'DNET', der ein SimpleDataSet mit Angabe des Datentyps zurückgibt:

```
<SOAP-ENV:Body>
  <tns:test_types_dnet_outResponse>
    <tns:test_types_dnet_outResult xsi:type='sqlresultstream:SqlRowSet'>
      <xsd:schema id='Schema2'
        xmlns:xsd='http://www.w3.org/2001/XMLSchema'
        xmlns:msdata='urn:schemas-microsoft.com:xml-msdata'>
        <xsd:element name='rowset' msdata:IsDataSet='true'>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name='row' minOccurs='0' maxOccurs='unbounded'>
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name='lvc' minOccurs='0' type='xsd:string' />
                    <xsd:element name='ub' minOccurs='0' type='xsd:unsignedByte' />
                    <xsd:element name='s' minOccurs='0' type='xsd:short' />
                    <xsd:element name='us' minOccurs='0' type='xsd:unsignedShort' />
                    <xsd:element name='i' minOccurs='0' type='xsd:int' />
                    <xsd:element name='ui' minOccurs='0' type='xsd:unsignedInt' />
                    <xsd:element name='l' minOccurs='0' type='xsd:long' />
                    <xsd:element name='ul' minOccurs='0' type='xsd:unsignedLong' />
                    <xsd:element name='f' minOccurs='0' type='xsd:float' />
                    <xsd:element name='d' minOccurs='0' type='xsd:double' />
                    <xsd:element name='bin' minOccurs='0' type='xsd:base64Binary' />
                    <xsd:element name='bool' minOccurs='0' type='xsd:boolean' />
                    <xsd:element name='num' minOccurs='0' type='xsd:decimal' />
                    <xsd:element name='dc' minOccurs='0' type='xsd:decimal' />
                    <xsd:element name='date' minOccurs='0' type='xsd:date' />
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </tns:test_types_dnet_outResult>
    <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>

<diffgr:diffgram xmlns:msdata='urn:schemas-microsoft-com:xml-msdata'
xmlns:diffgr='urn:schemas-microsoft-com:xml-diffgram-v1'>
  <rowset>
    <row>
      <lvc>Hello World</lvc>
      <ub>128</ub>
      <s>-99</s>
      <us>33000</us>
      <i>-2147483640</i>
      <ui>4294967295</ui>
      <l>-9223372036854775807</l>
      <ul>18446744073709551615</ul>
      <f>3.25</f>
      <d>.55555555555555555582</d>
      <bin>QUJD</bin>
      <bool>1</bool>
      <num>123456.123457</num>
      <dc>-1.756000</dc>
      <date>2006-05-29-04:00</date>
    </row>
  </rowset>
</diffgr:diffgram>
</tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>
```


SQL Anywhere-Datentyp	XML-Schematyp	XML-Beispiel
INTEGER	int	-2147483640
UNSIGNED INTEGER	unsignedInt	4294967295
NUMERIC	decimal	123456.123457
REAL	float	3.25
SMALLINT	short	-99
UNSIGNED SMALLINT	unsignedShort	33000
TINYINT	unsignedByte	128
MONEY	decimal	12345678.9900
SMALLMONEY	decimal	12.3400
DATE	date	2006-11-21-05:00
DATETIME	dateTime	2006-05-21T09:00:00.000-08:00
SMALLDATETIME	dateTime	2007-01-15T09:00:00.000-08:00
TIME	time	14:14:48.980-05:00
TIMESTAMP	dateTime	2007-01-12T21:02:14.420-06:00
TIMESTAMP WITH TIME ZONE	dateTime	2007-01-12T21:02:14.420-06:00
BINARY	base64Binary	AAAAZg==
IMAGE	base64Binary	AAAAZg==
LONG BINARY	base64Binary	AAAAZg==
VARBINARY	base64Binary	AAAAZg==

Wenn ein oder mehrere Parameter vom Typ NCHAR, NVARCHAR, LONG NVARCHAR oder NTEXT sind, erfolgt die Antwortausgabe in UTF8. Wenn die Clientdatenbank den UTF-8-Zeichensatz verwendet, gibt es keine Änderung im Verhalten (da NCHAR- und CHAR-Datentypen dieselben sind). Wenn die Datenbank allerdings nicht den UTF-8-Zeichensatz verwendet, werden alle Parameter, die nicht vom NCHAR-Datentyp sind, zu UTF8 konvertiert. Der Wert des XML-Deklarationszeichensatzes und des Content-Type HTTP-Headers werden dem verwendeten Zeichensatz entsprechen.

Zuordnung von XML-Schematypen zu Java-Datentypen

XML-Schematyp	Java-Datentyp
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration

XML-Schematyp	Java-Datentyp
xsd:NOTATION	javax.xml.namespace.QName

Strukturierte SOAP-Datentypen

XML-Rückgabewerte

Der SQL Anywhere-Server als Webdienstclient kann sich mit einer Funktion oder Prozedur mit einem Webdienst verbinden.

Eine Zeichenfolgendarstellung in einer Ergebnismenge kann für einfache Rückgabe-Datentypen ausreichen. Die Verwendung einer gespeicherten Prozedur kann in diesem Fall geeignet sein.

Für die Rückgabe komplexer Daten wie etwa Arrays oder Strukturen ist der Einsatz von Webdienstfunktionen vorzuziehen. Bei Funktionsdeklarationen kann die RETURN-Klausel einen XML-Datentyp angeben. Der zurückgegebene XML-Code kann mit OPENXML analysiert werden, um die gewünschten Elemente zu extrahieren.

Die Rückgabe von XML-Datentypen wie dateTime erfolgt innerhalb der Ergebnismenge wortgetreu. Falls beispielsweise eine TIMESTAMP-Spalte in einer Ergebnismenge enthalten ist, wird sie als XML-Zeichenfolge des Typs dateTime formatiert (2006-12-25T12:00:00.000-05:00), nicht als Zeichenfolge (2006-12-25 12:00:00.000).

XML-Parameterwerte

Der SQL Anywhere XML-Datentyp wird für die Verwendung als Parameter innerhalb von Webdienstfunktionen und -prozeduren unterstützt. Für einfache Typen wird das Parameterelement automatisch erstellt, wenn der Hauptteil für die SOAP-Anforderung generiert wird. Für Parameter des Typs XML kann dies jedoch nicht durchgeführt werden, da die XML-Darstellung des Elements möglicherweise Attribute benötigt, die zusätzliche Daten bereitstellen. Daher muss der Name des Stammelements dem Parameternamen entsprechen, wenn XML für einen Parameter generiert wird, dessen Datentyp XML ist.

```
<inputHexBinary xsi:type="xsd:hexBinary">414243</inputHexBinary>
```

Der XML-Typ zeigt, wie ein Parameter als XML-Typ hexBinary gesendet wird. Der SOAP-Endpunkt erwartet, dass der Parameternamen (oder in der XML-Terminologie der Stammelementname) inputHexBinary ist.

Namespace-Deklarationen

Um komplexe Strukturen und Arrays zu erstellen, sind Kenntnisse der SQL Anywhere-Referenz-Namespace erforderlich. Die hier aufgeführten Präfixe entsprechen den Namespace-Deklarationen, die für einen SQL Anywhere SOAP-Anforderungsrahmen generiert wurden.

SQL Anywhere-XML-Präfix	Namespace
xsd	http://www.w3.org/2001/XMLSchema

SQL Anywhere-XML-Präfix	Namespace
xsi	http://www.w3.org/2001/XMLSchema-Instanz
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
m	Namespace, wie in der NAMESPACE-Klausel definiert

Beispiele komplexer Datentypen

Mit den folgenden drei Beispielen wird gezeigt, wie Webdienstclientfunktionen mit Parametern erstellt werden, die ein Array, eine Struktur und ein Array von Strukturen darstellen. Die Webdienstfunktionen kommunizieren mit den SOAP-Operationen (oder RPC-Funktionsnamen) echoFloatArray, echoStruct und echoStructArray. Der gemeinsame Namespace für Systemoffenheitstests ist <http://soapinterop.org/>. Er macht es möglich, eine bestimmte Funktion mit alternativen Systemoffenheitsservern zu testen, indem einfach die URL-Klausel in den ausgewählten SOAP-Endpunkt geändert wird.

Diese Beispiele wurden erstellt, um Anforderungen an den SOAP ToolKit 3.0 Round 2-Systemoffenheits-Testserver von Microsoft unter <http://mssoapinterop.org/stkV3> zu stellen.

Alle Beispiele verwenden eine Tabelle zum Generieren der XML-Daten. Im folgenden Abschnitt wird gezeigt, wie diese Tabelle eingerichtet wird.

```
CREATE LOCAL TEMPORARY TABLE SoapData
(
    seqno INT DEFAULT AUTOINCREMENT,
    i INT,
    f FLOAT,
    s LONG VARCHAR
) ON COMMIT PRESERVE ROWS;

INSERT INTO SoapData (i,f,s)
VALUES (99,99.999,'Ninety-Nine');

INSERT INTO SoapData (i,f,s)
VALUES (199,199.999,'Hundred and Ninety-Nine');
```

Die folgenden drei Funktionen senden SOAP-Anforderungen an den Systemoffenheitsserver. In diesem Beispiel werden Anforderungen an den Microsoft Soap Interop Server gesendet:

```
CREATE FUNCTION echoFloatArray(inputFloatArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStruct(inputStruct XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStructArray(inputStructArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';
```

Abschließend werden die drei Beispieranweisungen zusammen mit der XML-Darstellung ihrer Parameter gezeigt:

1. Die Parameter im folgenden Beispiel stellen ein Array dar.

```
SELECT echoFloatArray(
  XMLELEMENT( 'inputFloatArray',
    XMLATTRIBUTES( 'xsd:float[2]' as "SOAP-ENC:arrayType" ),
    (
      SELECT XMLAGG( XMLELEMENT( 'number', f ) ORDER BY segno )
      FROM SoapData
    )
  )
);
```

Die gespeicherte Prozedur echoFloatArray sendet den folgenden XML-Code an den Systemoffenheitsserver.

```
<inputFloatArray SOAP-ENC:arrayType="xsd:float[2]">
<number>99.9990005493164</number>
<number>199.998992919922</number>
</inputFloatArray>
```

Der Systemoffenheitsserver liefert folgende Antwort.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:echoFloatArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result SOAPSDK3:arrayType="SOAPSDK1:float[2]"
        SOAPSDK3:offset="[0]"
        SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK3:float>99.9990005493164</SOAPSDK3:float>
        <SOAPSDK3:float>199.998992919922</SOAPSDK3:float>
      </Result>
    </SOAPSDK4:echoFloatArrayResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Falls die Antwort in einer Variablen gespeichert wurde, kann sie mit OPENXML analysiert werden.

```
SELECT * FROM OPENXML( resp, '//*:Result/*' )
WITH ( varFloat FLOAT 'text()' );
```

varFloat
99.9990005493
199.9989929199

2. Die Parameter im folgenden Beispiel stellen eine Struktur dar.

```
SELECT echoStruct(
  XMLELEMENT( 'inputStruct',
```

```

        (
        SELECT XMLFOREST( s as varString,
                           i as varInt,
                           f as varFloat )
        FROM SoapData
        WHERE seqno=1
        )
    );

```

Die gespeicherte Prozedur `echoStruct` sendet den folgenden XML-Code an den Systemoffenheitsserver.

```

<inputStruct>
  <varString>Ninety-Nine</varString>
  <varInt>99</varInt>
  <varFloat>99.9990005493164</varFloat>
</inputStruct>

```

Der Systemoffenheitsserver liefert folgende Antwort.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:echoStructResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result href="#idl1"/>
    </SOAPSDK4:echoStructResponse>
    <SOAPSDK5:SOAPStruct
      xmlns:SOAPSDK5="http://soapinterop.org/xsd"
      id="idl1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK5:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK5:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Falls die Antwort in einer Variablen gespeichert wurde, kann sie mit OPENXML analysiert werden.

```

SELECT * FROM OPENXML( resp, '/*:Body/*:SOAPStruct' )
WITH (
  varString LONG VARCHAR 'varString',
  varInt INT 'varInt',
  varFloat FLOAT 'varFloat' );

```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493

3. Die Parameter im folgenden Beispiel stellen ein Array von Strukturen dar.

```

SELECT echoStructArray(
  XMLELEMENT( 'inputStructArray',

```

```
XMLATTRIBUTES( 'http://soapinterop.org/xsd' AS "xmlns:q2",
                'q2:SOAPStruct[2]' AS "SOAP-ENC:arrayType" ),
(
  SELECT XMLAGG(
    XMLElement('q2:SOAPStruct',
      XMLFOREST( s as varString,
                  i as varInt,
                  f as varFloat )
    )
  ORDER BY seqno
)
FROM SoapData
)
);
```

Die gespeicherte Prozedur echoFloatArray sendet den folgenden XML-Code an den Systemoffenheitsserver.

```
<inputStructArray xmlns:q2="http://soapinterop.org/xsd"
  SOAP-ENC:arrayType="q2:SOAPStruct[2]">
  <q2:SOAPStruct>
    <varString>Ninety-Nine</varString>
    <varInt>99</varInt>
    <varFloat>99.9990005493164</varFloat>
  </q2:SOAPStruct>
  <q2:SOAPStruct>
    <varString>Hundred and Ninety-Nine</varString>
    <varInt>199</varInt>
    <varFloat>199.998992919922</varFloat>
  </q2:SOAPStruct>
</inputStructArray>
```

Der Systemoffenheitsserver liefert folgende Antwort.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    <SOAPSDK4:echoStructArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/"
      <Result xmlns:SOAPSDK5="http://soapinterop.org/xsd"
        SOAPSDK3:arrayType="SOAPSDK5:SOAPStruct[2]"
        SOAPSDK3:offset="[0]" SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK5:SOAPStruct href="#id1"/>
        <SOAPSDK5:SOAPStruct href="#id2"/>
      </Result>
    </SOAPSDK4:echoStructArrayResponse>
    <SOAPSDK6:SOAPStruct
      xmlns:SOAPSDK6="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK6:SOAPStruct">
        <varString>Ninety-Nine</varString>
        <varInt>99</varInt>
        <varFloat>99.9990005493164</varFloat>
      </SOAPSDK6:SOAPStruct>
    <SOAPSDK7:SOAPStruct
      xmlns:SOAPSDK7="http://soapinterop.org/xsd"
```

```

id="id2"
SOAPSDK3:root="0"
SOAPSDK2:type="SOAPSDK7:SOAPStruct">
<varString>Hundred and Ninety-Nine</varString>
<varInt>199</varInt>
<varFloat>199.998992919922</varFloat>
</SOAPSDK7:SOAPStruct>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Falls die Antwort in einer Variablen gespeichert wurde, kann sie mit OPENXML analysiert werden.

```

SELECT * FROM OPENXML( resp, '/*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );

```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493
Hundred and Ninety-Nine	199	199.9989929199

Für Klauselwerte verwendete Ersetzungsparameter

Deklarierte Parameter für eine gespeicherte Prozedur oder Funktion werden automatisch für Platzhalter in einer Klauseldefinition eingesetzt, und zwar jedes Mal, wenn die gespeicherte Prozedur oder Funktion ausgeführt wird. Ersetzungsparameter ermöglichen die Erstellung allgemeiner Webdienstprozeduren, die Klauseln zur Laufzeit dynamisch konfigurieren. Alle Teilzeichenfolgen, die ein Ausrufzeichen '!' enthalten, gefolgt vom Namen eines der deklarierten Parameter, werden durch den Wert dieses Parameters ersetzt. Auf diese Weise können ein oder mehrere Parameterwerte ersetzt werden, um die Werte einer oder mehrerer Klauseln während der Laufzeit abzuleiten.

Für das Ersetzen von Parametern müssen folgende Regeln eingehalten werden:

- Alle zum Ersetzen verwendeten Parameter müssen alphanumerisch sein. Unterstriche sind nicht zulässig.
- Einem Ersetzungsparameter muss unmittelbar ein nicht alphanumerisches Zeichen bzw. ein Abschlusszeichen folgen. Zum Beispiel wird **!sizeXL** nicht durch den Wert des Parameters "size" ersetzt, weil X ein alphanumerisches Zeichen ist.
- Ein Ersetzungsparameter ohne entsprechenden Parameternamen wird ignoriert.
- Ein Ausrufezeichen (!) kann durch Hinzufügen eines weiteren Ausrufezeichens umgangen werden.

Die folgende Prozedur veranschaulicht die Parameterersetzung. URL und HTTP-Header-Definitionen müssen als Parameter übergeben werden.

```

CREATE PROCEDURE test(uid CHAR(128), pwd CHAR(128), headers LONG VARCHAR)
URL 'http://!uid:!pwd@localhost/myservice'
HEADER '!headers';

```

Sie können dann folgende Anweisung verwenden, um die **test**-Prozedur aufzurufen und eine HTTP-Anforderung zu initiieren:

```
CALL test('dba', 'sql', 'NewHeader1:value1\nNewHeader2:value2');
```

Es können jedes Mal, wenn diese Prozedur aufgerufen wird, unterschiedliche Werte verwendet werden.

Beispiel für ein Verschlüsselungszertifikat

Sie können die Parameterersetzung außerdem verwenden, um Verschlüsselungszertifikate aus einer Datei an eine gespeicherte Prozedur oder Funktion zu übergeben.

Das folgende Beispiel zeigt, wie Sie ein Zertifikat als Ersetzungszeichenfolge zu übergeben:

```
CREATE PROCEDURE secure(cert LONG VARCHAR)
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Root';
```

Das Zertifikat wird aus einer Datei gelesen und im folgenden Aufruf an **secure** übergeben.

```
CALL secure( xp_read_file('C:\\Users\\Public\\Documents\\SQL Anywhere 16\\
\\Samples\\Certificates\\rsaroot.crt') );
```

Dieses Beispiel dient nur zur Veranschaulichung. Das Zertifikat kann unter Verwendung des **file=**-Schlüsselworts für die CERTIFICATE-Klausel direkt aus einer Datei gelesen werden.

Beispiel für keinen passenden Parameternamen

Platzhalter ohne entsprechenden Parameternamen werden automatisch gelöscht.

Der Parameter "size" z.B. würde in der folgenden Prozedur nicht den Platzhalter ersetzen:

```
CREATE PROCEDURE orderitem (size CHAR(18))
URL 'HTTP://localhost/salesserver/order?size=!sizeXL'
TYPE 'SOAP:RPC';
```

In diesem Beispiel wird **!sizeXL** immer gelöscht, weil es ein Platzhalter ist, für den es keinen entsprechenden Parameter gibt.

Parameter können verwendet werden, um Platzhalter im Hauptteil der gespeicherten Funktion bzw. Prozedur zu dem Zeitpunkt zu ersetzen, an dem die Funktion bzw. Prozedur aufgerufen wird. Wenn es keine Platzhalter für eine bestimmte Variable gibt, wird der Parameter und sein Wert als Teil der Anforderung übergeben. Parameter und Werte, die auf diese Weise zum Ersetzen verwendet werden, werden nicht als Teil der Anforderung übergeben.

HTTP- und SOAP-Anforderungsstrukturen

Außer wenn sie während einer Parameterersetzung verwendet werden, werden alle Parameter für eine Funktion oder Prozedur als Teil der Webdienstanforderung übergeben. Das Format, in dem sie übergeben werden, hängt vom Typ der Webdienstanforderung ab.

Parameterwerte, die keine Zeichen- oder Binärdatentypen sind, werden in eine Zeichenfolgendarstellung konvertiert, bevor sie der Anforderung hinzugefügt werden. Dieser Prozess ist mit dem Casting des Werts

in einen Zeichentyp äquivalent. Die Konvertierung wird entsprechend der Optionseinstellungen für Datentypformatierung zum Zeitpunkt des Funktions- bzw. Prozeduraufrufs durchgeführt. Insbesondere kann die Konvertierung von Optionen wie precision, scale und timestamp_format betroffen sein.

HTTP-Anforderungsstrukturen

Parameter für Typ HTTP:GET werden URL-kodiert und in der URL platziert. Parameternamen werden wortgetreu als Namen für HTTP-Variablen verwendet. Die folgende Prozedur z.B. deklariert zwei Parameter:

```
CREATE PROCEDURE test(a INTEGER, b CHAR(128))
  URL 'HTTP://localhost/myservice'
  TYPE 'HTTP:GET';
```

Wenn diese Prozedur mit den zwei Werten 123 und "xyz" aufgerufen wird, ist der für die Anforderung verwendete URL mit dem unten gezeigten äquivalent:

```
HTTP://localhost/myservice?a=123&b=xyz
```

Wenn der Typ HTTP:POST ist, werden die Parameter und ihre Werte URL-kodiert und in den Hauptteil der Anforderung platziert. Nach den Headern erscheint der folgende Text im Hauptteil der HTTP-Anforderung für die zwei Parameter und Werte:

```
a=123&b=xyz
```

SOAP-Anforderungsstrukturen

An SOAP-Anforderungen übergebene Parameter werden als Teil des Anforderungshauptteils gebündelt, wie es von der SOAP-Spezifikation verlangt wird:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:test>
      <m:a>123</m:a>
      <m:b>abc</m:b>
    </m:test>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Siehe auch

- [„Für Klauselwerte verwendete Ersetzungsparameter“ auf Seite 829](#)

Webclientanforderungen protokollieren

Webdienst-Clientinformationen, einschließlich HTTP-Anforderungen und Transportdaten, können in der Webdienst-Clientlogdatei protokolliert werden. Die Webdienst-Clientlogdatei kann mit der Serveroption -zoc oder mithilfe der sa_server_option-Systemprozedur angegeben werden:

```
CALL sa_server_option( 'WebClientLogFile', 'clientinfo.txt' );
```

Die Protokollierung wird automatisch aktiviert, wenn sie die Serveroption -zoc festlegen. Sie können das Protokollieren in diese Datei aktivieren und deaktivieren, indem Sie die sa_server_option-Systemprozedur verwenden:

```
CALL sa_server_option( 'WebClientLogging', 'ON' );
```

Siehe auch

- „Datenbankserveroption -zoc“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „sa_server_option-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Fehlercodereferenz für Webdienste

Der HTTP-Server generiert Standard-Webdienstfehler, wenn Anforderungen fehlschlagen. Diesen Fehlern werden in Übereinstimmung mit den Protokollstandards Nummern zugeordnet.

Im Folgenden sind einige der häufig auftretenden Fehler aufgeführt:

Nr.	Name	SOAP-Fehler	Beschreibung
301	Seite dauerhaft an anderer Stelle	Server	Die angeforderte Seite ist dauerhaft an eine andere Stelle verschoben worden. Der Server leitet die Anforderung automatisch an den neuen Standort um.
304	Seite nicht geändert	Server	Der Server hat aufgrund von Informationen in der Anforderung entschieden, dass die angeforderten Daten seit der letzten Anforderung nicht geändert wurden. Es ist daher nicht nötig, sie neuerlich zu senden.
307	Temporäre Umleitung	Server	Die angeforderte Seite ist verschoben worden, aber diese Änderung ist möglicherweise nicht permanent. Der Server leitet die Anforderung automatisch an den neuen Standort um.
400	Ungültige Abfrage	Client.BadRequest	Die HTTP-Abfrage ist nicht vollständig oder falsch formuliert.
401	Autorisierung erforderlich	Client.Authorization	Die Nutzung dieses Diensts erfordert eine Autorisierung. Es wurde jedoch kein gültiger Benutzername und kein gültiges Kennwort angegeben.
403	Verboten	Client.Forbidden	Sie verfügen nicht über die Berechtigung, auf diese Datenbank zuzugreifen.

Nr.	Name	SOAP-Fehler	Beschreibung
404	Seite nicht gefunden	Client.NotFound	Die angegebene Datenbank wird auf dem Server nicht ausgeführt oder der angegebene Webdienst ist nicht vorhanden.
408	Anforderungs-Timeout	Server.RequestTimeout	Die für Verbindungen maximal zulässige Leerlaufzeit wurde beim Empfangen der Anforderung überschritten.
411	HTTP-Länge erforderlich	Client.LengthRequired	Der Server erfordert, dass der Client eine Inhaltslängen-Angabe in die Anforderung aufnimmt. Dies tritt üblicherweise beim Übertragen von Daten auf den Server auf.
413	Abfrageentität zu groß	Server	Die Abfrage überschreitet die maximal zulässige Größe.
414	Abfrage-URI zu groß	Server	Die Länge des URIs überschreitet die maximal zulässige Länge.
500	Interner Serverfehler	Server	Ein interner Serverfehler ist aufgetreten. Die Anforderung konnte nicht verarbeitet werden.
501	Nicht implementiert	Server	Die HTTP-Abfragemethode ist keine GET-, HEAD- oder POST-Methode.
502	Fehlerhaftes Gateway	Server	Das angeforderte Dokument residiert auf einem Server eines Drittanbieters, und der Server empfing einen Fehler von diesem Server des Drittanbieters.
503	Dienst nicht verfügbar	Server	Die Anzahl der Verbindungen überschreitet das zulässige Maximum.

Fehler werden bei Fehlschlägen eines SOAP-Dienstes an den Client gemäß der folgenden Standards von SOAP Version 1.1 als SOAP-Fehler zurückgegeben:

- Wenn ein Fehler in der Anwendung auftritt, die die Anforderung verarbeitet, generiert die Anforderung einen **SQLCODE** und es wird ein SOAP-Fehler mit dem Fehlercode (faultcode) des Clients zurückgegeben, möglicherweise mit einer Unterkategorie, wie etwa "Procedure". Das faultstring-Element innerhalb des SOAP-Fehlers wird auf eine detaillierte Erklärung des Fehlers festgelegt und ein detail-Element enthält den numerischen **SQLCODE**-Wert.
- Im Fall eines Transportprotokollfehlers wird der faultcode-Fehlercode auf Client oder auf Server gesetzt, abhängig vom Fehler. Die faultstring-Fehlerzeichenfolge wird auf die HTTP-Transportmeldung gesetzt, wie z.B. 404 Not Found, und das detail-Element enthält den numerischen HTTP-Fehlercode.

- SOAP-Fehlermeldungen, die aufgrund von Anwendungsfehlern generiert werden, die einen SQLCODE-Wert zurückgeben, werden mit dem HTTP-Status 200 OK zurückgegeben.

Der entsprechende HTTP-Fehler wird in einem generierten HTML-Dokument zurückgegeben, wenn der Client nicht als SOAP-Client identifiziert werden kann.

Beispiele für HTTP-Webdienste

Mehrere Beispielimplementierungen von Webdiensten sind im Verzeichnis `%SQLANYSAMPI6%\SQLAnywhere\HTTP` abgelegt. Weitere Hinweise zu den Beispielen finden Sie hier: `%SQLANYSAMPI6%\SQLAnywhere\HTTP\readme.txt`.

Praktische Einführung: Webserver erstellen und über einen Webclient darauf zugreifen

Diese praktische Einführung veranschaulicht, wie Sie einen Webserver mit einem SQL Anywhere-Datenbankserver erstellen und anschließend über einen Webclient-Datenbankserver Anforderungen an diesen senden können.

Erforderliche Software

- SQL Anywhere 16

Kenntnisse und Erfahrungen

- XML-Kenntnisse
- Kenntnisse über MIME-Typen (Multipurpose Internet Mail Extensions)
- Grundkenntnisse über SQL Anywhere-Webdienste

Ziele

- Erstellen und Starten einer neuen SQL Anywhere-Webserver-Datenbank
- Erstellen eines Webdiensts.
- Einrichten einer Prozedur, die die in HTTP-Anforderungen enthaltenen Informationen zurückgibt.
- Erstellen und Starten einer neuen SQL Anywhere-Webclient-Datenbank
- Senden von HTTP:POST-Anforderungen vom Webclient an den Datenbankserver.
- Senden von HTTP-Antworten vom Webserver an den Webclient.

Privilegien

Die folgenden Privilegien sind erforderlich, um die Lektionen dieser praktischen Einführung auszuführen.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Empfohlene Hintergrundlektüre

- „SQL Anywhere als HTTP-Webserver“ auf Seite 755

Lektion 1: Webserver für das Empfangen von Anforderungen und das Senden von Antworten einrichten

Das Ziel dieser Lektion ist das Einrichten eines SQL Anywhere-Webservers, auf dem ein Webdienst ausgeführt wird.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Webserver erstellen und über einen Webclient darauf zugreifen](#)“.

Aufgabe

1. Erstellen Sie eine SQL Anywhere-Datenbank, die verwendet wird, um Webdienstdefinitionen zu speichern.

```
dbinit -dba DBA,sql echo
```

2. Starten Sie einen Netzwerk-Datenbankserver mit dieser Datenbank. Dieser Server fungiert als Webserver.

```
dbsrv16 -xs http(port=8082) -n echo echo.db
```

Der HTTP-Webserver ist so eingestellt, dass er Port 8082 auf Anforderungen abhört. Verwenden Sie eine andere Portnummer, wenn 8082 in Ihrem Netzwerk nicht zulässig ist.

3. Stellen Sie über Interactive SQL eine Verbindung mit dem Datenbankserver her.

```
dbisql -c "UID=DBA;PWD=sql;SERVER=echo"
```

4. Erstellen Sie einen neuen Webdienst zum Annehmen der eingehenden Anforderungen.

```
CREATE SERVICE EchoService
TYPE 'RAW'
AUTHORIZATION OFF
SECURE OFF
USER DBA
AS CALL Echo();
```

Diese Anweisung erstellt einen neuen Dienst mit dem Namen **EchoService**, der eine gespeicherte Prozedur mit dem Namen **Echo** aufruft, wenn ein Webclient eine Anforderung an den Dienst sendet. Sie generiert den Haupttext einer HTTP-Antwort ohne Formatierung (RAW) für den Webclient. Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

5. Erstellen Sie die **Echo**-Prozedur zum Bearbeiten von eingehenden Anforderungen.

```
CREATE OR REPLACE PROCEDURE Echo()  
BEGIN  
    DECLARE request_body LONG VARCHAR;  
    DECLARE request_mimetype LONG VARCHAR;  
  
    SET request_mimetype = http_header( 'Content-Type' );  
    SET request_body = isnull( http_variable('text'),  
http_variable('body') );  
    IF request_body IS NULL THEN  
        CALL sa_set_http_header('Content-Type', 'text/plain' );  
        SELECT 'failed'  
    ELSE  
        CALL sa_set_http_header('Content-Type', request_mimetype );  
        SELECT request_body;  
    END IF;  
END
```

Diese Prozedur formatiert den **Content-Type**-Header und den Hauptteil der Antwort, die an den Webclient gesendet wird.

Ergebnisse

Ein Webserver wird eingerichtet, um Anforderungen zu empfangen und Antworten zu senden.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 2: Anforderungen über einen Webclient senden und Antworten empfangen](#)“ auf Seite 836.

Siehe auch

- „CREATE SERVICE-Anweisung [HTTP-Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „CREATE PROCEDURE-Anweisung [Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „CREATE FUNCTION-Anweisung [Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „HTTP_VARIABLE-Funktion [Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „SQL Anywhere als HTTP-Webserver“ auf Seite 755

Lektion 2: Anforderungen über einen Webclient senden und Antworten empfangen

In dieser Lektion richten Sie einen Datenbankclient ein, um Anforderungen mit der POST-Methode an einen Webserver zu senden und die Antworten des Webservers zu empfangen.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie einen Webserver gemäß den Anweisungen in Lektion 1 eingerichtet haben. Weitere Hinweise zum Einrichten eines Datenbankservers für den Empfang der Anforderungen von dem in dieser Lektion beschriebenen Webclient finden Sie unter „[Lektion 1](#)“:

Webserver für das Empfangen von Anforderungen und das Senden von Antworten einrichten“ auf Seite 835.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Webserver erstellen und über einen Webclient darauf zugreifen](#)“.

Kontext und Bemerkungen

Diese Lektion enthält mehrere Verweise auf **localhost**. Verwenden Sie statt **localhost** den Hostnamen oder die IP-Adresse des Webserver aus Lektion 1, wenn der Webclient nicht auf demselben Computer ausgeführt wird wie der Webserver.

Aufgabe

1. Erstellen Sie eine SQL Anywhere-Datenbank, die verwendet wird, um Webclient-Prozeduren zu speichern.

```
dbinit -dba DBA,sql echo_client
```

2. Starten Sie einen Netzwerk-Datenbankserver mit dieser Datenbank. Dieser Server fungiert als Webclient.

```
dbsrv16 echo_client.db
```

3. Stellen Sie über Interactive SQL eine Verbindung mit dem Datenbankserver her.

```
dbisql -c "UID=DBA;PWD=sql;SERVER=echo_client"
```

4. Erstellen Sie eine neue gespeicherte Prozedur zum Senden von Anforderungen an einen Webdienst.

```
CREATE OR REPLACE PROCEDURE SendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
TYPE 'HTTP:POST:!mimeType';
```

Die **SendWithMimeType**-Prozedur hat drei Parameter. Der **value**-Parameter stellt den Hauptteil der Anforderung dar, die an den Webdienst gesendet werden soll. Der **urlSpec**-Parameter gibt die URL an, die für die Verbindung mit dem Webdienst verwendet werden soll. **mimeType** gibt an, welcher MIME-Typ für HTTP:POST verwendet werden soll.

5. Senden Sie eine Anforderung an den Webserver und rufen Sie die Antwort ab.

```
CALL SendWithMimeType('<hello>this is xml</hello>',
    'text/xml',
    'http://localhost:8082/EchoService'
);
```

Die **http://localhost:8082/EchoService**-Zeichenkette gibt an, dass der Webserver auf **localhost** ausgeführt wird und Port 8082 abhört. Der gewünschte Webdienst hat den Namen **EchoService**.

6. Probieren Sie einen anderen MIME-Typ aus und beobachten Sie die Antwort.

```
CALL SendWithMimeType('{ "menu": {   "id": "file", "value": "File",
"popup": {
    "menuitem": [{ "value": "New", "onclick": "CreateNew()" },
                  { "value": "Open", "onclick": "Open()" },
                  { "value": "Close", "onclick": "Close()" } ] } } }',
'application/json',
'http://localhost:8082/EchoService'
);
```

Ergebnisse

Ein Webclient wird eingerichtet, um Anforderungen mit der POST-Methode an einen Webserver zu senden und die Antworten des Webserver zu empfangen.

Beispiel

Das folgende Beispiel ist eine Ergebnismenge, wie sie von Interactive SQL angezeigt wird:

Attribut	Wert
Status	HTTP/1.1 200 OK
Body	<hello>this is xml</hello>
Server	SQLAnywhere/16.0.1234
Expires	Tue, 09 Oct 2012 21:06:01 GMT
Date	Tue, 09 Oct 2012 21:06:01 GMT
Content-Type	text/xml; charset=windows-1252
Connection	close

Der folgende Code zeigt das HTTP-Paket, das an den Webserver gesendet wird:

```
POST /EchoService HTTP/1.0
ASA-Id: 46758096650a44088c77237cc8719d5c
User-Agent: SQLAnywhere/16.0.1234
Accept-Charset: windows-1252, UTF-8, *
Date: Tue, 09 Oct 2012 21:06:01 GMT
Host: localhost:8082
Connection: close
Content-Type: text/xml; charset=windows-1252
Content-Length: 26

<hello>this is xml</hello>
```

Der folgende Code zeigt die Antwort vom Webserver:

```
HTTP/1.1 200 OK
Date: Tue, 09 Oct 2012 21:06:01 GMT
Connection: close
Expires: Tue, 09 Oct 2012 21:06:01 GMT
```

```
Content-Type: text/xml; charset=windows-1252
Server: SQLAnywhere/16.0.1234

<hello>this is xml</hello>
```

Siehe auch

- „CREATE PROCEDURE-Anweisung [Webdienste]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Praktische Einführung: Verwenden von SQL Anywhere für den Zugriff auf einen SOAP/DISH-Dienst

Diese praktische Einführung veranschaulicht, wie Sie einen SOAP-Server erstellen können, der einen vom Webclient gelieferten Temperaturwert von Fahrenheit in Celsius konvertiert.

Erforderliche Software

- SQL Anywhere 16

Kenntnisse und Erfahrungen

- Kenntnisse über SOAP
- Grundkenntnisse über SQL Anywhere-Webdienste

Ziele

- Erstellen und Starten einer neuen SQL Anywhere-Webserver-Datenbank
- Erstellen eines SOAP-Webdienstes
- Richten Sie eine Prozedur ein, die einen vom Client bereitgestellten Fahrenheit-Wert in einen Celsius-Wert konvertiert.
- Erstellen und Starten einer neuen SQL Anywhere-Webclient-Datenbank
- Senden Sie vom Webclient aus eine SOAP-Anforderung an den Datenbankserver.
- Senden Sie vom Datenbankserver aus eine SOAP-Antwort an den Webclient.

Privilegien

Die folgenden Privilegien sind erforderlich, um die Lektionen dieser praktischen Einführung auszuführen.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE
- SERVER OPERATOR

Empfohlene Hintergrundlektüre

- „SQL Anywhere als HTTP-Webserver“ auf Seite 755

Lektion 1: Einrichten eines Webserver für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten

In dieser Lektion richten Sie einen neuen Datenbankserver ein und erstellen einen SOAP-Dienst für die Verarbeitung von eingehenden SOAP-Anforderungen. Der Server erwartet SOAP-Anforderungen, die einen Fahrenheit-Temperaturwert liefern, der in Grad Celsius konvertiert wird.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Verwenden von SQL Anywhere für den Zugriff auf einen SOAP/DISH-Dienst](#)“.

Aufgabe

1. Erstellen Sie eine SQL Anywhere-Datenbank, die verwendet wird, um Webdienstdefinitionen zu speichern.

```
dbinit -dba DBA,sql ftc
```

2. Starten Sie einen Datenbankserver mit dieser Datenbank. Dieser Server fungiert als Webserver.

```
dbsrv16 -xs http(port=8082) -n ftc ftc.db
```

Der HTTP-Webserver ist so eingestellt, dass er Port 8082 auf Anforderungen abhört. Verwenden Sie eine andere Portnummer, wenn 8082 in Ihrem Netzwerk nicht zulässig ist.

3. Stellen Sie über Interactive SQL eine Verbindung mit dem Datenbankserver her.

```
dbisql -c "UID=DBA;PWD=sql;SERVER=ftc"
```

4. Erstellen Sie einen neuen DISH-Dienst zum Annehmen von eingehenden Anforderungen.

```
CREATE SERVICE soap_endpoint  
  TYPE 'DISH'  
  AUTHORIZATION OFF  
  SECURE OFF  
  USER DBA;
```

Diese Anweisung erstellt einen neuen DISH-Dienst namens **soap_endpoint**, der eingehende SOAP-Dienstanforderungen verarbeitet. Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

5. Erstellen Sie einen neuen SOAP-Dienst für die Konvertierung von Fahrenheit in Celsius.

```
CREATE SERVICE FtoCService  
  TYPE 'SOAP'  
  FORMAT 'XML'  
  AUTHORIZATION OFF
```

```

USER DBA
AS CALL FToCConverter( :fahrenheit );

```

Diese Anwendung erstellt einen neuen SOAP-Dienst mit dem Namen **FtoCService**, der als Ausgabe in XML formatierte Zeichenfolgen generiert. Es ruft eine gespeicherte Prozedur mit dem Namen **FToCConverter** auf, wenn ein Webclient eine SOAP-Anforderung an den Dienst sendet. Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

- Erstellen Sie die **FToCConverter**-Prozedur zum Bearbeiten von eingehenden SOAP-Anforderungen. Diese Prozedur führt die erforderlichen Berechnungen zur Konvertierung eines vom Client gelieferten Fahrenheit-Temperaturwerts in den entsprechenden Celsius-Temperaturwert aus.

```

CREATE OR REPLACE PROCEDURE FToCConverter( temperature FLOAT )
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    DECLARE alias LONG VARCHAR;
    DECLARE first_name LONG VARCHAR;
    DECLARE last_name LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE authinfo LONG VARCHAR;
    DECLARE namespace LONG VARCHAR;
    DECLARE mustUnderstand LONG VARCHAR;
header_loop:
    LOOP
        SET hd_key = NEXT_SOAP_HEADER( hd_key );
        IF hd_key IS NULL THEN
            -- no more header entries
            LEAVE header_loop;
        END IF;
        IF hd_key = 'Authentication' THEN
            SET hd_entry = SOAP_HEADER( hd_key );
            SET xpath = '/*: ' || hd_key || '/*:userName';
            SET namespace = SOAP_HEADER( hd_key, 1, '@namespace' );
            SET mustUnderstand = SOAP_HEADER( hd_key, 1,
'mustUnderstand' );
            BEGIN
                -- parse the XML returned in the SOAP header
                DECLARE crsr CURSOR FOR
                    SELECT * FROM OPENXML( hd_entry, xpath )
                        WITH ( alias LONG VARCHAR '@*:alias',
                            first_name LONG VARCHAR '*:first/text()',
                            last_name LONG VARCHAR '*:last/text()' );
                OPEN crsr;
                FETCH crsr INTO alias, first_name, last_name;
                CLOSE crsr;
            END;

            -- build a response header
            -- based on the pieces from the request header
            SET authinfo =
                XMLELEMENT( 'Authentication',
                    XMLATTRIBUTES(
                        namespace as xmlns,
                        alias,
                        mustUnderstand ),
                    XMLELEMENT( 'first', first_name ),
                    XMLELEMENT( 'last', last_name ) );
            CALL SA_SET_SOAP_HEADER( 'authinfo', authinfo );
        END IF;
    END LOOP;
END;

```

```
END LOOP header_loop;
SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) AS answer;
END;
```

Die Funktion `NEXT_SOAP_HEADER` wird in einer LOOP-Struktur verwendet, um eine Iteration durch alle Headernamen in einer SOAP-Anforderung durchzuführen. Die Schleife wird beendet, wenn die Funktion `NEXT_SOAP_HEADER` den Wert NULL zurückgibt.

Hinweis

Diese Funktion führt die Iteration durch die Header nicht notwendigerweise in derselben Reihenfolge durch, in der sie in der SOAP-Anforderung aufgeführt sind.

Die Funktion `SOAP_HEADER` gibt den Headerwert zurück oder NULL, wenn der Headername nicht vorhanden ist. Die **FToCConverter**-Prozedur sucht nach einem Header mit dem Namen **Authentication** und extrahiert die Headerstruktur, einschließlich der Attribute **@namespace** und **mustUnderstand**. Das Header-Attribut **@namespace** ist ein spezielles SQL Anywhere-Attribut, das zum Zugriff auf den Namespace (**xmlns**) des angegebenen Headereintrags verwendet wird.

Im Folgenden finden Sie ein XML-Zeichenfolge, die eine mögliche **Authentication**-Headerstruktur repräsentiert, wobei das Attribut **@namespace** den Wert **"SecretAgent"** und **mustUnderstand** den Wert 1 hat:

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

Die OPENXML-Systemprozedur in der SELECT-Anweisung führt eine syntaktische Analyse der XML-Header-Zeichenfolge **"/*:Authentication/*:userName"** durch, um den Attributwert **alias** und den Inhalt der Tags **first** und **last** zu extrahieren. Die Ergebnismenge wird mit einem Cursor verarbeitet, um die drei Spaltenwerte abzurufen.

Sie haben nun alle gewünschten Informationen, die an den Webdienst übergeben wurden. Sie haben die Temperatur in Fahrenheit und einige zusätzliche Attribute, die dem Webdienst in einem SOAP-Header übergeben wurden. Sie können den Namen und den Alias suchen, die übergeben wurden, um festzustellen, ob die Person für die Verwendung des Webdienstes autorisiert ist. Diese Übung wird jedoch nicht in dem Beispiel gezeigt.

Die SET-Anweisung wird zum Erstellen einer SOAP-Antwort im XML-Format verwendet, die an den Client gesendet wird. Im Folgenden finden Sie eine XML-Zeichenfolge, die eine mögliche SOAP-Antwort repräsentiert. Sie basiert auf dem weiter oben aufgeführten Beispiel mit der **Authentication**-Headerstruktur.

```
<Authentication xmlns="SecretAgent" alias="99" mustUnderstand="1">
  <first>Susan</first>
  <last>Hilton</last>
</Authentication>
```

Die Systemprozedur `SA_SET_SOAP_HEADER` wird zum Einrichten des SOAP-Antwort-Headers verwendet, der an den Client gesendet wird.

Die endgültige SELECT-Anweisung wird zum Konvertieren des gelieferten Fahrenheit-Werts in einen Celsius-Wert verwendet. Diese Informationen werden dann wieder an den Client weitergegeben.

Ergebnisse

Sie haben nun einen laufenden SQL Anywhere-Webserver, der einen Dienst für die Konvertierung der Temperaturen von Fahrenheit in Celsius bereitstellt. Dieser Dienst verarbeitet einen SOAP-Header vom Client und sendet eine SOAP-Antwort zurück an den Client.

Nächste Schritte

In der nächsten Lektion entwickeln Sie ein Beispiel für einen Client, der SOAP-Anforderungen an den Webserver senden und SOAP-Antworten vom Webserver empfangen kann. Gehen Sie weiter zu [„Lektion 2: Einrichten eines Webclients zum Senden von SOAP-Anforderungen und Empfangen von SOAP-Antworten“](#) auf Seite 843.

Siehe auch

- „DISH-Dienste erstellen“ auf Seite 764
- „CREATE SERVICE-Anweisung [SOAP-Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE PROCEDURE-Anweisung [Webdienste]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „SOAP_HEADER-Funktion [SOAP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „NEXT_SOAP_HEADER-Funktion [SOAP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Anforderungen an den SOAP-Namespace-URI“ auf Seite 809
- „OPENXML-Operator“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Lektion 2: Einrichten eines Webclients zum Senden von SOAP-Anforderungen und Empfangen von SOAP-Antworten

In dieser Lektion richten Sie einen Webclient ein, der SOAP-Anforderungen sendet und SOAP-Antworten empfängt.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie einen Webserver gemäß den Anweisungen in der vorherigen Lektion eingerichtet haben. Siehe [„Lektion 1: Einrichten eines Webservers für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten“](#) auf Seite 840.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: [„Praktische Einführung: Verwenden von SQL Anywhere für den Zugriff auf einen SOAP/DISH-Dienst“](#).

Kontext und Bemerkungen

Diese Lektion enthält mehrere Verweise auf **localhost**. Verwenden Sie statt **localhost** den Hostnamen oder die IP-Adresse des Webserver aus Lektion 1, wenn der Webclient nicht auf demselben Computer ausgeführt wird wie der Webserver.

Aufgabe

1. Führen Sie den folgenden Befehl aus, um eine SQL Anywhere-Datenbank zu erstellen:

```
dbinit -dba DBA,sql ftc_client
```

2. Starten Sie den Datenbankserver mit dem folgenden Befehl:

```
dbsrv16 ftc_client.db
```

3. Stellen Sie in Interactive SQL mit dem folgenden Befehl eine Verbindung zur Datenbank her:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=ftc_client"
```

4. Erstellen Sie eine neue gespeicherte Prozedur zum Senden von SOAP-Anforderungen an einen DISH-Dienst.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE OR REPLACE PROCEDURE FtoC( fahrenheit FLOAT,  
    INOUT inouthead LONG VARCHAR,  
    IN inheader LONG VARCHAR )  
URL 'http://localhost:8082/soap_endpoint'  
SET 'SOAP(OP=FtoCService)'  
TYPE 'SOAP:DOC'  
SOAPHEADER '!inouthead!inheader';
```

Die Zeichenfolge **http://localhost:8082/soap_endpoint** in der URL-Klausel gibt an, dass der Webserver auf **localhost** ausgeführt wird und Port 8082 überwacht. Der gewünschte DISH-Webdienst hat den Namen **soap_endpoint** und dient als SOAP-Endpunkt.

Die SET-Klausel gibt den Namen des SOAP-Vorgangs oder Diensts **FtoCService** an, der aufgerufen werden soll.

Das Standardformat für eine Webdienstanforderung ist SOAP:RPC. Das gewählte Format in diesem Beispiel ist "SOAP:DOC". Es ist ähnlich wie "SOAP:RPC", ermöglicht aber eine größere Vielfalt von Datentypen. SOAP-Anforderungen werden immer als XML-Dokumente gesendet. Die Methode zum Senden von SOAP-Anforderungen ist HTTP:POST.

Die Ersetzungsvariablen (**inouthead**, **inheader**) in einer Webdienst-Clientprozedur wie FtoC müssen alphanumerisch sein. Wenn der Webdienstclient als Funktion deklariert wird, sind seine Parameter auf den IN-Modus beschränkt (sie können von der aufgerufenen Funktion nicht zugewiesen werden). Aus diesem Grund werden OPENXML und andere Zeichenfolgenfunktionen verwendet, um die Informationen des SOAP-Antwort-Headers zu extrahieren.

5. Erstellen Sie eine Wrapper-Prozedur, die zwei spezielle SOAP-Anforderungs-Headereinträge erstellt, diese an die Prozedur **FtoC** übergibt und Serverantworten verarbeitet.

Führen Sie hierzu die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE OR REPLACE PROCEDURE FahrenheitToCelsius( Fahrenheit FLOAT )
BEGIN
  DECLARE io_header LONG VARCHAR;
  DECLARE in_header LONG VARCHAR;
  DECLARE result LONG VARCHAR;
  DECLARE err INTEGER;
  DECLARE crsr CURSOR FOR
    CALL FtoC( Fahrenheit, io_header, in_header );
  SET io_header =
    '<Authentication xmlns="SecretAgent" ' ||
    'mustUnderstand="1">' ||
    '<userName alias="99">' ||
    '<first>Susan</first><last>Hilton</last>' ||
    '</userName>' ||
    '</Authentication>';
  SET in_header =
    '<Session xmlns="SomeSession">' ||
    '123456789' ||
    '</Session>';

  MESSAGE 'send, soapheader=' || io_header || in_header;
  OPEN crsr;
  FETCH crsr INTO result, err;
  CLOSE crsr;
  MESSAGE 'receive, soapheader=' || io_header;
  SELECT Fahrenheit, Celsius
    FROM OPENXML(result, '//tns:answer', 1, result)
    WITH ("Celsius" FLOAT 'text()');
END;
```

Die erste SET-Anweisung erstellt die XML-Repräsentation eines SOAP-Headereintrags, um dem Webserver die Anmeldeinformationen des Benutzers mitzuteilen:

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

Die zweite SET-Anweisung legt die XML-Repräsentation eines SOAP-Headereintrags fest, um die Clientsitzungs-ID zu verfolgen:

```
<Session xmlns="SomeSession">123456789</Session>
```

6. Die OPEN-Anweisung sorgt dafür, dass die FtoC-Prozedur aufgerufen wird, die eine SOAP-Anforderung an den Webserver sendet und dann die Antwort vom Webserver verarbeitet. Die Antwort enthält einen Header, der in **inoutheader** zurückgegeben wird.

Ergebnisse

Sie haben nun einen Client, der SOAP-Anforderungen an den Webserver senden und SOAP-Antworten vom Webserver empfangen kann.

Nächste Schritte

In der nächsten Lektion senden Sie eine SOAP-Anforderung an den Webserver und überprüfen die SOAP-Antwort. Gehen Sie weiter zu [„Lektion 3: Senden einer SOAP-Anforderung und Empfangen einer SOAP-Antwort“](#) auf Seite 846.

Siehe auch

- „CREATE PROCEDURE-Anweisung [Webdienste]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „CREATE FUNCTION-Anweisung [Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Lektion 3: Senden einer SOAP-Anforderung und Empfangen einer SOAP-Antwort

In dieser Lektion rufen Sie die in der vorangehenden Lektion erstellte Wrapper-Prozedur auf, die eine SOAP-Anforderung an den in der ersten Lektion erstellte Webserver sendet.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie einen Webserver gemäß den Anweisungen in Lektion 1 eingerichtet haben. Siehe [„Lektion 1: Einrichten eines Webservers für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten“](#) auf Seite 840.

In dieser Lektion wird davon ausgegangen, dass Sie einen Webclient gemäß den Anweisungen in Lektion 2 eingerichtet haben. Siehe [„Lektion 2: Einrichten eines Webclients zum Senden von SOAP-Anforderungen und Empfangen von SOAP-Antworten“](#) auf Seite 843.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: [„Praktische Einführung: Verwenden von SQL Anywhere für den Zugriff auf einen SOAP/DISH-Dienst“](#).

Aufgabe

1. Stellen Sie in Interactive SQL eine Verbindung mit der Clientdatenbank her, wenn diese nicht bereits von der zweiten Lektion her geöffnet ist.

```
dbisql -c "UID=DBA;PWD=sql;SERVER=ftc_client"
```

2. Aktivieren der Protokollierung von SOAP-Anforderungen und Antworten.

Führen Sie hierzu die folgende SQL-Anweisung in Interactive SQL aus:

```
CALL sa_server_option('WebClientLogFile', 'soap.txt');  
CALL sa_server_option('WebClientLogging', 'ON');
```

Mit diesen Aufrufen können Sie den Inhalt der SOAP-Anforderung und Antwort prüfen. Die Anforderungen und Antworten werden in einer Datei namens *soap.txt* protokolliert.

3. Rufen Sie die Wrapper-Prozedur zum Senden einer SOAP-Anforderung und zum Empfangen einer SOAP-Antwort auf.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CALL FahrenheitToCelsius(212);
```

Dieser Aufruf gibt einen Fahrenheit-Wert von 212 an die Prozedur **FahrenheitToCelsius** weiter, welche ihrerseits diesen Wert zusammen mit zwei benutzerdefinierten SOAP-Headern an die Prozedur **FToC** weitergibt. Beide clientseitigen Prozeduren werden in der vorherigen Lektion erstellt.

Ergebnisse

Die Webdienstprozedur **FToC** sendet den Fahrenheit-Wert und den SOAP-Header an den Webserver. Die SOAP-Anforderung enthält folgende Daten.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" mustUnderstand="1">
      <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:FtoCService>
      <m:fahrenheit>212</m:fahrenheit>
    </m:FtoCService>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die FtoC-Prozedur empfängt dann die Antwort vom Webserver wobei auch eine auf dem Fahrenheit-Wert basierende Ergebnismenge enthalten ist. Die SOAP-Antwort enthält folgende Daten.

```
<SOAP-ENV:Envelope
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:tns='http://localhost:8082'>
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" alias="99" mustUnderstand="1">
      <first>Susan</first>
      <last>Hilton</last>
    </Authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <tns:FtoCServiceResponse>
      <tns:FtoCServiceResult xsi:type='xsd:string'>
        ftc"rowset"xmlns:tns="http://localhost:8082/
        <tns:row"xmlns:tns="http://localhost:8082/
        <tns:answer"xmlns:tns="http://localhost:8082/
        <tns:row"xmlns:tns="http://localhost:8082/
        <tns:rowset"xmlns:tns="http://localhost:8082/
      </tns:FtoCServiceResult>
      <tns:sqlcode>0</tns:sqlcode>
    </tns:FtoCServiceResponse>
```

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Der Inhalt von **<SOAP-ENV:Header>** wird in **inoutheader** zurückgegeben.

Wenn Sie die SOAP-Antwort überprüfen, können Sie sehen, dass die Ergebnismenge in der Antwort vom Webdienst **FToCService** kodiert ist. Die Ergebnismenge wird dekodiert und an die Prozedur **FahrenheitToCelsius** zurückgegeben. Die Ergebnismenge sieht wie folgt aus, wenn ein Fahrenheit-Wert von 212 an den Webserver übergeben wird:

```
<tns:rowset xmlns:tns="http://localhost:8082/ftc">
  <tns:row>
    <tns:answer>100
  </tns:answer>
</tns:row>
</tns:rowset>
```

Die SELECT-Anweisung in der Prozedur **FahrenheitToCelsius** verwendet die Funktion OPENXML zur syntaktischen Analyse der SOAP-Antwort und extrahiert den durch die **tns:answer**-Struktur definierten Celsius-Wert.

Die folgende wird in Interactive SQL angezeigt:

```
Fahrenheit    Celsius
      212           100
```

Beispiel

Hier sehen Sie einen weiteren Beispielaufwurf an den SOAP-Webdienst, mit dem ein Temperaturwert von Fahrenheit in Celsius konvertiert wird.

```
CALL FahrenheitToCelsius(32);
```

Die folgende wird in Interactive SQL angezeigt:

```
Fahrenheit    Celsius
      32           0
```

Siehe auch

- „sa_server_option-Systemprozedur“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Praktische Einführung: Verwenden von Visual C# für den Zugriff auf einen SOAP/DISH-Webdienst

In dieser praktischen Einführung wird erklärt, wie Sie eine Visual C#-Clientanwendung für den Zugriff auf SOAP/DISH-Dienste auf einem SQL Anywhere-Webserver erstellen.

Erforderliche Software

- SQL Anywhere 16
- Visual Studio

Kenntnisse und Erfahrungen

- Kenntnisse über SOAP
- Kenntnisse über .NET Framework
- Grundkenntnisse über SQL Anywhere-Webdienste

Ziele

- Erstellen und Starten einer neuen SQL Anywhere-Webserver-Datenbank
- Erstellen eines SOAP-Webdienstes
- Einrichten einer Prozedur, die die in SOAP-Anforderungen enthaltenen Informationen zurückgibt.
- Erstellen eines DISH-Webdienstes, der WSDL-Dokumente bereitstellt und als Proxy agiert.
- Einrichten von Visual C# auf dem Clientcomputer und importieren eines WSDL-Dokuments vom Webserver.
- Erstellen einer Java-Clientanwendung zum Abrufen von Informationen vom SOAP-Dienst unter Verwendung der Informationen aus dem WSDL-Dokument.

Privilegien

Die folgenden Privilegien sind erforderlich, um die Lektionen dieser praktischen Einführung auszuführen.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Empfohlene Hintergrundlektüre

- „SQL Anywhere als HTTP-Webserver“ auf Seite 755

Lektion 1: Einrichten eines Webserver für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten

In dieser Lektion richten Sie einen SQL Anywhere-Webserver ein, auf dem SOAP- und DISH-Webdienste ausgeführt werden und der die Anforderungen der Visual C#-Clientanwendung verarbeitet.

Voraussetzungen

Eine neue Version von Visual Studio ist erforderlich.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Verwenden von Visual C# für den Zugriff auf einen SOAP/DISH-Webdienst](#)“.

Aufgabe

1. Starten Sie die SQL Anywhere-Demodatenbank mit dem folgenden Befehl:

```
dbsrv16 -xs http(port=8082) "%SQLANYAMP16%\demo.db"
```

Dieser Befehl gibt an, dass der HTTP-Webserver an Port 8082 auf Anforderungen warten soll. Verwenden Sie eine andere Portnummer, wenn die Verwendung von 8082 in Ihrem Netzwerk nicht zulässig ist.

2. Stellen Sie in Interactive SQL mit dem folgenden Befehl eine Verbindung zum Datenbankserver her:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=demo"
```

3. Erstellen Sie einen neuen SOAP-Dienst zum Annehmen der eingehenden Anforderungen.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE SERVICE "SASoapTest/EmployeeList"  
  TYPE 'SOAP'  
  DATATYPE ON  
  AUTHORIZATION OFF  
  SECURE OFF  
  USER DBA  
  AS SELECT * FROM Employees;
```

Diese Anweisung erstellt einen neuen SOAP-Dienst mit dem Namen **SASoapTest/EmployeeList**, der als Ausgabe einen SOAP-Typ generiert. Der Dienst wählt alle Spalten aus der Tabelle **Employees** aus und gibt die Ergebnismenge an den Client zurück. Der Dienstname ist in Anführungszeichen gesetzt, weil darin ein Schrägstrich (/) vorkommt. Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

DATATYPE ON gibt an, dass explizite Datentypinformationen in der zurückgegebenen XML-Ergebnismenge und den Eingabeparametern generiert werden. Diese Option wirkt sich nicht auf das WSDL-Dokument aus, das generiert wird.

Da die FORMAT-Klausel nicht angegeben ist, wird das SOAP-Dienstformat durch das zugeordnete DISH-Dienstformat bestimmt, das im nächsten Schritt deklariert wird.

4. Erstellen Sie einen neuen DISH-Dienst, der als Proxy für den SOAP-Dienst agiert und das WSDL-Dokument generiert.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE SERVICE SASoapTest_DNET  
  TYPE 'DISH'  
  GROUP SASoapTest  
  FORMAT 'DNET'  
  AUTHORIZATION OFF  
  SECURE OFF  
  USER DBA;
```

DISH-Webdienste, auf die von .NET aus zugegriffen wird, sollten mit der FORMAT 'DNET'-Klausel deklariert werden. Die GROUP-Klausel kennzeichnet die SOAP-Dienste, die vom DISH-Dienst verarbeitet werden sollen. Der im vorherigen Schritt erstellte **EmployeeList**-Dienst ist Teil der

GROUP SASoapTest, weil er als **SASoapTest/EmployeeList** deklariert ist. Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

5. Vergewissern Sie sich, dass der DISH-Webdienst funktionsfähig ist, indem Sie über einen Webbrowser auf das zugehörige WSDL-Dokument zugreifen.

Öffnen Sie Ihren Webbrowser und gehen Sie zu http://localhost:8082/demo/SASoapTest_DNET.

Der DISH-Dienst generiert automatisch ein WSDL-Dokument, das im Browserfenster angezeigt wird.

Ergebnisse

Sie haben einen SQL Anywhere-Webserver eingerichtet, auf dem SOAP- und DISH-Webdienste ausgeführt werden, die Visual C#-Clientanwendungsanforderungen abwickeln können.

Nächste Schritte

In der nächsten Lektion erstellen Sie eine Visual C#-Anwendung für die Kommunikation mit dem Webserver. Gehen Sie weiter zu „[Lektion 2: Erstellen einer Visual C#-Anwendung für die Kommunikation mit dem Webserver](#)“ auf Seite 851.

Siehe auch

- „DISH-Dienste erstellen“ auf Seite 764
- „CREATE SERVICE-Anweisung [SOAP-Webdienst]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Lektion 2: Erstellen einer Visual C#-Anwendung für die Kommunikation mit dem Webserver

In dieser Lektion erstellen Sie eine Visual C#-Anwendung für die Kommunikation mit dem Webserver.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie einen Webserver gemäß den Anweisungen in Lektion 1 eingerichtet haben. Siehe „[Lektion 1: Einrichten eines Webservers für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten](#)“ auf Seite 849.

Eine neue Version von Visual Studio ist erforderlich, um diese Lektion durchzuführen.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Verwenden von Visual C# für den Zugriff auf einen SOAP/DISH-Webdienst](#)“.

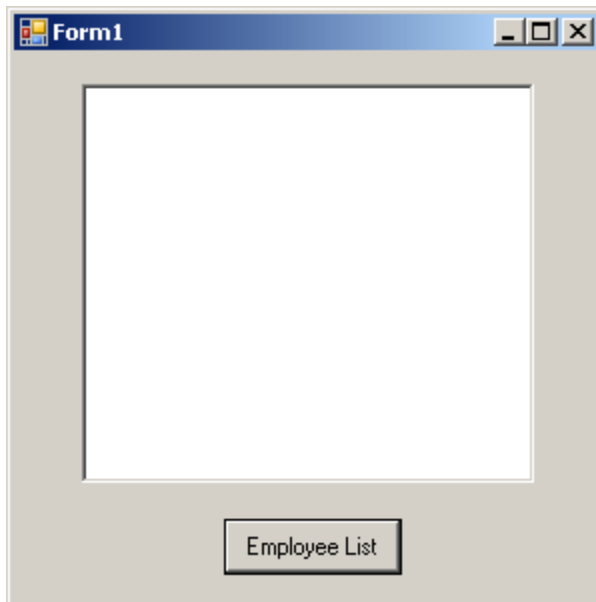
Kontext und Bemerkungen

Diese Lektion enthält mehrere Verweise auf **localhost**. Verwenden Sie statt **localhost** den Hostnamen oder die IP-Adresse des Webservers aus Lektion 1, wenn der Webclient nicht auf demselben Computer ausgeführt wird wie der Webserver.

Dieses Beispiel verwendet Funktionen aus .NET Framework 2.0.

Aufgabe

1. Starten Sie Visual Studio.
2. Erstellen Sie ein neues **Windows Forms-Anwendungsprojekt** in Visual C#.
Ein leeres Formular wird angezeigt.
3. Fügen Sie dem Projekt einen Webverweis hinzu.
 - a. Klicken Sie auf **Projekt » Dienstverweis hinzufügen**.
 - b. Klicken Sie im Fenster "Dienstverweis hinzufügen" auf die Option **Erweitert**.
 - c. Klicken Sie im Fenster "Einstellungen für Dienstverweise" auf die Option **Webverweis hinzufügen**.
 - d. Geben Sie im Fenster "Webverweis hinzufügen" die Zeichenfolge **http://localhost:8082/demo/SASoapTest_DNET** in das Feld **URL** ein.
 - e. Klicken Sie auf **Gehe zu** (oder den grünen Pfeil).
Visual Studio listet die im SASoapTest_DNET-Dienst verfügbare EmployeeList-Methode auf.
 - f. Klicken Sie auf **Verweis hinzufügen**.
Visual Studio fügt **localhost** zum Projekt **Webverweise** im Fenster **Projektmappen-Explorer** hinzu.
4. Füllen Sie das leere Formular mit den gewünschten Objekte für die Webclient-Anwendung mit Daten.
Ziehen Sie im Fenster **Toolbox** die ListBox- und Button-Objekte in das Formular und aktualisieren Sie die Textattribute so, dass das Formular dem in der folgenden Darstellung ähnelt:



5. Schreiben Sie eine Prozedur, die auf den Webverweis zugreift und die verfügbaren Methoden verwendet.

Doppelklicken Sie auf die Schaltfläche **Employee List** und fügen Sie den folgenden Code für das Schaltflächenereignis "Click" hinzu:

```
int sqlCode;

listBox1.Items.Clear();

localhost.SASoapTest_DNET proxy = new localhost.SASoapTest_DNET();
DataSet results = proxy.EmployeeList(out sqlCode);
DataTableReader dr = results.CreateDataReader();

listBox1.BeginUpdate();
while (dr.Read())
{
    for (int i = 0; i < dr.FieldCount; i++)
    {
        string dataTypeName = dr.GetDataTypeName(i);
        string dataName = dr.GetName(i);
        System.Type ftype = dr.GetFieldType(i);
        System.TypeCode typeCode = System.Type.GetTypeCode(ftype);

        string columnName = "(" + dataTypeName + ")" + dataName + "=";

        if (dr.IsDBNull(i))
        {
            listBox1.Items.Add(columnName + "(null)");
        }
        else
        {
            switch (typeCode)
            {
                case System.TypeCode.Int32:
                    Int32 intValue = dr.GetInt32(i);
                    listBox1.Items.Add(columnName + intValue);
                    break;
                case System.TypeCode.Decimal:
                    Decimal decValue = dr.GetDecimal(i);
                    listBox1.Items.Add(columnName +
decValue.ToString("c"));
                    break;
                case System.TypeCode.String:
                    string stringValue = dr.GetString(i);
                    listBox1.Items.Add(columnName + stringValue);
                    break;
                case System.TypeCode.DateTime:
                    DateTime dateValue = dr.GetDateTime(i);
                    listBox1.Items.Add(columnName + dateValue);
                    break;
                case System.TypeCode.Boolean:
                    Boolean boolValue = dr.GetBoolean(i);
                    listBox1.Items.Add(columnName + boolValue);
                    break;
                default:
                    listBox1.Items.Add(columnName + "(unsupported)");
                    break;
            }
        }
    }
    listBox1.Items.Add("");
}
```

```
}
listBox1.EndUpdate();
dr.Close();
```

6. Führen Sie die Anwendung aus.

Klicken Sie auf **Debuggen** » **Debuggen starten**.

7. Kommunizieren Sie mit dem Web-Datenbankserver.

Klicken Sie auf **Employee List**.

Das **ListBox**-Objekt zeigt die Ergebnismenge **EmployeeList** als (Typ)Name=Wert-Paare an. Die folgende Ausgabe zeigt wie ein Eintrag im **ListBox**-Objekt angezeigt wird:

```
(Int32)EmployeeID=102
(Int32)ManagerID=501
(String)Surname=Whitney
(String)GivenName=Fran
(Int32)DepartmentID=100
(String)Street=9 East Washington Street
(String)City=Cornwall
(String)State=NY
(String)Country=USA
(String)PostalCode=02192
(String)Phone=6175553985
(String)Status=A
(String)SocialSecurityNumber=017349033
(Decimal)Salary=$45,700.00
(DateTime)StartDate=8/28/1984 12:00:00 AM
(DateTime)TerminationDate=(null)
(DateTime)BirthDate=6/5/1958 12:00:00 AM
(Boolean)BenefitHealthInsurance=True
(Boolean)BenefitLifeInsurance=True
(Boolean)BenefitDayCare=False
(String)Sex=F
```

Der Salary-Betrag wird in das Währungsformat des Clients konvertiert.

Werten, die ein Datum ohne Uhrzeit enthalten, wird die Uhrzeit 00:00:00 oder Mitternacht zugewiesen (deren Anzeigeformat von den Sprachumgebungseinstellungen des Clients abhängt).

Werte, die NULL enthalten, werden mit der DataTableReader.IsDBNull-Methode getestet.

Ergebnisse

Die XML-Antwort vom Webserver enthält eine formatierte Ergebnismenge. Alle Spaltendaten werden in eine Zeichenfolgendarstellung der Daten konvertiert. Die folgende Ergebnismenge veranschaulicht wie Ergebnismengen formatiert werden, wenn sie an den Client gesendet werden:

```
<row>
<EmployeeID>102</EmployeeID>
<ManagerID>501</ManagerID>
<Surname>Whitney</Surname>
<GivenName>Fran</GivenName>
<DepartmentID>100</DepartmentID>
<Street>9 East Washington Street</Street>
<City>Cornwall</City>
```

```

<State>NY</State>
<Country>USA</Country>
<PostalCode>02192</PostalCode>
<Phone>6175553985</Phone>
<Status>A</Status>
<SocialSecurityNumber>017349033</SocialSecurityNumber>
<Salary>45700.000</Salary>
<StartDate>1984-08-28</StartDate>
<BirthDate>1958-06-05</BirthDate>
<BenefitHealthInsurance>1</BenefitHealthInsurance>
<BenefitLifeInsurance>1</BenefitLifeInsurance>
<BenefitDayCare>0</BenefitDayCare>
<Sex>F</Sex>
</row>

```

Die DATATYPE ON-Klausel wurde in der vorherigen Lektion angegeben, um Datentypinformationen in der zurückgegebenen XML-Ergebnismenge zu generieren. Im Folgenden finden Sie ein Fragment aus der Antwort des Webservers. Die Typinformationen stimmen mit dem Datentyp der Datenbankspalten überein.

```

<xsd:element name='EmployeeID' minOccurs='0' type='xsd:int' />
<xsd:element name='ManagerID' minOccurs='0' type='xsd:int' />
<xsd:element name='Surname' minOccurs='0' type='xsd:string' />
<xsd:element name='GivenName' minOccurs='0' type='xsd:string' />
<xsd:element name='DepartmentID' minOccurs='0' type='xsd:int' />
<xsd:element name='Street' minOccurs='0' type='xsd:string' />
<xsd:element name='City' minOccurs='0' type='xsd:string' />
<xsd:element name='State' minOccurs='0' type='xsd:string' />
<xsd:element name='Country' minOccurs='0' type='xsd:string' />
<xsd:element name='PostalCode' minOccurs='0' type='xsd:string' />
<xsd:element name='Phone' minOccurs='0' type='xsd:string' />
<xsd:element name='Status' minOccurs='0' type='xsd:string' />
<xsd:element name='SocialSecurityNumber' minOccurs='0' type='xsd:string' />
<xsd:element name='Salary' minOccurs='0' type='xsd:decimal' />
<xsd:element name='StartDate' minOccurs='0' type='xsd:date' />
<xsd:element name='TerminationDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BirthDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BenefitHealthInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitLifeInsurance' minOccurs='0' type='xsd:boolean' />
<xsd:element name='BenefitDayCare' minOccurs='0' type='xsd:boolean' />
<xsd:element name='Sex' minOccurs='0' type='xsd:string' />

```

Spalten, die nur das Datum enthalten, werden in der XML-Antwort als yyyy-mm-dd formatiert.

Spalten, die nur die Uhrzeit enthalten, werden in der XML-Antwort als hh:mm:ss.nnn-HH:MM oder hh:mm:ss.nnn+HH:MM formatiert. An die Zeichenfolge wird ein Zeitzone-Offset angehängt (-HH:MM oder +HH:MM).

Spalten, die sowohl das Datum als auch die Uhrzeit enthalten, werden in der XML-Antwort als yyyy-mm-ddThh:mm:ss.nnn-HH:MM oder yyyy-mm-ddThh:mm:ss.nnn+HH:MM formatiert. Das Datum wird durch den Buchstaben "T" von der Uhrzeit getrennt. An die Zeichenfolge wird ein Zeitzone-Offset angehängt (-HH:MM oder +HH:MM).

Spalten, die Datum, Uhrzeit und Zeitzone-Offset enthalten, werden in der XML-Antwort als yyyy-mm-ddThh:mm:ss.nnn-HH:MM oder yyyy-mm-ddThh:mm:ss.nnn+HH:MM formatiert. Das Datum wird durch den Buchstaben "T" von der Uhrzeit getrennt. An die Zeichenfolge wird ein Zeitzone-Offset angehängt (-HH:MM oder +HH:MM).

Im Folgenden finden Sie einige Beispiele für einen Server, der in der Pazifik-Zeitzone läuft:

```
<ADate>2013-01-27</ADate>  
<ATime>12:34:56.000-08:00</ATime>  
<ADateTime>2013-01-27T12:34:56.000-08:00</ADateTime>  
<ADateTimeWithZone>2013-01-27T12:34:56.000+06:00</ADateTimeWithZone>
```

Das Format, in der diese Datums- und Uhrzeitangaben in einer .NET-Anwendung angezeigt werden, hängt von der Zeitzone des Clients und Sprachumgebungseinstellungen ab. Im Folgenden finden Sie ein Beispiel für einen Client in der Eastern-Zeitzone mit einer US-Sprachumgebung (wobei davon ausgegangen wird, dass das heutige Datum der 28. Januar 2013 ist).

```
(DateTime)ADate=1/27/2013 12:00:00 AM  
(DateTime)ATime=1/28/2013 3:34:56 PM  
(DateTime)ADateTime=1/27/2013 3:34:56 PM  
(DateTime)ADateTimeWithZone=1/27/2013 1:34:56 AM
```

Beachten Sie, dass es keine separate Datums- und Uhrzeitangabe in der TypeCodeEnumeration gibt – es gibt nur TypeCode.DateTime. Dies erklärt, warum alle Ergebnisse das Datum und die Uhrzeit enthalten.

Die Datentypinformationen in der XML-Antwort für diese Datums- und Zeitangaben folgen:

```
<xsd:element name='ADate' minOccurs='0' type='xsd:date' />  
<xsd:element name='ATime' minOccurs='0' type='xsd:time' />  
<xsd:element name='ADateTime' minOccurs='0' type='xsd:dateTime' />  
<xsd:element name='ADateTimeWithZone' minOccurs='0' type='xsd:dateTime' />
```

Praktische Einführung: Verwenden von JAX-WS für den Zugriff auf einen SOAP/DISH-Webdienst

In dieser praktischen Einführung wird erläutert, wie Sie eine Clientanwendung für die Java-API für XML-Webdienste (JAX-WS) für den Zugriff auf SOAP/DISH Dienste auf einem SQL Anywhere-Webserver erstellen.

Erforderliche Software

- SQL Anywhere 16
- JDK 1.7.0
- JAX-WS 2.2.7 oder höher

Kenntnisse und Erfahrungen

- Kenntnisse über SOAP
- Kenntnisse über Java und JAX-WS
- Grundkenntnisse über SQL Anywhere-Webdienste

Ziele

- Erstellen und Starten einer neuen SQL Anywhere-Webserver-Datenbank

- Erstellen eines SOAP-Webdienstes
- Einrichten einer Prozedur, die die in SOAP-Anforderungen enthaltenen Informationen zurückgibt.
- Erstellen eines DISH-Webdienstes, der WSDL-Dokumente bereitstellt und als Proxy agiert.
- Verwenden Sie JAX-WS auf dem Clientcomputer, um ein WSDL-Dokument vom Webserver zu verarbeiten.
- Erstellen Sie eine Java-Clientanwendung, um anhand der Informationen aus dem WSDL-Dokument Informationen aus dem SOAP-Dienst abzurufen.

Privilegien

Die folgenden Privilegien sind erforderlich, um die Lektionen dieser praktischen Einführung auszuführen.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Empfohlene Hintergrundlektüre

- „SQL Anywhere als HTTP-Webserver“ auf Seite 755

Lektion 1: Einrichten eines Webserver für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten

In dieser Lektion richten Sie einen SQL Anywhere-Webserver ein, auf dem SOAP- und DISH-Webdienste ausgeführt werden und der die Anforderungen der JAX-WS-Clientanwendung verarbeitet.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Verwenden von JAX-WS für den Zugriff auf einen SOAP/DISH-Webdienst](#)“.

Kontext und Bemerkungen

In dieser Lektion wird der Webserver eingerichtet sowie ein einfacher Webdienst, den Sie in der nächsten Lektion verwenden werden. Es kann aufschlussreich sein, Proxy-Software zu verwenden, um das XML-Nachrichtenaufkommen zu beobachten. Der Proxy fügt sich zwischen Ihrer Clientanwendung und dem Webserver ein.

Aufgabe

1. Starten Sie die SQL Anywhere-Demodatenbank mit dem folgenden Befehl:

```
dbsrv16 -xs http(port=8082) "%SQLANYAMP16%\demo.db"
```

Dieser Befehl gibt an, dass der HTTP-Webserver an Port 8082 auf Anforderungen warten soll. Verwenden Sie eine andere Portnummer, wenn die Verwendung von 8082 in Ihrem Netzwerk nicht zulässig ist.

2. Stellen Sie in Interactive SQL mit dem folgenden Befehl eine Verbindung zum Datenbankserver her:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=demo"
```

3. Erstellen Sie eine gespeicherte Prozedur, die die Spalten der Tabelle **Employees** auflistet.

Führen Sie hierzu die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE OR REPLACE PROCEDURE ListEmployees()  
RESULT (  
    EmployeeID      INTEGER,  
    Surname          CHAR(20),  
    GivenName       CHAR(20),  
    StartDate       DATE,  
    TerminationDate DATE )  
BEGIN  
    SELECT EmployeeID, Surname, GivenName, StartDate, TerminationDate  
    FROM Employees;  
END;
```

Diese Anweisungen erstellen eine neue Prozedur mit dem Namen **ListEmployees**, die die Struktur der Ausgabe der Ergebnismenge definiert und bestimmte Spalten der Tabelle "Employees" auswählt.

4. Erstellen Sie einen neuen SOAP-Dienst zum Annehmen der eingehenden Anforderungen.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE SERVICE "WS/EmployeeList"  
    TYPE 'SOAP'  
    FORMAT 'CONCRETE' EXPLICIT ON  
    DATATYPE ON  
    AUTHORIZATION OFF  
    SECURE OFF  
    USER DBA  
    AS CALL ListEmployees();
```

Diese Anweisung erstellt einen neuen SOAP-Dienst mit dem Namen **WS/EmployeeList**, der als Ausgabe einen SOAP-Typ generiert. Der Dienst ruft die **ListEmployees**-Prozedur auf, wenn ein Webclient eine Anforderung an ihn sendet. Der Dienstname ist in Anführungszeichen gesetzt, weil darin ein Schrägstrich (/) vorkommt.

SOAP-Webdienste, auf die von JAX-WS aus zugegriffen wird, sollten mit der FORMAT 'CONCRETE'-Klausel deklariert werden. Die EXPLICIT ON-Klausel gibt an, dass der entsprechende DISH-Dienst ein XML-Schema generieren soll, das ein explizites DataSet-Objekt basierend auf der Ergebnismenge der Prozedur **ListEmployees** beschreibt. Die EXPLICIT-Klausel wirkt sich nur auf das generierte WSDL-Dokument aus.

DATATYPE ON gibt an, dass explizite Datentypinformationen in der zurückgegebenen XML-Ergebnismenge und den Eingabeparametern generiert werden. Diese Option wirkt sich nicht auf das WSDL-Dokument aus, das generiert wird.

Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

5. Erstellen Sie einen neuen DISH-Dienst, der als Proxy für den SOAP-Dienst agiert und das WSDL-Dokument generiert.

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
CREATE SERVICE WSDish
  TYPE 'DISH'
  FORMAT 'CONCRETE'
  GROUP WS
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA;
```

DISH-Webdienste, auf die von JAX-WS aus zugegriffen wird, sollten mit der FORMAT 'CONCRETE'-Klausel deklariert werden. Die GROUP-Klausel kennzeichnet die SOAP-Dienste, die vom DISH-Dienst verarbeitet werden sollen. Der im vorherigen Schritt erstellte EmployeeList-Dienst ist Teil der GROUP WS, weil er als WS/EmployeeList deklariert ist. Wenn Sie sich mit einer anderen Benutzer-ID angemeldet haben, muss die USER DBA-Klausel gemäß Ihrer Benutzer-ID geändert werden.

6. Vergewissern Sie sich, dass der DISH-Webdienst funktionsfähig ist, indem Sie über einen Webbrowser auf das zugehörige WSDL-Dokument zugreifen.

Öffnen Sie Ihren Webbrowser und gehen Sie zu <http://localhost:8082/demo/WSDish>.

Der DISH-Dienst generiert automatisch ein WSDL-Dokument, das im Browserfenster angezeigt wird. Überprüfen Sie das Objekt **EmployeeListDataset**, das der folgenden Ausgabe ähnelt:

```
<s:complexType name="EmployeeListDataset">
  <s:sequence>
    <s:element name="rowset">
      <s:complexType>
        <s:sequence>
          <s:element name="row" minOccurs="0" maxOccurs="unbounded">
            <s:complexType>
              <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="EmployeeID"
nillable="true" type="s:int" />
                <s:element minOccurs="0" maxOccurs="1" name="Surname" nillable="true"
type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="GivenName"
nillable="true" type="s:string" />
                <s:element minOccurs="0" maxOccurs="1" name="StartDate"
nillable="true" type="s:date" />
                <s:element minOccurs="0" maxOccurs="1" name="TerminationDate"
nillable="true" type="s:date" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:sequence>
</s:complexType>
```

EmployeeListDataset ist das explizite Objekt, das durch die Klauseln **FORMAT 'CONCRETE'** und **EXPLICIT ON** im SOAP-Dienst **EmployeeList** generiert wird. In einem späteren Schritt verwendet die **wsimport**-Anwendung diese Informationen, um eine SOAP 1.1-Clientschnittstelle für diesen Dienst zu generieren.

Ergebnisse

Sie haben einen SQL Anywhere-Webserver eingerichtet, auf dem SOAP- und DISH-Webdienste ausgeführt werden, die JAX-WS-Clientanwendungsanforderungen bearbeiten können.

Nächste Schritte

In der nächsten Lektion erstellen Sie eine Java-Anwendung für die Kommunikation mit dem Webserver. Gehen Sie weiter zu „[Lektion 2: Java-Anwendung für die Kommunikation mit dem Webserver erstellen](#)“ auf Seite 860.

Siehe auch

- „DISH-Dienste erstellen“ auf Seite 764
- „CREATE SERVICE-Anweisung [SOAP-Webdienst]“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]

Lektion 2: Java-Anwendung für die Kommunikation mit dem Webserver erstellen

In dieser Lektion verarbeiten Sie das aus dem DISH-Dienst generierte WSDL-Dokument und erstellen eine Java-Anwendung für den Zugriff auf Tabellendaten, basierend auf dem im WSDL-Dokument definierten Schema.

Voraussetzungen

Diese Lektion hängt von den in Lektion 1 ausgeführten Schritten ab. Siehe „[Lektion 1: Einrichten eines Webservers für den Empfang von SOAP-Anforderungen und das Senden von SOAP-Antworten](#)“ auf Seite 857.

In dieser Lektion wird davon ausgegangen, dass Sie die Rollen und Privilegien haben, die im Abschnitt "Privilegien" am Anfang dieser praktischen Einführung aufgeführt sind: „[Praktische Einführung: Verwenden von JAX-WS für den Zugriff auf einen SOAP/DISH-Webdienst](#)“.

Kontext und Bemerkungen

Zum Zeitpunkt des Erstellens dieser Dokumentation war JAX-WS in JDK 1.7.0 enthalten und die neueste Version von JAX-WS war 2.2.7. Die nachstehenden Schritte basieren auf dieser Version. Um zu ermitteln, ob JAX-WS in Ihrem JDK vorhanden ist, suchen Sie nach der Anwendung **wsimport** im JDK-Verzeichnis *bin*. Wenn sie dort nicht vorhanden ist, gehen Sie zu <http://jax-ws.java.net/>, um die neueste Version von JAX-WS herunterzuladen und zu installieren.

Diese Lektion enthält mehrere Verweise auf **localhost**. Verwenden Sie statt **localhost** den Hostnamen oder die IP-Adresse des Webserver aus Lektion 1, wenn der Webclient nicht auf demselben Computer ausgeführt wird wie der Webserver.

Aufgabe

1. Erstellen Sie an einer Eingabeaufforderung ein neues Arbeitsverzeichnis für Ihren Java-Code und die zugehörigen generierten Dateien. Wechseln Sie zu diesem neuen Verzeichnis.
2. Erstellen Sie mit dem folgenden Befehl die Schnittstelle, die den DISH-Webdienst aufruft und das WSDL-Dokument erstellt:

```
wsimport -keep "http://localhost:8082/demo/WSDish"
```

Die Anwendung `wsimport` ruft das WSDL-Dokument von der angegebenen URL ab. Sie generiert `.java` Dateien, um dafür eine Schnittstelle zu erstellen, und kompiliert sie dann in `.class`-Dateien.

Die Option **keep** gibt an, dass die `.java`-Dateien nach dem Generieren der Klassendateien nicht gelöscht werden sollen. Der generierte Java-Quellcode ermöglicht es Ihnen, die generierten Klassendateien zu verstehen.

Die Anwendung `wsimport` erstellt in Ihrem aktuellen Arbeitsverzeichnis eine neue Unterverzeichnisstruktur namens `localhost_8082\demo\ws`. Im Folgenden finden Sie eine Liste der Inhalte im Verzeichnis `ws`:

- *EmployeeList.class*
- *EmployeeList.java*
- *EmployeeListDataset\$Rowset\$Row.class*
- *EmployeeListDataset\$Rowset.class*
- *EmployeeListDataset.class*
- *EmployeeListDataset.java*
- *EmployeeListResponse.class*
- *EmployeeListResponse.java*
- *FaultMessage.class*
- *FaultMessage.java*
- *ObjectFactory.class*
- *ObjectFactory.java*
- *package-info.class*
- *package-info.java*
- *WSDish.class*
- *WSDish.java*
- *WSDishSoapPort.class*
- *WSDishSoapPort.java*

3. Schreiben Sie eine Java-Anwendung, die anhand des im generierten Quellcode definierten DataSet-Objektschemas auf Tabellendaten vom Datenbankserver zugreift.

Im Folgenden finden Sie ein Beispiel einer Java-Anwendung, die dies ausführt. Speichern Sie den Quellcode als Datei *SASoapDemo.java* im aktuellen Arbeitsverzeichnis. Ihr aktuelles Arbeitsverzeichnis muss das Verzeichnis sein, das das Unterverzeichnis *localhost* enthält.

```
// SASoapDemo.java illustrates a web service client that
// calls the WSDish service and prints out the data.

import java.util.*;
import javax.xml.ws.*;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import javax.xml.datatype.*;
import localhost._8082.demo.ws.*;

public class SASoapDemo
{
    public static void main( String[] args )
    {
        try {
            WSDish service = new WSDish();

            Holder<EmployeeListDataset> response =
                new Holder<EmployeeListDataset>();
            Holder<Integer> sqlcode = new Holder<Integer>();

            WSDishSoapPort port = service.getWSDishSoap();

            // This is the SOAP service call to EmployeeList
            port.employeeList( response, sqlcode );

            EmployeeListDataset result = response.value;
            EmployeeListDataset.Rowset rowset = result.getRowset();

            List<EmployeeListDataset.Rowset.Row> rows = rowset.getRow();

            String fieldType;
            String fieldName;
            String fieldValue;
            Integer fieldInt;
            XMLGregorianCalendar fieldDate;

            for ( int i = 0; i < rows.size(); i++ ) {
                EmployeeListDataset.Rowset.Row row = rows.get( i );

                fieldType = row.getEmployeeID().getDeclaredType().getSimpleName();
                fieldName = row.getEmployeeID().getName().getLocalPart();
                fieldInt = row.getEmployeeID().getValue();
                System.out.println( "(" + fieldType + ")" + fieldName +
                                    "=" + fieldInt );

                fieldType = row.getSurname().getDeclaredType().getSimpleName();
                fieldName = row.getSurname().getName().getLocalPart();
                fieldValue = row.getSurname().getValue();
                System.out.println( "(" + fieldType + ")" + fieldName +
                                    "=" + fieldValue );

                fieldType = row.getGivenName().getDeclaredType().getSimpleName();
                fieldName = row.getGivenName().getName().getLocalPart();
                fieldValue = row.getGivenName().getValue();
                System.out.println( "(" + fieldType + ")" + fieldName +
                                    "=" + fieldValue );

                fieldType = row.getStartDate().getDeclaredType().getSimpleName();
```

```

        fieldName = row.getStartDate().getName().getLocalPart();
        fieldDate = row.getStartDate().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                           "=" + fieldDate );

        if ( row.getTerminationDate() == null ) {
            fieldType = "unknown";
            fieldName = "TerminationDate";
            fieldDate = null;
        } else {
            fieldType =
                row.getTerminationDate().getDeclaredType().getSimpleName();
            fieldName = row.getTerminationDate().getName().getLocalPart();
            fieldDate = row.getTerminationDate().getValue();
        }
        System.out.println( "(" + fieldType + ")" + fieldName +
                           "=" + fieldDate );
        System.out.println();
    }
}
catch (Exception x) {
    x.printStackTrace();
}
}
}

```

Diese Anwendung gibt alle vom Server gelieferten Spaltendaten in der Standardsystemausgabe aus.

Hinweis

Diese Anwendung geht davon aus, dass Ihr SQL Anywhere-Webserver entsprechend der Anweisungen in der ersten Lektion Port 8082 überwacht. Ersetzen Sie den Teil **8082** der Codezeile **import localhost_8082.demo.ws.*** durch die von Ihnen beim Start des SQL Anywhere-Webservers angegebene Portnummer.

Weitere Hinweise zu den in dieser Anwendung verwendeten Java-Methoden finden Sie in der API-Dokumentation der javax.xml.bind.JAXBElement-Klasse unter <http://docs.oracle.com/javase/>.

4. Kompilieren Sie Ihre Java-Anwendung mit dem folgenden Befehl:

```
javac SASoapDemo.java
```

5. Führen Sie die Anwendung mit dem folgenden Befehl aus:

```
java SASoapDemo
```

6. Die Anwendung sendet ihre Anforderung an den Webserver. Sie empfängt eine Antwort mit einer XML-Ergebnismenge, die aus einem **EmployeeListResult** mit einer Zeilengruppe mit mehreren Zeileneinträgen besteht.

Ergebnisse

Im Folgenden finden Sie ein Beispiel für die Ausgabe aus SASoapDemo:

```

(Integer)EmployeeID=102
(String)Surname=Whitney
(String)GivenName=Fran
(XMLGregorianCalendar)StartDate=1984-08-28

```

```
(unknown)TerminationDate=null

(Integer)EmployeeID=105
(String)Surname=Cobb
(String)GivenName=Matthew
(XMLGregorianCalendar)StartDate=1985-01-01
(unknown)TerminationDate=null
.
.
(Integer)EmployeeID=1740
(String)Surname=Nielsen
(String)GivenName=Robert
(XMLGregorianCalendar)StartDate=1994-06-24
(unknown)TerminationDate=null

(Integer)EmployeeID=1751
(String)Surname=Ahmed
(String)GivenName=Alex
(XMLGregorianCalendar)StartDate=1994-07-12
(XMLGregorianCalendar)TerminationDate=2008-04-18
```

Die Spalte **TerminationDate** wird nur gesendet, wenn ihr Wert nicht NULL ist. Die Java-Anwendung ist dafür konzipiert, zu erkennen, wenn die Spalte **TerminationDate** nicht vorhanden ist. Bei diesem Beispiel wurde die letzte Zeile in der Employees-Tabelle geändert, um ein Nicht-NULL-Beendigungsdatum festzulegen.

Im Folgenden finden Sie ein Beispiel für eine SOAP-Antwort vom Webserver. Das SQLCODE-Ergebnis der ausgeführten Abfrage ist in der Antwort enthalten.

```
<tns:EmployeeListResponse>
  <tns:EmployeeListResult xsi:type='tns:EmployeeListDataset'>
    <tns:rowset>
      <tns:row> ... </tns:row>
      .
      .
      .
    <tns:row>
      <tns:EmployeeID xsi:type="xsd:int">1751</tns:EmployeeID>
      <tns:Surname xsi:type="xsd:string">Ahmed</tns:Surname>
      <tns:GivenName xsi:type="xsd:string">Alex</tns:GivenName>
      <tns:StartDate xsi:type="xsd:dateTime">1994-07-12</tns:StartDate>
      <tns:TerminationDate xsi:type="xsd:dateTime">2010-03-22</
tns:TerminationDate>
    </tns:row>
  </tns:rowset>
</tns:EmployeeListResult>
<tns:sqlcode>0</tns:sqlcode>
</tns:EmployeeListResponse>
```

Spaltennamen und Datentypen sind in jeder Zeilengruppe enthalten.

Dreischichtige Datenverarbeitung und verteilte Transaktionen

Sie können SQL Anywhere als Datenbankserver oder als **Ressourcen-Manager** einsetzen, der an verteilten Transaktionen teilnimmt, die von einem Transaktionsserver koordiniert werden.

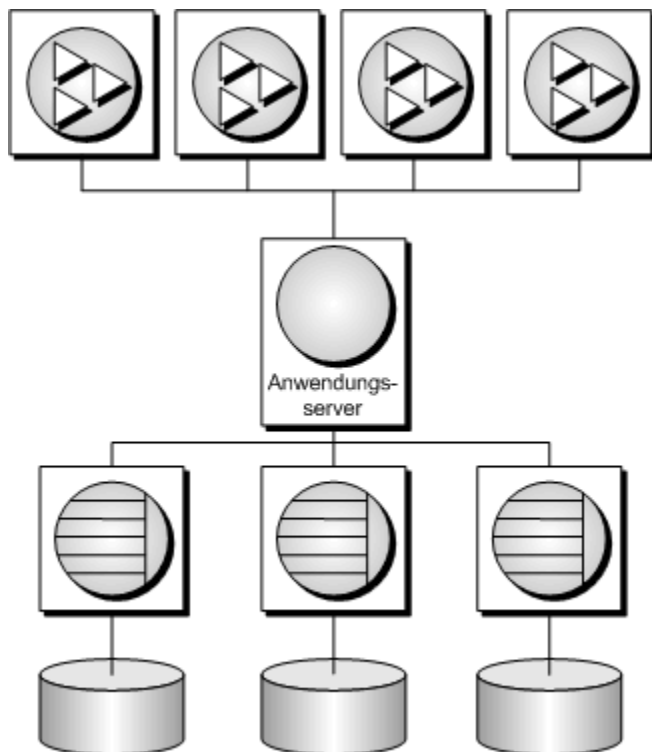
Eine dreischichtige Umgebung, in der ein Anwendungsserver zwischen Clientanwendung und einer Reihe von Ressourcen-Managern sitzt, ist eine übliche Umgebung für verteilte Transaktionen. Sybase EAServer und einige andere Anwendungsserver sind ebenfalls Transaktionsserver.

Sybase EAServer und Microsoft Transaction Server verwenden beide den Microsoft Distributed Transaktionskoordinator (DTC) zum Koordinieren der Transaktionen. SQL Anywhere bietet Unterstützung für verteilte Transaktionen, die vom DTC-Service kontrolliert werden, sodass Sie SQL Anywhere mit einem dieser Anwendungsserver oder einem anderen DTC-basierten Produkt verwenden können.

Wenn Sie SQL Anywhere in eine dreischichtige Umgebung integrieren, muss der Großteil der Arbeit vom Anwendungsserver erledigt werden. Dieser Abschnitt enthält eine Einführung in die Konzepte und die Architektur der dreischichtigen Datenverarbeitung sowie ein Überblick über die relevanten SQL Anywhere-Funktionen. Es wird nicht beschrieben, wie Ihr Anwendungsserver für die Arbeit mit SQL Anywhere konfiguriert werden muss. Weitere Hinweise finden Sie in der Dokumentation Ihres Anwendungsservers.

Dreischichtige Datenverarbeitungsarchitektur

Bei der dreischichtigen Datenverarbeitung wird die Anwendungslogik auf einem Anwendungsserver wie einem Sybase EAServer gespeichert, der sich zwischen dem Ressourcen-Manager und der Clientanwendung befindet. In vielen Situationen kann ein einziger Anwendungsserver auf mehrere Ressourcen-Manager zugreifen. Bei Internetanwendungen ist die Clientseite browserbasiert, und der Anwendungsserver ist im Allgemeinen eine Webserver-Erweiterung.



Sybase EAServer speichert die Anwendungslogik in Form von Komponenten und stellt diese den Clientanwendungen zur Verfügung. Bei den Komponenten kann es sich um PowerBuilder-Komponenten, JavaBeans oder COM-Komponenten handeln.

Weitere Hinweise finden Sie in der Dokumentation zu Sybase EAServer.

Verteilte Transaktionen in dreischichtiger Datenverarbeitung

Wenn Clientanwendungen oder Anwendungsserver mit einer einfachen Transaktionsverarbeitungs-Datenbank arbeiten, wie etwa SQL Anywhere, wird außerhalb der Datenbank selbst keine Transaktionslogik benötigt. Wenn jedoch mit mehreren Ressourcen-Managern gearbeitet wird, muss die Transaktionssteuerung die in die Transaktion einbezogenen Ressourcen abdecken. Anwendungsserver bieten ihren Clientanwendungen Transaktionslogik, sodass Gruppen von Vorgängen in kleinsten Einheiten ausgeführt werden.

Viele Transaktionsserver, einschließlich Sybase EAServer, verwenden den Microsoft Distributed Transaktionskoordinator (DTC), um den Clientanwendungen Transaktionsdienste anzubieten. DTC verwendet **OLE-Transaktionen**, die ihrerseits das Protokoll **Zwei-Phasen-Commit** für die Koordinierung von Transaktionen mit mehreren Ressourcen-Managern verwenden. Um die in diesem Abschnitt beschriebenen Funktionen verwenden zu können, müssen Sie DTC installiert haben.

SQL Anywhere in verteilten Transaktionen

SQL Anywhere kann an von DTC koordinierten Transaktionen teilnehmen. Das bedeutet, dass Sie SQL Anywhere-Datenbanken in verteilten Transaktionen mit einem Transaktionsserver wie etwa Sybase EAServer oder Microsoft Transaction Server verwenden können. Außerdem können Sie DTC direkt in Ihren Anwendungen zur Koordinierung von Transaktionen über mehrere Ressourcen-Manager einsetzen.

Begriffe im Zusammenhang mit verteilten Transaktionen

In diesem Abschnitt werden gewisse Kenntnisse über verteilte Transaktionen vorausgesetzt. Hinweise finden Sie in der Dokumentation zum Transaktionsserver. In diesem Abschnitt werden allgemein übliche Begriffe beschrieben.

- **Ressourcen-Manager** sind Dienste, die in eine Transaktion einbezogene Daten verwalten.

Der SQL Anywhere-Datenbankserver kann in einer verteilten Transaktion als Ressourcen-Manager agieren, wenn über ADO.NET, OLE DB oder ODBC darauf zugegriffen wird. Der .NET-Datenprovider, der OLE DB-Provider und der ODBC-Treiber von SQL Anywhere agieren auf dem Clientcomputer als Ressourcen-Manager-Proxys. Der SQL Anywhere-.NET-Datenprovider unterstützt verteilte Transaktionen mit DbProviderFactory und TransactionScope.

- Anstatt direkt mit dem Ressourcen-Manager können Anwendungskomponenten mit **Ressourcen-Verteilern** kommunizieren, die ihrerseits Verbindungen oder Verbindungs-Pools zu den Ressourcen-Managern verwalten.

SQL Anywhere unterstützt zwei Ressourcen-Verteiler: den ODBC-Treibermanager und OLE DB.

- Wenn eine Transaktionskomponente eine Datenbankverbindung anfordert (über einen Ressourcen-Manager), **bezieht** der Anwendungsserver alle Datenbankverbindungen ein, die an der Transaktion teilnehmen. DTC und der Ressourcen-Verteiler führen den Einbeziehungsvorgang aus.

Zwei-Phasen-Commit

Verteilte Transaktionen werden mit Zwei-Phasen-Commit verwaltet. Wenn die Arbeit der Transaktion abgeschlossen ist, fragt der Transaktions-Manager (DTC) alle in die Transaktion einbezogenen Ressourcen-Manager, ob sie bereit sind, die Transaktion festzuschreiben. Diese Phase wird **Vorbereiten** zum Festschreiben genannt.

Wenn alle Ressourcen-Manager antworten, dass sie zum Festschreiben bereit sind, sendet DTC eine Anforderung zum Festschreiben an jeden einzelnen Ressourcen-Manager und antwortet seinem Client, dass die Transaktion abgeschlossen ist. Wenn einer oder mehrere Ressourcen-Manager nicht antworten oder antworten, dass sie die Transaktion nicht festschreiben können, wird die gesamte Arbeit der Transaktion über alle Ressourcen-Manager zurückgesetzt.

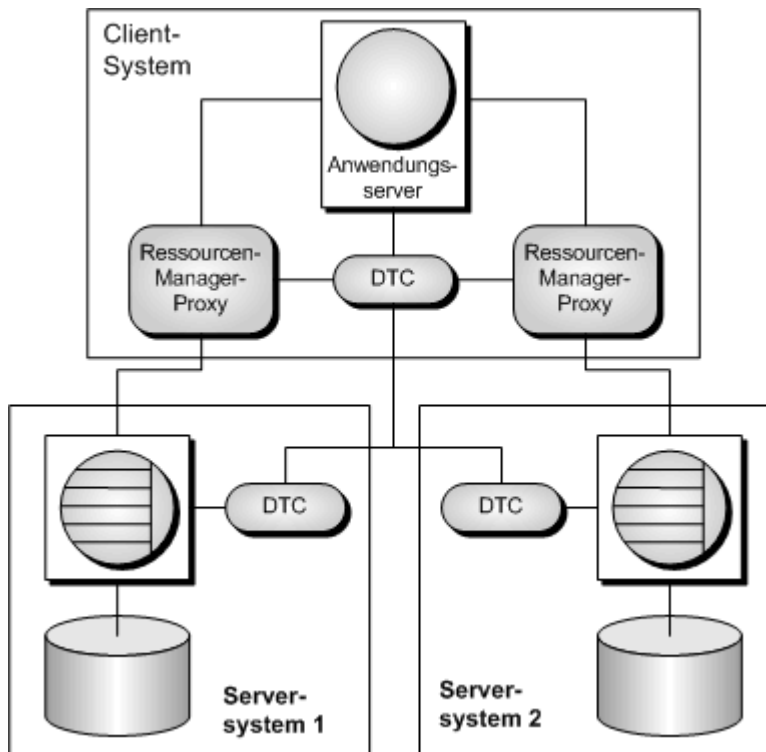
So verwenden Anwendungsserver DTC

Sybase EAServer und Microsoft Transaction Server sind beides Komponentenserver. Die Anwendungslogik wird in Form von Komponenten gespeichert und den Clientanwendungen zur Verfügung gestellt.

Jede Komponente hat ein Transaktionsattribut, das darauf hinweist, wie die Komponente an Transaktionen teilnimmt. Beim Erstellen der Komponente muss die Arbeit der Transaktion in die Komponente einprogrammiert werden: Die Ressourcen-Manager-Verbindungen, die Vorgänge mit den Daten, für die jeder einzelne Ressourcen-Manager verantwortlich ist. Sie müssen jedoch nicht die Transaktionsverwaltungslogik in die Komponente einbauen. Wenn das Transaktionsattribut so gesetzt ist, dass darauf hingewiesen wird, dass die Komponente eine Transaktionsverwaltung benötigt, verwendet EA Server DTC, um die Transaktion einzubeziehen und den Zwei-Phasen-Commit-Vorgang zu verwalten.

Verteilte Transaktionsarchitektur

Das folgende Diagramm veranschaulicht die Architektur von verteilten Transaktionen. In diesem Fall ist der Ressourcen-Manager-Proxy entweder ADO.NET, ODBC oder OLE DB.



In diesem Fall wird ein einzelner Ressourcen-Verteiler verwendet. Der Anwendungsserver fordert DTC auf, eine Transaktion vorzubereiten. DTC und der Ressourcen-Verteiler beziehen jede einzelne Verbindung in die Transaktion ein. Jeder Ressourcen-Manager muss in Kontakt mit dem DTC und der Datenbank sein, damit die Arbeit durchgeführt und der DTC gegebenenfalls über seinen Transaktionsstatus benachrichtigt werden kann.

Auf jedem Computer muss ein DTC-Dienst (Distributed Transaction Coordinator) laufen, damit die verteilten Transaktionen ausgeführt werden können. Sie können DTC aus dem Microsoft Windows-Fenster **Dienste** starten oder stoppen. Der DTC-Dienst-Task heißt **MSDTC**.

Weitere Hinweise finden Sie in der Dokumentation zu DTC bzw. EAServer.

Verteilte Transaktionen

Wenn SQL Anywhere in eine verteilte Transaktion einbezogen ist, gibt das Programm die Kontrolle an den Transaktionsserver weiter und gewährleistet so, dass keine implizite Transaktionsverwaltung ausgeführt wird. Die folgenden Bedingungen werden automatisch von SQL Anywhere auferlegt, wenn er an verteilten Transaktionen teilnimmt:

- Autocommit wird automatisch deaktiviert, wenn es verwendet wurde.
- Datendefinitions-Anweisungen (als Nebenwirkung festgeschrieben) werden während der verteilten Transaktionen deaktiviert.
- Ein explizites COMMIT oder ROLLBACK von der Anwendung direkt an SQL Anywhere anstatt über den Transaktionskoordinator führt zu einer Fehlermeldung. Die Transaktion wird jedoch nicht abgebrochen.
- Eine Verbindung kann jeweils nur an einer verteilten Transaktion teilnehmen.
- Zu dem Zeitpunkt, wenn die Verbindung in eine verteilte Transaktion einbezogen wird, darf es keine nicht festgeschriebenen Vorgänge geben.

DTC-Isolationsstufen

DTC weist eine Reihe von Isolationsstufen auf, die der Anwendungsserver angibt. Diese Isolationsstufen entsprechen folgendermaßen den Isolationsstufen von SQL Anywhere:

DTC-Isolationsstufe	SQL Anywhere-Isolationsstufe
ISOLATIONLEVEL_UNSPECIFIED	0
ISOLATIONLEVEL_CHAOS	0
ISOLATIONLEVEL_READUNCOMMITTED	0
ISOLATIONLEVEL_BROWSE	0
ISOLATIONLEVEL_CURSORSTABILITY	1
ISOLATIONLEVEL_READCOMMITTED	1
ISOLATIONLEVEL_REPEATABLE_READ	2
ISOLATIONLEVEL_SERIALIZABLE	3
ISOLATIONLEVEL_ISOLATED	3

Wiederherstellung nach verteilten Transaktionen

Wenn der Datenbankserver einen Fehler aufweist, während nicht festgeschriebene Vorgänge auf Ausführung warten, müssen diese Vorgänge beim Neustart entweder zurückgesetzt oder festgeschrieben werden, damit die atomare Natur der Transaktion geschützt wird.

Wenn während der Wiederherstellung nicht festgeschriebene Vorgänge einer verteilten Transaktion gefunden werden, versucht der Datenbankserver eine Verbindung mit DTC herzustellen und fordert, wieder in die auf Ausführung wartenden bzw. in die zweifelhaften Transaktionen einbezogen zu werden. Wenn die Wiedereinbeziehung abgeschlossen ist, weist DTC den Datenbankserver an, die ausstehenden Vorgänge zurückzusetzen oder festzuschreiben.

Wenn der Wiedereinbeziehungsvorgang fehlschlägt, kann SQL Anywhere nicht wissen, ob die zweifelhaften Vorgänge festgeschrieben oder zurückgesetzt werden sollten, und die Wiederherstellung schlägt fehl. Wenn die Datenbank, unabhängig vom unsicheren Status der Daten, wieder in einen solchen Status hergestellt werden soll, können Sie die Wiederherstellung mit den folgenden Datenbankserveroptionen erzwingen:

- **-tmf** Wenn DTC nicht geladen werden kann, werden die ausstehenden Vorgänge zurückgesetzt und die Wiederherstellung wird fortgesetzt.
- **-tmt** Wenn die Wiedereinbeziehung vor der angegebenen Zeit nicht gelingt, werden die ausstehenden Vorgänge zurückgesetzt und die Wiederherstellung wird fortgesetzt.

Siehe auch

- „Datenbankserveroption -tmf“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Datenbankserveroption -tmt“ [[SQL Anywhere Server - Datenbankadministration](#)]

Datenbanktools-Schnittstelle (DBTools)

SQL Anywhere enthält Sybase Central und eine Gruppe von Dienstprogrammen für die Datenbankverwaltung. Diese Dienstprogramme für das Datenbankmanagement übernehmen bestimmte Aufgaben, z.B. das Sichern und Erstellen von Datenbanken, die Konvertieren von Transaktionslogs in SQL usw.

Unterstützte Plattformen

Alle Dienstprogramme für das Datenbankmanagement verwenden als Shared Library die **DBTools-Bibliothek**. Sie wird für Windows-Betriebssysteme und für Linux, Unix und Mac OS X bereitgestellt. Bei Windows lautet der Name dieser Bibliothek *dbtool16.dll*. Bei Linux und Unix lautet der Name dieser Bibliothek *libdbtool16_r.so*. Bei Mac OS X lautet der Name dieser Bibliothek *libdbtool16_r.dylib*.

Sie können Ihre eigenen Dienstprogramme für das Datenbankmanagement entwickeln oder von Datenbankmanagementfunktionen in Ihre Anwendungen integrieren, indem Sie die DBTools-Bibliothek aufrufen. In diesem Abschnitt wird die Schnittstelle zur DBTools-Bibliothek beschrieben. Vorausgesetzt wird, dass Sie wissen, wie Sie Bibliotheksroutinen aus der von Ihnen verwendeten Entwicklungsumgebung aufrufen.

Die DBTools-Bibliothek hat Funktionen oder Eintrittspunkte für jedes einzelne Dienstprogramm für das Datenbankmanagement. Außerdem müssen Funktionen vor dem Aufruf anderer DBTools-Funktionen sowie im Anschluss an die Verwendung anderer DBTools-Funktionen aufgerufen werden.

Windows Mobile

Die Bibliothek *dbtool16.dll* wird für Windows Mobile geliefert, enthält aber nur Programmeintrittspunkte für DBToolsInit, DBToolsFin, DBRemoteSQL und DBSynchronizeLog. Es werden keine anderen Programmeintrittspunkte für Windows Mobile bereitgestellt.

Die *dbtools.h*-Header-Datei

Die DBTools-Header-Datei enthält eine Liste der Eintrittspunkte zur DBTools-Bibliothek und liefert die Strukturen, mit denen Daten an die Bibliothek übergeben oder von ihr bezogen werden. Die Datei *dbtools.h* wird im Unterverzeichnis *SDK\Include* des SQL Anywhere-Installationsverzeichnisses gespeichert. In der Datei *dbtools.h* finden Sie die neuesten Informationen über die Eintrittspunkte und die Strukturbestandteile.

Die Header-Datei *dbtools.h* bezieht u.a. folgende andere Dateien ein:

- ***sqlca.h*** Diese Datei ist für die Auflösung diverser Makros, nicht für SQLCA selbst, vorgesehen.
- ***dllapi.h*** Diese Datei definiert Präprozessor-Makros für betriebssystem- und sprachenabhängige Makros.
- ***dbtlvers.h*** Diese Datei definiert den Präprozessor-Makro `DB_TOOLS_VERSION_NUMBER` und andere versionsspezifische Makros.

Die *sqldef.h*-Header-Datei

Die Header-Datei *sqldef.h* enthält Fehlerrückgabewerte.

Die *dbrmt.h*-Header-Datei

Die in SQL Anywhere enthaltene Header-Datei *dbrmt.h* beschreibt den DBRemoteSQL-Eintrittspunkt in der DBTools-Bibliothek und außerdem die Struktur, die für die Übergabe der Informationen vom und an den DBRemoteSQL-Eintrittspunkt verwendet wird. Die Datei *dbrmt.h* wird im Unterverzeichnis *SDK\Include* des SQL Anywhere-Installationsverzeichnisses gespeichert. In der Datei *dbrmt.h* finden Sie die neuesten Informationen über den DBRemoteSQL-Eintrittspunkt und die Strukturelemente.

DBTools-Importbibliotheken

Damit die DBTools-Funktionen verwendet werden können, müssen Sie Ihre Anwendung mit einer DBTools-**Importbibliothek** linken, die die erforderlichen Funktionsdefinitionen enthält.

Für Unix-Systeme ist keine Importbibliothek erforderlich. Verbinden Sie sich direkt mit *libdbtool16.so* (ohne Threading) bzw. *libdbtool16_r.so* (mit Threading).

Importbibliotheken

Mit SQL Anywhere für Windows und Windows Mobile werden Importbibliotheken für die DBTools-Schnittstelle bereitgestellt. Für Windows befinden sich diese in den Unterverzeichnissen *SDK\Lib\x86* und *SDK\Lib\x64* des SQL Anywhere-Installationsverzeichnisses. Die Importbibliothek für Windows Mobile befindet sich im Unterverzeichnis *SDK\Lib\CE\Arm.50* des SQL Anywhere-Installationsverzeichnisses. Folgende DBTools-Importbibliotheken werden geliefert:

Compiler	Bibliothek
Microsoft Windows	<i>dbtlstm.lib</i>
Microsoft Windows Mobile	<i>dbtool16.lib</i>

Initialisierung und Finalisierung der DBTools-Bibliothek

Bevor Sie andere DBTools-Funktionen verwenden, müssen Sie DBToolsInit aufrufen. Wenn Sie die DBTools-Bibliothek nicht mehr verwenden, müssen Sie DBToolsFini aufrufen.

Die Funktionen DBToolsInit und DBToolsFini dienen hauptsächlich dazu, der DBTools-Bibliothek das Laden und Entladen der SQL Anywhere-Meldungsbibliothek zu ermöglichen. Die Meldungsbibliothek enthält lokalisierte Versionen aller Fehlermeldungen und Eingabeaufforderungen, die DBTools intern verwendet. Wenn DBToolsFini nicht aufgerufen wird, kann die Referenznummer der Meldungsbibliothek nicht heruntergezählt werden, sodass diese Meldungsbibliothek nicht entladen wird. Achten Sie daher immer darauf, dass zu jedem Aufruf von DBToolsInit als Gegenstück DBToolsFini aufgerufen wird.

Das folgenden Programmcodebeispiel zeigt, wie DBTools initialisiert und wieder beendet wird:

```
// Declarations
a_dbtools_info  info;
short          ret;
```

```

//Initialize the a_dbtools_info structure
memset( &info, 0, sizeof( a_dbtools_info ) );
info.errorrtn = (MSG_CALLBACK)MyErrorRtn;

// initialize the DBTools library
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
    // library initialization failed
    ...
}
// call some DBTools routines ...
...
// finalize the DBTools library
DBToolsFini( &info );

```

DBTools-Funktionsaufrufe

Alle Tools werden aufgerufen, indem erst eine Struktur ausgefüllt und dann eine Funktion (oder ein **Eintrittspunkt**) in der DBTools-Bibliothek aufgerufen wird. Jeder Eintrittspunkt nimmt einen Zeiger zu einer bestimmten Struktur als Argument.

Im folgenden Beispiel wird gezeigt, wie die DBBackup-Funktion in einem Windows-Betriebssystem verwendet wird.

```

// Initialize the structure
a_backup_db backup_info;
memset( &backup_info, 0, sizeof( backup_info ) );

// Fill out the structure
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.output_dir = "c:\\\\backup";
backup_info.connectparms = "UID=DBA;PWD=sql;DBF=demo.db";

backup_info.confirmrtn = (MSG_CALLBACK) ConfirmRtn ;
backup_info.errorrtn = (MSG_CALLBACK) ErrorRtn ;
backup_info.msgrtn = (MSG_CALLBACK) MessageRtn ;
backup_info.statusrtn = (MSG_CALLBACK) StatusRtn ;
backup_info.backup_database = TRUE;

// start the backup
DBBackup( &backup_info );

```

Siehe auch

- „DBTools-Strukturen“ auf Seite 878

Callback-Funktionen

Einige Elemente in der DBTools-Struktur sind vom Typ MSG_CALLBACK. Es handelt sich dabei um Zeiger auf Callback-Funktionen.

Nutzung der Callback-Funktionen

Mit Callback-Funktionen können DBTools-Funktionen die Kontrolle der Verarbeitung an die aufrufende Anwendung des Benutzers zurückgeben. Die DBTools-Bibliothek verwendet Callback-Funktionen zur Verarbeitung von Nachrichten, die von den DBTools-Funktionen an den Benutzer gesandt werden, für vier Zwecke:

- **Bestätigung** Wird aufgerufen, wenn eine Aktion vom Benutzer bestätigt werden muss. Falls zum Beispiel das Sicherungsverzeichnis nicht vorhanden ist, fragt die Tools-Bibliothek, ob es erzeugt werden soll.
- **Fehlermeldung** Wird aufgerufen, um eine Nachricht zu bearbeiten, wenn ein Fehler auftritt, zum Beispiel, wenn ein Vorgang nicht genügend Speicherplatz hat
- **Informationsnachricht** Wird aufgerufen, um den Benutzer zu informieren, wie zum Beispiel eine Nachricht mit den Namen der Tabelle, die gerade entladen wird
- **Statusinformationen** Wird aufgerufen, um den Status eines Vorgangs anzuzeigen, wie zum Beispiel, wieviel Prozent einer Tabelle entladen sind

Callback-Funktion einer Struktur zuordnen

Sie können der Struktur direkt eine Callback-Routine zuordnen. Die folgende Anweisung ist ein Beispiel für die Verwendung einer Sicherungsstruktur:

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

MSG_CALLBACK ist in der Header-Datei *dllapi.h* definiert, die in SQL Anywhere enthalten ist. Tools-Routinen können Nachrichten an die aufrufende Anwendung übergeben, die dann in der betreffenden Benutzeroberfläche dargestellt werden. Dabei kann es sich um eine Fensterumgebung, die Standardausgabe eines zeichenbasierten Systems oder andere Benutzeroberflächen handeln.

Beispiel für eine Callback-Funktion mit Bestätigung

Im folgenden Beispiel fragt eine Routine den Benutzer nach einer Bestätigung mit YES oder NO und gibt die Antwort des Benutzers zurück:

```
extern short _callback ConfirmRtn(
    char * question )
{
    int ret = IDNO;
    if( question != NULL ) {
        ret = MessageBox( HwndParent, question,
            "Confirm", MB_ICONEXCLAMATION|MB_YESNO );
    }
    return( ret == IDYES );
}
```

Beispiel für eine Callback-Funktion mit Fehlermeldung

Im folgenden Beispiel zeigt eine Routine zur Behandlung einer Fehlermeldung die Fehlermeldung in einem Fenster an.

```
extern short _callback ErrorRtn(
    char * errorstr )
{

```

```

        if( errorstr != NULL ) {
            MessageBox( HwndParent, errorstr, "Backup Error", MB_ICONSTOP|
MB_OK );
        }
        return( 0 );
    }

```

Beispiel für eine Callback-Funktion mit Nachricht

Eine typische Implementierung einer Callback-Funktion, die eine Nachricht am Bildschirm ausgibt:

```

extern short _callback MessageRtn(
    char * messagestr )
{
    if( messagestr != NULL ) {
        OutputMessageToWindow( messagestr );
    }
    return( 0 );
}

```

Beispiel für eine Callback-Funktion mit Statusmeldung

Eine Status-Callback-Routine wird aufgerufen, wenn ein Tool den Status eines Vorgangs anzeigen muss (zum Beispiel wie viel Prozent einer Tabelle entladen sind). Im folgenden Beispiel sehen Sie eine typische Implementierung, die die Nachricht lediglich am Bildschirm ausgibt:

```

extern short _callback StatusRtn(
    char * statusstr )
{
    if( statusstr != NULL ) {
        OutputMessageToWindow( statusstr );
    }
    return( 0 );
}

```

Versionsnummern und Kompatibilität

Jede Struktur enthält ein Element, das die Versionsnummer angibt. Im Versionsfeld sollten Sie die Versionsnummer der DBTools-Bibliothek angeben, für die Ihre Anwendung entwickelt wurde, bevor Sie eine DBTools-Funktion aufrufen. Die aktuelle Version der DBTools-Bibliothek wird definiert, wenn Sie die Header-Datei *dbtools.h* einbeziehen.

Das folgende Beispiel weist die aktuelle Version einer Instanz der *a_backup_db*-Struktur zu:

```

backup_info.version = DB_TOOLS_VERSION_NUMBER;

```

Die Versionsnummer ermöglicht es der Anwendung, auch in Zukunft mit neueren Versionen der DBTools-Bibliothek weiterzuarbeiten. Die DBTools-Funktionen sorgen mithilfe der Versionsnummern, die von Ihrer Anwendung zur Verfügung gestellt werden, dafür, dass Ihre Anwendung auch dann weiterarbeiten kann, wenn neue Elemente zur DBTools-Struktur hinzugefügt wurden.

Wenn eine der DBTools-Strukturen aktualisiert oder eine neuere Version der Software freigegeben wird, wird die Versionsnummer erhöht. Wenn Sie *DB_TOOLS_VERSION_NUMBER* verwenden und die Anwendung mit einer neuen Version der DBTools-Header-Datei neu erstellen, müssen Sie eine neue Version der DBTools-Bibliothek bereitstellen.

Bitfelder

Viele DBTools-Strukturen verwenden Bitfelder, um Boolesche Informationen kompakt zu speichern. Die Sicherungsstruktur hat zum Beispiel folgende Bitfelder:

```
a_bit_field    backup_database : 1;
a_bit_field    backup_logfile  : 1;
a_bit_field    no_confirm      : 1;
a_bit_field    quiet           : 1;
a_bit_field    rename_log      : 1;
a_bit_field    truncate_log     : 1;
a_bit_field    rename_local_log: 1;
a_bit_field    server_backup    : 1;
```

Jedes Bitfeld ist ein Bit lang, angezeigt durch eine 1 rechts neben dem Doppelpunkt in der Strukturdeklaration. Der hier verwendete Datentyp hängt vom Wert ab, der am Anfang von *dbtools.h* der Spalte *a_bit_field* zugeordnet wurde, sowie vom jeweiligen Betriebssystem.

Sie ordnen dem Bitfeld einen Wert von 0 oder 1 zu, um einen Booleschen Wert an die Struktur zu übergeben.

Ein Beispiel für DBTools

Dieses Beispiel sowie Kompilierungsanweisungen finden Sie im Verzeichnis *%SQLANYSAMPI6%\SQLAnywhere\DBTools*. Das Beispielprogramm selbst befindet sich in *main.cpp*. Das Beispiel veranschaulicht, wie die DBTools-Bibliothek zum Sichern der Datenbank eingesetzt werden kann.

```
#define WIN32

#include <stdio.h>
#include <string.h>
#include "windows.h"
#include "sqldef.h"
#include "dbtools.h"
extern short _callback ConfirmCallback( char * str )
{
    if( MessageBox( NULL, str, "Backup",
        MB_YESNO|MB_ICONQUESTION ) == IDYES )
    {
        return 1;
    }
    return 0;
}
extern short _callback MessageCallback( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback StatusCallback( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
}
```

```

    return 0;
}
extern short _callback ErrorCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
typedef void (CALLBACK *DBTOOLSPROC)( void * );
typedef short (CALLBACK *DBTOOLSFUNC)( void * );

// Main entry point into the program.
int main( int argc, char * argv[] )
{
    a_dbtools_info  dbt_info;
    a_backup_db     backup_info;
    char            dir_name[ _MAX_PATH + 1];
    char            connect[ 256 ];
    HINSTANCE       hinst;
    DBTOOLSFUNC     dbbackup;
    DBTOOLSFUNC     dbtoolsinit;
    DBTOOLSPROC     dbtoolsfini;
    short           ret_code;

    // Always initialize to 0 so new versions
    // of the structure will be compatible.
    memset( &dbt_info, 0, sizeof( a_dbtools_info ) );
    dbt_info.errorrtn = (MSG_CALLBACK)MessageCallBack;;

    memset( &backup_info, 0, sizeof( a_backup_db ) );
    backup_info.version = DB_TOOLS_VERSION_NUMBER;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.confirmrtn = (MSG_CALLBACK)ConfirmCallBack;
    backup_info.errorrtn = (MSG_CALLBACK)ErrorCallBack;
    backup_info.msgrtn = (MSG_CALLBACK)MessageCallBack;
    backup_info.statusrtn = (MSG_CALLBACK)StatusCallBack;
    if( argc > 1 )
    {
        strncpy( dir_name, argv[1], _MAX_PATH );
    }
    else
    {
        // DBTools does not expect (or like) a trailing slash
        strcpy( dir_name, "c:\\temp" );
    }
    backup_info.output_dir = dir_name;
    if( argc > 2 )
    {
        strncpy( connect, argv[2], 255 );
    }
    else
    {
        strcpy( connect, "DSN=SQL Anywhere 16 Demo" );
    }
    backup_info.connectparms = connect;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.backup_database = 1;
    backup_info.backup_logfile = 1;
    backup_info.rename_log = 0;
    backup_info.truncate_log = 0;

```

```
hinst = LoadLibrary( "dbtool16.dll" );
if( hinst == NULL )
{
    // Failed
    return EXIT_FAIL;
}
dbbackup = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
    "_DBBackup@4" );
dbtoolsinit = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
    "_DBToolsInit@4" );
dbtoolsfini = (DBTOOLSPROC) GetProcAddress( (HMODULE)hinst,
    "_DBToolsFini@4" );
ret_code = (*dbtoolsinit)( &dbt_info );
if( ret_code != EXIT_OKAY ) {
    return ret_code;
}
ret_code = (*dbbackup)( &backup_info );
(*dbtoolsfini)( &dbt_info );
FreeLibrary( hinst );
return ret_code;
}
```

SQL Anywhere-Datenbanktools - C-API-Referenz

Headerdateien

- `dbtools.h`
- `dbrmt.h`

DBTools-Strukturen

In diesem Abschnitt sind die Strukturen aufgeführt, mit deren Hilfe Informationen mit der DBTools-Bibliothek ausgetauscht werden. Die Strukturen sind alphabetisch aufgeführt. Abgesehen von der Struktur `a_remote_sql` sind alle diese Strukturen in `dbtools.h` definiert. Die Struktur `a_remote_sql` ist in `dbrmt.h` definiert.

Viele der Strukturelemente entsprechen Befehlszeilenoptionen des entsprechenden Dienstprogramms. Einige Strukturen haben beispielsweise ein Element namens "quiet", das mit den Werten 0 oder 1 belegt sein kann. Dieses Element entspricht der Option des Vorgangs "quiet" (-q), der von vielen Dienstprogrammen verwendet wird.

Die Datenstrukturen, die in dieser Datei definiert sind, können mit der DBTools-API für die Hauptversion von SQL Anywhere verwendet werden, für die die Datei gilt. Anwendungen, die beispielsweise mit der `dbtools.h`-Datei der Version 10.0.0 erstellt werden, können auf `dbtool9.dll` oder `dbtool11.dll` NICHT zugreifen.

Innerhalb einer Hauptversion werden Änderungen an den Strukturen so vorgenommen, dass die mit früheren oder späteren Versionen von `dbtools.h` für dieselbe Hauptversion erstellten Anwendungen funktionieren. Anwendungen, die mit früheren Versionen erstellt wurden, haben keinen Zugriff auf neue Felder, daher bieten DBTools Standardwerte, die dasselbe Verhalten ermöglichen wie frühere Versionen. Im Allgemeinen bedeutet dies, dass neue Felder in Nebenversionen am Ende einer Struktur erscheinen.

Das Verhalten für Anwendungen, die mit späteren Versionen innerhalb derselben Hauptversion erstellt wurden, hängt vom Wert im Feld "Version" der Struktur ab. Wenn die angegebene Versionsnummer einer früheren Version entspricht, kann die Anwendung eine frühere Version der DBTools-DLL aufrufen, als wäre die Anwendung mit dieser Version von dbtools.h erstellt worden. Wenn die angegebene Versionsnummer die aktuelle Version ist, führt die Verwendung einer früheren Version der DBTools-DLL zu einem Fehler.

Unter *dbtlvers.h* finden Sie die Definitionen der Versionsnummern.

DBBackup-Methode

Sichert eine Datenbank.

Syntax

```
_crtn short _entry DBBackup(const a_backup_db * pdb)
```

Parameter

- **pdb** Zeiger auf eine korrekt initialisierte a_backup_db-Struktur.

Rückgabe

Rückgabecode, wie in „[Exit-Codes der Softwarekomponenten](#)“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbbackup verwendet.

Die Funktion DBBackup verwaltet alle clientseitigen Datenbanksicherungs-Aufgaben. Die Beschreibung dieser Aufgaben finden Sie unter „[Sicherungsdienstprogramm \(dbbackup\)](#)“ [*SQL Anywhere Server - Datenbankadministration*].

Zur Durchführung einer serverseitigen Sicherung verwenden Sie die Anweisung BACKUP DATABASE.

Siehe auch

- [a_backup_db-Struktur \[Datenbanktools\]](#) auf Seite 894
- „[BACKUP-Anweisung](#)“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

DBChangeLogName-Methode

Ändert den Namen der Transaktionslogdatei.

Syntax

```
_crtn short _entry DBChangeLogName(const a_change_log * pcl)
```

Parameter

- **pcl** Zeiger auf eine korrekt initialisierte a_change_log-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dblog verwendet.

Die Option -t des Transaktionslog-Dienstprogramms (dblog) ändert den Namen des Transaktionslogs. DBChangeLogName liefert eine Programmierschnittstelle zu dieser Funktion.

Siehe auch

- [a_change_log-Struktur \[Datenbanktools\] auf Seite 897](#)
- „Transaktionslog-Dienstprogramm (dblog)“ [*SQL Anywhere Server - Datenbankadministration*]
- „ALTER DATABASE-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

DBCCreate-Methode

Erstellt eine Datenbank.

Syntax

```
_crtn short _entry DBCreate(a_create_db * pcdb)
```

Parameter

- **pcdb** Zeiger auf eine korrekt initialisierte a_create_db-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbinit verwendet.

Siehe auch

- [a_create_db-Struktur \[Datenbanktools\] auf Seite 899](#)
- „Dienstprogramm Initialisierung (dbinit)“ [*SQL Anywhere Server - Datenbankadministration*]

DBCCreatedVersion-Methode

Ermittelt die Version von SQL Anywhere, die zur Erstellung einer Datenbankdatei verwendet wurde, ohne die Datenbank zu starten.

Syntax

```
_crtn short _entry DBCreatedVersion(a_db_version_info * pdvi)
```

Parameter

- **pdvi** Zeiger auf eine korrekt initialisierte a_db_version_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Derzeit unterscheidet diese Funktion nur zwischen Datenbanken, die mit Version 9 oder einer früheren Version erstellt wurden, und solchen, die mit Version 10 oder später erstellt wurden.

Es werden keine Versionsinformationen angegeben, wenn der Rückgabecode ein negatives Ausführungsergebnis anzeigt.

Siehe auch

- [a_db_version_info-Struktur \[Datenbanktools\] auf Seite 905](#)
- [Version-Enumeration \[Datenbanktools\] auf Seite 894](#)
- [„CREATE DATABASE-Anweisung“ \[SQL Anywhere Server - SQL-Referenzhandbuch\]](#)

DBErase-Methode

Löscht eine Datenbankdatei bzw. ein Transaktionslog.

Syntax

```
_crtn short _entry DBErase(const an_erase_db * pedb)
```

Parameter

- **pedb** Zeiger auf eine korrekt initialisierte an_erase_db-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dberase verwendet.

Siehe auch

- [an_erase_db-Struktur \[Datenbanktools\] auf Seite 936](#)
- [„Löschen-Dienstprogramm \(dberase\)“ \[SQL Anywhere Server - Datenbankadministration\]](#)

DBInfo-Methode

Gibt Informationen über eine Datenbankdatei zurück.

Syntax

```
_crtn short _entry DBInfo(a_db_info * pdbi)
```

Parameter

- **pdbi** Zeiger auf eine korrekt initialisierte a_db_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbinfo verwendet.

Siehe auch

- [a_db_info-Struktur \[Datenbanktools\] auf Seite 903](#)
- [DBInfoDump-Methode \[Datenbanktools\] auf Seite 882](#)
- [DBInfoFree-Methode \[Datenbanktools\] auf Seite 882](#)
- [„Informations-Dienstprogramm \(dbinfo\)“ \[SQL Anywhere Server - Datenbankadministration\]](#)

DBInfoDump-Methode

Gibt Informationen über eine Datenbankdatei zurück.

Syntax

```
_crtn short _entry DBInfoDump(a_db_info * pdbi)
```

Parameter

- **pdbi** Zeiger auf eine korrekt initialisierte a_db_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbinfo verwendet, wenn die Option -u benutzt wird.

Siehe auch

- [a_db_info-Struktur \[Datenbanktools\] auf Seite 903](#)
- [DBInfo-Methode \[Datenbanktools\] auf Seite 881](#)
- [DBInfoFree-Methode \[Datenbanktools\] auf Seite 882](#)

DBInfoFree-Methode

Gibt nach dem Aufruf der Funktion DBInfoDump Ressourcen frei.

Syntax

```
_crtn short _entry DBInfoFree(a_db_info * pdbi)
```

Parameter

- **pdbi** Zeiger auf eine korrekt initialisierte a_db_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Siehe auch

- [a_db_info-Struktur \[Datenbanktools\]](#) auf Seite 903
- [DBInfo-Methode \[Datenbanktools\]](#) auf Seite 881
- [DBInfoDump-Methode \[Datenbanktools\]](#) auf Seite 882

DBLicense-Methode

Ändert die Lizenzdaten des Datenbankservers oder teilt sie mit.

Syntax

```
_crtn short _entry DBLicense(const a_dblic_info * pdi)
```

Parameter

- **pdi** Zeiger auf eine korrekt initialisierte a_dblic_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Siehe auch

- [a_dblic_info-Struktur \[Datenbanktools\]](#) auf Seite 906
- „Dienstprogramm für die Serverlizenzierung (dblic)“ [*SQL Anywhere Server - Datenbankadministration*]

DBLogFileInfo-Methode

Gibt die Logdatei- und Spiegellogdatei-Pfade für eine nicht laufende Datenbankdatei zurück.

Syntax

```
_crtn short _entry DBLogFileInfo(const a_log_file_info * plfi)
```

Parameter

- **plfi** Zeiger auf eine korrekt initialisierte a_log_file_info Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Beachten Sie, dass diese Methode nur für Datenbanken funktioniert, die mit SQL Anywhere 10.0.0 und höher erstellt wurden.

Siehe auch

- [a_log_file_info-Struktur \[Datenbanktools\]](#) auf Seite 907

DBRemoteSQL-Methode

Greift auf den SQL Remote-Nachrichtenagenten zu.

Syntax

```
_crtn short _entry DBRemoteSQL(a_remote_sql * prs)
```

Parameter

- **prs** Zeiger auf eine korrekt initialisierte a_remote_sql-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Weitere Hinweise über diese Features finden Sie unter „SQL Remote-Nachrichtenagent-Dienstprogramm (dbremote)“ [[SQL Remote](#)].

Siehe auch

- [a_remote_sql-Struktur \[Datenbanktools\]](#) auf Seite 908
- „SQL Remote-Systeme“ [[SQL Remote](#)]

DBSynchronizeLog-Methode

Synchronisiert eine Datenbank mit einem MobiLink-Server.

Syntax

```
_crtn short _entry DBSynchronizeLog(const a_sync_db * psdb)
```

Parameter

- **psdb** Zeiger auf eine korrekt initialisierte a_sync_db-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Siehe auch

- [a_sync_db-Struktur \[Datenbanktools\]](#) auf Seite 917
- „DBTools-Schnittstelle für dbmlsync“ [[MobiLink - Clientadministration](#)]

DBToolsFini-Methode

Vermindert einen Referenzzähler und gibt Ressourcen frei, nachdem eine Anwendung die DBTools-Bibliothek nicht mehr benötigt.

Syntax

```
_crtn short _entry DBToolsFini(const a_dbtools_info * pdi)
```

Parameter

- **pdi** Zeiger auf eine korrekt initialisierte a_dbtools_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Die Funktion DBToolsFini muss am Ende jeder Anwendung aufgerufen werden, die die DBTools-Schnittstelle verwendet. Falls diese Funktion nicht aufgerufen wird, kann das zum Verlust von Speicherressourcen führen.

Siehe auch

- [a_dbtools_info-Struktur \[Datenbanktools\] auf Seite 907](#)
- [DBToolsInit-Methode \[Datenbanktools\] auf Seite 885](#)

DBToolsInit-Methode

Bereitet die DBTools-Bibliothek zur Benutzung vor.

Syntax

```
_crtn short _entry DBToolsInit(const a_dbtools_info * pdi)
```

Parameter

- **pdi** Zeiger auf eine korrekt initialisierte a_dbtools_info-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Der hauptsächliche Zweck der Funktion DBToolsInit ist es, die Meldungsbibliothek von SQL Anywhere zu laden. Die Meldungsbibliothek enthält lokalisierte Versionen der Fehlermeldungen und Eingabeaufforderungen, die von den Funktionen in der DBTools-Bibliothek verwendet werden.

Die Funktion DBToolsInit muss zu Beginn jeder Anwendung aufgerufen werden, die die DBTools-Schnittstelle verwendet. Erst dann können weitere DBTools-Funktionen verwendet werden.

Siehe auch

- [a_dbtools_info-Struktur \[Datenbanktools\]](#) auf Seite 907
- [DBToolsFini-Methode \[Datenbanktools\]](#) auf Seite 885
- [„Ein Beispiel für DBTools“](#) auf Seite 876

DBToolsVersion-Methode

Gibt die Versionsnummer der DBTools-Bibliothek zurück.

Syntax

```
_crtn short _entry DBToolsVersion(void)
```

Bemerkungen

Verwenden Sie die Funktion DBToolsVersion, um zu überprüfen, dass die DBTools-Bibliothek nicht älter ist als diejenige, mit der Ihre Anwendung entwickelt wurde. Anwendungen können mit neueren Versionen von DBTools laufen, nicht aber mit älteren Versionen.

Siehe auch

- [„Versionsnummern und Kompatibilität“](#) auf Seite 875

DBTranslateLog-Methode

Übersetzt eine Transaktionslogdatei in SQL.

Syntax

```
_crtn short _entry DBTranslateLog(const a_translate_log * ptl)
```

Parameter

- **ptl** Zeiger auf eine korrekt initialisierte a_translate_log-Struktur.

Rückgabe

Rückgabecode, wie in [„Exit-Codes der Softwarekomponenten“](#) auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbtran verwendet.

Siehe auch

- [a_translate_log-Struktur \[Datenbanktools\]](#) auf Seite 929
- [„Dienstprogramm zur Logkonvertierung \(dbtran\)“ \[SQL Anywhere Server - Datenbankadministration\]](#)

DBTruncateLog-Methode

Kürzt eine Transaktionslogdatei.

Syntax

```
_crtn short _entry DBTruncateLog(const a_truncate_log * ptl)
```

Parameter

- **ptl** Zeiger auf eine korrekt initialisierte a_truncate_log-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbbackup verwendet.

Siehe auch

- [a_truncate_log-Struktur \[Datenbanktools\]](#) auf Seite 933
- „Sicherungsdienstprogramm (dbbackup)“ [*SQL Anywhere Server - Datenbankadministration*]
- „BACKUP-Anweisung“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

DBUnload-Methode

Entlädt eine Datenbank.

Syntax

```
_crtn short _entry DBUnload(an_unload_db * pudb)
```

Parameter

- **pudb** Zeiger auf eine korrekt initialisierte an_unload_db-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird von den Dienstprogrammen dbunload und dbxtract verwendet.

Siehe auch

- [an_unload_db-Struktur \[Datenbanktools\]](#) auf Seite 937
- „Dienstprogramm zum Entladen (dbunload)“ [*SQL Anywhere Server - Datenbankadministration*]
- „Extraktionsdienstprogramm (dbxtract)“ [*SQL Remote*]

DBUpgrade-Methode

Installiert ein Upgrade für eine Datenbankdatei.

Syntax

```
_crtn short _entry DBUpgrade(const an_upgrade_db * pudb)
```

Parameter

- **puadb** Zeiger auf eine korrekt initialisierte an_upgrade_db-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbupgrad verwendet.

Siehe auch

- an_upgrade_db-Struktur [Datenbanktools] auf Seite 943
- „Dienstprogramm zum Upgrade (dbupgrad)“ [SQL Anywhere Server - Datenbankadministration]
- „ALTER DATABASE-Anweisung“ [SQL Anywhere Server - SQL-Referenzhandbuch]

DBValidate-Methode

Validiert eine Datenbank oder Teile einer Datenbank.

Syntax

```
_crtn short _entry DBValidate(const a_validate_db * pvdb)
```

Parameter

- **pvdb** Zeiger auf eine korrekt initialisierte a_validate_db-Struktur.

Rückgabe

Rückgabecode, wie in „Exit-Codes der Softwarekomponenten“ auf Seite 945 beschrieben.

Bemerkungen

Diese Funktion wird vom Dienstprogramm dbvalid verwendet.

Vorsicht: Die Validierung einer Tabelle oder einer ganzen Datenbank darf nur durchgeführt werden, wenn keine Verbindungen Änderungen in der Datenbank durchführen, weil sonst möglicherweise Fehler über eine Datenbankbeschädigung gemeldet werden, obwohl eine solche nicht vorliegt.

Siehe auch

- a_validate_db-Struktur [Datenbanktools] auf Seite 934
- „Validierungs-Dienstprogramm (dbvalid)“ [SQL Anywhere Server - Datenbankadministration]
- „VALIDATE-Anweisung“ [SQL Anywhere Server - SQL-Referenzhandbuch]

Autotune-Enumeration

Wird in der a_backup_db-Struktur verwendet, um die automatische Optimierung von Writer-Threads zu steuern.

Syntaxenum **Autotune****Mitglieder**

Mitgliedsname	Beschreibung	Wert
BACK- UP_AUTO_TUNE_UNSPECI- FIED	Verwenden Sie dies, wenn Sie keine AUTO TUNE WRITERS-Klausel angeben möchten.	0
BACKUP_AUTO_TUNE_ON	Verwenden Sie dies, um die AUTO TUNE WRITERS ON-Klausel zu generieren.	
BACKUP_AUTO_TUNE_OFF	Verwenden Sie dies, um die AUTO TUNE WRITERS OFF-Klausel zu generieren.	

Siehe auch

- [a_backup_db-Struktur \[Datenbanktools\] auf Seite 894](#)
- [DBBackup-Methode \[Datenbanktools\] auf Seite 879](#)

Checkpoint-Enumeration

Wird in der a_backup_db-Struktur verwendet, um das Kopieren des Checkpoint-Logs zu steuern.

Syntaxenum **Checkpoint****Mitglieder**

Mitgliedsname	Beschreibung	Wert
BACKUP_CHKPT_LOG_COPY	Zum Generieren der WITH CHECKPOINT LOG COPY-Klausel verwenden	0
BACKUP_CHKPT_LOG_NOCOPY	Zum Generieren der WITH CHECKPOINT LOG NOCOPY-Klausel verwenden	
BACKUP_CHKPT_LOG_RECOVER	Zum Generieren der WITH CHECKPOINT LOG RECOVER-Klausel verwenden.	
BACKUP_CHKPT_LOG_AUTO	Zum Generieren der WITH CHECKPOINT LOG AUTO-Klausel verwenden	
BACK- UP_CHKPT_LOG_DEFAULT	Verwenden, um WITH CHECKPOINT-Klauseln auszulassen	

Siehe auch

- [a_backup_db-Struktur \[Datenbanktools\]](#) auf Seite 894
- [DBBackup-Methode \[Datenbanktools\]](#) auf Seite 879

History-Enumeration

Wird in der a_backup_db-Struktur verwendet, um das Aktivieren des Sicherungsverlaufs zu steuern.

Syntax

enum **History**

Mitglieder

Mitgliedsname	Beschreibung	Wert
BACKUP_HISTORY_UNSPECIFIED	Verwenden Sie dies, wenn Sie keine HISTORY-Klausel angeben möchten.	0
BACKUP_HISTORY_ON	Verwenden Sie dies, um die HISTORY ON-Klausel zu generieren.	
BACKUP_HISTORY_OFF	Verwenden Sie dies, um die HISTORY OFF-Klausel zu generieren.	

Siehe auch

- [a_backup_db-Struktur \[Datenbanktools\]](#) auf Seite 894
- [DBBackup-Methode \[Datenbanktools\]](#) auf Seite 879

Padding-Enumeration

Die Padding-Enumeration (zum Auffüllen mit Leerzeichen) legt die blank_pad-Einstellung in a_create_db fest.

Syntax

enum **Padding**

Mitglieder

Mitgliedsname	Beschreibung
NO_BLANK_PADDING	Kein Auffüllen mit Leerzeichen
BLANK_PADDING	Auffüllen mit Leerzeichen

Siehe auch

- [a_create_db-Struktur \[Datenbanktools\] auf Seite 899](#)
- [DBCCreate-Methode \[Datenbanktools\] auf Seite 880](#)

Unit-Enumeration

Wird in der `a_create_db` structure verwendet, um den Wert von `db_size_unit` anzugeben.

Syntax

```
enum Unit
```

Mitglieder

Mitgliedsname	Beschreibung
DBSP_UNIT_NONE	Einheiten werden nicht angegeben.
DBSP_UNIT_PAGES	Die Größe wird in Seiten angegeben.
DBSP_UNIT_BYTES	Die Größe wird in Byte angegeben.
DBSP_UNIT_KILOBYTES	Die Größe wird in kB angegeben.
DBSP_UNIT_MEGABYTES	Die Größe wird in MB angegeben.
DBSP_UNIT_GIGABYTES	Die Größe wird in GB angegeben.
DBSP_UNIT_TERABYTES	Die Größe wird in TByte angegeben.

Siehe auch

- [a_create_db-Struktur \[Datenbanktools\] auf Seite 899](#)
- [DBCCreate-Methode \[Datenbanktools\] auf Seite 880](#)

Unload-Enumeration

Der Entladungstyp, der von der Struktur "an_unload-db" (siehe `an_unload_db` structure) verwendet wird.

Syntax

```
enum Unload
```

Mitglieder

Mitgliedsname	Beschreibung
UNLOAD_ALL	Sowohl Daten als auch Schema entladen

Mitgliedsname	Beschreibung
UNLOAD_DATA_ONLY	Daten entladen. Schema nicht entladen. Entspricht der dbunload-Option -d.
UNLOAD_NO_DATA	Keine Daten. Nur Schema entladen. Entspricht der dbunload-Option -n.
UNLOAD_NO_DATA_FULL_SCRIPT	Keine Daten. LOAD/INPUT-Anweisungen in das Neulade-skript aufnehmen. Entspricht der dbunload-Option -nl.
UNLOAD_NO_DATA_NAME_ORDER	Keine Daten. Objekte werden nach Namen geordnet ausgegeben.

Siehe auch

- [an_unload_db-Struktur \[Datenbanktools\] auf Seite 937](#)
- [DBUnload-Methode \[Datenbanktools\] auf Seite 887](#)

UserList-Enumeration

Typ einer Benutzerliste, wie er unter a_translate_log structure beschrieben ist.

Syntax

```
enum UserList
```

Mitglieder

Mitgliedsname	Beschreibung
DBTRAN_INCLUDE_ALL	Alle Benutzervorgänge einschließen
DBTRAN_INCLUDE_SOME	Nur Vorgänge der in der Benutzerliste aufgeführten Benutzer einschließen
DBTRAN_EXCLUDE_SOME	Vorgänge der in der Benutzerliste aufgeführten Benutzer ausschließen

Siehe auch

- [a_translate_log-Struktur \[Datenbanktools\] auf Seite 929](#)
- [DBTranslateLog-Methode \[Datenbanktools\] auf Seite 886](#)

Validierung-Enumeration

Der ausgeführte Validierungstyp gemäß a_validate_db structure.

Syntaxenum **Validation****Mitglieder**

Mitgliedsname	Beschreibung	Wert
VALIDATE_NORMAL	Nur mit der Standardprüfung validieren	0
VALIDATE_DATA	(Veraltet)	
VALIDATE_INDEX	(Veraltet)	
VALIDATE_EXPRESS	Mit Expressprüfung validieren. Entspricht der dbvalid-Option -fx.	
VALIDATE_FULL	(Veraltet)	
VALIDATE_CHECKSUM	Datenbankprüfsummen validieren. Entspricht der dbvalid-Option -s.	
VALIDATE_DATABASE	Datenbank validieren. Entspricht der dbvalid-Option -d.	
VALIDATE_COMPLETE	Alle möglichen Validierungsaktivitäten ausführen.	

Siehe auch

- [a_validate_db-Struktur \[Datenbanktools\] auf Seite 934](#)
- [DBValidate-Methode \[Datenbanktools\] auf Seite 888](#)
- [„Erkennung von Beschädigungen mithilfe von Prüfsummen“ \[SQL Anywhere Server - Datenbankadministration\]](#)

Verbosity-Enumeration

Die Verbosity-Enumeration legt die Ausführlichkeit der Ausgabe fest.

Syntaxenum **Verbosity****Mitglieder**

Mitgliedsname	Beschreibung
VB_QUIET	Keine Ausgabe
VB_NORMAL	Normaler Ausgabeumfang
VB_VERBOSE	Ausgabe ausführlich darstellen, nützlich für die Fehlersuche

Siehe auch

- [a_create_db-Struktur \[Datenbanktools\] auf Seite 899](#)
- [DBCCreate-Methode \[Datenbanktools\] auf Seite 880](#)

Version-Enumeration

Wird in der `a_db_version_info` structure verwendet, um die Version von SQL Anywhere anzugeben, mit der die Datenbank ursprünglich erstellt wurde.

Syntax

```
enum Version
```

Mitglieder

Mitgliedsname	Beschreibung	Wert
VERSION_UNKNOWN	Es ist nicht möglich, die Version von SQL Anywhere anzugeben, mit der die Datenbank ursprünglich erstellt wurde.	0
VERSION_PRE_10	Die Datenbank wurde mit SQL Anywhere 9 oder früher erstellt.	9
VERSION_10	Die Datenbank wurde mit SQL Anywhere 10 erstellt.	10
VERSION_11	Die Datenbank wurde mit SQL Anywhere 11 erstellt.	11
VERSION_12	Die Datenbank wurde mit SQL Anywhere 12 erstellt.	12
VERSION_16	Die Datenbank wurde mit SQL Anywhere 16 erstellt.	16

Siehe auch

- [a_db_version_info-Struktur \[Datenbanktools\] auf Seite 905](#)
- [DBCCreatedVersion-Methode \[Datenbanktools\] auf Seite 880](#)

a_backup_db-Struktur

Enthält die Informationen, die gebraucht werden, um Sicherungsaufgaben mit der DBTools-Bibliothek auszuführen.

Syntax

```
typedef struct a_backup_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
auto_tune_writers	char	AUTO TUNE WRITER aktivieren oder deaktivieren. Muss BACKUP_AUTO_TUNE_UNSPECIFIED, BACKUP_AUTO_TUNE_ON oder BACKUP_AUTO_TUNE_OFF sein. Verwenden Sie dies, um die AUTO TUNE WRITERS OFF-Klausel zu generieren. Durch die dbbackup-Option -aw[-] festgelegt
backup_comment	const char *	Kommentar für die WITH COMMENT-Klausel.
backup_database	a_bit_field	Sichern der Datenbankdatei. Von dbbackup-Option -d auf TRUE gesetzt.
backup_history	char	Sicherungsverlauf. Muss BACKUP_HISTORY_UNSPECIFIED, BACKUP_HISTORY_ON oder BACKUP_HISTORY_OFF sein. Durch die dbbackup-Option -h[-] festgelegt
backup_interrupted	char	Angabe, dass der Vorgang bei Nicht-Null unterbrochen wurde.
backup_logfile	a_bit_field	Sichern der Transaktionslogdatei. Von dbbackup-Option -t auf TRUE gesetzt.
chkpt_log_type	char	Kopieren des Checkpoint-Logs steuern. Muss einen der folgenden Werte haben: BACKUP_CHKPT_LOG_COPY, BACKUP_CHKPT_LOG_NOCOPY, BACKUP_CHKPT_LOG_RECOVER, BACKUP_CHKPT_LOG_AUTO oder BACKUP_CHKPT_LOG_DEFAULT. Von dbbackup-Option -k festgelegt.
confirmrtn	MSG_CALLBACK	Adresse einer Bestätigungsanforderungs-Callback-Routine oder NULL.

Mitgliedsname	Typ	Beschreibung
connectparms	const char *	<p>Parameter für die Verbindung zur Datenbank.</p> <p>Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLA-ny16\bin32\dbsrv16.exe".</p> <p>Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLA-ny16\bin32\dbsrv16.exe".</p>
errortrn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
hotlog_filename	const char *	<p>Dateiname für die Live-Sicherungsdatei</p> <p>Von dbbackup-Option -l festgelegt.</p>
msgtrn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
no_confirm	a_bit_field	<p>Ohne Bestätigung durchführen.</p> <p>Von dbbackup-Option -y auf TRUE gesetzt.</p>
output_dir	const char *	Suchpfad des Ausgabeverzeichnis für Sicherungen, zum Beispiel: "c:\backup".
page_blocksize	a_sql_uint32	<p>Anzahl der Seiten in Datenblöcken.</p> <p>Wenn der Wert 0 ist, lautet der Standardwert 128. Von dbbackup-Option -b festgelegt.</p>
progress_messages	a_bit_field	<p>Zeigt Meldungen über den Verarbeitungsfortschritt an.</p> <p>Von dbbackup-Option -p auf TRUE gesetzt.</p>
quiet	a_bit_field	<p>Verarbeitung ohne Ausgabe von Meldungen</p> <p>Von dbbackup-Option -q auf TRUE gesetzt.</p>
rename_local_log	a_bit_field	<p>Lokale Sicherung des Transaktionslogs umbenennen</p> <p>Von dbbackup-Option -n auf TRUE gesetzt.</p>

Mitgliedsname	Typ	Beschreibung
rename_log	a_bit_field	Transaktionslog umbenennen. Von dbbackup-Option -r auf TRUE gesetzt.
server_backup	a_bit_field	Backup auf dem Server mit BACKUP DATABASE. Von dbbackup-Option -s auf TRUE gesetzt.
statusrtn	MSG_CALL-BACK	Adresse einer Statusmeldung-Callback-Routine oder NULL.
truncate_log	a_bit_field	Das Transaktionslog löschen Von dbbackup-Option -x auf TRUE gesetzt.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).
wait_after_end	a_bit_field	Warten nach dem Ende. Durch die dbbackup-Option -wa auf TRUE gesetzt.
wait_before_start	a_bit_field	Warten vor dem Start. Durch die dbbackup-Option -wb auf TRUE gesetzt.

Siehe auch

- [Checkpoint-Enumeration \[Datenbanktools\] auf Seite 889](#)
- [DBBackup-Methode \[Datenbanktools\] auf Seite 879](#)
- [„Callback-Funktionen“ auf Seite 873](#)
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)

a_change_log-Struktur

Enthält die Informationen, die gebraucht werden, um dblog-Aufgaben mit der DBTools-Bibliothek auszuführen.

Syntax

```
typedef struct a_change_log
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
change_logname	a_bit_field	Auf TRUE setzen, um den Transaktionslognamen zu ändern. Von dblog-Option -x oder -t auf TRUE gesetzt.
change_mirrorname	a_bit_field	Auf TRUE setzen, um den Spiegellognamen zu ändern. Von dblog-Option -m, -n oder -r auf TRUE gesetzt.
dbname	const char *	Datenbankdateiname.
encryption_key	char *	Der Chiffrierschlüssel für die Datenbankdatei. Entspricht dblog-Option -ek oder -ep.
errorrtn	MSG_CALL-BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
generation_number	unsigned short	Die neue Generationsnummer. Reserviert, 0 verwenden.
ignore_dbsync_trunc	a_bit_field	Bei Verwendung von dbmlsync wird hiermit der Ausgangspunkt für die Option delete_old_logs zurückgesetzt, mit dem nicht mehr benötigte Transaktionslogs gelöscht werden können. Von dblog-Option -is auf TRUE gesetzt.
ignore_ltm_trunc	a_bit_field	Reserviert. Verwenden Sie FALSE.
ignore_remote_trunc	a_bit_field	Für SQL Remote. Setzt den Ausgangspunkt für die Option delete_old_logs zurück, mit dem nicht mehr benötigte Transaktionslogs gelöscht werden können. Von dblog-Option -ir auf TRUE gesetzt.
logname	const char *	Transaktionslogdateiname oder NULL, wenn es kein Log gibt.
mirrorname	const char *	Der neue Name des Transaktionslogspiegels. Entspricht der dblog-Option -m.
msggrtn	MSG_CALL-BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
query_only	a_bit_field	Wenn 1, nur den Namen des Transaktionslogs anzeigen. Wenn 0, Änderung des Lognamens erlauben.

Mitgliedsname	Typ	Beschreibung
quiet	a_bit_field	Verarbeitung ohne Ausgabe von Meldungen Von dblog-Option -q auf TRUE gesetzt.
set_generation_number	a_bit_field	Reserviert. Verwenden Sie FALSE.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).
zap_current_offset	char *	Den aktuellen Ausgangspunkt auf den angegebenen Wert ändern. Dies ist nur für das Zurücksetzen eines Transaktionslogs nach Entladen und Aktualisieren zur Abstimmung mit den Einstellungen für dbremote oder dbmlsync vorgesehen. Entspricht dblog-Option -x.
zap_starting_offset	char *	Den Ausgangspunkt auf den angegebenen Wert ändern. Dies ist nur für das Zurücksetzen eines Transaktionslogs nach Entladen und Aktualisieren zur Abstimmung mit den Einstellungen für dbremote oder dbmlsync vorgesehen. Entspricht dblog-Option -z.

Siehe auch

- [DBChangeLogName-Methode \[Datenbanktools\] auf Seite 879](#)

a_create_db-Struktur

Enthält die Informationen, die gebraucht werden, um eine Datenbank mit der DBTools-Bibliothek zu erstellen.

Syntax

```
typedef struct a_create_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
accent_sensitivity	char	Entweder 'y', 'n' oder 'f' (ja, nein, französisch). Generiert eine der Klauseln ACCENT RESPECT, ACCENT IGNORE oder ACCENT FRENCH.

Mitgliedsname	Typ	Beschreibung
avoid_view_collisions	a_bit_field	Auf TRUE setzen, um die Erzeugung der Watcom SQL-Kompatibilitäts-Ansichten SYS.SYSCOLUMNS und SYS.SYSINDEXES zu vermeiden. Von der dbinit-Option -k auf TRUE gesetzt.
blank_pad	a_bit_field	Muss NO_BLANK_PADDING oder BLANK_PADDING sein. Leerzeichen in Zeichenfolgenvergleichen beachten und entsprechende Indexinformationen speichern. Siehe Padding-Enumeration. Entspricht der dbinit-Option -b.
case_sensitivity_use_default	a_bit_field	Auf TRUE setzen, um die standardmäßige Berücksichtigung von Groß- und Kleinschreibung für die Sprachumgebung zu verwenden. Dies bezieht sich nur auf UCA. Wenn dieser Wert auf TRUE gesetzt ist, wird die CASE RESPECT-Klausel der CREATE DATABASE-Anweisung nicht hinzugefügt.
checksum	a_bit_field	Auf TRUE für ON, auf FALSE für OFF. Generiert die CHECKSUM ON- oder die CHECKSUM OFF-Klausel. Von der dbinit-Option -s auf TRUE gesetzt.
data_store_type	const char *	Reserviert. NULL verwenden.
db_size	unsigned int	Wenn nicht NULL, wird die Klausel DATABASE SIZE generiert. Entspricht der dbinit-Option -dbs.
db_size_unit	int	Mit db_size verwendet, muss DBSP_UNIT_NONE, DBSP_UNIT_PAGES, DBSP_UNIT_BYTES, DBSP_UNIT_KILOBYTES, DBSP_UNIT_MEGABYTES, DBSP_UNIT_GIGABYTES oder DBSP_UNIT_TERABYTES sein. Wenn der Wert nicht DBSP_UNIT_NONE ist, wird das entsprechende Schlüsselwort generiert (DATABASE SIZE 10 MB wird z.B. generiert, wenn db_size den Wert 10 hat und db_size_unit den Wert DBSP_UNIT_MEGABYTES hat).
dba_pwd	char *	Wenn nicht NULL, wird die Klausel DBA PASSWORD xxx generiert. Entspricht der dbinit-Option -dba.
dba_uid	char *	Wenn nicht NULL, wird die Klausel DBA USER xxx generiert. Entspricht der dbinit-Option -dba.

Mitgliedsname	Typ	Beschreibung
dbname	const char *	Datenbankdateiname.
default_collation	const char *	Die Kollation für die Datenbank. Entspricht der dbinit-Option -Z.
Kodierung	const char *	Die Zeichensatzkodierung. Entspricht der dbinit-Option -ze.
encrypt	a_bit_field	Wenn auf TRUE gesetzt, wird die ENCRYPTED ON-Klausel generiert. Wenn encrypted_tables ebenfalls gesetzt ist, wird die Klausel ENCRYPTED TABLES ON generiert. Von den dbinit-Optionen -e? auf TRUE gesetzt.
encrypted_tables	a_bit_field	Auf TRUE setzen, um die Tabellen zu verschlüsseln. Wenn mit encrypt verwendet, wird die Klausel ENCRYPTED TABLE ON anstelle der Klausel ENCRYPTED ON generiert. Von der dbinit-Option -et auf TRUE gesetzt.
encryption_algorithm	const char *	Der Verschlüsselungsalgorithmus (AES, AES256, AES_FIPS oder AES256_FIPS). Bei der Verwendung dieses Werts mit encrypt und encryption_key wird die ALGORITHM-Klausel generiert. Entspricht der dbinit-Option -ea.
encryption_key	const char *	Der Chiffrierschlüssel für die Datenbankdatei. Wenn mit encrypt verwendet, wird die KEY-Klausel generiert. Entspricht der dbinit-Option -ek.
errortn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
iq_params	void *	Reserviert. NULL verwenden.
jconnect	a_bit_field	Auf TRUE setzen, um von für jConnect erforderliche Systemprozeduren einzuschließen. Von der dbinit-Option -i auf FALSE gesetzt.
logname	const char *	Neuer Transaktionslogname. Entspricht der dbinit-Option -t.
mirrorname	const char *	Name des Transaktionslogsiegels. Entspricht der dbinit-Option -m.

Mitgliedsname	Typ	Beschreibung
msgsrtn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
nchar_collation	const char *	Der NCHAR COLLATION-Wert für die Datenbank, wenn nicht NULL. Entspricht der dbinit-Option -zn.
page_size	unsigned short	Die Seitengröße der Datenbank. Entspricht der dbinit-Option -p.
respect_case	a_bit_field	Groß- und Kleinschreibung in Zeichenfolgevergleichen beachten und entsprechende Indexinformationen speichern. Von der dbinit-Option -c auf TRUE gesetzt.
startline	const char *	Befehlszeile zum Starten des Datenbankservers. Beispiel: "c:\SQLAny16\bin32\dbsrv16.exe". Beim Wert NULL, ist der START-Parameter für SQL Anywhere standardmäßig "dbeng16 -gp <Seitengröße> -c 10M", wobei die Seitengröße unten angegeben wird. Beachten Sie, dass "-c 10M" angehängt wird, wenn Seitengröße >= 2048.
sys_proc_definer	a_bit_field	Auf TRUE setzen, um das SQL SECURITY-Modell für gespeicherte Systemprozeduren Version 12.0.1 oder früher anzugeben. Durch die dbinit-Option -pd auf TRUE gesetzt.
verbose	char	Siehe Verbosity-Enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [Padding-Enumeration \[Datenbanktools\] auf Seite 890](#)
- [Unit-Enumeration \[Datenbanktools\] auf Seite 891](#)
- [Verbosity-Enumeration \[Datenbanktools\] auf Seite 893](#)
- [DBCCreate-Methode \[Datenbanktools\] auf Seite 880](#)
- [„Erkennung von Beschädigungen mithilfe von Prüfsummen“ \[SQL Anywhere Server - Datenbankadministration\]](#)

a_db_info-Struktur

Enthält die Informationen, die gebraucht werden, um dbinfo-Informationen mit der DBTools-Bibliothek zurückzugeben.

Syntax

```
typedef struct a_db_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
bit_map_pages	a_sql_uint32	Anzahl von Bitmap-Seiten in der Datenbank
charcollationspecbuffer	char *	Zeiger auf den Zeichenfolgenpuffer der CHAR-Kollation.
charcollationspecbufsize	unsigned short	Größe von charcollationspecbuffer (mindestens 256+1).
charencodingbuffer	char *	Zeiger auf den char-Kodierung-Zeichenfolgenpuffer.
charencodingbufsize	unsigned short	Größe von charencodingbuffer (mindestens 50+1).
checksum	a_bit_field	Wenn auf TRUE gesetzt, sind Prüfsummen aktiviert (eine Prüfsumme für jede Datenbankseite).
connectparms	const char *	<p>Parameter für die Verbindung zur Datenbank.</p> <p>Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Zum Beispiel: "START=c:\SQLAny16\bin32\dbsrv16.exe".</p> <p>Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".</p>
dbbufsize	unsigned short	Größe von dbnamebuffer (Beispiel: _MAX_PATH).
dbnamebuffer	char *	Zeiger auf den Datenbankdateinamenpuffer.
encrypted_tables	a_bit_field	Wenn auf TRUE gesetzt, werden verschlüsselten Tabellen unterstützt.

Mitgliedsname	Typ	Beschreibung
errorrtn	MSG_CALL-BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
file_size	a_sql_uint32	Größe der Datenbankdatei (in Seiten).
free_pages	a_sql_uint32	Anzahl der freien Seiten
logbufsize	unsigned short	Größe von lognamebuffer (zum Beispiel _MAX_PATH).
lognamebuffer	char *	Zeiger auf den Transaktionslogdateinamenpuffer.
mirrorbufsize	unsigned short	Größe von mirrornamebuffer (zum Beispiel _MAX_PATH).
mirrornamebuffer	char *	Zeiger auf den Spiegeldateinamenpuffer.
msgsrtn	MSG_CALL-BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
ncharcollations-pecbuffer	char *	Zeiger auf den Zeichenfolgenpuffer der NCHAR-Kollation.
ncharcollations-pecbufsize	unsigned short	Größe von ncharcollationspecbuffer (mindestens 256+1).
ncharencoding-buffer	char *	Zeiger auf nchar-Kodierungszeichenfolgepuffer.
ncharencoding-bufsize	unsigned short	Größe von ncharencodingbuffer (mindestens 50+1).
other_pages	a_sql_uint32	Anzahl der Seiten, die keine Tabellenseiten, Indexseiten, freie Seiten oder Bitmap-Seiten sind
page_usage	a_bit_field	Auf TRUE setzen, um Seitennutzungsstatistiken auszugeben, sonst FALSE. Von dbinfo-Option -u auf TRUE gesetzt.
quiet	a_bit_field	Auf TRUE setzen, um ohne Meldungsbestätigung zu arbeiten. Von dbinfo-Option -q auf TRUE gesetzt.
statusrtn	MSG_CALL-BACK	Adresse einer Statusmeldung-Callback-Routine oder NULL.
sysinfo	a_sysinfo	Inline-a_sysinfo-Struktur.

Mitgliedsname	Typ	Beschreibung
totals	a_table_info *	Zeiger auf eine a_table_info-Struktur
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [a_table_info-Struktur \[Datenbanktools\] auf Seite 928](#)
- [a_sysinfo-Struktur \[Datenbanktools\] auf Seite 928](#)
- [DBInfo-Methode \[Datenbanktools\] auf Seite 881](#)
- [DBInfoDump-Methode \[Datenbanktools\] auf Seite 882](#)
- [DBInfoFree-Methode \[Datenbanktools\] auf Seite 882](#)
- [„Callback-Funktionen“ auf Seite 873](#)
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)
- [„Erkennung von Beschädigungen mithilfe von Prüfsummen“ \[SQL Anywhere Server - Datenbankadministration\]](#)

a_db_version_info-Struktur

Speichert Angaben darüber, welche Version von SQL Anywhere zur Erstellung der Datenbank verwendet wurde.

Syntax

```
typedef struct a_db_version_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
created_version	char	Auf VERSION_UNKNOWN, VERSION_PRE_10, usw. setzen. Zeigt die Serverversion, die die Datenbankdatei erstellt hat.
errortrn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
filename	const char *	Name der zu prüfenden Datenbankdatei
msgtrn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [DBCcreatedVersion-Methode \[Datenbanktools\]](#) auf Seite 880
- [Version-Enumeration \[Datenbanktools\]](#) auf Seite 894

a_dblic_info-Struktur

Enthält Angaben zu den Lizenzdaten.

Syntax

```
typedef struct a_dblic_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
compname	char *	Name des Unternehmens für die Lizenz.
conncount	a_sql_int32	Maximale Anzahl von lizenzierten Verbindungen Zur Aktivierung 1000000L als Standardeinstellung verwenden.
errortrn	MSG_CALL- BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
exename	char *	Name der Programm- oder Lizenzdatei des Servers
installkey	char *	Reserviert. Auf NULL gesetzt. Von dblic-Option -k festgelegt.
msgtrn	MSG_CALL- BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
nodecount	a_sql_int32	Anzahl der Knotenlizenzen.
query_only	a_bit_field	Auf TRUE setzen, um nur die Lizenzinformationen anzuzeigen. Auf FALSE setzen, um eine Änderung der Informationen zuzulassen.
quiet	a_bit_field	Auf TRUE setzen, um ohne Meldungsausgabe zu arbeiten. Von dblic-Option -q auf TRUE gesetzt.

Mitgliedsname	Typ	Beschreibung
type	a_license_type	Werte finden Sie unter lictype.h. LICENSE_TYPE_PERSEAT, LICENSE_TYPE_CONCURRENT oder LICENSE_TYPE_PERCPU.
username	char *	Benutzername für die Lizenz.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Bemerkungen

Sie dürfen diese Informationen nur gemäß Ihrem Lizenzvertrag verwenden.

Siehe auch

- [DBLicense-Methode \[Datenbanktools\] auf Seite 883](#)

a_dbtools_info-Struktur

DBTools Informations-Callback-Funktion wird verwendet, um DBTools-Bibliotheksaufrufe zu initialisieren und zu finalisieren.

Syntax

```
typedef struct a_dbtools_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
errortn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.

Siehe auch

- [DBToolsInit-Methode \[Datenbanktools\] auf Seite 885](#)
- [DBToolsFini-Methode \[Datenbanktools\] auf Seite 885](#)

a_log_file_info-Struktur

Wird verwendet, um die Logdatei und die Spiegellogdateiinformationen für eine nicht laufende Datenbank zu erhalten.

Syntax

```
typedef struct a_log_file_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
dbname	const char *	Datenbankdateiname.
encryption_key	const char *	Der Chiffrierschlüssel für die Datenbankdatei.
errorrtn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
logname	char *	Puffer für Transaktionslogdateinamen oder NULL.
logname_size	size_t	Größe des Puffers für Transaktionslogdateinamen oder Null.
mirrorname	char *	Buffer für Spiegellogdateinamen oder NULL.
mirrorname_size	size_t	Größe des Puffers für Spiegellogdateinamen oder Null.
reserved	void *	Reserviert für den internen Gebrauch und muss auf NULL gesetzt werden.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [DBLogFileInfo-Methode \[Datenbanktools\] auf Seite 883](#)

a_name-Struktur

Legt eine Variablenliste mit Namen fest.

Syntax

```
typedef struct a_name
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
name	char	Mindestens ein Byte, das den Namen enthält.
next	struct a_name *	Zeiger auf den nächsten Namen in der Liste oder NULL.

a_remote_sql-Struktur

Enthält Informationen, die für den Gebrauch des Dienstprogramms dbremote mit der DBTools-Bibliothek benötigt werden.

Syntax

```
typedef struct a_remote_sql
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
apply	a_bit_field	Normalerweise auf TRUE gesetzt. Wenn die Option nicht aktiviert ist, werden Nachrichten durchsucht, aber nicht angewendet. Entspricht der dbremote-Option -a.
argv	char **	Zeiger auf eine syntaktisch analysierte Befehlszeile (ein Vektor von Zeigern auf Zeichenfolgen). Ist der Wert nicht NULL, ruft DBRemoteSQL eine Nachrichtenroutine auf, um die einzelnen Befehlszeilenargumente anzuzeigen, ausgenommen Argumente mit dem Präfix -c, -cq oder -ek.
batch	a_bit_field	Wenn auf TRUE gesetzt, wird das Beenden nach dem Anwenden der Nachricht und dem Log-Scan erzwungen (dies ist das Gleiche wie wenn mindestens ein Benutzer den Sendezeitpunkt "immer" eingestellt hat). Wenn FALSE, wird der Ausführungsmodus von den Sendezeiten der entfernten Benutzer bestimmt.
confirmrtn	MSG_CALLBACK	Adresse einer Bestätigungsanforderungs-Callback-Routine oder NULL.
connectparms	char *	Parameter für die Verbindung zur Datenbank. Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db". Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLAny16\bin32\dbeng16.exe". Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbeng16.exe".
debug	a_bit_field	Wenn auf TRUE gesetzt, ist die Debug-Ausgabe eingeschlossen.

Mitgliedsname	Typ	Beschreibung
debug_dump_size	a_sql_uint32	Reserviert für den internen Gebrauch und muss auf 0 gesetzt werden.
debug_page_offsets	a_bit_field	Reserviert für den internen Gebrauch und muss auf FALSE gesetzt werden.
default_window_title	char *	Ein Zeiger auf die Zeichenfolge des Standardfenstertitels
deleted	a_bit_field	Normalerweise auf TRUE gesetzt. Wenn dies nicht gesetzt wird, werden Nachrichten nicht gelöscht, nachdem sie angewendet wurden. Entspricht der dbremote-Option -p.
encryption_key	char *	Zeiger auf einen Chiffrierschlüssel. Entspricht der dbremote-Option -ek.
errorrtn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
frequency	a_sql_uint32	Reserviert für den internen Gebrauch und muss auf 0 gesetzt werden.
full_q_scan	a_bit_field	Reserviert für den internen Gebrauch und muss auf FALSE gesetzt werden.
include_scan_range	char *	Reserviert für den internen Gebrauch und muss auf NULL gesetzt werden.
latest_backup	a_bit_field	Wenn auf TRUE gesetzt, werden nur gesicherte Logs verarbeitet. Senden Sie keine Vorgänge aus einem Live-Log. Entspricht der dbremote-Option -u.
link_debug	a_bit_field	Wenn auf TRUE gesetzt, wird die Fehlersuche für Verknüpfungen aktiviert.
locale	char *	Reserviert für den internen Gebrauch und muss auf NULL gesetzt werden.

Mitgliedsname	Typ	Beschreibung
log_file_name	const char *	<p>Zeiger auf den Namen des DBRemoteSQL-Ausgabe-logs, in das die Nachrichten-Callbacks ihre Ausgabe schreiben.</p> <p>Wenn send TRUE ist, wird das Fehlerlog an die konsolidierte Datenbank gesendet (ausgenommen, dieser Zeiger ist NULL).</p>
log_size	a_sql_uint32	<p>DBRemoteSQL benennt das Online-Transaktionslog um und startet es neu, wenn die Größe des Online-Transaktionslogs größer als dieser Wert ist.</p> <p>Entspricht der dbremote-Option -x.</p>
logrtn	MSG_CALLBACK	<p>Zeiger auf eine Funktion, die die betreffende Nachricht in eine Logdatei schreibt.</p> <p>Diese Nachrichten müssen vom Benutzer nicht gesehen werden.</p>
max_length	a_sql_uint32	<p>Auf die maximale Länge (in Byte) setzen, die eine Nachricht haben kann.</p> <p>Dies betrifft Senden und Empfangen. Der empfohlene Wert ist 50000. Entspricht der dbremote-Option -l.</p>
memory	a_sql_uint32	<p>Auf die maximale Größe (in Byte) der Speicherpuffer setzen, die beim Erstellen von zu sendenden Nachrichten verwendet werden soll.</p> <p>Der empfohlene Wert ist mindestens $2 * 1024 * 1024$. Entspricht der dbremote-Option -m.</p>
mirror_logs	char *	<p>Zeiger auf den Namen des Verzeichnisses, das die Offline-Spiegeltransaktionslogs enthält.</p> <p>Entspricht der dbremote-Option -ml.</p>
more	a_bit_field	Dies sollte auf TRUE gesetzt werden.

Mitgliedsname	Typ	Beschreibung
msgqueue rtn	MSG_QUEUE_CALLBACK	<p>Funktion, die von DBRemoteSQL aufgerufen wird, wenn in den Ruhezustand gewechselt werden soll.</p> <p>Durch diesen Parameter wird die Dauer der Ruhezeit in Millisekunden festgelegt. Diese Funktion sollte nachstehende Ergebnisse zurückgeben, wie es in dllapi.h definiert ist.</p> <ul style="list-style-type: none"> • MSGQ_SLEEP_THROUGH zeigt an, dass die Routine über die angeforderte Anzahl von Millisekunden im Ruhezustand war. In der Regel ist dies der Wert, den Sie zurückgeben sollten. • MSGQ_SHUTDOWN_REQUESTED zeigt an, dass Sie die Synchronisation so schnell wie möglich beenden möchten.
msg rtn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
no_user_interaction	a_bit_field	Wenn auf TRUE gesetzt, ist keine Benutzerinteraktion erforderlich.
operations	a_sql_uint32	<p>Dieser Wert wird verwendet, wenn Nachrichten übernommen werden.</p> <p>Festschreibungen werden ignoriert, bis DBRemoteSQL mindestens diese Anzahl von nicht festgeschriebenen Vorgängen (Einfügungen, Löschungen, Aktualisierungen) hat. Entspricht der dbremote-Option -g.</p>
patience_retry	a_sql_uint32	<p>Diese Option auf die Anzahl der Abfragen eingehender Nachrichten setzen, die DBRemoteSQL warten soll, bevor angenommen wird, dass eine erwartete Nachricht verloren gegangen ist.</p> <p>Wenn beispielsweise patience_retry 3 ist, versucht DBRemoteSQL bis zu drei Mal, die fehlende Nachricht zu erhalten. Anschließend wird eine Neusendeanforderung gesendet. Der empfohlene Wert ist 1. Entspricht der dbremote-Option -rp.</p>

Mitgliedsname	Typ	Beschreibung
progress_index_rtn	SET_PROGRESS_CALLBACK	<p>Zeiger auf eine Funktion, die den Status der Fortschrittsleiste aktualisiert.</p> <p>Diese Funktion hat zwei ganzzahlige Argumente ohne Vorzeichen, nämlich index und max. Beim ersten Aufruf haben die Werte den Mindest- bzw. Höchstwert (z.B. 0, 100). Bei nachfolgenden Aufrufen ist das erste Argument der aktuelle Indexwert (z.B. zwischen 0 und 100) und das zweite Argument ist immer 0.</p>
progress_msg_rtn	MSG_CALLBACK	Zeiger auf eine Funktion, die eine Fortschrittmeldung anzeigt
queueparms	char *	Reserviert für den internen Gebrauch und muss auf NULL gesetzt werden.
receive	a_bit_field	<p>Wenn auf TRUE gesetzt, werden Nachrichten empfangen.</p> <p>Wenn receive und send jeweils FALSE sind, werden beide als TRUE behandelt. Es wird empfohlen, die Einstellung receive und send von auf FALSE zu setzen. Entspricht der dbremote-Option -r.</p>
receive_delay	a_sql_uint32	<p>Diese Option auf die Zeitdauer (in Sekunden) setzen, die zwischen Abfragen neuer eingehender Nachrichten gewartet werden soll.</p> <p>Der empfohlene Wert ist 60. Entspricht der dbremote-Option -rd.</p>
remote_output_file_name	char *	<p>Zeiger auf den Namen der entfernten DBRemoteSQL-Ausgabedatei.</p> <p>Entspricht der dbremote-Option -ro oder -rt.</p>
rename_log	a_bit_field	Wenn auf TRUE gesetzt, werden Logs umbenannt und neu gestartet (nur DBRemoteSQL).

Mitgliedsname	Typ	Beschreibung
resend_urgency	a_sql_uint32	<p>Die Zeitdauer (in Sekunden) setzen, die DBRemoteSQL wartet, nachdem erkannt wurde, dass ein Benutzer einen Rescan durchführen muss, bevor das Log vollständig durchsucht werden kann.</p> <p>Auf Null setzen, damit DBRemoteSQL einen guten Wert basierend auf den Benutzer-Sendezeiten und anderer gesammelter Informationen wählen kann. Entspricht der dbremote-Option -ru.</p>
scan_log	a_bit_field	Reserviert für den internen Gebrauch und muss auf FALSE gesetzt werden.
send	a_bit_field	<p>Wenn auf TRUE gesetzt, werden Nachrichten gesendet.</p> <p>Wenn receive und send jeweils FALSE sind, werden beide als TRUE behandelt. Es wird empfohlen, die Einstellung receive und send von auf FALSE zu setzen. Entspricht der dbremote-Option -s.</p>
send_delay	a_sql_uint32	<p>Die Zeitdauer (in Sekunden) zwischen Analysen der Logdatei bezüglich neuer zu sendender Vorgänge.</p> <p>Auf Null setzen, wenn DBRemoteSQL einen guten Wert basierend auf Benutzer-Sendezeiten wählen soll. Entspricht der dbremote-Option -sd.</p>
set_window_title_rtn	SET_WINDOW_TITLE_CALLBACK	<p>Zeiger auf eine Funktion, die den Titel des Fensters zurücksetzt (nur Windows).</p> <p>Der Titel könnte "Datenbankname (Empfangen, Durchsuchen oder Senden) - Standardfenstertitel" sein.</p>
threads	a_sql_uint32	<p>Die Anzahl von Worker-Threads, die für das Zuweisen von Nachrichten verwendet werden sollen.</p> <p>Dieser Wert darf 50 nicht überschreiten. Entspricht der dbremote-Option -w.</p>
transaction_logs	char *	<p>Sollte das Verzeichnis der Offline-Transaktionslogs identifizieren (nur DBRemoteSQL).</p> <p>Entspricht dem Argument transaction_logs_directory von dbremote.</p>

Mitgliedsname	Typ	Beschreibung
Trigger	a_bit_field	Dies sollte normalerweise gelöscht werden (FALSE). Wenn auf TRUE gesetzt, werden Triggeraktionen repliziert. Hier ist mit Vorsicht vorzugehen.
truncate_remote_output_file	a_bit_field	Wenn auf TRUE gesetzt, wird die entfernte Ausgabe-datei gekürzt und nichts an sie angehängt. Entspricht der dbremote-Option -rt.
Nicht verwendet	a_bit_field	Reserviert für den internen Gebrauch und muss auf FALSE gesetzt werden.
use_hex_offsets	a_bit_field	Wenn auf TRUE gesetzt, werden Log-Offsets in hexadezimaler Schreibweise angezeigt. Andernfalls wird die dezimale Schreibweise verwendet.
use_relative_offsets	a_bit_field	Wenn auf TRUE gesetzt, werden Log-Offsets als relativ zum Start der aktuellen Logdatei angezeigt. Wenn auf FALSE gesetzt werden Log-Offsets vom Anfang an angezeigt.
verbose	a_bit_field	Wenn aktiviert, werden zusätzliche Informationen erzeugt. Entspricht der dbremote-Option -v.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).
warningrtn	MSG_CALLBACK	Zeiger auf eine Funktion, die die angegebene Warnmeldung anzeigt. Wenn NULL, wird stattdessen die Funktion errorrtn aufgerufen.

Bemerkungen

Das Dienstprogramm dbremote legt die folgenden Standardwerte vor der Verarbeitung von Befehlszeilenoptionen fest:

- version = DB_TOOLS_VERSION_NUMBER
- argv = (an Anwendung übergebener Argumentvektor)
- deleted = TRUE

- apply = TRUE
- more = TRUE
- link_debug = FALSE
- max_length = 50000
- memory = 2 * 1024 * 1024
- frequency = 1
- threads = 0
- receive_delay = 60
- send_delay = 0
- log_size = 0
- patience_retry = 1
- resend_urgency = 0
- log_file_name = (in Befehlszeile gesetzt)
- truncate_remote_output_file = FALSE
- remote_output_file_name = NULL
- no_user_interaction = TRUE (wenn Benutzerschnittstelle nicht verfügbar ist)
- errorrtn = (Adresse einer geeigneten Routine)
- msggrtn = (Adresse einer geeigneten Routine)
- confirgrtn = (Adresse einer geeigneten Routine)
- msgqueuerrtn = (Adresse einer geeigneten Routine)
- logrtn = (Adresse einer geeigneten Routine)
- warningrtn = (Adresse einer geeigneten Routine)
- set_window_title_rtn = (Adresse einer geeigneten Routine)
- progress_msg_rtn = (Adresse einer geeigneten Routine)
- progress_index_rtn = (Adresse einer geeigneten Routine)

Siehe auch

- [DBRemoteSQL-Methode \[Datenbanktools\] auf Seite 884](#)

a_sync_db-Struktur

Enthält Informationen, die für den Gebrauch des Dienstprogramms dbmlsync mit der DBTools-Bibliothek benötigt werden.

Syntax

```
typedef struct a_sync_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
allow_outside_connect	a_bit_field	Reserviert. Verwenden Sie 0.
allow_schema_change	a_bit_field	Auf TRUE setzen, um auf Schemaänderungen zwischen Synchronisierungen zu prüfen. Entspricht der dbmlsync-Option -sc.
apply_dnld_file	const char *	Name der anzuwendenden Download-Datei. Entspricht der dbmlsync-Option -ba oder NULL, wenn die Option nicht angegeben ist.
argv	char **	Das argv-Array für diese Ausführung. Das letzte Element des Arrays muss NULL sein.
autoclose	a_bit_field	Auf TRUE setzen, um das Fenster beim Beenden zu schließen. Entspricht der dbmlsync-Option -qc.
background_retry	a_sql_int32	Anzahl der Neuversuche für eine unterbrochene Hintergrundsynchro-nisation. Entspricht der dbmlsync-Option -bkr.
background_sync	a_bit_field	Auf TRUE setzen, um eine Hintergrundsynchro-nisation durchzuführen. Entspricht der dbmlsync-Option -bk.
cache_verbosity	a_bit_field	Reserviert. Verwenden Sie 0.
ce_argv	char **	Reserviert. Verwenden Sie NULL.

Mitgliedsname	Typ	Beschreibung
ce_reproc_argv	char **	Reserviert. Verwenden Sie NULL.
changing_pwd	a_bit_field	Auf TRUE setzen, wenn ein neues MobiLink-Kennwort festgelegt wird. Siehe new_mlpasword Feld. Entspricht der dbmsync-Option -mn.
confirmrtn	MSG_CALLBACK	Adresse einer Bestätigungsanforderungs-Callback-Routine oder NULL.
connectparms	char *	Parameter für die Verbindung zur Datenbank. Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db". Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLAny16\bin32\dbsrv16.exe". Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".
connectparms_allocated	a_bit_field	Reserviert. Verwenden Sie 0.
continue_download	a_bit_field	Auf TRUE setzen, um einen zuvor fehlgeschlagenen Download fortzusetzen. Entspricht der dbmsync-Option -dc.
create_dnlld_file	const char *	Name der zu erstellenden Download-Datei. Entspricht der dbmsync-Option -bc bzw. NULL, wenn die Option nicht angegeben ist.
debug	a_bit_field	Reserviert. Verwenden Sie 0.
debug_dump_char	a_bit_field	Reserviert. Verwenden Sie 0.
debug_dump_hex	a_bit_field	Reserviert. Verwenden Sie 0.
debug_dump_size	a_sql_uint32	Reserviert. Verwenden Sie 0.

Mitgliedsname	Typ	Beschreibung
debug_page_offsets	a_bit_field	Reserviert. Verwenden Sie 0.
default_window_title	char *	Name des Programmes, der in der Titelzeile des Fensters anzugeben ist (z.B. DBMLSync).
dl_insert_width	a_sql_uint32	Reserviert. Verwenden Sie 0.
dl_use_put	a_bit_field	Reserviert. Verwenden Sie 0.
dlg_info_msg	a_sql_uint32	Reserviert. Verwenden Sie 0.
dnld_fail_len	a_sql_uint32	Reserviert. Verwenden Sie 0.
dnld_file_extra	const char *	Eingabe einer zusätzlichen Zeichenfolge, die in die Download-Datei eingefügt werden soll. Entspricht der dbmlsync-Option -be.
dnld_gen_num	a_bit_field	Auf TRUE setzen, um die Generationsnummer bei Übernahme der Downloaddatei zu aktualisieren. Entspricht der dbmlsync-Option -bg.
dnld_read_size	a_sql_uint32	Festlegen der Größe des Lesevorgangs für den Download. Entspricht der dbmlsync Option -drs.
download_only	a_bit_field	Auf TRUE setzen, um die Synchronisation mit reinem Download durchzuführen. Entspricht der dbmlsync-Option -ds.
encrypted_stream_opts	const char *	Reserviert. Verwenden Sie NULL.
encryption_key	char *	Der Chiffrierschlüssel für die Datenbankdatei. Entspricht der dbmlsync-Option -ek.
entered_dialog	a_bit_field	Reserviert. Verwenden Sie 0.
errortn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.

Mitgliedsname	Typ	Beschreibung
est_upld_row_cnt	a_sql_uint32	Festlegen der geschätzten Upload-Zeilenzahl (für die Optimierung). Entspricht der dbmlsync-Option -urc.
extended_options	char *	Erweiterte Optionen in der Form "Schlüsselwort = Wert;...". Entspricht der dbmlsync-Option -e.
hide_conn_str	a_bit_field	FALSE, um die Verbindungszeichenfolge zu zeigen, TRUE, um die Verbindungszeichenfolge auszublenden. Entspricht der dbmlsync-Option -vc.
hide_ml_pwd	a_bit_field	FALSE, um das MobiLink-Kennwort zu zeigen, TRUE, um das MobiLink-Kennwort auszublenden. Entspricht der dbmlsync-Option -vp.
hovering_frequency	a_sql_uint32	Legt die Logscan-Abrufperiode in Sekunden fest. Gewöhnlich 60. Entspricht der dbmlsync Option -pp.
ignore_debug_interrupt	a_bit_field	Reserviert. Verwenden Sie 0.
ignore_hook_errors	a_bit_field	Auf TRUE setzen, um Fehler zu ignorieren, die in Hook-Funktionen auftreten. Entspricht der dbmlsync Option -eh.
ignore_hovering	a_bit_field	Auf TRUE setzen, um Logscan-Abruf zu deaktivieren. Entspricht der dbmlsync Option -p.
ignore_scheduling	a_bit_field	Auf TRUE setzen, um die Abfolgeplanung zu ignorieren. Entspricht der dbmlsync Option -is.
include_scan_range	const char *	Reserviert. Verwenden Sie NULL.

Mitgliedsname	Typ	Beschreibung
init_cache	a_sql_uint32	Anfängliche Cachegröße. Entspricht der dbmsync Option -ci.
init_cache_suffix	char	Suffix für anfängliche Cachegröße ('B' für Byte, 'P' für Prozentsatz oder "0", falls nicht festgelegt).
kill_other_connections	a_bit_field	Auf TRUE setzen, um Verbindungen mit Sperren auf zu synchronisierende Tabellen zu entfernen. Entspricht der dbmsync Option -d.
last_upload_def	a_syncpub *	Reserviert. Verwenden Sie NULL.
lite_blob_handling	a_bit_field	Reserviert. Verwenden Sie 0.
log_file_name	const char *	Datenbankserver-Nachrichtenlogname. Entspricht der dbmsync-Option -o oder -ot.
log_size	a_sql_uint32	Größe der Logdatei in Byte beim Umbenennen und Neustarten des Transaktionslogs. 0 für nicht festgelegte Größe angeben. Entspricht der dbmsync-Option -x.
logrtn	MSG_CALLBACK	Adresse einer Logging-Callback-Routine zum Schreiben von Nachrichten ausschließlich in die Logdatei oder NULL.
max_cache	a_sql_uint32	Maximale Cachegröße. Entspricht der dbmsync-Option -cm.
max_cache_suffix	char	Suffix für maximale Cachegröße ('B' für Byte, 'P' für Prozentsatz oder "0", falls nicht festgelegt).
min_cache	a_sql_uint32	Minimale Cachegröße. Entspricht der dbmsync-Option -cl.
min_cache_suffix	char	Suffix für minimale Cachegröße ('B' für Byte, 'P' für Prozentsatz oder "0", falls nicht festgelegt).

Mitgliedsname	Typ	Beschreibung
mlpassword	char *	Das MobiLink-Kennwort oder NULL, wenn die Option nicht angegeben ist. Entspricht der dbmlsync-Option -mp.
msgqueue rtn	MSG_QUEUE_CALLBACK	Funktion, die von "DBMLSync" aufgerufen wird, wenn in den Ruhezustand gewechselt werden soll. Durch diesen Parameter wird die Dauer der Ruhezeit in Millisekunden festgelegt. Diese Funktion sollte nachstehende Ergebnisse zurückgeben, wie es in dllapi.h definiert ist. <ul style="list-style-type: none"> • MSGQ_SLEEP_THROUGH zeigt an, dass die Routine über die angeforderte Anzahl von Millisekunden im Ruhezustand war. In der Regel ist dies der Wert, den Sie zurückgeben sollten. • MSGQ_SHUTDOWN_REQUESTED zeigt an, dass Sie die Synchronisation so schnell wie möglich beenden möchten. • MSGQ_SYNC_REQUESTED zeigt an, dass die Routine für die angeforderte Anzahl von Millisekunden im Ruhezustand war und die nächste Synchronisation unmittelbar beginnen sollte, wenn derzeit keine Synchronisation durchgeführt wird.
msg rtn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
new_mlpassword	char *	Das neue MobiLink-Kennwort oder NULL, wenn die Option nicht angegeben ist. Entspricht der dbmlsync-Option -mn.
no_offline_logscan	a_sql_uint32	Auf TRUE setzen, um Offline-Logscan zu deaktivieren (Verwendung mit -x nicht möglich). Entspricht der dbmlsync-Option -do.
no_schema_cache	a_bit_field	Reserviert. Verwenden Sie 0.
no_stream_compress	a_bit_field	Reserviert. Verwenden Sie 0.

Mitgliedsname	Typ	Beschreibung
offline_dir	const char *	Transaktionslogverzeichnis. Letztes Element in dbmlsync-Befehlszeile.
output_to_file	a_bit_field	Reserviert. Verwenden Sie 0.
output_to_mobile_link	a_bit_field	Reserviert. Verwenden Sie 1.
persist_connection	a_bit_field	Auf TRUE setzen, um die MobiLink-Verbindung zwischen Synchronisationen dauerhaft zu machen. Auf FALSE setzen, um die MobiLink-Verbindung zwischen Synchronisationen zu schließen. Entspricht der dbmlsync Option -pc{+ -}.
ping	a_bit_field	Auf TRUE setzen, um MobiLink-Server zu pinggen. Entspricht der dbmlsync-Option -pi.
preload_dlls	char *	Reserviert. Verwenden Sie NULL.
progress_index_rtn	SET_PROGRESS_CALLBACK	Funktion zum Aktualisieren des Status des Fortschrittsbalkens aufgerufen.
progress_msg_rtn	MSG_CALLBACK	Funktion zum Ändern des Textes im Statusfenster oberhalb des Fortschrittsbalkens aufgerufen.
prompt_again	a_bit_field	Reserviert. Verwenden Sie 0.
prompt_for_encrypt_key	a_bit_field	Reserviert. Verwenden Sie 0.
protocol_add_cli_bit_to_cli_both	a_bit_field	Reserviert. Verwenden Sie 0.
protocol_add_cli_bit_to_cli_max	a_bit_field	Reserviert. Verwenden Sie 0.
protocol_add_serv_bit_to_cli_both	a_bit_field	Reserviert. Verwenden Sie 0.
protocol_add_serv_bit_to_cli_max	a_bit_field	Reserviert. Verwenden Sie 0.
protocol_add_serv_bit_to_serv_both	a_bit_field	Reserviert. Verwenden Sie 0.

Mitgliedsname	Typ	Beschreibung
proto-col_add_serv_bit_to_serv_max	a_bit_field	Reserviert. Verwenden Sie 0.
raw_file	const char *	Reserviert. Verwenden Sie NULL.
rename_log	a_bit_field	Auf TRUE setzen, um das Transaktionslog umzubenennen und neu zu starten. Siehe log_size-Feld. Entspricht der dbmlsync-Option -x.
reserved	a_bit_field	Reserviert. Verwenden Sie 0.
retry_remote_ahead	a_bit_field	Auf TRUE setzen, um Neuübermittlung des Uploads mit dem Offset der entfernten Datenbank bei nicht übereinstimmendem Verarbeitungsschritt zu bewirken, wenn Offset der entfernten Datenbank größer als in der konsolidierten Datenbank ist. Entspricht der dbmlsync Option -ra.
retry_remote_behind	a_bit_field	Auf TRUE setzen, um Neuübermittlung des Uploads mit dem Offset der entfernten Datenbank bei nicht übereinstimmendem Verarbeitungsschritt anzufordern. Entspricht der dbmlsync-Option -r oder -rb.
server_mode	a_bit_field	Auf TRUE setzen, wenn Servermodus ausgeführt werden soll. Entspricht der dbmlsync-Option -sm.
server_port	a_sql_uint32	Setzen des Kommunikationsports, wenn Servermodus ausgeführt wird. Entspricht der dbmlsync-Option -ppo.
set_window_title_rtn	SET_WINDOW_TITLE_CALLBACK	Funktion zum Aufrufen der Änderung des Titels des dbmlsync-Fensters (nur Windows).
status_rtn	STATUS_CALLBACK	Reserviert. Verwenden Sie NULL.
strictly_free_memory	a_bit_field	Reserviert. Verwenden Sie 0.

Mitgliedsname	Typ	Beschreibung
strictly_ignore_trigger_ops	a_bit_field	Reserviert. Verwenden Sie 0.
sync_opt	char *	Reserviert. Verwenden Sie NULL.
sync_params	char *	Parameter für Benutzerauthentifizierung. Entspricht der dbmlsync-Option -ap.
sync_profile	char *	Auszuführendes Synchronisationsprofil. Entspricht der dbmlsync Option -sp.
trans_upload	a_bit_field	Auf TRUE setzen, um Upload jeder Datenbanktransaktion getrennt auszuführen. Entspricht der dbmlsync Option -tu.
upld_fail_len	a_sql_uint32	Reserviert. Verwenden Sie 0.
upload_defs	a_syncpub *	Verknüpfte Liste von zu synchronisierenden Publikationen/Subskriptionen.
upload_only	a_bit_field	Auf TRUE setzen, um die Synchronisation mit reinem Upload durchzuführen. Entspricht der dbmlsync Option -uo.
usage_rtn	USAGE_CALLBACK	Reserviert. Verwenden Sie NULL.
use_fixed_cache	a_bit_field	Reserviert. Verwenden Sie 0.
use_hex_offsets	a_bit_field	Reserviert. Verwenden Sie 0.
use_relative_offsets	a_bit_field	Reserviert. Verwenden Sie 0.
used_dialog_allocation	a_bit_field	Reserviert. Verwenden Sie 0.
user_name	char *	Der zu synchronisierende MobiLink-Benutzer. Entspricht der dbmlsync-Option -u.
verbose	a_bit_field	Reserviert. Verwenden Sie 0.
verbose_download	a_bit_field	Reserviert. Verwenden Sie 0.
verbose_download_data	a_bit_field	Reserviert. Verwenden Sie 0.

Mitgliedsname	Typ	Beschreibung
verbose_hook	a_bit_field	Auf TRUE setzen, um Hook-Skriptinformationen anzuzeigen. Entspricht der dbmlsync Option -vs.
verbose_minimum	a_bit_field	Auf TRUE setzen, um die Ausführlichkeitsstufe auf ein Minimum zu beschränken. Entspricht der dbmlsync Option -v.
verbose_msgid	a_bit_field	Setzen Sie dies auf TRUE, um Nachrichten-IDs anzuzeigen. Entspricht der dbmlsync-Option -vi.
verbose_option_info	a_bit_field	Auf TRUE setzen, um Befehlszeilen- und erweiterte Optionen anzuzeigen. Entspricht der dbmlsync Option -vo.
verbose_protocol	a_bit_field	Reserviert. Verwenden Sie 0.
verbose_row_cnts	a_bit_field	Auf TRUE setzen, um Upload-/Download-Zeilenzahl anzuzeigen. Entspricht der dbmlsync Option -vn.
verbose_row_data	a_bit_field	Auf TRUE setzen, um Upload-/Download-Zeilwerte anzuzeigen. Entspricht der dbmlsync Option -vr.
verbose_server	a_bit_field	Reserviert. Verwenden Sie 0.
verbose_upload	a_bit_field	Auf TRUE setzen, um Upload-Datenstrominformationen anzuzeigen. Entspricht der dbmlsync Option -vu.
verbose_upload_data	a_bit_field	Reserviert. Verwenden Sie 0.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).
warningrtn	MSG_CALLBACK	Funktion zum Anzeigen von Warnmeldungen.

Bemerkungen

Einige Mitglieder entsprechen Funktionen, die über das Dienstprogramm dbmlsync zugänglich sind. Nicht verwendeten Mitgliedern sollte abhängig vom Datentyp der Wert 0, FALSE oder NULL zugeordnet werden.

Siehe auch

- DBSynchronizeLog-Methode [Datenbanktools] auf Seite 884
- „dbmlsync-Syntax“ [MobiLink - Clientadministration]
- „DBTools-Schnittstelle für dbmlsync“ [MobiLink - Clientadministration]
- „Callback-Funktionen“ auf Seite 873
- „Verbindungsparameter“ [SQL Anywhere Server - Datenbankadministration]

a_syncpub-Struktur

Enthält Informationen, die für das Dienstprogramm dbmlsync benötigt werden.

Syntax

```
typedef struct a_syncpub
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
ext_opt	char *	Erweiterte Optionen in der Form "Schlüsselwort = Wert;...". Dies sind dieselben Optionen wie bei der dbmlsync-Option -eu.
next	struct a_syncpub *	Zeiger auf den nächsten Knoten in der Liste, NULL für den letzten Knoten
pub_name	char *	Publikationsname(n), durch Kommas getrennt (nicht mehr empfohlen). Dies ist dieselbe Zeichenfolge wie bei der dbmlsync-Option -n. Nur einer der Werte pub_name und subscription können Nicht-NULL sein.
subscription	char *	Subskriptionsname(n), durch Kommas getrennt. Dies ist dieselbe Zeichenfolge wie bei der dbmlsync-Option -s. Nur einer der Werte pub_name und subscription können Nicht-NULL sein.

Siehe auch

- [a_sync_db-Struktur \[Datenbanktools\]](#) auf Seite 917
- [DBSynchronizeLog-Methode \[Datenbanktools\]](#) auf Seite 884
- [„DBTools-Schnittstelle für dbmlsync“ \[MobiLink - Clientadministration\]](#)

a_sysinfo-Struktur

Enthält Informationen, die für den Gebrauch der Dienstprogramme dbinfo und dbunload mit der DBTools-Bibliothek benötigt werden.

Syntax

```
typedef struct a_sysinfo
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
blank_padding	a_bit_field	1, falls Auffüllen mit Leerzeichen in der Datenbank verwendet wird, andernfalls 0.
case_sensitivity	a_bit_field	1, falls die Datenbank Groß- und Kleinschreibung beachtet, andernfalls 0.
default_collation	char	Die Kollationssequenz für die Datenbank.
encryption	a_bit_field	1, falls die Datenbank verschlüsselt ist, andernfalls 0.
page_size	unsigned short	Die Seitengröße für die Datenbank.
valid_data	a_bit_field	1, um anzuzeigen, dass die anderen Bitfelder gültig sind.

Siehe auch

- [a_db_info-Struktur \[Datenbanktools\]](#) auf Seite 903
- [DBInfo-Methode \[Datenbanktools\]](#) auf Seite 881
- [DBInfoDump-Methode \[Datenbanktools\]](#) auf Seite 882
- [DBInfoFree-Methode \[Datenbanktools\]](#) auf Seite 882

a_table_info-Struktur

Enthält Informationen zu einer Tabelle, die als Teil der a_db_info-Struktur gebraucht wird.

Syntax

```
typedef struct a_table_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
index_pages	a_sql_uint32	Anzahl der Indexseiten
index_used	a_sql_uint32	Anzahl der in den Indexseiten verwendeten Byte
index_used_pct	a_sql_uint32	Verwendung des Indexbereichs als Prozentsatz
next	struct a_table_info *	Nächste Tabelle in der Liste
table_id	a_sql_uint32	ID-Nummer für diese Tabelle
Tabellenname	char *	Der Name der Tabelle
table_pages	a_sql_uint32	Anzahl der Tabellenseiten
table_used	a_sql_uint32	Anzahl der in den Tabellenseiten verwendeten Byte
table_used_pct	a_sql_uint32	Verwendung des Tabellenbereichs als Prozentsatz

Siehe auch

- [a_db_info-Struktur \[Datenbanktools\] auf Seite 903](#)
- [DBInfo-Methode \[Datenbanktools\] auf Seite 881](#)
- [DBInfoDump-Methode \[Datenbanktools\] auf Seite 882](#)
- [DBInfoFree-Methode \[Datenbanktools\] auf Seite 882](#)

a_translate_log-Struktur

Enthält Informationen, die für die Konvertierung des Transaktionslogs mit der DBTools-Bibliothek gebraucht werden.

Syntax

```
typedef struct a_translate_log
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
ansi_sql	a_bit_field	Auf TRUE setzen, um SQL-Transaktionen nach dem ANSI-Standard zu erstellen. Von der dbtran-Option -s auf TRUE gesetzt.
chronological_order	a_bit_field	Reserviert. Auf FALSE setzen.

Mitgliedsname	Typ	Beschreibung
comment_trigger_trans	a_bit_field	Auf TRUE setzen, um von Triggern ausgelöste Transaktionen als Kommentare einzubeziehen. Von der dbtran-Option -z auf TRUE gesetzt.
confirmrtn	MSG_CALL-BACK	Adresse einer Bestätigungsanforderungs-Callback-Routine oder NULL.
connectparms	const char *	Parameter für die Verbindung zur Datenbank. Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db". Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLAny16\bin32\dsrv16.exe". Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dsrv16.exe".
debug	a_bit_field	Reserviert. Auf FALSE setzen.
debug_dump_char	a_bit_field	Reserviert. Auf FALSE setzen.
debug_dump_hex	a_bit_field	Reserviert. Auf FALSE setzen.
debug_dump_size	a_sql_uint32	Reserviert. Verwenden Sie 0.
debug_page_offsets	a_bit_field	Reserviert. Auf FALSE setzen.
debug_sql_remote	a_bit_field	Reserviert. Auf FALSE setzen.
encryption_key	const char *	Der Chiffrierschlüssel für die Datenbankdatei. Entspricht der dbtran-Option -ek.
errorrtn	MSG_CALL-BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
extra_audit	a_bit_field	Reserviert. Auf FALSE setzen.
force_chaining	a_bit_field	Reserviert. Auf FALSE setzen.
force_recovery	a_bit_field	Reserviert. Auf FALSE setzen.

Mitgliedsname	Typ	Beschreibung
generate_reciprocals	a_bit_field	Reserviert. Auf FALSE setzen.
include_audit	a_bit_field	Reserviert. Auf FALSE setzen.
include_destination_sets	const char *	Reserviert. Verwenden Sie NULL.
include_publications	const char *	Reserviert. Verwenden Sie NULL.
include_scan_range	const char *	Reserviert. Verwenden Sie NULL.
include_source_sets	const char *	Reserviert. Verwenden Sie NULL.
include_subsets	a_bit_field	Reserviert. Auf FALSE setzen.
include_tables	const char *	Reserviert. Verwenden Sie NULL.
include_trigger_trans	a_bit_field	Auf TRUE setzen, um von Triggern ausgelöste Transaktionen einzubeziehen. Von den dbtran-Optionen -t, -g und -sr auf TRUE gesetzt.
leave_output_on_error	a_bit_field	Auf TRUE setzen, um die generierte SQL-Datei zu verlassen, wenn ein Logfehler erkannt wird. Von der dbtran-Option -k auf TRUE gesetzt.
logname	const char *	Der Name der Transaktionslogdatei. Wenn NULL, ist kein Log vorhanden.
logrtn	MSG_CALLBACK	Adresse einer Logging-Callback-Routine zum Schreiben von Nachrichten ausschließlich in die Logdatei oder NULL.
logs_dir	const char *	Transaktionslogverzeichnis. Entspricht der dbtran-Option -m. Der sqlname-Zeiger muss angegeben sein und connectparms muss NULL sein.
match_mode	a_bit_field	Reserviert. Auf FALSE setzen.
match_pos	const char *	Reserviert. Verwenden Sie NULL.
msg rtn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.

Mitgliedsname	Typ	Beschreibung
omit_comments	a_bit_field	Reserviert. Auf FALSE setzen.
queueparms	const char *	Reserviert. Verwenden Sie NULL.
quiet	a_bit_field	Auf TRUE setzen, um ohne Meldungs Ausgabe zu arbeiten. Von der dbtran-Option -q auf TRUE gesetzt.
recovery_bytes	a_sql_uint32	Reserviert. Verwenden Sie 0.
recovery_ops	a_sql_uint32	Reserviert. Verwenden Sie 0.
remove_rollback	a_bit_field	Auf FALSE setzen, wenn Rollback-Transaktionen in die Ausgabe einbezogen werden sollen. Von der dbtran-Option -a auf FALSE gesetzt.
replace	a_bit_field	Auf TRUE setzen, um die SQL-Datei ohne Bestätigung zu ersetzen. Von der dbtran-Option -y auf TRUE gesetzt.
repserver_users	const char *	Reserviert. Verwenden Sie NULL.
show_undo	a_bit_field	Reserviert. Auf FALSE setzen.
since_checkpoint	a_bit_field	Auf TRUE setzen, um die Ausgabe des letzten Checkpoints zu erhalten. Von der dbtran-Option -f auf TRUE gesetzt.
since_time	a_sql_uint32	Ausgabe aus dem letzten Checkpoints vor der Zeit. Die Anzahl von Minuten seit 1. Januar 0001. Entspricht der dbtran-Option -j.
sqlname	const char *	Name der SQL-Ausgabedatei. Wenn NULL, basiert der Name auf den Namen der Transaktionslogdatei. Entspricht der dbtran-Option -n.
statusrtn	MSG_CALLBACK	Adresse einer Statusmeldung-Callback-Routine oder NULL.
use_hex_offsets	a_bit_field	Reserviert. Auf FALSE setzen.
use_relative_offsets	a_bit_field	Reserviert. Auf FALSE setzen.

Mitgliedsname	Typ	Beschreibung
userlist	p_name	Eine verknüpfte Liste von Benutzernamen. Entspricht den dbtran-Optionen -u Benutzer1,... oder -x Benutzer1,... Transaktionen für aufgeführte Benutzer auswählen oder auslassen.
userlisttype	char	Auf DBTRAN_INCLUDE_ALL setzen; ausgenommen, Sie möchten eine Liste von Benutzern einbeziehen oder ausschließen. DBTRAN_INCLUDE_SOME für -u oder DBTRAN_EXCLUDE_SOME für -x.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [a_name-Struktur \[Datenbanktools\]](#) auf Seite 908
- [UserList-Enumeration \[Datenbanktools\]](#) auf Seite 892
- [DBTranslateLog-Methode \[Datenbanktools\]](#) auf Seite 886
- [„Callback-Funktionen“](#) auf Seite 873
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)

a_truncate_log-Struktur

Enthält Informationen, die für das Kürzen des Transaktionslogs mit der DBTools-Bibliothek gebraucht werden.

Syntax

```
typedef struct a_truncate_log
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
connectparms	const char *	<p>Parameter für die Verbindung zur Datenbank.</p> <p>Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLA-ny16\bin32\dbsrv16.exe".</p> <p>Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLA-ny16\bin32\dbsrv16.exe".</p>
errorrtn	MSG_CALL-BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
msgtrtn	MSG_CALL-BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
quiet	a_bit_field	<p>Auf TRUE setzen, um ohne Meldungs Ausgabe zu arbeiten.</p> <p>Von dbbackup-Option -q auf TRUE gesetzt.</p>
server_backup	a_bit_field	<p>Auf TRUE setzen, um Sicherung auf dem Server mit BACKUP DATABASE anzuzeigen.</p> <p>Von der dbbackup-Option -s auf TRUE gesetzt, wenn dbbackup -x angegeben ist.</p>
truncate_interrupted	char	Wenn nicht null, wurde die Kürzung unterbrochen.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [DBTruncateLog-Methode \[Datenbanktools\] auf Seite 886](#)
- [„Callback-Funktionen“ auf Seite 873](#)
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)

a_validate_db-Struktur

Enthält Informationen, die für Datenbankvalidierung mit der DBTools-Bibliothek gebraucht werden.

Syntax

```
typedef struct a_validate_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
connectparms	const char *	<p>Parameter für die Verbindung zur Datenbank.</p> <p>Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLA-ny16\bin32\dsrv16.exe".</p> <p>Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLA-ny16\bin32\dsrv16.exe".</p>
errorrtn	MSG_CALL-BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
index	a_bit_field	<p>Auf TRUE setzen, um Indizes zu validieren.</p> <p>Das Feld "tables" zeigt auf eine Liste von Indizes. Von der dbvalid Option -i auf TRUE gesetzt. Von der dbvalid Option -t auf FALSE gesetzt.</p>
msgrtn	MSG_CALL-BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
quiet	a_bit_field	<p>Auf TRUE setzen, um ohne Meldungs Ausgabe zu arbeiten.</p> <p>Von der dbvalid Option -q auf TRUE gesetzt.</p>
statusrtn	MSG_CALL-BACK	Adresse einer Statusmeldung-Callback-Routine oder NULL.
Tabellen	p_name	<p>Zeiger auf eine verknüpfte Liste mit Tabellennamen oder Indexnamen (wenn das Feld "index" auf TRUE gesetzt ist).</p> <p>Dies wird durch das Objektlistennamenargument von dbvalid eingestellt.</p>

Mitgliedsname	Typ	Beschreibung
type	char	Der Typ der durchzuführenden Validierung. Eine Auswahl von VALIDATE_NORMAL, VALIDATE_EXPRESS, VALIDATE_CHECKSUM, etc. Siehe Validierung-Enumeration.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [a_name-Struktur \[Datenbanktools\] auf Seite 908](#)
- [DBValidate-Methode \[Datenbanktools\] auf Seite 888](#)
- [Validation-Enumeration \[Datenbanktools\] auf Seite 892](#)
- [„Callback-Funktionen“ auf Seite 873](#)
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)

an_erase_db-Struktur

Enthält die Informationen, die gebraucht werden, um eine Datenbank mit der DBTools-Bibliothek zu löschen.

Syntax

```
typedef struct an_erase_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
confirmrtn	MSG_CALLBACK	Adresse einer Bestätigungsanforderungs-Callback-Routine oder NULL.
dbname	const char *	Datenbankdateiname.
encryption_key	const char *	Der Chiffrierschlüssel für die Datenbankdatei. Entspricht dberase-Option -ek oder -ep.
erase	a_bit_field	Ohne Bestätigung löschen (1) oder mit Bestätigung (0) Von der dberase-Option -y auf TRUE gesetzt.
errorrtn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
msgrtn	MSG_CALLBACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.

Mitgliedsname	Typ	Beschreibung
quiet	a_bit_field	Ohne die Ausgabe von Nachrichten arbeiten (1) oder Nachrichten ausgeben (0) Von der dberase-Option -q auf TRUE gesetzt.
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [DBErase-Methode \[Datenbanktools\] auf Seite 881](#)
- [„Callback-Funktionen“ auf Seite 873](#)

an_unload_db-Struktur

Enthält die Informationen, die gebraucht werden, um eine Datenbank mit der DBTools-Bibliothek zu entladen oder um eine entfernte Datenbank für SQL Remote zu extrahieren.

Syntax

```
typedef struct an_unload_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
compress_output	a_bit_field	Auf TRUE setzen, um Tabellendatendateien zu komprimieren. Von der dbunload-Option -cp auf TRUE gesetzt.
confirmrtn	MSG_CALLBACK	Adresse einer Bestätigungsanforderungs-Callback-Routine oder NULL.
connectparms	const char *	Parameter für die Verbindung zur Datenbank. Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db". Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Beispiel: "START=c:\SQLAny16\bin32\dbsrv16.exe". Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".

Mitgliedsname	Typ	Beschreibung
debug	a_bit_field	Reserviert. Auf FALSE setzen.
display_create	a_bit_field	Auf TRUE setzen, um Datenbankerstellungsbefehl (sql oder dbinit) anzuzeigen. Von dbunload-Option -cm sql oder -cm dbinit auf TRUE gesetzt.
display_create_dbinit	a_bit_field	Auf TRUE setzen, um den dbinit-Datenbankerstellungsbefehl anzuzeigen. Von der dbunload-Option -cm auf TRUE gesetzt.
encrypted_tables	a_bit_field	Auf TRUE setzen, um verschlüsselte Tabellen in neuer Datenbank zu aktivieren (mit -an oder -ar). Von der dbunload/dbxtract-Option -et auf TRUE gesetzt.
encryption_algorithm	const char *	Der Verschlüsselungsalgorithmus: "simple", "aes", "aes256", "aes_fips", "aes256_fips". Oder NULL für keinen. Von der dbunload/dbxtract-Option -ea gesetzt.
encryption_key	const char *	Der Chiffrierschlüssel für die Datenbankdatei. Von der dbunload/dbxtract-Option -ek oder -ep gesetzt.
errorrtn	MSG_CALLBACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
escape_char	char	Das Escapezeichen (in der Regel "\"). Wird verwendet, wenn escape_char_present TRUE. Von der dbunload/dbxtract-Option -p auf TRUE gesetzt.
escape_char_present	a_bit_field	Auf TRUE setzen, um anzuzeigen, dass das Escapezeichen in escape_char definiert ist. Von der dbunload/dbxtract-Option -p auf TRUE gesetzt.
exclude_foreign_keys	a_bit_field	Auf TRUE setzen, um Fremdschlüssel auszuschließen. Von der dbxtract-Option -xf auf TRUE gesetzt.
exclude_hooks	a_bit_field	Auf TRUE setzen, um Prozedureinstiege auszuschließen. Von der dbxtract-Option -xh auf TRUE gesetzt.

Mitgliedsname	Typ	Beschreibung
exclude_procedures	a_bit_field	Auf TRUE setzen, um gespeicherte Prozeduren auszuschließen. Von der dbxtract-Option -xp auf TRUE gesetzt.
exclude_tables	a_bit_field	Auf FALSE setzen, um anzuzeigen, dass die Liste Tabellen enthält, die einbezogen werden müssen. Auf FALSE setzen, um anzuzeigen, dass die Liste Tabellen enthält, die ausgeschlossen werden müssen. Von der dbunload-Option -e auf TRUE gesetzt.
exclude_triggers	a_bit_field	Auf TRUE setzen, um Trigger auszuschließen. Von der dbxtract-Option -xt auf TRUE gesetzt.
exclude_views	a_bit_field	Auf TRUE setzen, um Ansichten auszuschließen. Von der dbxtract-Option -xv auf TRUE gesetzt.
extract	a_bit_field	Auf TRUE setzen, wenn eine entfernte Datenbankextraktion ausgeführt wird. Von dbunload auf FALSE gesetzt. Von dbxtract auf TRUE gesetzt.
genscript	a_bit_field	Reserviert. Auf FALSE setzen.
include_where_subscribe	a_bit_field	Auf TRUE setzen, um vollqualifizierte Publikationen zu extrahieren. Von der dbxtract-Option -f auf TRUE gesetzt.
isolation_level	unsigned short	Die Isolationsstufe, mit der gearbeitet werden soll. Von dbxtract-Option -l festgelegt.
isolation_set	a_bit_field	Auf TRUE setzen, um anzuzeigen, dass isolation_level für alle Extraktionen eingerichtet wurde. Von dbxtract-Option -l auf TRUE gesetzt.
locale	const char *	Reserviert. Verwenden Sie NULL.
make_auxiliary	a_bit_field	Auf TRUE setzen, um Hilfskatalog zu erstellen (für Diagnoseprotokollierung). Von der dbunload-Option -k auf TRUE gesetzt.

Mitgliedsname	Typ	Beschreibung
ms_filename	const char *	Reserviert. Verwenden Sie NULL.
ms_reserve	int	Reserviert. Verwenden Sie 0.
ms_size	int	Reserviert. Verwenden Sie 0.
msgsrtn	MSG_CALL- BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
no_confirm	a_bit_field	Auf TRUE setzen, um eine bestehende SQL-Skriptdatei ohne Bestätigung zu ersetzen. Von der dbunload/dbxtract-Option -y gesetzt.
no_reload_status	a_bit_field	Auf TRUE setzen, um das Neuladen von Statusmeldungen für Tabellen und Indizes zu unterdrücken. Von der dbunload-Option -qr auf TRUE gesetzt.
notemp_size	long	Reserviert. Verwenden Sie 0.
preserve_identity_values	a_bit_field	Auf TRUE setzen, um Identity-Werte für AUTOINCREMENT-Spalten zu bewahren. Von der dbunload-Option -l auf TRUE gesetzt.
preserve_ids	a_bit_field	Auf TRUE setzen, um Benutzer-IDs zu bewahren. Dies ist die Standardeinstellung. Von der dbunload-Option -e auf FALSE gesetzt.
profiling_uses_single_dbpace	a_bit_field	Auf TRUE setzen, alles in eine einzelne DBSpace-Datei zu komprimieren (für Diagnoseprotokollierung). Von der dbunload-Option -kd auf TRUE gesetzt.
recompute	a_bit_field	Auf TRUE setzen, um berechnete Spalten erneut zu berechnen. Von der dbunload-Option -dc auf TRUE gesetzt.
refresh_mat_view	a_bit_field	Auf TRUE setzen, um Anweisungen zu generieren, die eine Aktualisierung von Textindizes und gültigen materialisierten Ansichten auslösen. Von der dbunload/dbxtract-Option -g auf TRUE gesetzt.

Mitgliedsname	Typ	Beschreibung
reload_connect-parms	char *	Verbindungsparameter wie Benutzer-ID, Kennwort und Datenbank für Reload-Datenbank. Von der dbunload/dbxtract-Option -ac gesetzt.
reload_db_filename	char *	Name der neuen Datenbankdatei, die erstellt und neu geladen werden muss. Von der dbunload/dbxtract-Option -an gesetzt.
reload_db_logname	char *	Dateiname des neuen Datenbank-Transaktionslogs oder NULL. Von dbxtract-Option -al festgelegt.
reload_filename	const char *	Name, der für die Reload-SQL-Skriptdatei (z.B. reload.sql) verwendet wird. Von der dbunload-Option -r gesetzt.
reload_page_size	unsigned short	Die Seitengröße der neu geladenen Datenbank. Von der dbunload-Option -ap gesetzt.
remote_dir	const char *	Wie temp_dir, aber für internes serverseitiges Entladen.
remove_encrypted_tables	a_bit_field	Auf TRUE setzen, um Verschlüsselung aus verschlüsselten Tabellen zu entfernen. Von der dbunload/dbxtract-Option -er auf TRUE gesetzt.
replace_db	a_bit_field	Auf TRUE setzen, um die Datenbank zu ersetzen. Von der dbunload-Option -ar auf TRUE gesetzt.
runscript	a_bit_field	Reserviert. Auf FALSE setzen.
schema_reload	a_bit_field	Reserviert. Auf FALSE setzen.
site_name	const char *	der Standortname zur Verwendung durch dbxtract. Sonst NULL.
start_subscriptions	a_bit_field	Auf TRUE setzen, um Subskriptionen zu starten. Dies ist die Standardeinstellung für dbxtract. Von der dbxtract-Option -b auf FALSE gesetzt.
startline	const char *	Reserviert. Verwenden Sie NULL.

Mitgliedsname	Typ	Beschreibung
startline_name	a_bit_field	Reserviert. Auf FALSE setzen.
startline_old	const char *	Reserviert. Verwenden Sie NULL.
statusrtn	MSG_CALL-BACK	Adresse einer Statusmeldung-Callback-Routine oder NULL.
subscriber_username	const char *	Der Subskribentename zur Verwendung durch dbxtract. Sonst NULL.
suppress_statistics	a_bit_field	Auf TRUE setzen, um die Aufnahme von Spaltenstatistiken zu unterdrücken. Von der dbunload-Option -ss auf TRUE gesetzt.
sysinfo	a_sysinfo	Reserviert. Verwenden Sie NULL.
table_list	p_name	Selektive Tabellenliste. Von den dbunload-Optionen -e und -t gesetzt.
table_list_provided	a_bit_field	Auf TRUE setzen, um anzuzeigen, dass eine Liste der Tabellen angegeben wurde. Siehe table_list-Feld. Von den dbunload-Option -e oder -t auf TRUE gesetzt.
temp_dir	const char *	Verzeichnis zum Entladen der Datendateien
template_name	const char *	Der Vorlagenname zur Verwendung durch dbxtract. Sonst NULL.
unload_interrupted	char	Reserviert. Auf 0 gesetzt.
unload_type	char	Unload-Enumeration festlegen (UNLOAD_ALL usw.). Von den dbunload/dbxtract-Optionen -d, -k, -n gesetzt.
unordered	a_bit_field	Für ungeordnete Daten auf TRUE gesetzt. Indizes werden für das Entladen der Daten nicht verwendet. Von der dbunload/dbxtract-Option -u gesetzt.

Mitgliedsname	Typ	Beschreibung
use_internal_reload	a_bit_field	Auf TRUE setzen, um ein internes Neuladen durchzuführen. Dies ist die Standardeinstellung. Von den dbunload/dbxtract-Optionen -ii und -xi auf TRUE gesetzt. Von den dbunload/dbxtract-Optionen auf -ix und -xx gesetzt.
use_internal_unload	a_bit_field	Auf TRUE setzen, um ein internes Entladen durchzuführen. Von der dbunload/dbxtract-Option -i? auf TRUE gesetzt. Von der dbunload/dbxtract-Option -x? auf FALSE gesetzt.
verbose	char	Siehe Verbosity-Enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Bemerkungen

Die Felder, die vom Dienstprogramm dbxtract von SQL Remote verwendet werden, sind gekennzeichnet.

Siehe auch

- [Verbosity-Enumeration \[Datenbanktools\]](#) auf Seite 893
- [DBUnload-Methode \[Datenbanktools\]](#) auf Seite 887
- [„Callback-Funktionen“](#) auf Seite 873
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)

an_upgrade_db-Struktur

Enthält die Informationen, die gebraucht werden, um eine Datenbank mit der DBTools-Bibliothek zu aktualisieren.

Syntax

```
typedef struct an_upgrade_db
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
connectparms	const char *	<p>Parameter für die Verbindung zur Datenbank.</p> <p>Sie haben die Form von Verbindungszeichenfolgen wie: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>Der Datenbankserver würde durch den START-Parameter der Verbindungszeichenfolge gestartet. Zum Beispiel: "START=c:\SQLAny16\bin32\dbsrv16.exe".</p> <p>Eine vollständige Verbindungszeichenfolge mit dem START-Parameter würde wie folgt aussehen: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny16\bin32\dbsrv16.exe".</p>
errorrtn	MSG_CALL-BACK	Adresse einer Fehlermeldungs-Callback-Routine oder NULL.
jconnect	a_bit_field	<p>Auf TRUE setzen, um ein Upgrade der Datenbank durchzuführen, damit sie jConnect-Prozeduren akzeptiert.</p> <p>Von der dbupgrad-Option -i auf FALSE gesetzt.</p>
msgrtn	MSG_CALL-BACK	Adresse einer Informationsnachricht-Callback-Routine oder NULL.
quiet	a_bit_field	<p>Auf TRUE setzen, um ohne Meldungs Ausgabe zu arbeiten.</p> <p>Von der dbupgrad-Option -q auf TRUE gesetzt.</p>
restart	a_bit_field	<p>Setzen Sie dies auf TRUE, um die Datenbank nach dem Upgrade neu zu starten.</p> <p>Durch die dbupgrad-Option -nrs auf FALSE gesetzt.</p>
statusrtn	MSG_CALL-BACK	Adresse einer Statusmeldung-Callback-Routine oder NULL.

Mitgliedsname	Typ	Beschreibung
sys_proc_definer	unsigned short	<p>Ordnen Sie den Wert 0 zu, um ein Upgrade der Datenbank durchzuführen, damit das SQL SECURITY-Modell vor Version 16 für veraltete gespeicherte Systemprozeduren vorhanden ist, wenn Sie ein Upgrade aus Versionen vor 16.0 durchführen.</p> <p>Wenn Sie ein Upgrade einer Datenbank ab Version 16.0 durchführen, behalten Sie das aktuelle SQL SECURITY-Modell bei. (Dies entspricht dem Nicht-Angeben von -pd.).</p> <p>Ordnen Sie den Wert 1 zu, um ein Upgrade der Datenbank durchzuführen, damit das SQL SECURITY-Modell vor Version 16 für veraltete gespeicherte Systemprozeduren vorhanden ist. (Dies entspricht -pd y.)</p> <p>Ordnen Sie den Wert 2 zu, um ein Upgrade der Datenbank durchzuführen, damit das SQL SECURITY-Modell vor Version 16 für veraltete gespeicherte Systemprozeduren vorhanden ist. (Dies entspricht -pd n.)</p>
version	unsigned short	DBTools-Versionsnummer (DB_TOOLS_VERSION_NUMBER).

Siehe auch

- [DBUpgrade-Methode \[Datenbanktools\] auf Seite 887](#)
- [„Callback-Funktionen“ auf Seite 873](#)
- [„Verbindungsparameter“ \[SQL Anywhere Server - Datenbankadministration\]](#)

Exit-Codes der Softwarekomponenten

Alle Eintrittspunkte der DBTools-Bibliothek verwenden die folgenden Exit-Codes. Die SQL Anywhere-Dienstprogramme (dbbackup, dbspawn, dbsrv16 usw.) verwenden ebenfalls diese Exit-Codes.

Code	Status	Erklärung
0	EXIT_OKAY	Erfolg
1	EXIT_FAIL	Allgemeiner Fehler
2	EXIT_BAD_DATA	Ungültiges Dateiformat
3	EXIT_FILE_ERROR	Datei nicht gefunden, kann nicht geöffnet werden
4	EXIT_OUT_OF_MEMORY	Kein Speicher mehr

Code	Status	Erklärung
5	EXIT_BREAK	Beendet vom Benutzer
6	EXIT_COMMUNICATIONS_FAIL	Fehlgeschlagene Kommunikationen
7	EXIT_MISSING_DATABASE	Erforderlicher Datenbankname fehlt
8	EXIT_PROTOCOL_MISMATCH	Falsches Client/Server-Protokoll
9	EXIT_UNABLE_TO_CONNECT	Keine Verbindung mit dem Datenbankserver möglich
10	EXIT_ENGINE_NOT_RUNNING	Datenbankserver läuft nicht
11	EXIT_SERVER_NOT_FOUND	Datenbankserver nicht gefunden
12	EXIT_BAD_ENCRYPT_KEY	Fehlender oder falscher Chiffrierschlüssel
13	EXIT_DB_VER_NEWER	Zum Ausführen der Datenbank ist Server-Upgrade erforderlich
14	EXIT_FILE_INVALID_DB	Datei ist keine Datenbank
15	EXIT_LOG_FILE_ERROR	Logdatei fehlte oder anderer Fehler
16	EXIT_FILE_IN_USE	Datei wird benutzt
17	EXIT_FATAL_ERROR	Schwerwiegender Fehler aufgetreten
18	EXIT_MISSING_LICENSE_FILE	Fehlende Server-Lizenzdatei
19	EXIT_BACKGROUND_SYNC_ABORTED	Hintergrundsynchronisation abgebrochen, um Vorgänge mit höherer Priorität durchzuführen
20	EXIT_FILE_ACCESS_DENIED	Datenbank kann nicht gestartet werden, weil der Zugriff verweigert wird
255	EXIT_USAGE	Ungültige Parameter in der Befehlszeile
21	EXIT_SERVER_NAME_IN_USE	Ein anderer Server mit demselben Namen läuft bereits.

Diese Exit-Codes sind in der Datei `%SQLANY16%\sdk\include\sqldef.h` enthalten.

Deployment von Datenbanken und Anwendungen

Wenn Sie eine Datenbankanwendung fertig gestellt haben, müssen Sie ein Deployment der Anwendung für Ihre Endbenutzer vornehmen. Je nach der Art, wie Ihre Anwendung SQL Anywhere verwendet (als eingebettete Datenbank, im Client-/Servermodus usw.), müssen Sie die Komponenten von SQL Anywhere in das Deployment Ihrer Anwendung einbeziehen. Es kann außerdem erforderlich sein, ein Deployment der Konfigurationsdaten vorzunehmen, wie etwa Datenquellennamen, damit die Anwendung mit SQL Anywhere kommunizieren kann.

Hinweis

Die Weitergabe von Dateien unterliegt Ihrer Lizenzvereinbarung mit Sybase. In diesem Kapitel enthaltene Ausführungen können die Bestimmungen Ihrer Lizenzvereinbarung weder aufheben noch ändern. Bevor Sie daher ein Deployment vornehmen, prüfen Sie bitte Ihre Lizenzvereinbarung.

Folgende Bereiche des Deployments werden in diesem Abschnitt besprochen:

- Ermittlung der erforderlichen Dateien je nach Anwendungsplattform und Plattformarchitektur
- Konfiguration der Clientanwendungen

Eine großer Teil des Abschnitts befasst sich mit einzelnen Dateien und wo sie platziert werden müssen. Es wird jedoch empfohlen, SQL Anywhere-Komponenten unter Verwendung des **Deployment-Assistenten** oder dialogfrei zu installieren. Weitere Hinweise finden Sie unter [„Deployment-Assistent“ auf Seite 952](#) und [„Dialogfreie Installationen mit dem SQL Anywhere-Installationsprogramm“ auf Seite 956](#).

Deployment-Typen

Welche Dateien Sie in das Deployment einbeziehen müssen, hängt vom gewählten Deployment-Typ ab. Nachstehend werden einige denkbare Modelle beschrieben:

- **Deployment von Clients** Sie können nur die Clientkomponenten von SQL Anywhere in das Deployment für die Endbenutzer aufnehmen, sodass diese sich mit einem zentral untergebrachten Netzwerk-Datenbankserver verbinden können.
- **Deployment des Netzwerkservers** Sie können Netzwerkservers in Zweigstellen einrichten und dann das Deployment der Clients für alle Benutzer in diesen Büros vornehmen.
- **Deployment eingebetteter Datenbanken** Sie können das Deployment einer Datenbank vornehmen, die mit einem Personal Datenbankserver läuft. In diesem Fall müssen der Client und der Personal Server auf dem Computer des Endbenutzers installiert werden.
- **Deployment über SQL Remote** Das Deployment einer SQL Remote-Anwendung ist eine Erweiterung des Deployment-Modells einer eingebetteten Datenbank.

- **MobiLink-Deployment** Hinweise zum Deployment von MobiLink-Servern finden Sie unter „[Deployment von MobiLink-Anwendungen](#)“ [[MobiLink - Serveradministration](#)].
- **Deployment von Administrationstools** Sie können ein Deployment von Interactive SQL, Sybase Central und anderen Tools zur Verwaltung vornehmen.

Möglichkeiten zur Weitergabe von Dateien

Es gibt zwei Möglichkeiten, SQL Anywhere bereitzustellen:

- **SQL Anywhere-Installer verwenden** Sie können den Installer den Endbenutzern zur Verfügung stellen. Wenn der Endbenutzer die richtigen Installationsoptionen wählt, erhält er garantiert alle erforderlichen Dateien.

Dies ist die einfachste Lösung für viele Fälle des Deployments. In diesem Fall müssen Sie den Endbenutzern nur noch eine Methode zur Verbindungsaufnahme mit dem Datenbankserver bereitstellen (z.B. eine ODBC-Datenquelle).

Weitere Hinweise finden Sie unter „[Deployment-Assistent](#)“ auf [Seite 952](#) oder „[Dialogfreie Installationen mit dem SQL Anywhere-Installationsprogramm](#)“ auf [Seite 956](#).

- **Entwicklung Ihrer eigenen Installation** Es kann Gründe dafür geben, dass Sie ein eigenes Installationsprogramm entwickeln wollen, das Dateien von SQL Anywhere enthält. Diese Option ist viel komplizierter, und daher richtet sich der größte Teil dieses Abschnitts an jene, die eigene Installationen entwickeln müssen.

Wenn SQL Anywhere bereits passend für den Servertyp und das Betriebssystem der Clientanwendung installiert wurde, stehen die erforderlichen Dateien in dem entsprechend benannten Unterverzeichnis bereit, das im Installationsverzeichnis von SQL Anywhere angelegt wurde. Das Unterverzeichnis *bin32* des Installationsverzeichnisses enthält z.B. die Dateien, die erforderlich sind, den Server auf 32-Bit-Windows-Betriebssystemen auszuführen.

Gleichgültig welche Option Sie wählen: Sie müssen sich dabei immer an die Bestimmungen Ihrer Lizenz halten.

Installationsverzeichnisse und Dateinamen

Damit eine Anwendung nach dem Deployment richtig funktioniert, müssen die Clientanwendungen ermitteln können, wo die erforderlichen Dateien untergebracht sind. Die beim Deployment installierten Dateien müssen in derselben Struktur gespeichert werden wie in der Installation von SQL Anywhere.

In der Praxis bedeutet das, dass unter Windows die meisten Programmdateien in ein einziges Verzeichnis gehören. Unter Windows werden Dateien für den Datenbankserver und für den Client z.B. in einem einzigen Verzeichnis installiert, nämlich dem Unterverzeichnis *bin32* des Installationsverzeichnisses von SQL Anywhere.

Eine vollständige Beschreibung der Standorte, an denen die Software nach Dateien sucht, finden Sie unter „[Wie SQL Anywhere die Dateien findet](#)“ [[SQL Anywhere Server - Datenbankadministration](#)].

Linux-, Unix- und Mac OS X-Deploymentmethoden

Das Deployment von Anwendungen unter Unix unterscheidet sich von der unter Windows üblichen in folgender Weise:

- **Verzeichnisstruktur** Bei Linux-, Unix- und Mac OS X-Installationen lautet die Verzeichnisstruktur standardmäßig wie folgt:

Verzeichnis	Inhalt
<i>/opt/sqlanywhere16/bin32 und /opt/sqlanywhere16/bin64</i>	Programmdateien, Lizenzdateien
<i>/opt/sqlanywhere16/lib32 und /opt/sqlanywhere16/lib64</i>	Shared Objects und Bibliotheken
<i>/opt/sqlanywhere16/res</i>	Textdateien

Unter AIX lautet das Stammverzeichnis standardmäßig */usr/lpp/sqlanywhere16* und nicht */opt/sqlanywhere16*.

Unter Mac OS X lautet das Stammverzeichnis standardmäßig */Applications/SQLAnywhere16/System* und nicht */opt/sqlanywhere16*.

Je nach der Komplexität Ihres Deployments können Sie alle für Ihre Anwendung benötigten Dateien in einem einzigen Verzeichnis zusammenfassen. Es handelt sich hierbei um eine einfachere Deployment-Option, vor allem, wenn für Ihr Deployment nur eine geringe Anzahl von Dateien erforderlich ist. Dieses Verzeichnis kann dasselbe Verzeichnis sein, das Sie auch für Ihre eigene Anwendung verwenden.

- **Dateisuffixe** In den Tabellen dieses Abschnitts werden die Shared Objects mit dem Suffix *.so* oder *.so.1* aufgelistet. Die Versionsnummer 1 kann bei späteren Versionen höher sein. Zur Vereinfachung der Darstellung wird die Versionsnummer häufig nicht angegeben.

Unter AIX umfasst das Suffix keine Versionsnummer, sodass es einfach *.so* lautet.

- **Symbolische Verknüpfungen** Jedes Shared Object ist als symbolische Verknüpfung (symlink) zu einer Datei gleichen Namens mit dem zusätzlichen Suffix *.1* (eins) installiert. Die Datei *libdblib16.so* ist beispielsweise eine Symbolverknüpfung zur Datei *libdblib16.so.1* im selben Verzeichnis.

Das Versionssuffix *.1* kann bei späteren Versionen höher sein und die Symbolverknüpfung muss entsprechend umgeleitet werden.

Unter Mac OS X sollten Sie eine symbolische jnilib-Verknüpfung für jede dylib erstellen, die Sie direkt von Ihrer Java-Clientanwendung laden wollen.

- **Anwendungen mit und ohne Threading** Die meisten Shared Objects werden in zwei Formen geliefert, von denen eine mit den zusätzlichen Zeichen *_r* vor dem Dateisuffix versehen ist. Zusätzlich zu *libdblib16.so.1* gibt es beispielsweise eine Datei namens *libdblib16_r.so.1*. In diesem Fall müssen

Anwendungen mit Threading mit dem Shared Object verknüpft werden, dessen Name das Suffix *_r* besitzt, während Anwendungen ohne Threading mit dem Shared Object verknüpft werden müssen, dessen Name kein Suffix *_r* hat. Gelegentlich wird eine dritte Form eines Shared Objects mit *_n* vor dem Dateisuffix verwendet. Dies ist eine Version eines Shared Objects, die mit Anwendungen ohne Threading verwendet wird.

- **Zeichensatzkonvertierung** Um die Zeichensatzkonvertierung für Datenbankserver zu verwenden, müssen Sie die folgenden Dateien einbeziehen:
 - *libdbicu16.so.1*
 - *libdbicu16_r.so.1*
 - *libdbicudt16.so.1*
 - *sqlany.cvf*
- **Umgebungsvariablen** Unter Linux/Unix und Mac OS müssen Umgebungsvariablen für das System so eingerichtet werden, dass SQL Anywhere-Anwendungen und -Bibliotheken gefunden werden. Es wird empfohlen, dass Sie die für Ihre Shell geeignete Datei, d.h. also *sa_config.sh* oder *sa_config.csh* (in den Verzeichnissen */opt/sqlanywhere16/bin32* und */opt/sqlanywhere16/bin64*), als Vorlage zum Einstellen der erforderlichen Umgebungsvariablen verwenden. Die Umgebungsvariablen, die von diesen Dateien eingerichtet werden, sind z.B. PATH, LD_LIBRARY_PATH und SQLANY16.

Eine Beschreibung der Art und Weise, wie SQL Anywhere Dateien findet, finden Sie unter „Wie SQL Anywhere die Dateien findet“ [[SQL Anywhere Server - Datenbankadministration](#)].

Namenskonventionen für Dateien

SQL Anywhere benutzt einheitliche Konventionen, damit Sie die Systemkomponenten leichter erkennen und zu Gruppen zusammenfassen können.

Diese Konventionen sind wie folgt aufgebaut:

- **Versionsnummer** Die SQL Anywhere-Versionsnummer ist im Dateinamen der Hauptserverkomponenten angegeben (Programmdateien, DLL-Dateien, Shared Objects, Lizenzdateien usw.).

Die Datei *dbsrv16.exe* ist beispielsweise eine Programmdatei der Version 16 für Windows.

- **Sprache** Die in einer Sprachen-Ressourcenbibliothek verwendete Sprache wird durch einen Sprachcode im Dateinamen angezeigt. Die zwei Zeichen vor der Versionsnummer weisen auf die in der Bibliothek verwendete Sprache hin. *dblg16.dll* ist beispielsweise die Nachrichten-Ressourcenbibliothek für die Sprache Englisch. Diese Codes aus zwei Buchstaben sind im ISO-Standard 639-1 festgelegt.

Weitere Hinweise zur Sprachenkennzeichnung finden Sie unter „Dienstprogramm für die Sprachauswahl (dblang)“ [[SQL Anywhere Server - Datenbankadministration](#)].

Eine Liste der in SQL Anywhere verfügbaren Sprachen finden Sie unter „Lokalisierte Versionen von SQL Anywhere“ [[SQL Anywhere Server - Datenbankadministration](#)].

Andere Dateitypen erkennen

Die folgende Tabelle zeigt die Plattform und die Funktion der SQL Anywhere-Dateien entsprechend ihrer Dateierweiterung. In SQL Anywhere wurden, wo immer möglich, die Standard-Dateierweiterungen verwendet.

Dateierweiterung	Plattform	Dateityp
<i>.bat, .cmd</i>	Windows	Batch-Befehlsdateien
<i>.chm, .chw</i>	Windows	Datei des Hilfesystems
<i>.dll</i>	Windows	Dynamische Verknüpfungsbibliothek (Dynamic Link Library)
<i>.exe</i>	Windows	Programmdatei
<i>.ini</i>	Alle	Initialisierungsdatei
<i>.lic</i>	Alle	Lizenzdatei
<i>.lib</i>	Ändert sich je nach Entwicklungstool	Statische Laufzeitbibliotheken für die Erstellung von Embedded SQL-Programmdateien
<i>.res</i>	Linux, Unix, Mac OS X	Sprachen-Ressourcendatei für Umgebungen außer Windows
<i>.so</i>	Linux, Unix	Shared Object- oder Shared Library-Dateien (das Äquivalent einer Windows-DLL)
<i>.bundle, .dylib</i>	Mac OS X	Shared Object-Datei (das Äquivalent einer Windows-DLL)

Datenbankdateinamen

SQL Anywhere-Datenbanken bestehen aus zwei Elementen:

- **Datenbankdatei** Sie wird verwendet, um Informationen in organisierter Form zu speichern. Dieser Datei ist standardmäßig die Dateierweiterung *.db* zugeordnet. Es können auch zusätzliche DBSpace-Dateien vorhanden sein. Diese Dateien können eine beliebige Dateierweiterung haben oder auch gar keine.
- **Transaktionslogdatei** Sie wird benutzt, um alle Änderungen aufzuzeichnen, die an den Daten in der Datenbankdatei vorgenommen werden. Für diese Datei wird standardmäßig die Dateierweiterung *.log* verwendet. Sie wird von SQL Anywhere erzeugt, wenn keine solche Datei vorhanden ist und die Verwendung einer Logdatei definiert wurde. Ein Transaktionslog-Spiegel hat die standardmäßige Erweiterung *.mlg*.

Diese Dateien werden vom relationalen Datenbank-Managementsystem von SQL Anywhere aktualisiert, gepflegt und verwaltet.

Deployment-Assistent

Der SQL Anywhere-**Deployment-Assistent** ist das bevorzugte Tool zum Erstellen von Deployments von SQL Anywhere für Windows. Der **Deployment-Assistent** kann Installationsdateien erstellen, die einige oder alle der folgenden Komponenten umfassen:

- Clientschnittstellen, wie ODBC
- SQL Anywhere-Server, einschließlich FernDatenzugriff, Datenbanktools und Verschlüsselung
- Relationale UltraLite-Datenbank
- MobiLink-Server, Client, und Verschlüsselung
- Administrationstools, wie Interactive SQL und Sybase Central

Sie können mit dem **Deployment-Assistenten** sowohl eine Microsoft Windows Installer-Paketdatei als auch eine Microsoft Windows Installer Merge Modul-Datei erstellen:

- **Microsoft Windows Installer-Paketdatei** Eine Datei mit den Anweisungen und den erforderlichen Daten für die Installation einer Anwendung. Eine Installer-Paketdatei hat die Erweiterung *.msi*.
- **Microsoft Windows Installer Merge Module-Datei** Ein vereinfachter Typ einer Microsoft Installer-Paketdatei, die alle Dateien, Ressourcen, Registrierungseinträge und die Setup-Logik für die Installation einer gemeinsam genutzten Komponente enthält. Ein Merge Modul hat die Erweiterung *.msm*.

Ein Merge Modul kann nicht alleine installiert werden, da ihm einige erforderliche Datenbanktabellen fehlen, die in einer Installer-Paketdatei enthalten sind. Merge-Module können zusätzliche Tabellen enthalten, die nur von ihm verwendet werden. Um die von einem Merge Modul bereitgestellten Informationen für eine Anwendung zu installieren, muss das Modul zuerst in der Installer-Paketdatei der Anwendung zusammengeführt werden (*.msi*). Ein Merge Modul besteht aus den folgenden Teilen:

- Eine Merge Modul-Datenbank, die die Installationseigenschaften und die vom Merge Modul bereitgestellte Setup-Logik enthält.
- Merge Modul-Zusammenfassungsdaten (Summary Information Stream), die das Modul beschreiben.
- Eine *MergeModule.CAB*-Cabinet-Datei, die als Datenstrom im Merge Modul gespeichert ist. Diese Cabinet-Datei enthält alle Dateien, die von den vom Merge Modul bereitgestellten Komponenten erforderlich sind. Jede vom Merge Modul bereitgestellte Datei muss in einer Cabinet-Datei gespeichert werden, die als Datenstrom im strukturierten Speicher des Merge Moduls eingebettet sind. In einem Standard-Merge-Modul ist der Name dieser Cabinet-Datei immer *MergeModule.CAB*.

Mit dem **Deployment-Assistenten** können Sie Teilmengen der in SQL Anywhere enthaltenen Komponenten auswählen. Jede Komponente hat Abhängigkeiten zu anderen Komponenten, daher können Dateien, die vom Assistenten ausgewählt werden, Dateien aus anderen Kategorien einbeziehen.

Um zu ermitteln, welche Dateien in den einzelnen auswählbaren Komponenten enthalten sind, erstellen Sie ein MSI-Installationsprogramm-Image und wählen Sie alle Komponenten aus. Eine Logdatei wird erstellt, die angibt, welche Dateien in der jeweiligen Komponente enthalten sind. Diese Textdatei kann mit einem Texteditor überprüft werden. Es gibt Überschriften wie **Feature: SERVER32_TOOLS** und **Feature: CLIENT64_TOOLS**, die den **Deployment-Assistent**-Komponenten entsprechen. Wenn Sie sich die Datei ansehen, erhalten Sie eine Vorstellung davon, was in den einzelnen Gruppen enthalten ist.

Um den **Deployment-Assistenten** auszuführen, klicken Sie auf **Start » Programme » SQL Anywhere 16 » Administrationstools » Deployment auf Windows**.

Notieren Sie sich den Produktcode, der benutzt wurde, um das Deployment-Paket zu erstellen. Sie können den Produktcode zu einem späteren Zeitpunkt verwenden, um das Paket zu deinstallieren.

Hinweis

Die Weitergabe von Dateien wird durch die Lizenzvereinbarung geregelt. Sie müssen bestätigen, dass Sie über die erforderliche Lizenz zur Weitergabe von SQL Anywhere-Dateien verfügen. Überprüfen Sie Ihre Lizenzvereinbarung, bevor Sie fortfahren.

Siehe auch

- „Ausführen des Deployment-Assistenten“ auf Seite 953
- „Installationsprogramm für Deployment-Pakete“ auf Seite 954
- „Dialogfreie Installationen mit dem SQL Anywhere-Installationsprogramm“ auf Seite 956

Ausführen des Deployment-Assistenten

Der **Deployment-Assistent** von SQL Anywhere wird verwendet, um das Deployment von Datenbanken und Anwendungen für Windows vorzunehmen.

Voraussetzungen

Windows-Betriebssystem

Kontext und Bemerkungen

Deployment von Datenbanken und Anwendungen

Aufgabe

1. Um den **Deployment-Assistenten** auszuführen, klicken Sie auf **Start » Programme » SQL Anywhere 16 » Administrationstools » Deployment auf Windows**.
2. Befolgen Sie die Anweisungen des Assistenten.

3. Notieren Sie sich den Produktcode, der benutzt wurde, um das Deployment-Paket zu erstellen. Ein Beispiel für einen Produktcode ist {7C99D24E-AE1B-4770-9015-65B805950E3D}. Sie können den Produktcode zu einem späteren Zeitpunkt verwenden, um das Paket zu deinstallieren.

Ergebnisse

Der **Zielpfad**, den sie ausgewählt haben, enthält das Deployment-Paket (z.B. *sqlany 16.msi*) und eine Logdatei (*sqlany 16.msi.log*).

Nächste Schritte

Nachdem Sie ein Deployment-Paket erstellt haben, müssen Sie es testen, indem Sie es installieren.

Beispiel

Sie können auch den **Deployment-Assistenten** aus dem Unterverzeichnis *Deployment* Ihrer SQL Anywhere-Installation ausführen. Beispiel:

```
%SQLANY16%\Deployment\DeploymentWizard.exe
```

Siehe auch

- „Deployment-Assistent“ auf Seite 952
- „Installationsprogramm für Deployment-Pakete“ auf Seite 954

Installationsprogramm für Deployment-Pakete

Installiert oder deinstalliert Deployments unter Windows.

Syntax

```
msiexec [ options ] [ SQLANYDIR=path ]
```

Option	Beschreibung
/log <i>Logdateiname</i>	Dieser Parameter weist den Microsoft Windows Installer an, die Vorgänge in einer Datei (z.B. <i>sqlany 16.log</i>) zu protokollieren. Die Logdatei ist bei der Ermittlung von Problemen sinnvoll.
/package <i>Paketname</i>	Dieser Parameter weist den Microsoft Windows Installer an, das angegebene Paket zu installieren (in diesem Fall <i>sqlany.msi</i>).
/qn	Dieser Parameter weist den Microsoft Windows Installer an, die Deinstallation im Hintergrund und ohne Benutzereingabe auszuführen.

Option	Beschreibung
<code>/uninstall Paketname Produktcode</code>	Dieser Parameter weist den Microsoft Windows Installer an, das Produkt, dem die angegebene MSI-Datei oder der angegebene Produktcode zugeordnet ist, zu deinstallieren. Der <i>Produktcode</i> wird erstellt, wenn der Deployment-Assistent ausgeführt wird, um das Deployment-Paket zu erstellen.
<code>SQLANYDIR=Pfad</code>	Der Wert dieses Parameters legt den Pfad des Verzeichnisses fest, in dem die Software installiert werden soll. Diese Option ist nützlich für die dialogfreie Installation. Es wird ignoriert für /uninstall .

Bemerkungen

Die Option `/qn` kann verwendet werden, um den Vorgang ohne Meldungen für den Benutzer durchzuführen. Unter Windows Vista und höheren Versionen von Windows sind zum Installieren oder Deinstallieren von Software Administratorrechte erforderlich. Wenn die Option `/qn` verwendet wird, wird die automatische Anforderung der Administratorrechte unterdrückt.

Siehe auch

- „Deployment-Assistent“ auf Seite 952
- „Ausführen des Deployment-Assistenten“ auf Seite 953

Beispiel

Mit dem folgenden Befehl wird eine Deploymentdatei installiert.

```
msiexec /package sqlany16.msi
```

Der folgende Befehl führt eine dialogfreie Installation der Deploymentdatei im angegebenen Ordner durch und protokolliert die Ergebnisse in einer Datei. Seien Sie mit der Verwendung der Option `/qn` sollten Sie vorsichtig, da das Computersystem ohne Vorankündigung heruntergefahren und neu gestartet werden kann.

```
msiexec /qn /package sqlany16.msi /log sqlany16.log SQLANYDIR=c:\sa16
```

Der folgende Befehl führt eine dialogfreie Deinstallation mit dem angegebenen Paket durch und protokolliert die Ergebnisse in einer Datei.

```
msiexec /qn /uninstall sqlany16.msi /log sqlany16.log
```

Der folgende Befehl führt eine dialogfreie Deinstallation mit dem angegebenen Produktcode durch.

```
msiexec.exe /qn /uninstall {7C99D24E-AE1B-4770-9015-65B805950E3D}
```

Dialogfreie Installationen mit dem SQL Anywhere-Installationsprogramm

Dialogfreie Installationen werden ohne Eingriff von Seiten des Benutzers ausgeführt. Der Benutzer wird nicht darüber informiert, dass eine Installation ausgeführt wird. Bei Windows-Betriebssystemen können Sie den Installer von SQL Anywhere so aufrufen, dass die SQL Anywhere-Installation dialogfrei ausgeführt wird.

Es sind folgende Optionen für das SQL Anywhere-Installationsprogramm *setup.exe* verfügbar:

- **/L:Sprach_ID** Die Sprach-ID ist eine Sprachumgebungsnummer, die die Sprache für die Installation repräsentiert. Die Sprachumgebungs-ID 1033 kennzeichnet z.B. US-Englisch, die Sprachumgebungs-ID 1031 kennzeichnet Deutsch, die Sprachumgebungs-ID 1036 kennzeichnet Französisch, die Sprachumgebungs-ID 1041 kennzeichnet Japanisch und die Sprachumgebungs-ID 2052 kennzeichnet Vereinfachtes Chinesisch.
- **/S** Diese Option blendet das Initialisierungsfenster aus. Verwenden Sie diese Option zusammen mit **/V**.
- **/V** Mit dieser Option können Sie Parameter für das Microsoft Windows Installer-Tool MSIEXEC angeben.

Im folgenden Befehlszeilen-Beispiel wird davon ausgegangen, dass sich das Verzeichnis für das Installations-Image im Verzeichnis *pkg\SQLAnywhere* auf der CD im Laufwerk *d:* befindet.

```
d:\pkg\sqlanywhere\setup.exe /l:1033 /s "/v: /qn  
REGKEY=QEDEV-B888A-6L123-45678-90123 INSTALLDIR=c:\sa16 DIR_SAMPLES=c:  
\sa16\Samples"
```

Hinweis

Die *setup.exe* im oben stehenden Befehl ist jene, die sich in demselben Verzeichnis wie die *SQLANY32.msi*- und *SQLANY64.msi*-Dateien befindet. Die *setup.exe* im übergeordneten Verzeichnis dieser Dateien unterstützt dialogfreie Installationen NICHT.

Bei der Verwendung der Option *MSIEXEC /qn* sollten Sie vorsichtig vorgehen, da das Computersystem ohne Vorankündigung herunterfahren und neu starten kann.

Die folgenden Eigenschaften beziehen sich auf das Installationsprogramm von SQL Anywhere:

- **INSTALLDIR** Der Wert dieses Parameters ist der Pfad des Verzeichnisses, in dem die Software installiert wird.
- **DIR_SAMPLES** Der Wert dieses Parameters ist der Pfad des Verzeichnisses, in dem die Beispielprogramme installiert werden.
- **DIR_SQLANY_MONITOR** Der Wert dieses Parameters ist der Pfad des Verzeichnisses, in dem die *SQL Anywhere-Monitor-Datenbank* installiert wird.
- **USERNAME** Der Wert dieses Parameters ist der Benutzername, der für diese Installation aufgezeichnet werden soll (z.B. *USERNAME="John Smith"*).

- **COMPANYNAME** Der Wert dieses Parameters ist der Firmenname, der für diese Installation aufgezeichnet werden soll (z.B. COMPANYNAME="Smith Holdings").
- **REGKEY** Der Wert dieses Parameters muss ein gültiger Softwareregistrierungsschlüssel sein.
- **REGKEY_ADD_1** Der Wert dieses Parameters muss ein gültiger Softwareregistrierungsschlüssel für eine Add-On-Funktion wie die FIPS-Verschlüsselung sein. Diese Eigenschaft gilt nur, wenn Sie optionale Add-On-Funktionen haben, die während dieser Ausführung des Installationsprogramms installiert werden sollen.
- **REGKEY_ADD_2** Der Wert dieses Parameters muss ein gültiger Softwareregistrierungsschlüssel für eine Add-On-Funktion wie die FIPS-Verschlüsselung sein. Diese Eigenschaft gilt nur, wenn Sie optionale Add-On-Funktionen haben, die während dieser Ausführung des Installationsprogramms installiert werden sollen.
- **REGKEY_ADD_3** Der Wert dieses Parameters muss ein gültiger Softwareregistrierungsschlüssel für eine Add-On-Funktion wie die FIPS-Verschlüsselung sein. Diese Eigenschaft gilt nur, wenn Sie optionale Add-On-Funktionen haben, die während dieser Ausführung des Installationsprogramms installiert werden sollen.

Das folgende Beispiel zeigt, wie die SQL Anywhere-Installationseigenschaften anzugeben sind:

```
d:\pkg\sqlanywhere\setup.exe /S "/v: /qn
USERNAME=\"John Smith\"
COMPANYNAME=\"Smith Holdings\"
REGKEY=QEDEV-B888A-6L123-45678-90123
REGKEY_ADD_1=<an add-on software registration key>
REGKEY_ADD_2=<another add-on software registration key>
INSTALLDIR=c:\sal6
DIR_SAMPLES=c:\sal6\Samples
DIR_SQLANY_MONITOR=c:\sal6\Monitor"
```

Der Text dieses Beispiels wird hier aus Platzgründen über mehrere Zeilen verteilt. Tatsächlich muss der gesamte Text in einer einzigen Zeile eingegeben werden. Beachten Sie, dass die inneren Anführungszeichen mit dem Backslash als Escapezeichen als Anführungszeichen dargestellt werden müssen.

Die folgenden Eigenschaften gelten für das Installationsprogramm für SQL Anywhere-Monitor (in d:\pkg\Monitor\setup.exe):

- **INSTALLDIR** Der Wert dieses Parameters ist der Pfad des Verzeichnisses, in dem die Software installiert wird.
- **DIR_SQLANY_MONITOR** Der Wert dieses Parameters ist der Pfad des Verzeichnisses, in dem die SQL Anywhere-Monitor-Datenbank installiert wird.
- **REGKEY** Der Wert dieses Parameters muss ein gültiger Softwareinstallationsschlüssel sein.
- **REGKEY_ADD_1** Der Wert dieses Parameters muss ein gültiger Softwareinstallationsschlüssel für eine Add-On-Funktion wie die FIPS-Verschlüsselung sein. Diese Eigenschaft gilt nur, wenn Sie optionale Add-On-Funktionen haben, die während dieser Ausführung des Installationsprogramms installiert werden sollen.

- **REGKEY_ADD_2** Der Wert dieses Parameters muss ein gültiger Softwareinstallationsschlüssel für eine Add-On-Funktion wie die FIPS-Verschlüsselung sein. Diese Eigenschaft gilt nur, wenn Sie optionale Add-On-Funktionen haben, die während dieser Ausführung des Installationsprogramms installiert werden sollen.

Das folgende Beispiel zeigt, wie die Installationseigenschaften für SQL Anywhere-Monitor anzugeben sind:

```
d:\software\monitor\setup.exe /S "/v: /qn
REGKEY=<SQLAnywhere Monitor registration key>
REGKEY_ADD_1=<an add-on software registration key>
REGKEY_ADD_2=<another add-on software registration key>
INSTALLDIR=c:\sal6"
```

Neben dem Festlegen der oben beschriebenen Eigenschaftswerte haben Sie noch die Möglichkeit, die folgenden SQL Anywhere-Komponenten oder -Funktionen über die Befehlszeile auszuwählen:

Funktion	Eigenschaft	x86-Standardwert	x64-Standardwert
Administrationstools (32-Bit)	AT32	1	0
Administrationstools (64-Bit)	AT64	0	1
CAC-Authentifizierung	CAC *		
FIPS-zertifizierte Verschlüsselung	FIPS *		
Hochverfügbarkeit	HA *		
In-Memory-Modus	IM *		
MobiLink (32-Bit)	ML32	1	0
MobiLink (64-Bit)	ML64	0	1
Scale-Out mit Schreibschutz	SON *		
Relay Server (32-Bit)	RS32	1	0
Relay Server (64-Bit)	RS64	0	1
Beispiele	SAMPLES	1	1
SQL Anywhere (32-Bit)	SA32	1	0
SQL Anywhere (64-Bit)	SA64	0	1
SQL Anywhere für Windows Mobile	MOBILE	1	1
SQL Anywhere-Monitor (32-Bit)	SM32	1	0

Funktion	Eigenschaft	x86-Standardwert	x64-Standardwert
SQL Anywhere-Monitor (64-Bit)	SM64	0	1
SQL Remote (32-Bit)	SR32	1	0
SQL Remote (64-Bit)	SR64	0	1
UltraLite	UL	1	1

* Diese Funktionen erfordern zusätzlich einen entsprechenden Softwareregistrierungsschlüssel.

Setzen Sie den Eigenschaftswert auf 1, um die Funktion auszuwählen oder auf 0, um sie wegzulassen. Um zum Beispiel MobiLink (in der 32-Bit-Version) auszulassen, setzen Sie ML32=0. Um die 32-Bit-Administrationstools zu wählen, setzen Sie AT32=1. Mithilfe dieser Eigenschaften heben Sie die Auswahl der Standardwerte auf.

Sie brauchen den Auswahlstatus der Funktion nicht anzugeben, wenn der Standardwert gelten soll. Um zum Beispiel die Standardauswahl auf einem 64-Bit-Computer aufzuheben und die 32-Bit-MobiLink-Funktion, nicht aber die Beispiele zu installieren, verwenden Sie die folgende Befehlszeile:

```
setup.exe "/v ml32=1 samples=0"
```

Bei Eigenschaftsnamen wird die Groß- und Kleinschreibung nicht berücksichtigt.

Ihr Registrierungsschlüssel schränkt möglicherweise die verfügbaren Funktionen ein. Sie können diese Einschränkungen nicht mithilfe der Befehlszeilenoptionen aufheben. Wenn Ihr Registrierungsschlüssel beispielsweise die Installation der FIPS-Verschlüsselung nicht zulässt, ist es nicht möglich, diese Funktion durch Eingabe von FIPS=1 in der Befehlszeile auszuwählen.

Um ein MSI-Log zu generieren, fügen Sie hinter /v: den folgenden Parameter hinzu.

```
/l*v! logfile
```

Im oben stehenden Beispiel ist logfile der vollständige Pfad und der Dateiname der Logdatei. Der Pfad muss bereits vorhanden sein. Diese Option generiert ein extrem ausführliches Log und führt dazu, dass die Installation wesentlich länger dauert. Weitere Hinweise zum Reduzieren der Logdatei-Ausgabe finden Sie unter <http://msdn.microsoft.com/de-de/library/aa367988.aspx>.

Neben einer dialogfreien Installation kann auch eine dialogfreie Deinstallation ausgeführt werden. Das folgende Beispiel zeigt eine entsprechende Befehlszeile.

```
msiexec.exe /qn /uninstall {3D7AB509-4FC2-4570-9913-A8F1CAD8C743} /l*v %temp%\sauninstall.log
```

Im obigen Beispiel rufen Sie das Microsoft Windows Installer-Tool direkt auf.

- **/qn** Dieser Parameter weist den Microsoft Windows Installer an, die Deinstallation im Hintergrund und ohne Benutzereingabe auszuführen.

- **/uninstall <Produktcode>** Dieser Parameter weist den Microsoft Windows Installer an, das Produkt, dem der angegebene Produktcode zugeordnet ist, zu deinstallieren. Der oben angezeigte Code gilt für die SQL Anywhere-Software.
- **/!v %temp%\sauninstall.log** Dieser Parameter weist den Microsoft Windows Installer an, eine Logdatei am angegebenen Ort zu erstellen.

Die Produktcodes für SQL Anywhere 16.0 lauten:

- **{3D7AB509-4FC2-4570-9913-A8F1CAD8C743}** SQL Anywhere-Software
- **{37B83CDB-F75E-4428-A9C2-995D92105EEC}** SQL Anywhere-Software (nur Client)
- **{1AF08ED0-804C-4AFF-BB6C-D980D81AD594}** SQL Anywhere-Monitor

Anforderungen für das Deployment von Clientanwendungen

Um das Deployment einer Clientanwendung vorzunehmen, die mit einem Netzwerk-Datenbankserver betrieben wird, müssen Sie jedem Endbenutzer folgende Elemente zur Verfügung stellen:

- **Clientanwendung** Die Anwendungssoftware selbst ist von der Datenbanksoftware unabhängig und wird hier nicht beschrieben.
- **Datenbank-Interface-Dateien** Die Clientanwendung benötigt die Dateien für die Datenbankschnittstelle, die sie benutzt (.NET, ADO, OLE DB, ODBC, JDBC, Embedded SQL oder Open Client).
- **Verbindungsinformationen** Jede Clientanwendung benötigt Verbindungsinformationen für die Datenbank.

Die erforderlichen Interface-Dateien und Verbindungsinformationen richten sich nach der Schnittstelle, die von Ihrer Anwendung benutzt wird. Jede Schnittstelle wird in den folgenden Abschnitten im Einzelnen beschrieben.

Die einfachste Möglichkeit des Deployments von Clients ist die Verwendung des **Deployment-Assistenten**. Weitere Hinweise finden Sie unter „[Deployment-Assistent](#)“ auf Seite 952.

Deployment eines .NET Clients

Die einfachste Möglichkeit, .NET-Assemblies bereitzustellen, ist die Verwendung des **Deployment-Assistenten**. Weitere Hinweise finden Sie unter „[Deployment-Assistent](#)“ auf Seite 952.

Um die Installation der .NET-Assemblies auf dem Clientsystem abzuschließen, müssen Sie folgendermaßen vorgehen.

- Stellen Sie sicher, dass Visual Studio nicht läuft.

- Verwenden Sie das SetupVSPackage-Tool zur Installation der .NET-Assemblys. SetupVSPackage erfordert Administratorrechte für Windows Vista und höher. Wenn Sie eine Eingabeaufforderung verwenden, vergewissern Sie sich, dass die Administratorrechte vorhanden sind.
- Führen Sie für .NET 2.0/3.x folgenden Befehl aus: `%SQLANY16%\Assembly\v2\SetupVSPackage.exe /install`.
- Führen Sie für .NET 4.x folgenden Befehl aus: `%SQLANY16%\Assembly\v4\SetupVSPackage.exe /install`.
- Standardmäßig verwendet SetupVSPackage SQL Anywhere-Registrierungseinstellungen, um den Speicherort der .NET-Assemblys zu ermitteln. Sie können den Speicherort der SQL Anywhere-Installation mit der **salocation**-Option angeben.

`%SQLANY16%\Assembly\v2\SetupVSPackage.exe /install /salocation %SQLANY16%`

Die Kurzform für **salocation** ist **sal**.

Die SetupVSPackage-Anwendung aktualisiert den globalen Assembly-Cache und die Windows-Microsoft.NET-Datei *machine.config*. Wenn SQL Server 2008 oder später auf dem System installiert ist, installiert SetupVSPackage auch zwei Zuordnungsdateien namens *MSSqlToSA.xml* und *SAToMSSql10.xml* im SQL Server-Ordner *DTS\MappingFiles*.

Die SetupVSPackage-Anwendung kopiert die Datei *SSDLToSA16.tt* in das Visual Studio 2010-Verzeichnis. Sie wird zum Generieren einer Datenbankschema-DDL für Entity Data Models verwendet. Beim Generieren der Datenbankschema-DLL für Entity Data Models muss der Benutzer die DDL-Generierungseigenschaft auf diese Datei setzen.

In diesem Abschnitt werden die Dateien beschrieben, die Sie für die Endbenutzer bereitstellen müssen, wenn Sie Ihre eigene Installation erstellen. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).

Jeder .NET-Clientcomputer muss folgende Elemente aufweisen:

- **Eine betriebsbereite Installation von .NET 2.0, 3.0, 3.5, 4.0 oder 4.5** Microsoft .NET-Assemblys und Anweisungen für ihre Verteilung werden von der Microsoft Corporation bereitgestellt. Sie werden hier nicht im Einzelnen beschrieben.
- **SQL Anywhere-Provider für .NET Framework 2.0/3.0** Die SQL Anywhere-Installation speichert die Windows-Assemblys für die .NET Framework-Versionen 2.0 und 3.0 im Unterverzeichnis *Assembly\v2* des SQL Anywhere-Installationsverzeichnisses. Die anderen Dateien werden im Verzeichnis der Betriebssystem-Binärdateien für das SQL Anywhere-Installationsverzeichnis gespeichert (z.B. *bin32* und *bin64*). Die folgenden Dateien sind erforderlich.

`iAnywhere.Data.SQLAnywhere.dll`
`policy.16.0.iAnywhere.Data.SQLAnywhere.dll`
`dblg[LL]16.dll`
`dbicu16.dll`
`dbicudt16.dll`

- **SQL Anywhere-Provider für .NET Framework 3.5** Die SQL Anywhere-Installation speichert die Windows-Assemblys für die .NET Framework-Version 3.5 im Unterverzeichnis *Assembly\V3.5* des SQL Anywhere-Installationsverzeichnis. Die anderen Dateien werden im Verzeichnis der Betriebssystem-Binärdateien für das SQL Anywhere-Installationsverzeichnis gespeichert (z.B. *bin32* und *bin64*). Die folgenden Dateien sind erforderlich.

```
iAnywhere.Data.SQLAnywhere.v3.5.dll  
policy.16.0.iAnywhere.Data.SQLAnywhere.v3.5.dll  
dblg[LL]16.dll  
dbicul6.dll  
dbicudt16.dll
```

- **SQL Anywhere-Provider für .NET Framework 4.0/4.5** Die SQL Anywhere-Installation speichert die Windows-Assemblys für die .NET Framework-Versionen 4.0 und 4.5 im Unterverzeichnis *Assembly\V4* des SQL Anywhere-Installationsverzeichnis. Die anderen Dateien werden im Verzeichnis der Betriebssystem-Binärdateien für das SQL Anywhere-Installationsverzeichnis gespeichert (z.B. *bin32* und *bin64*). Die folgenden Dateien sind erforderlich.

```
iAnywhere.Data.SQLAnywhere.v4.0.dll  
policy.16.0.iAnywhere.Data.SQLAnywhere.v4.0.dll  
dblg[LL]16.dll  
dbicul6.dll  
dbicudt16.dll
```

- **SQL Anywhere-Provider für .NET Compact Framework 2.0** Die SQL Anywhere-Installation platziert die Windows Mobile-Assemblys für das .NET Compact Framework Version 2.0 in *CE\Assembly\V2*. Die anderen Dateien werden im Unterverzeichnis der Windows Mobile-Binärdateien für das SQL Anywhere-Installationsverzeichnis gespeichert (z.B. *CE\Arm.50*). Die folgenden Dateien sind erforderlich.

```
iAnywhere.Data.SQLAnywhere.dll  
iAnywhere.Data.SQLAnywhere.gac  
dblg[LL]16.dll  
dbicul6.dll
```

- **SQL Anywhere-Provider für .NET Compact Framework 3.5** Die SQL Anywhere-Installation platziert die Windows Mobile-Assemblys für das .NET Compact Framework Version 3.5 in *CE\Assembly\V3.5*. Die anderen Dateien werden im Unterverzeichnis der Windows Mobile-Binärdateien für das SQL Anywhere-Installationsverzeichnis gespeichert (z.B. *CE\Arm.50*). Die folgenden Dateien sind erforderlich.

```
iAnywhere.Data.SQLAnywhere.v3.5.dll  
dblg[LL]16.dll  
dbicul6.dll
```

Weitere Hinweise zum Deployment des SQL Anywhere .NET-Datenproviders finden Sie unter „Deployment von SQL Anywhere .NET-Datenprovider“ auf Seite 72 und „ClickOnce- und .NET-Datenprovider-DLLs für nicht verwalteten Code“ auf Seite 963.

ClickOnce- und .NET-Datenprovider-DLLs für nicht verwalteten Code

Mit dem ClickOnce-Deployment können Sie .NET-Anwendungen auf einem Webserver oder einem Netzlaufwerk publizieren, um die Installation zu vereinfachen. Microsoft Visual Studio unterstützt das Publizieren und Aktualisieren von mit der ClickOnce-Technologie bereitgestellten Anwendungen.

Zur Unterstützung des Deployments für die nicht verwalteten Codeteile des SQL Anywhere .NET-Datenproviders wird eine Beispiel-Deployment-Anwendung bereitgestellt, die veranschaulicht, wie Sie DLLs als Ressourcen zur bereitzustellenden .NET-Anwendung hinzufügen können. Der Quellcode für diese Beispielanwendung befindet sich im Ordner `%SQLANY16%\SQLAnywhere\ADO.NET\DeployUtility`.

- **Build.cmd** Diese Batchdatei zeigt, wie Sie die nicht verwalteten DLLs für den SQL Anywhere .NET-Datenprovider als Ressourcen zur Programmdatei Ihrer Anwendung hinzufügen. Insbesondere wird gezeigt, wie Sie die 32-Bit-DLLs, `dblg16.dll`, `dbicu16.dll` und `dbicudt16.dll`, als Ressourcen zu einer .NET-Anwendung hinzufügen. Diese Methode kann auf die 64-Bit-Versionen der DLLs erweitert werden.
- **Program.cs** Dieses Beispielprogramm zeigt, wie Sie auf die DLLs zugreifen, die als Ressourcen zu Ihrer Anwendung hinzugefügt wurden, und sie in ein Verzeichnis schreiben, in dem Sie vom SQL Anywhere .NET-Datenprovider verwendet werden können. Diese Methode kann optimiert werden, um 32-Bit- und 64-Bit-DLLs zu verarbeiten.

SQL Anywhere .NET-Datenprovider entfernen

Wenn Sie die aktuelle Version des SQL Anywhere .NET-Datenproviders deinstallieren möchten, müssen Sie folgendermaßen vorgehen.

- Stellen Sie sicher, dass Visual Studio nicht läuft.
- Verwenden Sie das SetupVSPackage-Tool zur Deinstallation der .NET-Assemblys. SetupVSPackage erfordert Administratorrechte für Windows Vista und höher. Wenn Sie eine Eingabeaufforderung verwenden, vergewissern Sie sich, dass die Administratorrechte vorhanden sind.
- Führen Sie für .NET 2.0/3.x folgenden Befehl aus: `%SQLANY16%\Assembly\v2\SetupVSPackage.exe /uninstall`.
- Führen Sie für .NET 4.x folgenden Befehl aus: `%SQLANY16%\Assembly\v4\SetupVSPackage.exe /uninstall`.

Die Kurzform für `/uninstall` ist `/u`.

Wenn alle Builds von Version 16.0 des SQL Anywhere .NET-Datenproviders deinstallieren möchten, müssen Sie folgendermaßen vorgehen.

- Stellen Sie sicher, dass Visual Studio nicht läuft.

- Verwenden Sie das SetupVSPackage-Tool zur Deinstallation aller Builds der .NET-Assemblies. SetupVSPackage erfordert Administratorrechte für Windows Vista und höher. Wenn Sie eine Eingabeaufforderung verwenden, vergewissern Sie sich, dass die Administratorrechte vorhanden sind.
- Führen Sie für .NET 2.0/3.x folgenden Befehl aus: `%SQLANY16%\Assembly\v2\SetupVSPackage.exe /uninstallall`.
- Führen Sie für .NET 4.x folgenden Befehl aus: `%SQLANY16%\Assembly\v4\SetupVSPackage.exe /uninstallall`.

Die Kurzform für `/uninstallall` ist `/ua`.

Deployment von OLE DB- und ADO-Client

Die einfachste Möglichkeit, OLE DB-Clientbibliotheken bereitzustellen, ist die Verwendung des **Deployment-Assistenten**.

In diesem Abschnitt werden die Dateien beschrieben, die Sie für die Endbenutzer bereitstellen müssen, wenn Sie Ihre eigene Installation erstellen.

Jeder OLE DB-Clientcomputer muss folgende Elemente aufweisen:

- **Eine funktionierende OLE DB-Installation** OLE DB-Dateien und Anweisungen für ihre Verteilung können bei der Microsoft Corporation bezogen werden. Sie werden hier nicht im Einzelnen beschrieben.
- **SQL Anywhere OLE DB-Provider** Die folgende Tabelle enthält die Dateien, die für den SQL Anywhere OLE DB-Provider erforderlich sind. Diese Dateien sollten in nur einem Verzeichnis abgelegt werden. Die SQL Anywhere-Installation platziert sie alle in das Betriebssystem-Unterverzeichnis des SQL Anywhere-Installationsverzeichnisses (zum Beispiel `bin32` oder `bin64`). Für Windows gibt es zwei Provider-DLLs. Die zweite DLL (`dboledba16.dll`) ist eine unterstützende DLL, die Schemaunterstützung bereitstellt.

Beschreibung	Windows
OLE DB-Treiberdatei	<code>dboledb16.dll</code>
OLE DB-Treiberdatei	<code>dboledba16.dll</code>
Sprachen-Ressourcenbibliothek	<code>dblg[LL]16.dll</code>
Fenster "Verbinden"	<code>dbcon16.dll</code>
Agent für Vorgänge mit erweiterten Berechtigungen	<code>dbelevate16.exe</code> (nur Windows Vista oder später)

Diese Tabelle zeigt eine Datei mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren,

müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie [LL] durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).

OLE DB-Provider benötigen viele Registrierungseinträge. Sie können diese erstellen, indem Sie die DLLs *dboledb16.dll* und *dboledb16.dll* sich über das Dienstprogramm regsvr32 selbst registrieren lassen.

Bei Windows Vista oder späteren Versionen von Windows müssen Sie den SQL Anywhere-Agenten für Vorgänge mit erweiterten Berechtigungen einbeziehen, der die erforderlichen Privilegien für die Registrierung bzw. Deregistrierung von DLLs unterstützt. Diese Datei ist nur als Teil der Installations- bzw. Deinstallationsprozedur für den OLE DB-Provider erforderlich.

Bei Windows-Clients wird empfohlen, Microsoft MDAC 2.7 oder höher zu verwenden.

Anpassung des OLE DB-Providers

Wenn der OLE DB-Provider installiert wird, muss die Windows-Registrierung geändert werden. Hierzu wird gewöhnlich die in den OLE DB-Provider integrierte Selbstregistrierungsfunktion verwendet. Sie können z.B. das Windows-Tool regsvr32 verwenden. Eine Standardgruppe von Registrierungseinträgen wird vom Provider erstellt.

In einer typischen Verbindungszeichenfolge ist eine der Komponenten das Providerattribut. Um anzuzeigen, dass der SQL Anywhere OLE DB-Provider verwendet werden soll, geben Sie den Namen des Providers an. Im Folgenden sehen Sie ein Visual Basic-Beispiel:

```
connectString = "Provider=SAOLEDB;DSN=SQL Anywhere 16 Demo"
```

In ADO bzw. OLE DB gibt es zahlreiche weitere Wege, den Provider namentlich zu referenzieren. Im folgenden C++-Beispiel wird neben dem Providernamen auch die zu verwendende Version angegeben.

```
hr = db.Open(_T("SAOLEDB.16"), &dbinit);
```

Der Providernamen wird in der Registrierung gesucht. Wenn Sie die Registrierung auf Ihrem Computersystem untersuchen, finden Sie den Eintrag in HKEY_CLASSES_ROOT für SAOLEDB.

```
[HKEY_CLASSES_ROOT\SAOLEDB]
@="SQL Anywhere OLE DB Provider"
```

Er hat zwei Sub-Schlüssel, die einen Klassenbezeichner (ClsId) und die aktuelle Version (CurVer) des Providers enthalten. Hier sehen Sie ein Beispiel.

```
[HKEY_CLASSES_ROOT\SAOLEDB\ClsId]
@="{41dfe9f7-db91-11d2-8c43-006008d26a6f}"

[HKEY_CLASSES_ROOT\SAOLEDB\CurVer]
@="SAOLEDB.16"
```

Es sind mehrere ähnliche Einträge vorhanden. Sie werden verwendet, um eine spezifische Instanz eines OLE DB-Providers zu identifizieren. Wenn Sie Clsid in der Registrierung unter HKEY_CLASSES_ROOT\CLSID suchen und die Sub-Schlüssel überprüfen, stellen Sie fest, dass einer der Einträge den Speicherort der Provider-DLL angibt.

```
[HKEY_CLASSES_ROOT\CLSID\
{41dfe9f3-db91-11d2-8c43-006008d26a6f}\
```

```
InprocServer32]

@="c:\\sa16\\bin64\\dboledb16.dll"
"ThreadingModel"="Both"
```

Hier liegt das Problem vor, dass die Struktur sehr monolithisch ist. Würden Sie die SQL Anywhere-Software von Ihrem System deinstallieren, würden die Registrierungseinträge für den OLE DB-Provider entfernt und die Provider-DLL würde von der Festplatte entfernt. Die Anwendungen, die den Provider benötigen, könnten nicht mehr ausgeführt werden.

Wenn Anwendungen verschiedener Hersteller denselben OLE DB-Provider verwenden, würde jede Installation des Providers die gemeinsamen Registrierungseinstellungen überschreiben. Die Version des Providers, den Sie für Ihre Anwendung vorgesehen haben, würde in diesem Fall von einer neueren (oder älteren!) Version des Providers ersetzt.

Dies könnte zu einer unerwünschten Instabilität führen. Um dieses Problem zu vermeiden, kann der SQL Anywhere OLE DB-Provider angepasst werden.

OLE DB-Provider anpassen

Erstellen Sie einen eindeutigen OLE DB-Provider, den Sie in das Deployment Ihrer Anwendung einbeziehen können, indem Sie eine eindeutige Gruppe von GUIDs erstellen, den Provider benennen und eindeutige DLL-Namen wählen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Kopieren Sie den folgenden Code in einen Texteditor und speichern Sie ihn als `.reg`-Datei:

```
REGEDIT4
; Special registry entries for a private OLE DB provider.
[HKEY_CLASSES_ROOT\myoledb16]
@="Custom SQL Anywhere OLE DB Provider 16.0"
[HKEY_CLASSES_ROOT\myoledb16\Clsid] @="{GUID1}"
; Datal of the following GUID must be 3 greater than the
; previous, for example, 41dfe9f3 + 3 => 41dfe9ee.
[HKEY_CLASSES_ROOT\myoledba16]
@="Custom SQL Anywhere OLE DB Provider 16.0"
[HKEY_CLASSES_ROOT\myoledba16\Clsid] @="{GUID4}"
; Current version (or version independent prog ID)
; entries (what you get when you have "SQLAny"
; instead of "SQLAny.16")
[HKEY_CLASSES_ROOT\SQLAny]
@="SQL Anywhere OLE DB Provider"
[HKEY_CLASSES_ROOT\SQLAny\Clsid]
@="{GUID1}"
[HKEY_CLASSES_ROOT\SQLAny\CurVer]
@="SQLAny.16"
[HKEY_CLASSES_ROOT\SQLAnyEnum]
@="SQL Anywhere OLE DB Provider Enumerator"
[HKEY_CLASSES_ROOT\SQLAnyEnum\Clsid]
@="{GUID2}" [HKEY_CLASSES_ROOT\SQLAnyEnum\CurVer]
@="SQLAnyEnum.16" [HKEY_CLASSES_ROOT\SQLAnyErrorLookup]
```

```

@="SQL Anywhere OLE DB Provider Extended Error Support"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\Clsid]
@="{GUID3}"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\CurVer]
@="SQLAnyErrorLookup.16"
[HKEY_CLASSES_ROOT\SQLAnyA]
@="SQL Anywhere OLE DB Provider Assist"
[HKEY_CLASSES_ROOT\SQLAnyA\Clsid]
@="{GUID4}"
[HKEY_CLASSES_ROOT\SQLAnyA\CurVer]
@="SQLAnyA.16"
; Standard entries (Provider=SQLAny.16)
[HKEY_CLASSES_ROOT\SQLAny.16]
@="Sybase SQL Anywhere OLE DB Provider 16.0"
[HKEY_CLASSES_ROOT\SQLAny.16\Clsid]
@="{GUID1}"
[HKEY_CLASSES_ROOT\SQLAnyEnum.16]
@="Sybase SQL Anywhere OLE DB Provider Enumerator 16.0"
[HKEY_CLASSES_ROOT\SQLAnyEnum.16\Clsid]
@="{GUID2}"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.16]
@="Sybase SQL Anywhere OLE DB Provider Extended Error Support 16.0"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.16\Clsid]
@="{GUID3}"
[HKEY_CLASSES_ROOT\SQLAnyA.16]
@="Sybase SQL Anywhere OLE DB Provider Assist 16.0"
[HKEY_CLASSES_ROOT\SQLAnyA.16\Clsid]
@="{GUID4}"
; SQLAny (Provider=SQLAny.16)
[HKEY_CLASSES_ROOT\CLSID\{GUID1}]
@="SQLAny.16"
"OLEDB_SERVICES"=dword:ffffffff
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors]
@="Extended Error Service"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors\{GUID3}]
@="Sybase SQL Anywhere OLE DB Provider Error Lookup"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\InprocServer32]
@="d:\myopath\bin32\myoledb16.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\OLE DB Provider]
@="Sybase SQL Anywhere OLE DB Provider 16.0"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ProgID]
@="SQLAny.16"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\VersionIndependentProgID]
@="SQLAny"
; SQLAnyErrorLookup
[HKEY_CLASSES_ROOT\CLSID\{GUID3}]
@="Sybase SQL Anywhere OLE DB Provider Error Lookup 16.0"
@="SQLAnyErrorLookup.16"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\InprocServer32]
@="d:\myopath\bin32\myoledb16.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\ProgID]
@="SQLAnyErrorLookup.16"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\VersionIndependentProgID]
@="SQLAnyErrorLookup"
; SQLAnyEnum [HKEY_CLASSES_ROOT\CLSID\{GUID2}]
@="SQLAnyEnum.16"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\InprocServer32]
@="d:\myopath\bin32\myoledb16.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\OLE DB Enumerator]
@="Sybase SQL Anywhere OLE DB Provider Enumerator"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\ProgID]

```

```
@="SQLAnyEnum.16"  
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\VersionIndependentProgID]  
@="SQLAnyEnum"  
; SQLAnyA [HKEY_CLASSES_ROOT\CLSID\{GUID4}]  
@="SQLAnyA.16"  
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\InprocServer32]  
@="d:\\mypath\\bin32\\myoledba16.dll"  
"ThreadingModel"="Both"  
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\ProgID]  
@="SQLAnyA.16"  
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\VersionIndependentProgID]  
@="SQLAnyA"
```

Bei Namen von Registrierungswerten wird die Groß- und Kleinschreibung berücksichtigt.

- 2. Verwenden Sie das Microsoft Visual Studio-Dienstprogramm uuidgen, um 4 sequenzielle UUIDs (GUIDs) zu erstellen.

Führen Sie beispielsweise den folgenden Befehl an einer Eingabeaufforderung aus:

```
uuidgen -n4 -s -x >oledbs.txt
```

Die 4 UUIDs bzw. GUIDs werden in der folgenden Sequenz zugewiesen:

- a. Die Provider-Klassen-ID (unten GUID1)
- b. Die Enum-Klassen-ID (unten GUID2)
- c. Die ErrorLookup-Klassen-ID (unten GUID3)
- d. Die Provider Assist-Klassen-ID (unten GUID4). Diese letzte GUID wird in Windows Mobile-Deployments nicht verwendet.

Es ist wichtig, dass die IDs sequenziell zugeordnet werden (was die Option -x in der Befehlszeile bewirkt).

Jede GUID sollte ähnlich wie die folgende angezeigt werden:

Name	GUID
GUID1	41dfe9f3-db92-11d2-8c43-006008d26a6f
GUID2	41dfe9f4-db92-11d2-8c43-006008d26a6f
GUID3	41dfe9f5-db92-11d2-8c43-006008d26a6f
GUID4	41dfe9f6-db92-11d2-8c43-006008d26a6f

Hinweis

Der erste Teil der GUID (z.B. 41dfe9f3), der inkrementiert wird.

- 3. Ändern Sie mit der Suchen/Ersetzen-Funktion eines Editors die Elemente GUID1, GUID2, GUID3 und GUID4 im Text in die entsprechende GUID. (Ersetzen Sie beispielsweise GUID1 durch 41dfe9f3-db92-11d2-8c43-006008d26a6f, wenn dies die GUID ist, die uuidgen für Sie generiert hat.)

4. Wählen Sie den Providernamen, den Sie in Ihrer Anwendung in Verbindungszeichenfolgen usw. verwenden (z.B. Provider=SQLAny).

Die folgenden Namen sind für SQL Anywhere reserviert und dürfen nicht als Providernamen verwendet werden:

Version 10 oder höher	Version 9 oder früher
SAOLEDB	ASAProv
SAErrorLookup	ASAEErrorLookup
SAEnum	ASAEnum
SAOLEDBA	ASAProvA

5. Ändern Sie mit der Suchen/Ersetzen-Funktion eines Editors alle Textstellen der Zeichenfolge "SQLAny" in den von Ihnen gewählten Providernamen. Dies umfasst auch alle Textstellen, bei denen "SQLAny" ein Teil einer längeren Zeichenfolge ist (z.B. SQLAnyEnum).

Angenommen Sie wählen den Providernamen "Acme". Die Namen, die im Registrierungseintrag HKEY_CLASSES_ROOT angezeigt werden, sind in der folgenden Tabelle zusammen mit den SQL Anywhere-Namen (zum Vergleich) aufgelistet.

SQL Anywhere-Provider	Ihr benutzerdefinierter Provider
SAOLEDB	Acme
SAErrorLookup	AcmeErrorLookup
SAEnum	AcmeEnum
SAOLEDBA	AcmeA

6. Erstellen Sie Kopien der SQL Anywhere-Provider-DLLs (*dboledb16.dll* und *dboledba16.dll*) unter anderen Namen.

```
copy dboledb16.dll myoledb16.dll
copy dboledba16.dll myoledba16.dll
```

Die Datei *dboledba16.dll* gibt es nicht für Windows Mobile.

Ein spezieller Registrierungsschlüssel durch das auf dem gewählten DLL-Namen basierende Skript erstellt. Der Name muss sich von den Namen der Standard-DLLs (z.B. *dboledb16.dll* oder *dboledba16.dll*) unterscheiden. Wenn Sie die Provider-DLL *myoledb16* nennen, sucht der Provider in HKEY_CLASSES_ROOT einen Registrierungseintrag mit demselben Namen. Dies gilt auch für die Provider-Schema-Assist-DLL. Wenn Sie die DLL *myoledba16* nennen, sucht der Provider in HKEY_CLASSES_ROOT einen Registrierungseintrag mit demselben Namen. Der gewählte Name muss eindeutig sein und darf von keinem anderen Benutzer verwendet werden. Hier einige Beispiele.

Gewählte DLL-Namen	Entsprechung HKEY_CLASSES_ROOT\Name
<i>myoledb16.dll</i>	HKEY_CLASSES_ROOT\myoledb16
<i>myoledba16.dll</i>	HKEY_CLASSES_ROOT\myoledba16
<i>acmeOledb.dll</i>	HKEY_CLASSES_ROOT\acmeOledb
<i>acmeOledba.dll</i>	HKEY_CLASSES_ROOT\acmeOledba
<i>SAcustom.dll</i>	HKEY_CLASSES_ROOT\SAcustom
<i>SAcustomA.dll</i>	HKEY_CLASSES_ROOT\SAcustomA

7. Ändern Sie mit der Suchen/Ersetzen-Funktion eines Editors alle Vorkommen der Zeichenfolgen *myoledb16* und *myoledba16* im Registrierungsskript in die beiden von Ihnen gewählten DLL-Namen.
8. Ändern Sie mit der Suchen/Ersetzen-Funktion eines Editors alle Textstellen der Zeichenfolge *d:\mypath\bin32* im Registrierungsskript in den Speicherort, an dem die DLLs installiert wurden. Verwenden Sie zwei Schrägstriche, wenn Sie einen einzelnen Schrägstrich angeben wollen. Dieser Schritt muss zum Zeitpunkt der Anwendungsinstallation ausgeführt werden.
9. Speichern Sie das Registrierungsskript auf der Festplatte und führen Sie es aus.
10. Probieren Sie den neuen Provider aus. Ändern Sie Ihre ADO/OLE DB-Anwendung so, dass sie den neuen Providernamen verwendet.

Ergebnisse

Ihr benutzerdefinierter OLE DB-Provider wurde konfiguriert.

Nächste Schritte

Führen Sie das Deployment des benutzerdefinierten OLE DB-Providers mit Ihrer Anwendung durch.

Deployment von ODBC-Clients

Die einfachste Möglichkeit, ODBC-Clients bereitzustellen, ist die Verwendung des **Deployment-Assistenten**.

Jeder ODBC-Clientcomputer muss folgende Elemente aufweisen:

- **ODBC-Treibermanager** Microsoft liefert einen ODBC-Treibermanager für Windows-Betriebssysteme. SQL Anywhere umfasst einen ODBC-Treibermanager für Linux, Unix und Mac OS X. Es gibt keinen ODBC-Treibermanager für Windows Mobile. ODBC-Anwendungen können ohne Treibermanager ausgeführt werden. Auf Plattformen, für die ein ODBC-Treibermanager verfügbar ist, wird dies jedoch nicht empfohlen.

- **Verbindungsinformationen** Die Clientanwendung muss Zugriff auf die Daten haben, aus denen sie die Informationen für die Verbindung mit dem Server bezieht. Diese Informationen sind normalerweise in der ODBC-Datenquelle enthalten.
- **ODBC-Treiber** Der SQL Anywhere ODBC-Treiber muss installiert sein.

Siehe auch

- „Deployment-Assistent“ auf Seite 952
- „Erforderliche Dateien für den ODBC-Treiber“ auf Seite 971

Erforderliche Dateien für den ODBC-Treiber

Die folgende Tabelle zeigt die Dateien, die für einen funktionierenden ODBC-Treiber von SQL Anywhere erforderlich sind. Diese Dateien sollten in nur einem Verzeichnis abgelegt werden. Die SQL Anywhere-Installation platziert sie alle in das Betriebssystem-Unterverzeichnis des SQL Anywhere-Installationsverzeichnisses (zum Beispiel *bin32* oder *bin64*).

Die ODBC-Treiberversion mit mehreren Threads für Linux-, Unix- und Mac OS X-Plattformen ist mit "MT" gekennzeichnet.

Plattform	Erforderliche Dateien
Windows	<i>dbodbc16.dll</i> <i>dbcon16.dll</i> <i>sacshelp16.chm</i> (optional) <i>dbicu16.dll</i> <i>dbicudt16.dll</i> <i>dblg[LL]16.dll</i> <i>dbelevate16.exe</i>
Windows Mobile	<i>dbodbc16.dll</i> <i>dbicu16.dll</i> (optional) <i>dbicudt16.dat</i> (optional) <i>dblg[LL]16.dll</i>

Plattform	Erforderliche Dateien
Linux, Solaris, HP-UX	<i>libdbodbc16.so.1</i> <i>libdbodbc16_n.so.1</i> <i>libdbodm16.so.1</i> <i>libdbtasks16.so.1</i> <i>libdbicu16.so.1</i> <i>libdbicudt16.so.1</i> <i>dblg[LL]16.res</i>
Linux, Solaris, HP-UX MT	<i>libdbodbc16.so.1</i> <i>libdbodbc16_r.so.1</i> <i>libdbodm16.so.1</i> <i>libdbtasks16_r.so.1</i> <i>libdbicu16_r.so.1</i> <i>libdbicudt16.so.1</i> <i>dblg[LL]16.res</i>
AIX	<i>libdbodbc16.so</i> <i>libdbodbc16_n.so</i> <i>libdbodm16.so</i> <i>libdbtasks16.so</i> <i>libdbicu16.so</i> <i>libdbicudt16.so</i> <i>dblg[LL]16.res</i>

Plattform	Erforderliche Dateien
AIX MT	<i>libdbodbc16.so</i> <i>libdbodbc16_r.so</i> <i>libdbodm16.so</i> <i>libdbtasks16_r.so</i> <i>libdbicu16_r.so</i> <i>libdbicudt16.so</i> <i>dblg[LL]16.res</i>
Mac OS X	<i>dbodbc16.bundle</i> <i>libdbodbc16.dylib</i> <i>libdbodbc16_n.dylib</i> <i>libdbodm16.dylib</i> <i>libdbtasks16.dylib</i> <i>libdbicu16.dylib</i> <i>libdbicudt16.dylib</i> <i>dblg[LL]16.res</i>
Mac OS X MT	<i>dbodbc16_r.bundle</i> <i>libdbodbc16.dylib</i> <i>libdbodbc16_r.dylib</i> <i>libdbodm16.dylib</i> <i>libdbtasks16_r.dylib</i> <i>libdbicu16_r.dylib</i> <i>libdbicudt16.dylib</i> <i>dblg[LL]16.res</i>

Hinweise

- Bei Linux- und Solaris-Plattformen sollten Sie eine Verknüpfung zu den *.so.l*-Dateien erstellen. Der Name der Verknüpfung muss mit dem Dateinamen übereinstimmen, wobei das Versionssuffix ".1" entfernt werden muss.
- Es gibt ODBC-Treiberversionen mit mehreren Threads (MT) für Linux-, Unix- und Mac OS X-Plattformen. Diese Dateinamen enthalten das Suffix "_r". Stellen Sie diese Dateien bereit, wenn die Anwendung sie benötigt.
- Bei Windows ist ein Treibermanager im Betriebssystem enthalten. Für Linux, Unix und Mac OS X stellt SQL Anywhere einen Treibermanager bereit. Der Dateiname beginnt mit *libdbodm16*.
- Für Windows kann die Hilfedatei *sacshelp16.chm* installiert werden, um die lokale kontextsensitive Hilfe bei Verwendung des Microsoft ODBC-Datenquellenadministrators bereitzustellen. Wenn die Hilfedatei nicht gefunden wurde, wird versucht, über den Standard-Internetbrowser des Systems eine Verbindung mit DocCommentXchange herzustellen.
- Bei Windows Vista oder späteren Versionen von Windows müssen Sie den SQL Anywhere-Agenten für Vorgänge mit erweiterten Berechtigungen einbeziehen (*dbelevate16.exe*), der die erforderlichen Privilegien für die Registrierung bzw. Deregistrierung des ODBC-Treibers unterstützt. Diese Datei ist nur als Teil der Installations- bzw. Deinstallationsprozedur für den ODBC-Provider erforderlich.
- Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung [LL]. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie [LL] durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).
- Unter Windows wird der Unterstützungscode für die Fenster **ODBC-Konfiguration für SQL Anywhere** und **Mit SQL Anywhere verbinden** (*dbcon16.dll*) benötigt, wenn Ihre Endbenutzer eigene Datenquellen erstellen oder beim Herstellen der Verbindung mit der Datenbank Benutzer-IDs und Kennwörter eingeben müssen sowie wenn das Fenster "Verbinden" aus anderen Gründen angezeigt werden muss.

ODBC-Treiberkonfiguration

Das Installationsprogramm muss nicht nur die Dateien des ODBC-Treibers auf die Festplatte kopieren, sondern auch eine Reihe von Einträgen in die Registrierung schreiben, damit der ODBC-Treiber richtig installiert wird.

Windows

Das SQL Anywhere-Installationsprogramm führt Änderungen in der Windows-Systemregistrierung durch, um den ODBC-Treiber anzumelden und zu konfigurieren. Wenn Sie ein Installationsprogramm für die Endbenutzer erstellen, müssen Sie dieselben Registrierungseinstellungen vornehmen.

Dies können Sie am einfachsten mithilfe der Selbstregistrierungsfunktion des ODBC-Treibers. Unter Windows verwenden Sie das Dienstprogramm *regsrv32*. Bei 64-Bit-Versionen von Windows können Sie sowohl die 64-Bit-Version als auch die 32-Bit-Version des ODBC-Treibers registrieren. Durch die

Verwendung der Selbstregistrierungsfunktion des ODBC-Treibers ist gewährleistet, dass die richtigen Registrierungseinträge erstellt werden. Der ODBC-Treiber für Windows Mobile erfordert keine Registrierung.

Geben Sie den 32-Bit- und 64-Bit-Versionen des SQL Anywhere-ODBC-Treibers benutzerdefinierte Namen. Dies erleichtert die Installation und die Registrierung von mehreren unabhängigen Kopien des SQL Anywhere-ODBC-Treibers mit regsvr32, und verhindert, dass die Registrierungseinstellungen überschrieben werden, wenn eine andere Anwendungsinstallation den SQL Anywhere-ODBC-Treiber registriert.

Um die Namen Ihrer 32-Bit- und 64-Bit-Versionen des ODBC-Treibers anzupassen, öffnen Sie eine Eingabeaufforderung und benennen Sie sie wie folgt um. Dabei ist *Benutzerdefinierter_Name* eine aussagekräftige Zeichenfolge, z.B. Ihr Firmenname:

```
ren "%SQLANY16%\bin32\dbodbc16.dll" dbodbc16custom-name.dll
ren "%SQLANY16%\bin64\dbodbc16.dll" dbodbc16custom-name.dll
```

Achten Sie darauf, beim Umbenennen der Dateien das Präfix dbodbc16 beizubehalten.

Um die 32-Bit- und 64-Bit-Versionen des ODBC-Treibers mit den benutzerdefinierten Namen zu registrieren, öffnen Sie eine Eingabeaufforderung und geben Sie die folgenden Befehle ein:

```
regsvr32 "%SQLANY16%\bin32\dbodbc16custom-name.dll"
regsvr32 "%SQLANY16%\bin64\dbodbc16custom-name.dll"
```

Sie können die vom ODBC-Treiber erstellten Registrierungseinträge mithilfe des Dienstprogramms regedit überprüfen.

Der ODBC-Treiber von SQL Anywhere wird im System durch eine Gruppe von Registrierungseinträgen im folgenden Registrierungsschlüssel angemeldet:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
    ODBC\
      ODBCINST.INI\
        SQL Anywhere 16
```

Im Folgenden sind Beispielwerte für 32-Bit-Windows zu sehen:

Name des Wertes	Typ des Wertes	Daten des Wertes
Driver	Zeichenfolge	<i>C:\Program Files\SQL Anywhere 16\bin32\dbodbc16Benutzerdefinierter_Name.dll</i>
Setup	Zeichenfolge	<i>C:\Program Files\SQL Anywhere 16\bin32\dbodbc16Benutzerdefinierter_Name.dll</i>

Es gibt auch einen Registrierungseintrag im folgenden Schlüssel:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
```

ODBC\
ODBCINST.INI\
ODBC Drivers

Der Wert ist wie folgt:

Name des Wertes	Typ des Wertes	Daten des Wertes
SQL Anywhere 16 - <i>Benutzerdefinierter_Name</i>	Zeichenfolge	Installiert

Windows 64-Bit

Bei Windows 64-Bit werden die Registrierungseinträge des 32-Bit-ODBC-Treibers ("SQL Anywhere 16 - *Benutzerdefinierter_Name*" und "ODBC Drivers") unter dem folgenden Schlüssel platziert:

HKEY_LOCAL_MACHINE\
SOFTWARE\
Wow6432Node\
ODBC\
ODBCINST.INI

Um diese Einträge anzuzeigen, müssen Sie eine 64-Bit-Version von regedit verwenden. Wenn Sie Wow6432Node unter 64-Bit-Windows nicht finden, verwenden Sie aktuell die 32-Bit-Version von regedit.

ODBC-Treiber von Drittanbietern

Wenn Sie einen ODBC-Treiber eines Drittanbieters auf einem anderen Betriebssystem als Windows verwenden, finden Sie in der Dokumentation dieses ODBC-Treibers Hinweise dazu, wie der ODBC-Treiber zu konfigurieren ist.

Deployment von Verbindungsinformationen

Das Deployment der Verbindungsinformationen für den ODBC-Client wird im Allgemeinen in Form einer ODBC-Datenquelle vorgenommen. Das Deployment einer ODBC-Datenquelle erfolgt auf folgende Weise:

- **Programmgesteuert** Fügen Sie die Beschreibung für eine Datenquelle in die Registrierung des Endbenutzers oder in die ODBC-Initialisierungsdateien ein.
- **Manuell** Stellen Sie Anweisungen für die Endbenutzer bereit, damit sie auf ihrem Computer eine geeignete Datenquelle einrichten können.

Unter Windows können Sie auf der Registerkarte "Benutzer-DSN" oder "System-DSN" eine Datenquelle manuell mit dem ODBC-Datenquellen-Administrator erstellen. Der ODBC-Treiber von SQL Anywhere zeigt das Konfigurationsfenster für die Eingabe der Einstellungen an. Einstellungen für die Datenquelle enthalten den Speicherort der Datenbankdatei, den Namen des Datenbankservers sowie Startparameter und andere Optionen. Die kontextsensitive Hilfe kann über die Schaltfläche **Hilfe** geöffnet werden. Über die Datei *sacshelp16.chm* wird Hilfetext lokal bereitgestellt. Wenn diese Datei nicht vorhanden ist, wird die Hilfe über den Standard-Internetbrowser des Systems von DocCommentXchange abgerufen.

Auf Unix-Plattformen können Sie eine Datenquelle manuell unter Verwendung des SQL Anywhere-Dienstprogramms dbdsn erstellen. Einstellungen für die Datenquelle enthalten den Speicherort der Datenbankdatei, den Namen des Datenbankservers sowie Startparameter und andere Optionen.

In diesem Abschnitt finden Sie die Informationen, die Sie für beide Ansätze benötigen.

Arten von Datenquellen (Windows)

Es gibt drei Arten von Datenquellen: Benutzerdatenquellen, Systemdatenquellen und Dateidatenquellen.

1. Die Definitionen der Benutzerdatenquellen werden in der Registrierung im Abschnitt für den aktuell im System angemeldeten Benutzer gespeichert.
2. Systemdatenquellen hingegen stehen allen Benutzern und den Diensten von Windows zur Verfügung, wobei die Dienste auch aktiv sind, wenn kein Benutzer angemeldet ist. Bei einer richtig konfigurierten Systemdatenquelle namens "MeineAnwendung" kann jeder Benutzer diese ODBC-Datenquelle verwenden, indem er in der ODBC-Verbindungszeichenfolge "DSN=MeineAnwendung" eingibt.
3. Dateidatenquellen werden nicht in der Registrierung, sondern auf der Festplatte gespeichert. Eine Verbindungszeichenfolge muss einen FileDSN-Verbindungsparameter liefern, um eine Dateidatenquelle benutzen zu können. Der Standardspeicherort für Dateidatenquellen wird durch den Registrierungseintrag **HKEY_CURRENT_USER\Software\ODBC\odbc.ini\ODBC File DSN\DefaultDSNDir** angegeben (oder **HKEY_LOCAL_MACHINE\Software\ODBC\odbc.ini\ODBC File DSN\DefaultDSNDir**, wenn ersterer nicht definiert ist). Der Pfad kann in den FileDSN-Verbindungsparameter aufgenommen werden, um das Auffinden der Dateidatenquelle zu unterstützen, wenn sie sich an anderer Stelle befindet.

Registrierungseinträge für die Datenquelle (Windows)

Definitionen von Benutzer- und Systemdatenquellen werden in der Windows-Registrierung gespeichert. Die einfachste Methode, um sicherzustellen, dass die Registrierungseinträge für Datenquellendefinitionen richtig erstellt werden, besteht darin, sie mithilfe des Dienstprogramms dbdsn zu erstellen. Andernfalls müssen Sie eine Gruppe von Registrierungswerten in einem bestimmten Registrierungsschlüssel angeben.

Für Benutzerdatenquellen lautet der Registrierungsschlüssel wie folgt:

```
HKEY_CURRENT_USER\  
  SOFTWARE\  
    ODBC\  
      ODBC.INI\  
        user-data-source-name
```

Für Systemdatenquellen unter 32-Bit-Windows lautet der Registrierungsschlüssel wie folgt:

```
HKEY_LOCAL_MACHINE\  
  SOFTWARE\  
    ODBC\  
      ODBC.INI\  
        system-data-source-name
```

Für Systemdatenquellen unter 64-Bit-Windows gibt es zwei Registrierungsschlüssel: einen für 64-Bit-Anwendungen und einen für 32-Bit-Anwendungen. Der 64-Bit-Registrierungsschlüssel lautet wie folgt:

```
HKEY_LOCAL_MACHINE\  
SOFTWARE\  
ODBC\  
ODBC.INI\  
system-data-source-name
```

Verwenden Sie unter 64-Bit-Windows die 64-Bit-Version von dbdsn, um den 64-Bit Registrierungsschlüssel zu erstellen.

Der 32-Bit-Registrierungsschlüssel lautet wie folgt:

```
HKEY_LOCAL_MACHINE\  
SOFTWARE\  
Wow6432Node\  
ODBC\  
ODBC.INI\  
system-data-source-name
```

Verwenden Sie unter 64-Bit-Windows die 32-Bit-Version von dbdsn, um den 32-Bit Registrierungsschlüssel zu erstellen.

Der Schlüssel enthält eine Gruppe von Registrierungswerten, die jeweils einem Verbindungsparameter entsprechen (mit Ausnahme der Treiber-Zeichenfolge). Die Treiber-Zeichenfolge wird vom Microsoft ODBC-Treibermanager automatisch zur Registrierung hinzugefügt, wenn eine Datenquelle mit dem Microsoft ODBC-Datenquellen-Administrator oder mit dem dbdsn-Dienstprogramm erstellt wird. Der SQL Anywhere 16 Demo-Schlüssel, der dem Systemdatenquellennamen (DSN) von SQL Anywhere 16 Demo entspricht, enthält z.B. die folgenden Einstellungen für 32-Bit-Windows:

Name des Wertes	Typ des Wertes	Daten des Wertes
AutoStop	Zeichenfolge	YES
DatabaseFile	Zeichenfolge	C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db
Description	Zeichenfolge	SQL Anywhere 16-Beispieldatenbank
Driver	Zeichenfolge	C:\Program Files\SQL Anywhere 16\bin32\dbodbc16.dll
PWD	Zeichenfolge	sql
ServerName	Zeichenfolge	demo16
StartLine	Zeichenfolge	C:\Program Files\SQL Anywhere 16\bin32\dbeng16.exe
UID	Zeichenfolge	DBA

Hinweis

Es wird empfohlen, dass Sie den ServerName-Parameter in Verbindungszeichenfolgen bei bereitgestellten Anwendungen einbeziehen. Damit wird sichergestellt, dass sich die Anwendung mit dem richtigen Server verbindet, wenn auf einem Computer mehrere SQL Anywhere-Datenbankserver laufen. Dadurch können Verbindungsfehler durch Zeitablauf verhindert werden.

Bei 64-Bit-Windows wird *bin32* durch *bin64* ersetzt.

Unter 64-Bit-Windows gibt es nur einen Registrierungseintrag für Benutzerdatenquellen, der von 64-Bit- und 32-Bit-Anwendungen gemeinsam genutzt wird.

Außerdem müssen Sie den Datenquellennamen der Liste von Datenquellen in der Registrierung hinzufügen. Für Benutzerdatenquellen benutzen Sie folgenden Schlüssel:

```
HKEY_CURRENT_USER\
  SOFTWARE\
    ODBC\
      ODBC.INI\
        ODBC Data Sources
```

Für Systemdatenquellen benutzen Sie folgenden Schlüssel:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
    ODBC\
      ODBC.INI\
        ODBC Data Sources
```

Der Wert verknüpft jede Datenquelle mit einem ODBC-Treiber. Der Name des Wertes ist der Datenquellennamen, und die Daten des Wertes sind der Name des ODBC-Treibers. Beispiel: Die von SQL Anywhere installierte Systemdatenquelle wird als SQL Anywhere 16 Demo benannt und hat den folgenden Wert:

Name des Wertes	Typ des Wertes	Daten des Wertes
SQL Anywhere 16 Demo	Zeichenfolge	SQL Anywhere 16

Vorsicht

Einstellungen für ODBC-Datenquellennamen können leicht eingesehen werden. Die Konfiguration von Benutzerdatenquellen kann vertrauliche Datenbankeinstellungen enthalten, z.B. Benutzer-IDs und Kennwörter. Diese Einstellungen werden in der Registrierung als reiner Text gespeichert und können mit den Windows-Registrierungseditoren (*regedit.exe* oder *regedt32.exe*) angezeigt werden. Diese Editoren werden von Microsoft zusammen mit dem Betriebssystem bereitgestellt. Sie können Kennwörter verschlüsseln oder von Benutzern verlangen, sie bei der Aufnahme der Verbindung einzugeben.

Erforderliche und optionale Verbindungsparameter

Sie können den Datenquellennamen in einer ODBC-Verbindungszeichenfolge in folgender Weise definieren:

```
DSN=ADataSourceName
```

Unter Windows werden, wenn ein DSN-Parameter in der Verbindungszeichenfolge geliefert wird, erst die Definitionen von Benutzerdatenquellen in der Windows-Registrierung durchsucht, dann die Systemdatenquellen. Dateidatenquellen werden nur durchsucht, wenn in der ODBC-Verbindungszeichenfolge der FileDSN-Verbindungsparameter angegeben wird.

Die folgende Tabelle zeigt die Auswirkungen für Benutzer und Anwendungsentwickler, wenn eine Datenquelle vorhanden ist und in der Verbindungszeichenfolge der Anwendung als DSN- oder FileDSN-Parameter angegeben wird.

Datenquelle	Eingabe in der Verbindungszeichenfolge	Eingabe durch den Benutzer
Enthält den ODBC-Treibernamen und Speicherort, den Namen der Datenbankdatei, des Datenbankservers, die Startparameter sowie Benutzer-ID und Kennwort	Keine zusätzlichen Informationen	Keine zusätzlichen Informationen
Enthält den ODBC-Treibernamen und den Speicherort, den Namen der Datenbankdatei bzw. des Datenbankservers und Startparameter	Keine zusätzlichen Informationen	Benutzer-ID und Kennwort, wenn nicht in der Zeichenfolge für die ODBC-Datenquelle angegeben
Enthält nur Namen und Speicherort des ODBC-Treibers	Der Name der Datenbankdatei (DBF=) bzw. des Datenbankservers (SERVER=). Optional können andere Verbindungsparameter enthalten sein, z.B. Userid (UID=) und Password (PWD=).	Benutzer-ID und Kennwort, wenn nicht in der Zeichenfolge für die ODBC-Verbindung oder ODBC-Datenquelle angegeben.
Nicht vorhanden	Der Name des zu verwendenden ODBC-Treibers (Driver=) und der Datenbankname (DBN=), die Datenbankdatei (DBF=) bzw. der Datenbankserver (SERVER=). Optional können andere Verbindungsparameter enthalten sein, z.B. Userid (UID=) und Password (PWD=).	Benutzererkennung und Kennwort, wenn nicht in der Zeichenfolge für die ODBC-Verbindung bereitgestellt

Siehe auch

- „Datenbankverbindungen“ [[SQL Anywhere Server - Datenbankadministration](#)]
- Open Database Connectivity (ODBC) SDK von Microsoft

Deployment von Embedded SQL-Clients

Die einfachste Möglichkeit, Embedded SQL-Clients bereitzustellen, ist die Verwendung des **Deployment-Assistenten**.

Das Deployment von Embedded SQL-Clients im System umfasst folgende Aufgaben:

- **Installierte Dateien** Jeder Clientcomputer muss die erforderlichen Dateien für eine SQL Anywhere Embedded SQL-Clientanwendung enthalten.
- **Verbindungsinformationen** Die Clientanwendung muss Zugriff auf die Daten haben, aus denen sie die Informationen für die Verbindung mit dem Server bezieht. Diese Informationen können in die ODBC-Datenquelle aufgenommen werden.

Siehe auch

- „Deployment-Assistent“ auf Seite 952

Erforderliche Dateien für Embedded SQL-Clients

Die folgende Tabelle zeigt, welche Dateien für Embedded SQL Clients benötigt werden.

Beschreibung	Windows	Linux/Unix	Mac OS X
Schnittstellenbibliothek	<i>dblib16.dll</i>	<i>libdblib16_r.so</i>	<i>libdblib16_r.dylib</i>
Optionale Verschlüsselungsunterstützung	<i>dbfips16.dll</i>	<i>libdbfips16_r.so</i>	<i>libdbfips16_r.dylib</i>
Optionale Verschlüsselungsunterstützung	<i>dbrsa16.dll</i>	<i>libdbrsa16_r.so</i>	<i>libdbrsa16_r.dylib</i>
Thread-Unterstützungsbibliothek	k.A.	<i>libdbtasks16_r.so</i>	<i>libdbtasks16_r.dylib</i>
Sprachen-Ressourcenbibliothek	<i>dblg[LL]16.dll</i>	<i>dblg[LL]16.res</i>	<i>dblg[LL]16.res</i>
Fenster "Verbinden"	<i>dbcon16.dll</i>	k.A.	k.A.

Hinweise

- Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).
- Für Anwendungen ohne Multi-Threading unter Linux/Unix verwenden Sie *libdblib16.so* und *libdbtasks16.so*.
- Bei Anwendungen ohne Multi-Threading unter Mac OS X verwenden Sie *libdblib16.dylib* und *libdbtasks16.dylib*.
- Wenn die Clientanwendung Verschlüsselung verwendet, muss die geeignete Verschlüsselungsunterstützung (*dbfips16.dll* oder *dbrsa16.dll*) ebenfalls einbezogen werden.

- Wenn die Clientanwendung eine ODBC-Datenquelle benutzt, um die Verbindungsparameter zu speichern, muss der Endbenutzer über eine funktionierende ODBC-Installation verfügen. Anweisungen für das Deployment von ODBC finden Sie im Microsoft ODBC SDK.
- Unter Windows wird der Unterstützungscod für die Fenster **ODBC-Konfiguration für SQL Anywhere** und **Mit SQL Anywhere verbinden** (*dbcon16.dll*) benötigt, wenn Ihre Endbenutzer eigene Datenquellen erstellen oder beim Herstellen der Verbindung mit der Datenbank Benutzer-IDs und Kennwörter eingeben müssen sowie wenn das Fenster "Verbinden" aus anderen Gründen angezeigt werden muss.

Siehe auch

- „Deployment von ODBC-Clients“ auf Seite 970

Verbindungsinformationen

Das Deployment von Embedded SQL-Verbindungsinformationen erfolgt auf folgende Weise:

- **Manuell** Stellen Sie Anweisungen für die Endbenutzer bereit, damit sie auf ihrem Computer eine geeignete Datenquelle einrichten können.
- **Datei** Übergeben Sie den Endbenutzern eine Datei, die Verbindungsinformationen in einem Format enthält, das Ihre Anwendung lesen kann.
- **ODBC-Datenquelle** Sie können die ODBC-Datenquelle verwenden, um darin die Verbindungsinformationen zu speichern.

Deployment von JDBC-Clients

Sie müssen eine Java-Laufzeitumgebung (JRE) installieren, um JDBC verwenden zu können. Empfohlen wird Version 1.7.0 oder höher.

Zusätzlich zu JRE benötigt jeder JDBC-Client den SQL Anywhere JDBC-Treiber oder jConnect.

SQL Anywhere JDBC 4.0-Treiber

Für das Deployment des SQL Anywhere JDBC 4.0-Treibers müssen Sie die Datei *sajdbc4.jar* bereitstellen, die sich im SQL Anywhere-Installationsverzeichnis im Ordner *java* befindet. Diese Datei muss sich im Classpath der Anwendung befinden.

Außerdem müssen Sie eine Reihe von Unterstützungsdateien für den JDBC-Treiber bereitstellen. Die folgende Tabelle zeigt die erforderlichen Dateien für verschiedene Plattformen. Diese Dateien sollten in nur einem Verzeichnis abgelegt werden. Die SQL Anywhere-Installation platziert sie alle in das Betriebssystem-Unterverzeichnis des SQL Anywhere-Installationsverzeichnisses (zum Beispiel *bin32*, *bin64*, *lib32* oder *lib64*). Die Wahl der 32-Bit- oder 64-Bit-Dateien hängt von der Bitversion der installierten Java VM ab.

Die JDBC-Treiber-Unterstützungsdateien für alle Plattformen arbeiten mit mehreren Threads.

Plattform	Erforderliche Dateien
Windows	<i>dbjdbc16.dll</i> <i>dbicu16.dll</i> <i>dbicudt16.dll</i> <i>dblg[LL]16.dll</i>
Linux, Solaris, HP-UX	<i>libdbjdbc16.so.1</i> <i>libdbtasks16_r.so.1</i> <i>libdbicu16_r.so.1</i> <i>libdbicudt16.so.1</i> <i>dblg[LL]16.res</i>
AIX	<i>libdbjdbc16.so</i> <i>libdbtasks16_r.so</i> <i>libdbicu16_r.so</i> <i>libdbicudt16.so</i> <i>dblg[LL]16.res</i>
Mac OS X	<i>libdbjdbc16.dylib</i> <i>libdbtasks16_r.dylib</i> <i>libdbicu16_r.dylib</i> <i>libdbicudt16.dylib</i> <i>dblg[LL]16.res</i>

- Bei Linux- und Solaris-Plattformen sollten Sie eine Verknüpfung zu den *.so.1*-Dateien erstellen. Der Name der Verknüpfung muss mit dem Dateinamen übereinstimmen, wobei das Versionssuffix ".1" entfernt werden muss.
- Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).

jConnect JDBC-Treiber

Für das Deployment des jConnect JDBC-Treibers müssen folgende Dateien mitgeliefert werden:

- Die Dateien des jConnect-Treibers. Eine Version der jConnect-Software und der jConnect-Dokumentation finden Sie unter <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>.
- Wenn Sie einen TDS-Client verwenden (entweder Open Client oder jConnect-basiert), können Sie das Verbindungskennwort entweder in Klartext oder verschlüsselt senden. Beim Senden in verschlüsselter Form wird ein TDS-Handshake für das Kennwort ausgeführt. Der Handshake umfasst die Verschlüsselung des privaten/öffentlichen Schlüssels. Die Unterstützung zum Generieren des privaten/öffentlichen RSA-Schlüsselpaares und zum Entschlüsseln des verschlüsselten Kennworts ist in einer speziellen Bibliothek enthalten. Die Position der Bibliotheksdatei muss vom SQL Anywhere-Datenbankserver in seinem Systempfad ermittelt werden können. Unter Windows heißt diese Datei *dbrsakp16.dll*. Es gibt sowohl 64-Bit- als auch 32-Bit-Versionen der DLL. In Linux- und Unix-Umgebungen ist die Datei eine Shared Library namens *libdbrsakp16.so*. Unter Mac OS X ist die Datei eine Shared Library namens *libdbrsakp16.dylib*. Die Datei ist nicht erforderlich, wenn Sie diese Funktion nicht verwenden.

URL zur JDBC-Datenbankverbindung

Ihre Java-Anwendung benötigt eine URL, um sich mit der Datenbank zu verbinden. Diese URL gibt den Treiber, den zu verwendenden Computer und den Port an, auf dem der Datenbankserver auf Daten wartet.

Siehe auch

- „jConnect-Treiber-Verbindungszeichenfolgen“ auf Seite 426

PHP Client-Deployment

Für das Deployment der SQL Anywhere-PHP-Erweiterung müssen Sie die folgenden Komponenten auf der Zielplattform installieren:

- PHP 5-Binärdateien für Ihre Plattform, die unter <http://www.php.net> zum Download zur Verfügung stehen. Bei Windows-Plattformen muss die threadsichere Version von PHP zusammen mit der SQL Anywhere-PHP-Erweiterung verwendet werden.

- Einen Webserver, z.B. den Apache HTTP-Server, wenn Sie PHP-Skripten in einem Webserver ausführen.

SQL Anywhere kann auf demselben Computer wie der Webserver ausgeführt werden oder auf einem anderen Computer.

- SQL Anywhere stellt vorgefertigte PHP-Erweiterungen für viele PHP-Versionen von 5.1.1 bis 5.4.8 bereit.
- Unterstützung für SQL Anywhere-Shared Objects oder -Shared Libraries.

Die folgende Tabelle gibt einen Überblick über die für PHP-Clients erforderlichen Dateien.

Beschreibung	Windows	Linux/Unix	Mac OS X
PHP-Installation (Drittanbieter)	<i>php.exe</i>	<i>php</i>	<i>php</i>
PHP 5.1.x-Aufrufe	<i>php-5.1.[1-*)_sqlanywhere.dll</i>	<i>php-5.1.[1-*)_sqlanywhere_r.so</i> oder Build aus Quellcode	Build aus Quellcode
PHP 5.2.x-Aufrufe	<i>php-5.2.[0-*)_sqlanywhere.dll</i> <i>php-5.2.1_sqlanywhere_nts.dll</i>	<i>php-5.2.[0-*)_sqlanywhere_r.so</i> oder Build aus Quellcode	Build aus Quellcode
PHP 5.3.x-Aufrufe	<i>php-5.3.[0-*)_sqlanywhere.dll</i> <i>php-5.3.[0-*)_sqlanywhere_nts.dll</i>	<i>php-5.3.[0-*)_sqlanywhere_r.so</i> oder Build aus Quellcode	Build aus Quellcode
PHP 5.4.x-Aufrufe	<i>php-5.4.[0-*)_sqlanywhere.dll</i> <i>php-5.4.[0-*)_sqlanywhere_nts.dll</i>	<i>php-5.4.[0-*)_sqlanywhere_r.so</i> oder Build aus Quellcode	Build aus Quellcode
Sprachen-Ressourcenbibliothek	<i>dblg[LL]16.dll</i>	<i>dblg[LL]16.res</i>	<i>dblg[LL]16.res</i>
Fenster "Verbinden"	<i>dbcon16.dll</i>	k.A.	k.A.
SQL Anywhere C-API-Laufzeitumgebung	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB (mit Threading)	<i>dblib16.dll</i>	<i>libdblib16_r.so</i>	<i>libdblib16_r.dylib</i>
Thread-Unterstützungsbibliothek	k.A.	<i>libdbtasks16_r.so</i>	<i>libdbtasks16_r.dylib</i>
ICU-Bibliothek	<i>dbicu16.dll</i>	<i>libdbicu16_r.so</i> ²	<i>libdbicu16_r.dylib</i> ²
ICU-Datenbibliothek	<i>dbicudt16.dll</i>	<i>libdbicudt16.so</i> ²	<i>libdbicudt16.dylib</i> ²
Optionale Verschlüsselungsunterstützung	<i>dbfips16.dll</i>	<i>libdbfips16_r.so</i>	<i>libdbfips16_r.dylib</i>
Optionale Verschlüsselungsunterstützung	<i>dbrsa16.dll</i>	<i>libdbrsa16_r.so</i>	<i>libdbrsa16_r.dylib</i>

Hinweise

- Sie finden die neuesten SQL Anywhere-PHP-Treiber unter <http://www.sybase.com/detail?id=1019698>.
- Die *php-5.x.y_sqlanywhere_nts.dll*-Dateien sind eine nicht-threadsichere Version der PHP-Erweiterungen.
- Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).
- Wenn die Clientanwendung Verschlüsselung verwendet, muss die geeignete Verschlüsselungsunterstützung einbezogen werden.
- Wenn die Clientanwendung eine ODBC-Datenquelle benutzt, um die Verbindungsparameter zu speichern, muss der Endbenutzer über eine funktionierende ODBC-Installation verfügen. Anweisungen für das Deployment von ODBC finden Sie im Microsoft ODBC SDK.
- Unter Windows wird der Unterstützungscod für die Fenster **ODBC-Konfiguration für SQL Anywhere** und **Mit SQL Anywhere verbinden** (*dbcon16.dll*) benötigt, wenn Ihre Endbenutzer eigene Datenquellen erstellen oder beim Herstellen der Verbindung mit der Datenbank Benutzer-IDs und Kennwörter eingeben müssen sowie wenn das Fenster "Verbinden" aus anderen Gründen angezeigt werden muss.
- Hinweise zur Installation von PHP unter Windows und Linux/Solaris finden Sie in den folgenden Themen.
- Ein Whitepaper über PHP finden Sie unter <http://www.sybase.com/detail?id=1057714>.

Siehe auch

- „Deployment von ODBC-Clients“ auf Seite 970

PHP-Erweiterungen

Unter Windows stellt SQL Anywhere threadsichere Erweiterungen für die PHP-Versionen 5.1.1 bis 5.4.8 bereit. Mit der SQL Anywhere-PHP-Erweiterung muss eine threadsichere PHP-Version verwendet werden. Die Erweiterungs-Dateinamen für die unterstützten PHP-Versionen entsprechen folgendem Muster:

```
php-5.x.y_sqlanywhere.dll
```

Unter Windows stellt SQL Anywhere nicht-threadsichere Erweiterungen für viele PHP-Versionen von 5.2.11 bis 5.4.8 bereit. Eine nicht-threadsichere Version von PHP kann bei Webservern, die mehrere Prozessoren, aber nicht mehrere Threads haben, eingesetzt werden, beispielsweise Apache und IIS. Die Erweiterungs-Dateinamen für die unterstützten PHP-Versionen entsprechen folgendem Muster:

```
php-5.x.y_sqlanywhere_nts.dll
```

Unter Linux und Solaris umfasst SQL Anywhere sowohl die 64-Bit-Versionen als auch die 32-Bit-Versionen der Erweiterungen für viele PHP-Versionen von 5.1.1 bis 5.4.8. Ebenfalls enthalten sind Erweiterungen mit und ohne Threading. Benutzen Sie die Erweiterung ohne Threading, wenn Sie die CGI-Version von PHP verwenden oder Apache 1.x. Wenn Sie Apache 2.x verwenden, benutzen Sie die Erweiterung mit Threading. Die Erweiterungs-Dateinamen für die unterstützten PHP-Versionen entsprechen folgendem Muster:

```
php-5.x.y_sqlanywhere[_r].so
```

"5.x.y" stellt die PHP-Version dar (z.B. 5.2.11). Bei Linux und Solaris wird an den Dateinamen der Version der PHP-Erweiterung mit Threading `_r` angehängt. Windows-Versionen werden als Dynamic Link Libraries und Linux/Solaris-Versionen als Shared Objects implementiert. Wenn Sie mit einer höheren Version von PHP arbeiten, dann können Sie die neuesten SQL Anywhere-PHP-Treiber unter <http://www.sybase.com/detail?id=1019698> abrufen.

PHP-Erweiterung unter Windows installieren

Kopieren Sie die DLL der PHP-Erweiterung aus dem SQL Anywhere-Installationsverzeichnis und fügen Sie zu Ihrer PHP-Installation hinzu, um die SQL Anywhere-PHP-Erweiterung unter Windows verwenden zu können. Fügen Sie optional Ihrer PHP-Initialisierungsdatei einen Eintrag hinzu, mit dem die Erweiterung geladen wird, damit Sie sie nicht in jedem Skript manuell laden müssen.

Voraussetzungen

PHP 5.0 oder später muss installiert sein.

Aufgabe

1. Suchen Sie die Datei *php.ini* für Ihre PHP-Installation und öffnen Sie sie in einem Texteditor.
2. Suchen Sie die Zeile mit dem **extension_dir**-Eintrag.
3. Wenn der Eintrag nicht vorhanden ist, wird empfohlen, dass Sie den **extension_dir**-Eintrag erstellen und aus Gründen der Systemsicherheit damit auf ein isoliertes Verzeichnis zeigen
4. Kopieren Sie eine der DLL-Dateien (*php-5.x.y_sqlanywhere.dll*) aus dem Unterverzeichnis *Bin32* Ihrer SQL Anywhere-Installation in das Verzeichnis, das im **extension_dir**-Eintrag in der Datei *php.ini* angegeben ist.

Hinweis

Die Zeichenfolge 5.x.y ist die PHP-Versionsnummer, die der von Ihnen installierten Version entspricht.

Wenn Ihre PHP-Version neuer als die SQL Anywhere-PHP-Erweiterungen ist, die von SQL Anywhere bereitgestellt werden, sollten Sie die neueste bereitgestellte Erweiterung verwenden. Eine SQL Anywhere-PHP-Erweiterung der Version 5.2.x funktioniert nicht mit einer PHP-Version 5.3.x.

5. Sie können optional die folgende Zeile in den Abschnitt "Dynamic Extensions" der *php.ini*-Datei einfügen, um den SQL Anywhere-PHP-Treiber automatisch zu laden. Ohne diesen Schritt muss der PHP-Treiber jedes Mal manuell geladen werden, wenn das Skript ihn benötigt.

Kopieren Sie den folgenden Eintrag:

```
extension=php-5.x.y_sqlanywhere.dll
```

5.x.y stellt die Versionsnummer der SQL Anywhere-PHP-Erweiterung dar, die im vorherigen Schritt kopiert wurde.

6. Speichern und schließen Sie die Datei *php.ini*.
7. Achten Sie darauf, dass das Unterverzeichnis *Bin32* Ihrer SQL Anywhere-Installation in Ihrem Pfad enthalten ist.

Ergebnisse

Die SQL Anywhere-PHP-Erweiterung wird in der Windows-Umgebung unterstützt und kann direkt verwendet werden.

Siehe auch

- „PHP-Testseiten erstellen und ausführen“ auf Seite 670
- „Konfiguration der SQL Anywhere-PHP-Erweiterung“ auf Seite 990

PHP-Erweiterung unter Linux/Solaris installieren

Kopieren Sie das Shared Object aus dem SQL Anywhere-Installationsverzeichnis und fügen Sie es zu Ihrer PHP-Installation hinzu, um die SQL Anywhere-PHP-Erweiterung unter Linux oder Solaris verwenden zu können. Fügen Sie optional Ihrer PHP-Initialisierungsdatei einen Eintrag hinzu, mit dem die Erweiterung geladen wird, damit Sie sie nicht in jedem Skript manuell laden müssen.

Voraussetzungen

PHP 5.0 oder später muss installiert sein.

Aufgabe

1. Suchen Sie die Datei *php.ini*-Ihrer PHP-Installation und öffnen Sie sie in einem Texteditor.

Suchen Sie die Zeile mit dem **extension_dir**-Eintrag.

Wenn der Eintrag nicht vorhanden ist, wird empfohlen, dass Sie den **extension_dir**-Eintrag erstellen und aus Gründen der Systemsicherheit damit auf ein isoliertes Verzeichnis zeigen

2. Kopieren Sie eines der Shared Objects (*php-5.x.y_sqlanywhere*) aus dem Unterverzeichnis *lib32* oder *lib64* Ihrer SQL Anywhere-Installation in das durch die **extension_dir**-Variable in der Datei *php.ini* angegebene Verzeichnis.

Welches Shared Object Sie wählen müssen, hängt von der installierten Version von PHP ab und davon, ob es sich um eine 32-Bit-Version oder eine 64-Bit-Version handelt.

Hinweis

Die Zeichenfolge 5.x.y ist die PHP-Versionsnummer, die der von Ihnen installierten Version entspricht.

Wenn Ihre PHP-Version neuer ist als das von SQL Anywhere bereitgestellte Shared Object, verwenden Sie das neueste bereitgestellte Shared Object. Eine SQL Anywhere-PHP-Erweiterung der Version 5.2.x funktioniert nicht mit einer PHP-Version 5.3.x.

Hinweise über die zu verwendende Version des Shared Object finden Sie unter „[PHP-Erweiterungen](#)“ auf Seite 986.

3. Sie können optional die folgende Zeile in den Abschnitt Dynamic Extensions der *php.ini*-Datei einfügen, um den SQL Anywhere PHP-Treiber automatisch zu laden. Ohne diesen Schritt muss der PHP-Treiber jedes Mal manuell geladen werden, wenn das Skript ihn benötigt.

Der Eintrag muss das kopierte Shared Object identifizieren, entweder:

```
extension=php-5.x.y_sqlanywhere.so
```

oder beim threadsicheren Shared Object:

```
extension=php-5.x.y_sqlanywhere_r.so
```

5.x.y ist die Versionsnummer für das PHP-Shared Object, das im vorherigen Schritt kopiert wurde.

4. Speichern und schließen Sie die Datei *php.ini*.
5. Überprüfen Sie, ob Ihre PHP-Ausführungsumgebung für SQL Anywhere eingerichtet ist.

Abhängig von der verwendeten Shell müssen Sie das Konfigurationsskript für die Umgebung Ihres Webserverns bearbeiten und den entsprechenden Befehl hinzufügen, um das Sourcing des SQL Anywhere-Konfigurationsskripts aus dem SQL Anywhere-Installationsverzeichnis durchzuführen.

Verwendete Shell	Verwendeter Befehl
sh, ksh oder bash	<code>./bin32/sa_config.sh</code>
csh oder tcsh	<code>source /bin32/sa_config.csh</code>

Die 32-Bit-Version der SQL Anywhere PHP-Erweiterung erfordert, dass das *bin 32*-Verzeichnis in Ihrem Pfad enthalten ist. Die 64-Bit-Version der SQL Anywhere-PHP-Erweiterung erfordert, dass das Verzeichnis *bin64* in Ihrem Pfad enthalten ist.

Die Konfigurationsdatei, in der diese Zeile eingefügt werden soll, hängt von den verschiedenen Webservern und den verschiedenen Linux-Distributionen ab. Hier sind einige Beispiele für den Apache-Server mit den angegebenen Distributionen:

- **RedHat/Fedora/CentOS** /etc/sysconfig/httpd
- **Debian/Ubuntu** /etc/apache2/envvars

Hinweis

Der Webserver muss nach der Bearbeitung seiner Umgebungskonfiguration neu gestartet werden.

Ergebnisse

Die SQL Anywhere-PHP-Erweiterung wird in der Linux/Solaris-Umgebung unterstützt und kann direkt verwendet werden.

Siehe auch

- „PHP-Testseiten erstellen und ausführen“ auf Seite 670
- „Konfiguration der SQL Anywhere-PHP-Erweiterung“ auf Seite 990

Die PHP-Erweiterung unter Unix und Mac OS X

Um die SQL Anywhere-PHP-Erweiterung unter anderen Versionen von Unix oder Mac OS X zu verwenden, müssen Sie die PHP-Erweiterung mithilfe des Quellcodes erstellen, der sich im Unterverzeichnis *sdk\php* Ihres SQL Anywhere-Installationsverzeichnis befindet.

Siehe auch

- „Erstellen der SQL Anywhere-PHP-Erweiterung unter Unix und Mac OS X“ auf Seite 678

Konfiguration der SQL Anywhere-PHP-Erweiterung

Das Verhalten des SQL Anywhere PHP-Treibers kann gesteuert werden, indem Sie Werte in der PHP-Initialisierungsdatei *php.ini* einstellen. Folgende Einträge werden unterstützt:

- **extension** Bewirkt, dass PHP beim Starten automatisch die SQL Anywhere-PHP-Erweiterung lädt. Das Hinzufügen dieses Eintrags zu Ihrer PHP-Initialisierungsdatei ist optional, aber wenn Sie ihn nicht hinzufügen, muss jedes Skript mit Programmcodezeilen beginnen, die sicherstellen, dass dieses Modul geladen wird. Der nachstehende Eintrag wird für Windows-Plattformen verwendet.

```
extension=php-5.x.y_sqlanywhere.dll
```

Verwenden Sie auf Linux-Plattformen einen der folgenden Einträge. Der zweite Eintrag ist threadsicher.

```
extension=php-5.x.y_sqlanywhere.so
```

```
extension=php-5.x.y_sqlanywhere_r.so
```

Bei diesen Einträgen steht 5.x.y für die PHP-Version.

Wenn die SQL Anywhere-Erweiterung nicht immer automatisch geladen wird, wenn PHP startet, müssen Sie jedem Skript die folgenden Codezeilen voranstellen. Dieser Code stellt sicher, dass die SQL Anywhere-PHP-Erweiterung geladen wird.

```
# Ensure that the SQL Anywhere PHP extension is loaded
if( !extension_loaded('sqlanywhere') ) {
    # Find out which version of PHP is running
    $version = phpversion();
    $extension_name = 'php-'. $version . '_sqlanywhere';
    if( strtoupper(substr(PHP_OS, 0, 3)) == 'WIN' ) {
        $extension_ext = '.dll';
    } else {
        $extension_ext = '.so';
    }
    dl( $extension_name.$extension_ext );
}
```

- **allow_persistent** Beständige Verbindungen zulassen, wenn dieser Eintrag auf ON gesetzt ist. Sie sind nicht zulässig, wenn dieser Eintrag auf OFF gesetzt ist. Der Standardwert ist ON.

```
sqlanywhere.allow_persistent=On
```

- **max_persistent** Legt die maximale Anzahl von dauerhaften Verbindungen fest. Der Standardwert ist -1, was die Verbindungszahl unbeschränkt läßt.

```
sqlanywhere.max_persistent=-1
```

- **max_connections** Legt die maximale Anzahl von Verbindungen fest, die durch die SQL Anywhere-PHP-Erweiterung gleichzeitig geöffnet werden können. Der Standardwert ist -1, was die Verbindungszahl unbeschränkt läßt.

```
sqlanywhere.max_connections=-1
```

- **auto_commit** Legt fest, ob der Datenbankserver automatisch einen Festschreibevorgang ausführt. Wenn dieser Eintrag den Wert ON hat, wird das Festschreiben unmittelbar nach der Ausführung der einzelnen Anweisungen durchgeführt. Wenn dieser Eintrag den Wert OFF hat, sollten Transaktionen manuell mit der sasql_commit- bzw. der sasql_rollback-Funktion beendet werden. Der Standardwert ist ON.

```
sqlanywhere.auto_commit=On
```

- **row_counts** Wenn dieser Eintrag den Wert ON hat, wird die genaue Anzahl der Zeilen zurückgegeben. Wenn er den Wert OFF hat, wird ein geschätzter Wert zurückgegeben. Der Standardwert ist OFF.

```
sqlanywhere.row_counts=Off
```

- **verbose_errors** Wenn dieser Eintrag den Wert ON hat, werden ausführliche Fehler und Warnungen zurückgegeben. Andernfalls müssen Sie die sasql_error- bzw. sasql_errorcode-Funktion aufrufen, um weitere Fehlerinformationen zu erhalten. Der Standardwert ist ON.

```
sqlanywhere.verbose_errors=On
```

Siehe auch

- „sasql_set_option“ auf Seite 702

Deployment von Open Client-Anwendungen

Um das Deployment von Open Client-Anwendungen vorzunehmen, muss auf jedem Computer Sybase Open Client installiert sein. Sie müssen die Open Client-Software getrennt bei Sybase erwerben. Sie enthält eigene Installationsanweisungen.

Wenn Sie einen TDS-Client verwenden (entweder Open Client oder jConnect-basiert), können Sie das Verbindungskennwort entweder in Klartext oder verschlüsselt senden. Beim Senden in verschlüsselter Form wird ein TDS-Handshake für das Kennwort ausgeführt. Der Handshake umfasst die Verschlüsselung des privaten/öffentlichen Schlüssels. Die Unterstützung zum Generieren des privaten/öffentlichen RSA-Schlüsselpaares und zum Entschlüsseln des verschlüsselten Kennworts ist in einer speziellen Bibliothek enthalten. Die Position der Bibliotheksdatei muss vom SQL Anywhere-Datenbankserver in seinem Systempfad ermittelt werden können. Unter Windows heißt diese Datei *dbrsakp16.dll*. Es gibt sowohl eine 64-Bit-Version als auch eine 32-Bit-Version der DLL. In Linux- und Unix-Umgebungen ist die Datei eine Shared Library namens *libdbrsakp16.so*. Unter Mac OS X ist die Datei eine Shared Library namens *libdbrsakp16_r.dylib*. Die Datei ist nicht erforderlich, wenn Sie diese Funktion nicht verwenden.

Verbindungsinformationen für Open Client-Clients sind in der Interface-Datei zu finden.

Deployment von Administrationstools

Je nach Ihrer Lizenzvereinbarung können Sie das Deployment einer Reihe von Administrationstools wie Interactive SQL, Sybase Central und des SQL Anywhere-Konsolendienstprogramms (dbconsole) vornehmen.

Die einfachste Möglichkeit, die Administrationstools bereitzustellen, ist die Verwendung des **Deployment-Assistenten**.

Hinweise zu Systemanforderungen für Administrationstools finden Sie unter <http://www.sybase.com/detail?id=1002288>.

Initialisierungsdateien können das Deployment von Administrationstools vereinfachen. Jede Programmdatei der Administrationstools (Sybase Central, Interactive SQL und das SQL Anywhere-Konsolendienstprogramm) kann eine entsprechende *.ini*-Datei haben. Wenn sie verwendet wird, sind keine Registrierungseinträge und keine feste Verzeichnisstruktur für den Speicherort der JAR-Dateien erforderlich. Diese *.ini*-Dateien befinden sich im gleichen Verzeichnis und haben dieselben Dateinamen wie die jeweiligen Programmdateien.

- ***dbconsole.ini*** Dies ist der Name der Initialisierungsdatei des Konsolendienstprogramms.
- ***dbisql.ini*** Dies ist der Name der Initialisierungsdatei von Interactive SQL.
- ***scjview.ini*** Dies ist der Name der Initialisierungsdatei von Sybase Central.

Die Initialisierungsdatei enthält die Details dazu, wie das Datenbankadministrationstool geladen wird. Die Initialisierungsdatei kann beispielsweise die folgenden Zeilen enthalten:

- **JRE_DIRECTORY=*Pfad*** Dies ist der Speicherort der erforderlichen JRE. Die Angabe von **JRE_DIRECTORY** ist erforderlich.
- **VM_ARGUMENTS=*alle erforderlichen VM-Argumente*** VM-Argumente werden durch Semikola (;) getrennt. Alle Pfadwerte, die Leerzeichen enthalten, müssen in Anführungszeichen gestellt werden. VM-Argumente können mithilfe der Option `-batch` des Administrationstools und in der erstellten Batchdatei ermittelt werden. Unter Windows generiert der Start von Sybase Central mit `scjview -batch` an einer Eingabeaufforderung die Datei `scjview.bat`, und der Start von Interactive SQL mit `dbisql -batch` generiert `dbisql.bat`. Die Angabe von **VM_ARGUMENTS** ist optional.
- **JAR_PATHS=*Pfad1;Pfad2;...*** Eine begrenzte Liste von Verzeichnissen, die die JAR-Dateien für das Programm enthalten. Sie werden durch Semikola (;) getrennt. Die Angabe von **JAR_PATHS** ist optional.
- **ADDITIONAL_CLASSPATH=*Pfad1;Pfad2;...*** Classpath-Werte werden durch Semikola (;) getrennt. Die Angabe von **ADDITIONAL_CLASSPATH** ist optional.
- **LIBRARY_PATHS=*Pfad1;Pfad2;...*** Dies sind Pfade zu den DLLs/Shared Objects. Sie werden durch Semikola (;) getrennt. Die Angabe von **LIBRARY_PATHS** ist optional.
- **APPLICATION_ARGUMENTS=*Arg1;Arg2;...*** Dies sind beliebige Anwendungsargumente. Sie werden durch Semikola (;) getrennt. Anwendungsargumente können mithilfe der Option `-batch` des Administrationstools und in der erstellten Batchdatei ermittelt werden. Unter Windows generiert der Start von Sybase Central mit `scjview -batch` an einer Eingabeaufforderung die Datei `scjview.bat`, und der Start von Interactive SQL mit `dbisql -batch` generiert `dbisql.bat`. Die Angabe von **APPLICATION_ARGUMENTS** ist optional.

Das folgende Beispiel zeigt eine Initialisierungsdatei für Sybase Central.

```
JRE_DIRECTORY=c:\jdk1.7.0\jre
VM_ARGUMENTS=-Xmx200m
JAR_PATHS=c:\scj\jars
ADDITIONAL_CLASSPATH=
LIBRARY_PATHS=c:\scj\bin
APPLICATION_ARGUMENTS=-screpository=c:\Users\Public\Documents\Sybase Central
16;-installdir=c:\scj
```

In diesem Szenario wird davon ausgegangen, dass eine Kopie der JRE im Verzeichnis `c:\jdk1.7.0\jre` enthalten ist. Die Sybase Central-Programmdatei und die Shared Librarys (DLLs), wie `jsyblib1600`, sind im Verzeichnis `c:\scj\bin` gespeichert. Die JAR-Dateien von SQL Anywhere sind sich im Verzeichnis `c:\scj\jars` gespeichert.

Hinweis

Beim Deployment von Anwendungen ist der Personal Datenbankserver (dbeng16) erforderlich, um Datenbanken mithilfe des Dienstprogramms `dbinit` erstellen zu können. Er ist auch erforderlich, wenn Sie Datenbanken von Sybase Central aus auf dem lokalen Computer erstellen und keine anderen Datenbankserver ausgeführt werden.

Siehe auch

- „Deployment-Assistent“ auf Seite 952

Deployment der Administrationstools unter Windows

In diesem Abschnitt wird erklärt, wie Interactive SQL (dbisql), Sybase Central (einschließlich der SQL Anywhere-, MobiLink- und UltraLite-Plug-Ins) sowie das SQL Anywhere-Konsolendienstprogramm (dbconsole) auf einem Windows-Computer ohne den Deployment-Assistenten installiert werden. Der Abschnitt richtet sich an Entwickler, die ein Installationsprogramm für diese Administrationstools erstellen wollen.

Diese Informationen gelten für alle Windows-Plattformen außer Windows Mobile. Die hier beschriebenen Anweisungen gelten speziell für die Version 16.0, nicht aber für frühere oder spätere Versionen der Software.

Hinweis

Die Weitergabe von Dateien wird durch die Lizenzvereinbarung geregelt. In diesem Kapitel enthaltene Ausführungen können die Bestimmungen Ihrer Lizenzvereinbarung weder aufheben noch ändern. Bevor Sie daher ein Deployment vornehmen, prüfen Sie bitte Ihre Lizenzvereinbarung.

Bevor Sie beginnen

Dieser Abschnitt setzt Kenntnisse der Windows-Registrierung und der REGEDIT-Anwendung voraus. Bei den Namen der Registrierungswerte wird die Groß- und Kleinschreibung berücksichtigt.

Vorsicht

Änderungen an der Windows-Registrierung sind gefährlich und Sie tun dies auf eigene Gefahr. Bevor Sie die Registrierung ändern, sollten Sie eine Sicherungskopie Ihres Systems erstellen.

Folgende Schritte sind für ein Deployment der Administrationstools erforderlich:

1. Entscheiden Sie, auf welche Elemente sich das Deployment erstrecken soll.
2. Kopieren Sie die erforderlichen Dateien.
3. Registrieren Sie die Administrationstools in Windows.
4. Aktualisieren Sie den Systempfad.
5. Registrieren Sie die Plug-Ins in Sybase Central.
6. Registrieren Sie den SQL Anywhere ODBC-Treiber in Windows.
7. Registrieren Sie die Online-Hilfedateien in Windows.

In den folgenden Abschnitten werden alle diese Schritte beschrieben.

Schritt 1: Entscheiden Sie, welche Software bereitgestellt werden soll

Sie können jede beliebige Kombination der folgenden Softwarepakete installieren:

- Interactive SQL
- Sybase Central mit dem SQL Anywhere-Plug-In
- Sybase Central mit dem MobiLink-Plug-In
- Sybase Central mit dem Relay Server-Plug-In
- Sybase Central mit dem UltraLite-Plug-In
- SQL Anywhere-Konsolendienstprogramm (dbconsole)

Die folgenden Komponenten sind auch erforderlich, wenn Sie eines der oben genannten Softwarepakete installieren:

- SQL Anywhere-ODBC-Treiber.
- Java Runtime Environment (JRE) Version 1.7.0. Je nach Zielplattform können Sie die 32-Bit-Version oder die 64-Bit-Version der JRE oder beide installieren.
- Die Java Access Bridge für das Microsoft Windows-Betriebssystem (wenn Sie Eingabehilfefunktionen der Administrationstools unter Windows-Plattformen aktivieren). Die Aktivierung dieser Funktion ermöglicht, dass die Administrationstools mit Technologien für Bildschirmlesegeräte verwendet werden können. Weitere Hinweise finden Sie unter <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136191.html>.

Die Anweisungen in den folgenden Abschnitten sind so strukturiert, dass Sie diese Softwarepakete einzeln oder zusammen ohne Konflikte installieren können.

Schritt 2: Kopieren Sie die erforderlichen Dateien

Für die Administrationstools ist eine bestimmte Verzeichnisstruktur erforderlich. Sie können den Verzeichnisbaum in ein beliebiges Verzeichnis auf einem beliebigen Laufwerk platzieren. Im Folgenden wird *c:\sa16* als Beispiel-Installationsordner verwendet. Die Software muss in einer Verzeichnisbaumstruktur mit dem folgenden Layout installiert sein:

Verzeichnis	Beschreibung
<i>sa16</i>	Der Stammordner. In den folgenden Schritten wird davon ausgegangen, dass Sie in <i>c:\sa16</i> installieren. Sie können das Verzeichnis aber beliebig auswählen (z.B. <i>C:\Programme\SQLAny16</i>).

Verzeichnis	Beschreibung
<i>sa16\Bin32</i>	Hier befinden sich die nativen 32-Bit-Windows-Komponenten, die vom Programm verwendet werden, darunter auch die Programme, die die Anwendungen starten.
<i>sa16\Bin32\jre170</i>	Die 32-Bit-Java-Laufzeitumgebung.
<i>sa16\Bin64</i>	Hier befinden sich die nativen 64-Bit-Windows-Komponenten, die vom Programm verwendet werden, darunter auch die Programme, die die Anwendungen starten.
<i>sa16\Bin64\jre170</i>	Die 64-Bit-Java-Laufzeitumgebung.
<i>sa16\Java</i>	Hier befinden sich die JAR-Dateien für die Java-Programme.

Die folgenden Tabellen listen die erforderlichen Dateien für alle Administrationstools und Sybase Central-Plug-Ins auf. Legen Sie eine Liste der erforderlichen Dateien an und kopieren Sie sie in die Verzeichnisstruktur, die oben beschrieben wurde.

Die Tabellen geben Dateien mit der Ordnerbezeichnung **BinXX** an. Es gibt 32-Bit- und 64-Bit-Versionen dieser Dateien in den Ordnern *Bin32* bzw. *Bin64*. Wenn Sie 32-Bit- und 64-Bit-Administrationstools installieren, müssen Sie die beiden Dateigruppen in jeweils entsprechenden Ordner installieren.

Die Tabellen geben Dateien mit der Bezeichnung **[LL]** an. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen hinzufügen. Weitere Hinweise finden Sie unter [Internationale Meldungsdateien und internationale kontextsensitive Hilfedateien auf Seite 999](#).

Die Administrationstools erfordern JRE 1.7.0. Eine spätere Patchversion von JRE sollte nur installiert werden, wenn dies aus bestimmten Gründen erforderlich ist.

Wenn Sie die 32-Bit-Administrationstools bereitstellen möchten, kopieren Sie die 32-Bit-Version der JRE-Dateien aus dem Verzeichnis *%SQLANY16%\Bin32\jre170*. Kopieren Sie die gesamte *jre170*-Verzeichnisstruktur einschließlich der Unterverzeichnisse.

Wenn Sie die 64-Bit-Administrationstools bereitstellen möchten, kopieren Sie die 64-Bit-Version der JRE-Dateien aus dem Verzeichnis *%SQLANY16%\Bin64\jre170*. Kopieren Sie die gesamte *jre170*-Verzeichnisstruktur einschließlich der Unterverzeichnisse.

Interactive SQL

Interactive SQL (dbisql) erfordert die folgenden Dateien.

```
BinXX\dbdsn.exe
BinXX\dbelevat16.exe
BinXX\dbicul16.dll
BinXX\dbicudt16.dll
```

```
BinXX\dbisql.com
BinXX\dbisql.exe
BinXX\dbjodbc16.dll
BinXX\dblg[LL]16.dll
BinXX\dblib16.dll
BinXX\dbodbc16.dll
BinXX\jsyblib1600.dll
BinXX\jre170\...
Java\batik*.jar
Java\isql.jar
Java\JComponents1600.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\js.jar
Java\ngdbc.jar
Java\jsyblib1600.jar
Java\pdf-transcoder.jar
Java\saip16.jar
Java\SCEditor1600.jar
Java\xerces_2_5_0.jar
Java\xml-apis-ext.jar
Java\xml-apis.jar
```

Einige der oben angegebenen Dateipfade enden mit "...". Das weist darauf hin, dass die gesamte Verzeichnisstruktur, einschließlich der Unterverzeichnisse, kopiert werden muss. Die 32-Bit-Version von Interactive SQL erfordert die 32-Bit-Version der JRE-Dateien (*Bin32\jre170*). Die 64-Bit-Version von Interactive SQL erfordert die 64-Bit-Version der JRE-Dateien (*Bin64\jre170*).

Sybase Central

Sybase Central (scjview) erfordert die folgenden Dateien.

```
BinXX\jsyblib1600.dll
BinXX\scjview.exe
BinXX\jre170\...
Java\jsyblib1600.jar
Java\salib.jar
Java\SCEditor1600.jar
Java\sybasecentral1600.jar
```

Einige der oben angegebenen Dateipfade enden mit "...". Das weist darauf hin, dass die gesamte Verzeichnisstruktur, einschließlich der Unterverzeichnisse, kopiert werden muss. Die 32-Bit-Version von Sybase Central erfordert die 32-Bit-Version der JRE-Dateien (*Bin32\jre170*). Die 64-Bit-Version von Sybase Central erfordert die 64-Bit-Version der JRE-Dateien (*Bin64\jre170*).

Sybase Central mit SQL Anywhere-Plug-In

Das SQL Anywhere-Plug-In von Sybase Central erfordert den Personal Server (*dbeng16.exe* und unterstützende Dateien) und die folgenden Dateien.

```
BinXX\dbdsn.exe
BinXX\dbfips16.dll
BinXX\dbicul16.dll
BinXX\dbicudt16.dll
BinXX\dbjodbc16.dll
BinXX\dblg[LL]16.dll
BinXX\dblib16.dll
BinXX\dbodbc16.dll
BinXX\dbput16.dll
BinXX\dbtool16.dll
```

```
BinXX\sbgse2.dll
Java\batik*.jar
Java\debugger.jar
Java\isql.jar
Java\JComponents1600.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\js.jar
Java\pdf-transcoder.jar
Java\sapplugin.jar
Java\SQLAnywhere.jpr
Java\xerces_2_5_0.jar
Java\xml-apis-ext.jar
Java\xml-apis.jar
```

Sybase Central mit MobiLink-Plug-In

Das MobiLink-Plug-In von Sybase Central erfordert die folgenden Dateien.

```
BinXX\dbdsn.exe
BinXX\dbicu16.dll
BinXX\dbicudt16.dll
BinXX\dblg[LL]16.dll
BinXX\dblib16.dll
BinXX\dbput16.dll
BinXX\dbtool16.dll
BinXX\mljodbc16.dll
Java\isql.jar
Java\JComponents1600.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\mldesign.jar
Java\mlplugin.jar
Java\MobiLink.jpr
Java\stax-api-1.0.jar
Java\velocity-dep.jar
Java\velocity.jar
Java\wstx-asl-3.2.6.jar
```

Sybase Central mit Relay Server-Plug-In

Das Relay Server-Plug-In von Sybase Central erfordert die folgenden Dateien.

```
Java\JComponents1600.jar
Java\jlogon.jar
Java\RelayServer.jpr
Java\rsplugin.jar
Java\rstool.jar
```

Sybase Central mit UltraLite-Plug-In

Das UltraLite-Plug-In von Sybase Central erfordert die folgenden Dateien.

```
BinXX\dbdsn.exe
BinXX\dbicu16.dll
BinXX\dbicudt16.dll
BinXX\dblg[LL]16.dll
BinXX\dblib16.dll
BinXX\dbput16.dll
BinXX\dbtool16.dll
BinXX\mlcrsa16.dll
BinXX\mlcrsafips16.dll
```

```

BinXX\ulfips16.dll
BinXX\ulscutil16.dll
BinXX\ulutils16.dll
Java\batik*.jar
Java\isql.jar
Java\JComponents1600.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\js.jar
Java\pdf-transcoder.jar
Java\ulplugin.jar
Java\UltraLite.jpr
Java\xerces_2_5_0.jar
Java\xml-apis-ext.jar
Java\xml-apis.jar

```

DBConsole

Die DBConsole-Anwendung erfordert die folgenden Dateien.

```

BinXX\dbconsole.exe
BinXX\dbdsn.exe
BinXX\dbelevate16.exe
BinXX\dbicul6.dll
BinXX\dbicudt16.dll
BinXX\dbjodbc16.dll
BinXX\dblg[LL]16.dll
BinXX\dblib16.dll
BinXX\dbodbc16.dll
BinXX\jsyblib1600.dll
BinXX\jre170\...
Java\DBConsole.jar
Java\JComponents1600.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\jsyblib1600.jar

```

Einige der oben angegebenen Dateipfade enden mit "...". Das weist darauf hin, dass die gesamte Verzeichnisstruktur, einschließlich der Unterverzeichnisse, kopiert werden muss. Die 32-Bit-Version von DBConsole erfordert die 32-Bit-Version der JRE-Dateien (*Bin32\jre170*). Die 64-Bit-Version von DBConsole erfordert die 64-Bit-Version der JRE-Dateien (*Bin64\jre170*).

Internationale Meldungsdateien und internationale kontextsensitive Hilfedateien

Alle angezeigten Hilfetexte und kontextsensitiven Meldungen für die Administrationstools sind in Französisch, Deutsch, Japanisch und vereinfachtem Chinesisch verfügbar. Die Ressourcen für die jeweiligen Sprachen befinden sich in getrennten Dateien. Die englischen Dateien enthalten das Kürzel **en** im Dateinamen. Die französischen Dateien haben denselben Namen, aber das Kürzel **fr** anstelle von **en**. Deutsche Dateinamen enthalten das Kürzel **de**, japanische das Kürzel **ja** und chinesische das Kürzel **zh**.

Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Meldungsdateien für die betreffenden Sprachen hinzufügen. Es gibt 32-Bit- und 64-Bit-Versionen dieser Dateien in den Ordnern Bin32 bzw. Bin64. Wenn Sie 32-Bit- und 64-Bit-Administrationstools installieren, müssen Sie die beiden Dateigruppen im jeweils entsprechenden Ordner installieren. Die lokalisierten Dateien sind:

<i>dblggen16.dll</i>	Englisch
----------------------	----------

<i>dblgde16.dll</i>	Deutsch
<i>dblgfr16.dll</i>	Französisch
<i>dblgja16.dll</i>	Japanisch
<i>dblgzh16.dll</i>	Vereinfachtes Chinesisch

Diese Dateien sind in den lokalisierten Versionen von SQL Anywhere enthalten.

Eingabehilfenkomponenten

Die Java Access Bridge stellt Eingabehilfen für die Java-basierten Administrationstools bereit (Sie können die Java Access Bridge für Microsoft Windows-Betriebssysteme auf der Java-Website von herunterladen). Nachstehend finden Sie eine Liste der Installationspfade für die Eingabehilfenkomponenten.

• 32-Bit-Administrationstools

<i>accessibility.properties</i>	<i>Bin32\jre170\lib</i>
<i>jaccess.jar</i>	<i>Bin32\jre170\lib\ext</i>
<i>access-bridge.jar</i>	<i>Bin32\jre170\lib\ext</i>
<i>JavaAccessBridge.dll</i>	<i>Bin32\jre170\bin</i>
<i>JAWTAccessBridge.dll</i>	<i>Bin32\jre170\bin</i>
<i>WindowsAccessBridge.dll</i>	%windir%\system32 (32-Bit-Windows) oder %windir%\SysWOW64 (64-Bit-Windows)

• 64-Bit-Administrationstools

<i>accessibility.properties</i>	<i>Bin64\jre170\lib</i>
<i>jaccess.jar</i>	<i>Bin64\jre170\lib\ext</i>
<i>access-bridge.jar</i>	<i>Bin64\jre170\lib\ext</i>
<i>JavaAccessBridge-64.dll</i>	<i>Bin64\jre170\bin</i>
<i>JAWTAccessBridge-64.dll</i>	<i>Bin64\jre170\bin</i>
<i>WindowsAccessBridge-64.dll</i>	%windir%\system32

Ihr Installationsprogramm muss eine *accessibility.properties*-Datei im Ordner *BinXX\jre170\lib* erstellen und muss wie unten dargestellt eine **assistive_technologies**-Option enthalten.

```
#
# Load the Java Access Bridge class into the JVM
#
assistive_technologies=com.sun.java.accessibility.AccessBridge
#screen_magnifier_present=true
```

Das SQL Anywhere-Installationsprogramm erstellt diese Datei, aber alle Zeilen werden kommentiert. Um die assistiven Technologien zu aktivieren, muss die Kommentarmarkierung der Zeile **assistive_technologies** entfernt werden (# muss entfernt werden).

Schritt 3: Registrieren Sie die Administrationstools in Windows

Sie müssen den nachstehenden Registrierungsschlüssel für die Administrationstools festlegen. Bei den Namen der Registrierungswerte wird die Groß- und Kleinschreibung berücksichtigt.

- In **HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\16.0**
 - **Location** Der voll qualifizierte Pfad zum Stamm des Installationsverzeichnis (zum Beispiel *C:\Program Files\SQL Anywhere 16*) in dem sich die Sybase Central-Dateien befinden.

Sie können die nachstehenden Registrierungsschlüssel für die Administrationstools festlegen. Die Registrierungsschlüssel sind optional. Sie müssen nur dann festgelegt werden, wenn die BS-Sprache eine andere ist als die Sprache, die von den Administrationstools verwendet wird. Bei den Namen der Registrierungsschlüssel wird die Groß- und Kleinschreibung berücksichtigt.

- In **HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\Sybase Central\16.0**
 - **Language** Der Zwei-Buchstaben-Code, der von Sybase Central verwendet wird. Es gilt einer der folgenden Werte: **EN, DE, FR, JA** bzw. **ZH** für Englisch, Deutsch, Französisch, Japanisch und vereinfachtes Chinesisch.
- In **HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\16.0**
 - **Language** Der aus zwei Buchstaben bestehende Code, der von SQL Anywhere und den anderen Administrationstools verwendet wird. Es gilt einer der folgenden Werte: **EN, DE, FR, JA** bzw. **ZH** für Englisch, Deutsch, Französisch, Japanisch und vereinfachtes Chinesisch.

Auf 64-Bit- Windows befinden sich die entsprechenden Registrierungseinträge für 32-Bit-Software in **SOFTWARE\Wow6432Node\Sybase**.

Die Pfade dürfen *nicht* mit einem Backslash enden.

Ihr Installer kann alle diese Informationen einschließen, indem Sie eine *.reg*-Datei erstellen und ausführen. Unter Verwendung des Beispiel-Installationsordners *c:\sa16* kann eine *.reg*-Beispieldatei wie folgt aussehen:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\Sybase Central\16.0]
"Language"="FR"

[HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\16.0]
```

```
"Location"="c:\\sal6"  
"Language"="FR"
```

Backslashes in Dateipfaden müssen in einer *.reg*-Datei mit einem zweiten Backslash als Escapezeichen eingegeben werden.

Schritt 4: Aktualisieren Sie den Systempfad

Damit die Administrationstools ausgeführt werden können, müssen die Verzeichnisse mit den *.exe*- und *.dll*-Dateien in den Pfad einbezogen werden. Sie müssen dem Systempfad das Verzeichnis *c:\sal6\Bin32* oder *c:\sal6\Bin64* hinzufügen.

Unter Windows wird der Systempfad im folgenden Registrierungsschlüssel gespeichert:

```
HKEY_LOCAL_MACHINE\  
SYSTEM\  
CurrentControlSet\  
Control\  
Session Manager\  
Environment\  
Path
```

Schritt 5: Erstellen Sie Verbindungsprofile für Sybase Central

Dieser Schritt enthält die Konfiguration von Sybase Central. Wenn Sie kein Deployment von Sybase Central vornehmen, können Sie diesen Schritt überspringen.

Wenn Sybase Central auf Ihrem System installiert wird, wird in der Repositorydatei ein Verbindungsprofil für **SQL Anywhere 16 Demo** erstellt. Wenn Sie keine Verbindungsprofile erstellen wollen, können Sie diesen Schritt überspringen.

Die folgenden Befehle werden zum Erstellen des **SQL Anywhere 16 Demo**-Verbindungsprofils verwendet. Verwenden Sie sie als Modell für die Erstellung Ihrer eigenen Verbindungsprofile.

```
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Name" "SQL Anywhere  
16 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/FirstTimeStart"  
"false"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Description"  
"Suitable Description"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/ProviderId"  
"sqlanywhere1600"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Provider" "SQL  
Anywhere 16"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Data/  
ConnectionProfileSettings" "DSN\eSQL^0020Anywhere^002016^0020Demo;UID  
\eDBA;PWD\^e35c624d517fb"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Data/  
ConnectionProfileName" "SQL Anywhere 16 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Data/  
ConnectionProfileType" "SQL Anywhere"
```

Die Verbindungsprofil-Zeichenfolgen und -Werte können aus der Repositorydatei extrahiert werden. Definieren Sie mit Sybase Central ein Verbindungsprofil und sehen Sie sich dann die Repositorydatei für die entsprechenden Zeilen an.

Im Folgenden sehen Sie einen Auszug aus der Repositorydatei, die mit dem oben beschriebenen Prozess erstellt wurde. Einige Einträge wurden zur einfacheren Lesbarkeit über mehrere Zeilen umgebrochen. In der Datei muss sich jeder Eintrag auf einer einzigen Zeile befinden:

```
# Version: 16.0.1154
# Thu Oct 04 12:07:53 EDT 2012
#
ConnectionProfiles/SQL Anywhere 16 Demo/Name=SQL Anywhere 16 Demo
ConnectionProfiles/SQL Anywhere 16 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 16 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 16 Demo/ProviderId=sqlanywhere1600
ConnectionProfiles/SQL Anywhere 16 Demo/Provider=SQL Anywhere 16
ConnectionProfiles/SQL Anywhere 16 Demo/Data/ConnectionProfileSettings=
    DSN\esQL^0020Anywhere^002016^0020Demo;
    UID\edBA;
    PWD\es35c624d517fb
ConnectionProfiles/SQL Anywhere 16 Demo/Data/ConnectionProfileName=
    SQL Anywhere 16 Demo
ConnectionProfiles/SQL Anywhere 16 Demo/Data/ConnectionProfileType=
    SQL Anywhere
```

Schritt 6: Registrieren Sie den SQL Anywhere-ODBC-Treiber

Sie müssen den SQL Anywhere-ODBC-Treiber installieren, damit er vom JDBC-Treiber der Administrationstools verwendet werden kann.

Siehe auch

- „ODBC-Treiberkonfiguration“ auf Seite 974

Deployment von Administrationstools unter Linux, Solaris und Mac OS X

In diesem Abschnitt wird erklärt, wie Interactive SQL (dbisql), Sybase Central (einschließlich der SQL Anywhere- und MobiLink-Plug-Ins) sowie das SQL Anywhere-Konsolendienstprogramm (dbconsole) auf Linux-, Solaris- und Mac OS X-Computern installiert werden. Der Abschnitt richtet sich an Entwickler, die ein Installationsprogramm für diese Administrationstools erstellen wollen.

Die hier beschriebenen Anweisungen gelten speziell für die Version 16.0, eventuell jedoch nicht für frühere oder spätere Versionen der Software.

Beachten Sie auch, dass das Befehlszeilen-Dienstprogramm dbisqlc unter Linux, Solaris, Mac OS X, HP-UX und AIX unterstützt wird. Siehe „Deployment von dbisqlc“ auf Seite 1015.

Hinweis

Die Weitergabe von Dateien wird durch die Lizenzvereinbarung geregelt. In diesem Kapitel enthaltene Ausführungen können die Bestimmungen Ihrer Lizenzvereinbarung weder aufheben noch ändern. Bevor Sie daher ein Deployment vornehmen, prüfen Sie bitte Ihre Lizenzvereinbarung.

Bevor Sie beginnen

Bevor Sie beginnen, müssen Sie SQL Anywhere auf einem Computer als Quelle für Programmdateien installieren. Dies ist die **Referenzinstallation** für Ihr Deployment.

Grundsätzlich sind folgende Schritte vorgesehen:

1. Entscheiden Sie, für welche Programme Sie das Deployment vornehmen wollen.
2. Kopieren Sie die erforderlichen Dateien.
3. Legen Sie Umgebungsvariable fest.
4. Registrieren Sie die Sybase Central-Plug-Ins.

In den folgenden Abschnitten werden alle diese Schritte beschrieben.

Schritt 1: Entscheiden Sie, welche Software bereitgestellt werden soll

Sie können jede beliebige Kombination der folgenden Softwarepakete installieren:

- Interactive SQL
- Sybase Central mit dem SQL Anywhere-Plug-In
- Sybase Central mit dem MobiLink-Plug-In
- SQL Anywhere-Konsolendienstprogramm (dbconsole)

Die folgenden Komponenten sind auch erforderlich, wenn Sie eines der oben genannten Softwarepakete installieren:

- SQL Anywhere-ODBC-Treiber
- Java Runtime Environment (JRE) Version 1.7.0. Es gibt 32-Bit- und 64-Bit-Versionen der JRE.

Hinweis

Um Ihre JRE-Version unter Mac OS X zu prüfen, gehen Sie in das **Apple**-Menü und klicken dort auf **Systemeinstellungen** » **Software Updates**. Klicken Sie auf **Installierte Aktualisierungen**, um eine Liste der durchgeführten Aktualisierungen aufzurufen. Wenn Java 1.7.0 nicht in der Liste enthalten ist, gehen Sie zu <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1637583.html>.

Die Anweisungen im folgenden Abschnitt sind so strukturiert, dass Sie diese fünf Softwarepakete einzeln oder insgesamt ohne Konflikte installieren können.

Schritt 2: Kopieren Sie die erforderlichen Dateien

Ihr Installationsprogramm muss eine Teilmenge der Dateien kopieren, die vom SQL Anywhere-Installationsprogramm installiert werden. Sie müssen dieselbe Verzeichnisstruktur verwenden.

Sie müssen die Berechtigungen für die Dateien beibehalten, wenn Sie sie aus der SQL Anywhere-Referenzinstallation kopieren. Im Allgemeinen können alle Benutzer und Gruppen alle Dateien lesen und ausführen.

Die Administrationstools erfordern JRE 1.7.0. Eine spätere Patchversion von JRE sollte nur installiert werden, wenn dies aus bestimmten Gründen erforderlich ist.

Unter Linux/Solaris erfordern die Administrationstools die 64-Bit- oder 32-Bit-Version der JRE (je nach Zielarchitektur). Der MobiLink-Server erfordert die 64-Bit-Version der JRE. Unter Mac OS X erfordern die Administrationstools die 64-Bit-Version der JRE. Nicht alle JRE-Plattformversionen sind Teil des Softwarepakets von SQL Anywhere. Die Plattformen, die in SQL Anywhere enthalten sind, unterstützen Linux unter x86/x64 und Solaris SPARC. Andere Plattformen sind vom jeweiligen Hersteller erhältlich. Wenn Sie beispielsweise eine Linux-Installation verwenden, kopieren Sie die gesamte *jre170*-Verzeichnisstruktur einschließlich der Unterverzeichnisse.

Wenn die benötigte Plattform im Lieferumfang von SQL Anywhere enthalten ist, kopieren Sie die JRE-Dateien aus einer SQL Anywhere-Installation. Kopieren Sie die gesamte Verzeichnisstruktur einschließlich der Unterverzeichnisse.

Die folgenden Tabellen listen die erforderlichen Dateien für alle Administrationstools und Sybase Central-Plug-Ins auf. Legen Sie eine Liste der erforderlichen Dateien an und kopieren Sie sie in die Verzeichnisstruktur, die oben beschrieben wurde.

Die Tabellen geben Dateien mit der Ordnerbezeichnung **binXX** an. Je nach Plattform gibt es 32-Bit- und 64-Bit-Versionen von diesen Dateien in den Ordnern *bin32* bzw. *bin64*. Wenn Sie 32-Bit- und 64-Bit-Administrationstools installieren, müssen Sie die beiden Dateigruppen in jeweils entsprechenden Ordner installieren.

Die Tabellen geben Dateien mit der Ordnerbezeichnung **libXX** an. Je nach Plattform gibt es 32-Bit- und 64-Bit-Versionen von diesen Dateien in den Ordnern *lib32* bzw. *lib64*. Wenn Sie 32-Bit- und 64-Bit-Administrationstools installieren, müssen Sie die beiden Dateigruppen in jeweils entsprechenden Ordner installieren.

Die Erstellung von mehreren Verknüpfungen ist für die Administrationstools und Sybase Central-Plug-Ins erforderlich.

Erstellen Sie für Linux und Solaris Symbolverknüpfungen für alle Shared Objects, deren Deployment Sie durchführen. Erstellen Sie außerdem eine Symbolverknüpfung in *\$SQLANY16/bin64* oder *\$SQLANY16/bin32*. Die Symbolverknüpfung für Linux und andere Systeme ist *jre170*. Hier einige Beispiele:

```
libdblib16_r.so -> $SQLANY16/lib32/libdblib16_r.so.1
jre170 -> $SQLANY16/bin32/jre170
```

Erstellen Sie für das MobiLink-Plug-In unter 64-Bit-Linux eine zusätzliche Symbolverknüpfung in *\$SQLANY16/bin64*. Die Symbolverknüpfung für Linux ist *jre170*.

```
jre170 -> $SQLANY16/bin64/jre170 (Linux)
```

Beachten Sie, dass unter Mac OS X Shared Objects die Erweiterung *.dylib* haben. Bei den folgenden dylib ist die Erstellung von Symbolverknüpfungen (symlink) erforderlich:

```
libdbjodbc16.jnilib -> libdbjodbc16.dylib  
libdblib16_r.jnilib -> libdblib16_r.dylib  
libdbput16_r.jnilib -> libdbput16_r.dylib  
libmljodbc16.jnilib -> libmljodbc16.dylib
```

Die Tabellen geben Dateien mit der Bezeichnung **[LL]** an. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen hinzufügen. Weitere Hinweise finden Sie unter [Internationale Meldungsdateien und internationale kontextsensitive Hilfedateien auf Seite 999](#).

Interactive SQL

Interactive SQL (dbisql) erfordert die folgenden Dateien.

```
binXX/dbisql  
binXX/jre170/...  
libXX/libdbicul16_r.so (.dylib)  
libXX/libdbicudt16.so (.dylib)  
libXX/libdbjodbc16.so (.dylib)  
libXX/libdblib16_r.so (.dylib)  
libXX/libdbodbc16_r.so (.dylib)  
libXX/libdbodml16.so (.dylib)  
libXX/libjsyblib1600_r.so (.dylib)  
res/dblg[LL]16.res  
java/batik*.jar  
java/isql.jar  
java/JComponents1600.jar  
java/jlogon.jar  
java/jodbc4.jar  
java/js.jar  
java/jsyblib1600.jar  
java/ngdbc.jar  
java/pdf-transcoder.jar  
java/saip16.jar  
java/SCEditor1600.jar  
java/xerces_2_5_0.jar  
java/xml-apis-ext.jar  
java/xml-apis.jar
```

Einige der oben angegebenen Dateipfade enden mit "...". Das weist darauf hin, dass die gesamte Verzeichnisstruktur, einschließlich der Unterverzeichnisse, kopiert werden muss. Die 32-Bit-Version von Interactive SQL erfordert die 32-Bit-Version der JRE-Dateien (*bin32\jre170*). Die 64-Bit-Version von Interactive SQL erfordert die 64-Bit-Version der JRE-Dateien (*bin64\jre170*).

Sybase Central

Sybase Central (scjview) erfordert die folgenden Dateien.

```
binXX/scjview  
binXX/jre170/...  
libXX/libjsyblib1600_r.so (.dylib)  
java/jsyblib1600.jar  
java/salib.jar  
java/SCEditor1600.jar  
java/sybasecentral1600.jar
```

Einige der oben angegebenen Dateipfade enden mit "...". Das weist darauf hin, dass die gesamte Verzeichnisstruktur, einschließlich der Unterverzeichnisse, kopiert werden muss. Die 32-Bit-Version von Sybase Central erfordert die 32-Bit-Version der JRE-Dateien (*bin32\jre170*). Die 64-Bit-Version von Sybase Central erfordert die 64-Bit-Version der JRE-Dateien (*bin64\jre170*).

Sybase Central mit SQL Anywhere-Plug-In

Das SQL Anywhere-Plug-In von Sybase Central erfordert den Personal Server (*dbeng16* und unterstützende Dateien) und die folgenden Dateien.

```
libXX/libdbfips16.so (.dylib)
libXX/libdbbicu16_r.so (.dylib)
libXX/libdbbicudt16.so (.dylib)
libXX/libdbbjodbc16.so (.dylib)
libXX/libdbblib16_r.so (.dylib)
libXX/libdbbodb16_r.so (.dylib)
libXX/libdbodm16.so (.dylib)
libXX/libdbbput16_r.so (.dylib)
libXX/libdbtasks16_r.so (.dylib)
libXX/libdbtool16_r.so (.dylib)
libXX/sbgse2.so
res/dblg[LL]16.res
java/batik*.jar
java/debugger.jar
java/isql.jar
java/JComponents1600.jar
java/jlogon.jar
java/jodbc4.jar
java/js.jar
java/pdf-transcoder.jar
java/sapplugin.jar
java/SQLAnywhere.jpr
java/xerces_2_5_0.jar
java/xml-apis-ext.jar
java/xml-apis.jar
```

Sybase Central mit MobiLink-Plug-In

Das MobiLink-Plug-In von Sybase Central erfordert die folgenden Dateien.

```
libXX/libdbbicu16_r.so (.dylib)
libXX/libdbbicudt16.so (.dylib)
libXX/libdbblib16_r.so (.dylib)
libXX/libdbbput16_r.so (.dylib)
libXX/libdbtasks16_r.so (.dylib)
libXX/libdbtool16_r.so (.dylib)
libXX/libmljodbc16.so (.dylib)
res/dblg[LL]16.res
java/isql.jar
java/JComponents1600.jar
java/jlogon.jar
java/jodbc4.jar
java/mldesign.jar
java/mlplugin.jar
java/MobiLink.jpr
java/stax-api-1.0.jar
java/velocity-dep.jar
java/velocity.jar
java/wstx-asl-3.2.6.jar
```

Sybase Central mit Relay Server-Plug-In

Das Relay Server-Plug-In von Sybase Central erfordert die folgenden Dateien.

```
java/JComponents1600.jar
java/jlogon.jar
java/RelayServer.jpr
java/rsplugin.jar
java/rstool.jar
```

Sybase Central mit UltraLite-Plug-In

Das UltraLite-Plug-In von Sybase Central ist nur unter Windows und Linux verfügbar. Das UltraLite-Plug-In von Sybase Central erfordert die folgenden Dateien.

```
libXX/libdbicul6_r.so (.dylib)
libXX/libdbicudt16.so (.dylib)
libXX/libdblib16_r.so (.dylib)
libXX/libdbput16_r.so (.dylib)
libXX/libdbtasks16_r.so (.dylib)
libXX/libdbtool16_r.so (.dylib)
libXX/libmlcrsa16.so (.dylib)
libXX/libmlcrsafips16.so (.dylib)
libXX/libulfips16.so (.dylib)
libXX/libulscutil16.so (.dylib)
libXX/libulutils16.so (.dylib)
res/dblg[LL]16.res
java/batik*.jar
java/isql.jar
java/JComponents1600.jar
java/jlogon.jar
java/jodbc4.jar
java/js.jar
java/pdf-transcoder.jar
java/ulplugin.jar
java/UltraLite.jpr
java/xerces_2_5_0.jar
java/xml-apis-ext.jar
java/xml-apis.jar
```

DBConsole

Die DBConsole-Anwendung erfordert die folgenden Dateien.

```
binXX/dbconsole
binXX/jrel70/...
libXX/libdbicul6_r.so (.dylib)
libXX/libdbicudt16.so (.dylib)
libXX/libdbjodbc16.so (.dylib)
libXX/libdblib16_r.so (.dylib)
libXX/libdbodbc16_r.so (.dylib)
libXX/libdbodm16.so (.dylib)
libXX/libjsyblib1600.so (.dylib)
res/dblg[LL]16.res
java/DBConsole.jar
java/JComponents1600.jar
java/jlogon.jar
java/jodbc4.jar
java/jsyblib1600.jar
```

Einige der oben angegebenen Dateipfade enden mit "...". Das weist darauf hin, dass die gesamte Verzeichnisstruktur, einschließlich der Unterverzeichnisse, kopiert werden muss. Die 32-Bit-Version von DBConsole erfordert die 32-Bit-Version der JRE-Dateien (*bin32\jre170*). Die 64-Bit-Version von DBConsole erfordert die 64-Bit-Version der JRE-Dateien (*bin64\jre170*).

Internationale Meldungsdateien und internationale kontextsensitive Hilfedateien

Nur für Linux-Systeme sind alle angezeigten Meldungen und kontextsensitiven Hilfetexte für die Administrationstools in Deutsch, Französisch, Japanisch und vereinfachtem Chinesisch verfügbar. Die Ressourcen für die jeweiligen Sprachen befinden sich in getrennten Dateien. Die englischen Dateien enthalten das Kürzel **en** im Dateinamen. Deutsche Dateinamen enthalten das Kürzel **de**, französische das Kürzel **fr**, japanische das Kürzel **ja** und chinesische das Kürzel **zh**.

Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Meldungsdateien für die betreffenden Sprachen hinzufügen. Die lokalisierten Dateien sind folgendermaßen benannt:

<i>dblggen16.res</i>	Englisch
<i>dblgde16_iso_1.res, dblgde16_utf8.res</i>	Deutsch (nur Linux)
<i>dblgja16_eucjis.res, dblgja16_sjis.res, dblgja16_utf8.res</i>	Japanisch (nur Linux)
<i>dblgzh16_cp936.res, dblgzh16_eucgb.res, dblgzh16_utf8.res</i>	Vereinfachtes Chinesisch (nur Linux)

Diese Dateien sind in den lokalisierten Versionen von SQL Anywhere enthalten.

Schritt 3: Legen Sie Umgebungsvariablen fest

Für die Administrationstools müssen bestimmte Umgebungsvariablen definiert oder geändert werden. In der Regel erfolgt dies in der Datei *sa_config.sh*, die vom SQL Anywhere-Installationsprogramm erstellt wird. Um die Datei *sa_config.sh* zu verwenden, kopieren Sie sie einfach und stellen Sie SQLANY16 so ein, dass es auf den Deployment-Speicherort verweist.

Andernfalls müssen Sie zum Einrichten der Umgebung folgende Schritte durchführen:

1. Setzen Sie die folgende Umgebungsvariable:

```
SQLANY16="SQL-Anywhere-install-dir"
```

2. Die PATH-Variable muss einen der folgenden Einträge enthalten:

- **Linux und Solaris (32-Bit)**

```
$SQLANY16/bin32:
```

- **Linux und Solaris (64-Bit)**

```
$SQLANY16/bin64:
```

3. LD_LIBRARY_PATH muss die folgenden Einträge enthalten:

- **Linux und Solaris (32-Bit)**

```
$SQLANY16/lib32:  
$SQLANY16/lib64:  
$SQLANY16/bin32/jre170/lib/i386/client:  
$SQLANY16/bin32/jre170/lib/i386/server:  
$SQLANY16/bin32/jre170/lib/i386:  
$SQLANY16/bin32/jre170/lib/i386/native_threads
```

- **Linux und Solaris (64-Bit)**

```
$SQLANY16/lib32:  
$SQLANY16/lib64:  
$SQLANY16/bin64/jre170/lib/amd64/client:  
$SQLANY16/bin64/jre170/lib/amd64/server:  
$SQLANY16/bin64/jre170/lib/amd64:  
$SQLANY16/bin64/jre170/lib/amd64/native_threads
```

Unter Mac OS X verwenden die Administrationstools einen Shell-Skript-Stub-Launcher namens *sa_java_stub_launcher.sh*, der bei der Installation generiert und im Ordner *Contents/MacOS* der einzelnen Java-Anwendungs-Bundles gespeichert wird. Das generierte Skript zieht die Datei *System/bin64/sa_config.sh* zum Einrichten der Umgebung heran und führt dann die *JavaApplicationStub*-Binärdatei aus. Diese startet die tatsächliche Java-Anwendung. Für das Deployment kann *sa_java_stub_launcher.sh* je nach Umgebung geändert werden. Sie können den Namen des Skripts ändern, indem Sie die Datei *Infos.plist* im Java-Anwendungs-Bundle sowie den Wert der Zeichenfolge der Schlüssel-*CFBundleExecutable* ändern.

Schritt 4: Erstellen Sie Verbindungsprofile für Sybase Central

Dieser Schritt enthält die Konfiguration von Sybase Central. Wenn Sie kein Deployment von Sybase Central vornehmen, können Sie diesen Schritt überspringen.

Wenn Sybase Central auf Ihrem System installiert wird, wird in der Repositorydatei ein Verbindungsprofil für **SQL Anywhere 16 Demo** erstellt. Wenn Sie keine Verbindungsprofile erstellen wollen, können Sie diesen Schritt überspringen.

Die folgenden Befehle werden zum Erstellen des **SQL Anywhere 16 Demo**-Verbindungsprofils verwendet. Verwenden Sie sie als Modell für die Erstellung Ihrer eigenen Verbindungsprofile.

```
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Name" "SQL Anywhere  
16 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/FirstTimeStart"  
"false"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Description"  
"Suitable Description"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/ProviderId"  
"sqlanywhere1600"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Provider" "SQL  
Anywhere 16"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Data/  
ConnectionProfileSettings" "DSN\eSQL^0020Anywhere^002016^0020Demo;UID  
\eDBA;PWD\xe35c624d517fb"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Data/  
ConnectionProfileName" "SQL Anywhere 16 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 16 Demo/Data/  
ConnectionProfileType" "SQL Anywhere"
```

Die Verbindungsprofil-Zeichenfolgen und -Werte können aus der Repositorydatei extrahiert werden. Definieren Sie mit Sybase Central ein Verbindungsprofil und sehen Sie sich dann die Repositorydatei für die entsprechenden Zeilen an.

Im Folgenden sehen Sie einen Auszug aus der Repositorydatei, die mit dem oben beschriebenen Prozess erstellt wurde. Einige Einträge wurden zur einfacheren Lesbarkeit über mehrere Zeilen umgebrochen. In der Datei muss sich jeder Eintrag auf einer einzigen Zeile befinden:

```
# Version: 16.0.1154
# Thu Oct 04 12:07:53 EDT 2012
#
ConnectionProfiles/SQL Anywhere 16 Demo/Name=SQL Anywhere 16 Demo
ConnectionProfiles/SQL Anywhere 16 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 16 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 16 Demo/ProviderId=sqlanywhere1600
ConnectionProfiles/SQL Anywhere 16 Demo/Provider=SQL Anywhere 16
ConnectionProfiles/SQL Anywhere 16 Demo/Data/ConnectionProfileSettings=
    DSN\eSQL^0020Anywhere^002016^0020Demo;
    UID\eDBA;
    PWD\e35c624d517fb
ConnectionProfiles/SQL Anywhere 16 Demo/Data/ConnectionProfileName=
    SQL Anywhere 16 Demo
ConnectionProfiles/SQL Anywhere 16 Demo/Data/ConnectionProfileType=
    SQL Anywhere
```

Konfiguration der Administrationstools

Mithilfe der Initialisierungsdatei *OEM.ini* können Sie festlegen, welche Funktionen von den Administrationstools angezeigt oder aktiviert werden. Diese Datei muss sich im gleichen Verzeichnis wie die JAR-Dateien befinden, die von den Administrationstools verwendet werden (z.B. *C:\Program Files\SQL Anywhere 16\Java*). Wenn diese Datei nicht gefunden wird, werden Standardwerte verwendet. Für Werte, die in *OEM.ini* fehlen, werden ebenfalls Standardwerte verwendet.

Hinweis

Wenn Sie ein erneutes Deployment der Administrationstools durchführen, können die Tools nicht nach SQL Anywhere-Software-Updates suchen. Die Option **Updates suchen** erscheint nicht in durch erneutes Deployment erstellten Versionen.

Im Folgenden finden Sie eine *OEM.ini*-Beispieldatei:

```
[errors]
# reportErrors type is boolean, default = true
reportErrors=true

[updates]
# checkForUpdates type is boolean, default = true
checkForUpdates=true

[preferences]
directory=preferences_files_directory

[dbisql]
allowPasswordsInFavorites=true
defaultShowResultsForAllStatements=false
defaultShowMultipleResultSets=false
```

```
disableExecuteAll=false
fastLauncherEnabled=true
# lockedPreferences is assigned a comma-separated
# list of one or more of the following option names:
#   autoCommit
#   autoRefetch
#   commitOnExit
#   disableResultsEditing
#   executeToolBarButtonSemantics
#   fastLauncherEnabled
#   maximumDisplayedRows
#   showMultipleResultSets
#   showResultsForAllStatements
lockedPreferences=showMultipleResultSets,commitOnExit
```

Jede Zeile, die mit dem Zeichen # beginnt, ist eine Kommentarzeile und wird ignoriert. Bei den angegebenen Optionsnamen und -werten wird die Groß- und Kleinschreibung berücksichtigt.

Die Datei *OEM.ini* ist in die folgenden Abschnitte unterteilt:

[errors]

Die in diesem Abschnitt festgelegte Option gilt für alle Administrationstools.

Wenn **reportErrors** den Wert FALSE hat, zeigt das Administrationstool dem Benutzer bei einem Absturz der Software kein Fenster an, mit dem er aufgefordert wird, Fehlerinformationen an den technischen Support zu senden. Stattdessen wird das Standard-Fenster angezeigt.

[updates]

Die in diesem Abschnitt festgelegte Option gilt für alle Administrationstools.

Wenn **checkForUpdates** den Wert FALSE hat, prüft das Administrationstool nicht automatisch auf SQL Anywhere-Software-Updates bzw. bietet dem Benutzer nicht die Option, dies nach eigenem Ermessen zu tun.

[preferences]

Die in diesem Abschnitt festgelegte Option gilt für alle Administrationstools.

Legen Sie mit der **directory**-Option das Verzeichnis fest, das von den Administrationstools verwendet wird, um benutzerspezifische Konfigurationsdateien zu speichern. Diese Dateien enthalten Informationen zu den Einstellungen und zum Verlauf der Administrationstools. Beispiel: Sie enthalten möglicherweise den Interactive SQL-Anweisungsverlauf, zuletzt geöffnete Dateien oder gespeicherten Fensterpositionen.

Sie müssen einen voll qualifizierten Verzeichnisnamen angeben (z.B. *c:\work\prefs*), der nicht mit einem Pfadtrennzeichen (Backslash unter Windows, Schrägstrich unter Linux, Unix und Mac OS X) endet.

Unter Windows ist die Standardeinstellung für das Verzeichnis der Benutzervoreinstellungen *%appdata%\sybase*. Mit dieser Einstellung können mehrere Benutzer auf einem System ihr eigenes Verzeichnis für die Registrierung der Voreinstellungen haben. Das Außerkraftsetzen dieser Einstellung in der Datei *OEM.ini* deaktiviert diese Fähigkeit.

[performancedata]

Die in diesem Abschnitt festgelegte Option gilt nur für Interactive SQL und Sybase Central.

- **showPerformanceDataUI** Wenn **showPerformanceDataUI** den Wert FALSE hat, stellen die Administrationstools für den Benutzer nicht die Option bereit, die automatische Einsendung von Performancedaten an das Softwareentwicklungsteam zu aktivieren oder zu deaktivieren. Siehe [„Regelmäßig Performancedaten senden“ \[SQL Anywhere Server - Datenbankadministration\]](#).

[dbisql]

Die in diesem Abschnitt festgelegten Optionen gelten nur für Interactive SQL.

- **allowPasswordsInFavorites** Wenn **allowPasswordsInFavorites** den Wert FALSE hat, entfernt Interactive SQL das Kontrollkästchen **Das Verbindungskennwort speichern** aus dem Fenster **Zu Favoriten hinzufügen**. Die Standardeinstellung ist TRUE, was bedeutet, dass das Kontrollkästchen vorhanden ist. Siehe [„SQL-Skriptdateien, SQL-Anweisungen und Verbindungen zu den Favoriten hinzufügen“ \[SQL Anywhere Server - Datenbankadministration\]](#).
- **defaultShowResultsForAllStatements** Diese Option ändert die Standardeinstellungen für die Art, wie die Ergebnisse in Interactive SQL angezeigt werden. Wenn **defaultShowResultsForAllStatements** auf TRUE gesetzt wird, lautet die Standardeinstellung für alle Datenbanktypen, Ergebnisse für alle Anweisungen anzuzeigen. Wenn diese Option auf FALSE gesetzt wird, lautet die Standardeinstellung für alle Datenbanktypen, nur die Ergebnismenge der letzten Anweisung anzuzeigen.

Ein optionales Präfix kann verwendet werden, um die Wirkung auf einen bestimmten Datenbanktyp zu begrenzen. Wenn kein Präfix vorhanden ist, gilt die Direktive für alle Datentypen. Die Präfixe lauten:

- **SQLAnywhere.**
- **UltraLite.**
- **SybaseIQ.**
- **HANA.**

Die folgende Optionseinstellung betrifft beispielsweise nur die Standardeinstellungen für SQL Anywhere-Datenbanken.

```
SQLAnywhere.defaultShowResultsForAllStatements=true
```

- **defaultShowMultipleResultSets** Diese Option ändert die Standardeinstellungen für die Art, wie die Ergebnisse in Interactive SQL angezeigt werden. Wenn **defaultShowMultipleResultSets** auf TRUE gesetzt wird, lautet die Standardeinstellung für alle Datenbanktypen, mehrere Ergebnismengen anzuzeigen. Wenn diese Option auf FALSE gesetzt wird, lautet die Standardeinstellung für alle Datenbanktypen, nur die erste Ergebnismenge anzuzeigen.

Ein optionales Präfix kann verwendet werden, um die Wirkung auf einen bestimmten Datenbanktyp zu begrenzen. Wenn kein Präfix vorhanden ist, gilt die Direktive für alle Datentypen. Die Präfixe lauten:

- **SQLAnywhere.**
- **SybaseIQ.**

- **disableExecuteAll** Wenn **disableExecuteAll** auf TRUE gesetzt wird, werden die Menüoption **SQL » Ausführen** und die Schnell taste F5 in Interactive SQL deaktiviert. Wenn die

Symbolleistenschaltfläche **Ausführen** für die Option **Ausführen** konfiguriert ist, wird sie ebenfalls deaktiviert. Aus diesem Grund sollten Sie in Interactive SQL die Symbolschaltfläche "Ausführen" auf **Markierte Anweisungen ausführen** festlegen und anschließend in der Datei *OEM.ini* die Option "executeToolBarButtonSemantics" setzen, damit die Benutzer die Symbolschaltfläche **Ausführen** nicht ändern können. Siehe „[SQL-Anweisungen ausführen \(Interactive SQL\)](#)“ [*SQL Anywhere Server - Datenbankadministration*].

- **fastLauncherEnabled** Wenn **fastLauncherEnabled** auf TRUE gesetzt wird, wird die Schnellladerfunktion aktiviert. Wenn Sie die Schnellladerfunktion deaktivieren möchten, setzen Sie die Option auf FALSE. Siehe „[Schnelllader-Option](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **lockedPreferences** Sie können die Einstellungen von Interactive SQL-Optionen sperren, sodass Benutzer sie nicht ändern können. Bei den Optionsnamen wird die Groß- und Kleinschreibung berücksichtigt. Im Folgenden finden Sie ein Beispiel:

```
[dbisql]
lockedPreferences=autoCommit
```

Sie können verhindern, dass Benutzer die folgenden Interactive SQL-Optionseinstellungen ändern:

- **autoCommit** Verhindert, dass der Benutzer die Option **Nach jeder Anweisung festschreiben** ändert. Siehe „[auto_commit-Option \[Interactive SQL\]](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **autoRefetch** Verhindert, dass der Benutzer die Option **Ergebnisse automatisch erneut abfragen** ändert. Siehe „[auto_refetch-Option \[Interactive SQL\]](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **commitOnExit** Verhindert, dass der Benutzer die Option **Beim Beenden oder Trennen festschreiben** ändert. Siehe „[commit_on_exit-Option \[Interactive SQL\]](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **disableResultsEditing** Verhindert, dass der Benutzer die Option **Bearbeitung deaktivieren** ändert.
- **executeToolBarButtonSemantics** Verhindert, dass der Benutzer das Verhalten der Symbolschaltfläche **Ausführen** ändert. Siehe „[SQL-Anweisungen ausführen \(Interactive SQL\)](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **fastLauncherEnabled** Verhindert, dass der Benutzer die Schnelllader-Option ändert. Siehe „[Schnelllader-Option](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **maximumDisplayedRows** Verhindert, dass der Benutzer die Option **Maximale Anzahl von anzuzeigenden Zeilen** ändert. Siehe „[isql_maximum_displayed_rows-Option \[Interactive SQL\]](#)“ [*SQL Anywhere Server - Datenbankadministration*].
- **showMultipleResultSets** Verhindert, dass der Benutzer die Option **Nur die erste Ergebnismenge anzeigen** oder **Alle Ergebnismengen anzeigen** ändert. Darüber hinaus wird durch diese Einstellung verhindert, dass der Benutzer die Option "isql_show_multiple_result_sets"

festlegt. Siehe „[isql_show_multiple_result_sets-Option \[Interactive SQL\]](#)“ [*SQL Anywhere Server - Datenbankadministration*].

- **showResultsForAllStatements** Verhindert, dass der Benutzer die Option **Ergebnisse der letzten Anweisung anzeigen** oder **Ergebnisse aller Anweisungen anzeigen** ändert. Siehe „[Rückgabe von mehreren Ergebnismengen](#)“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*].

Deployment von dbisqlc

Wenn die bereitgestellte Anwendung auf Computern mit begrenzten Ressourcen läuft und eine Abfrageausführung oder ein Skript-Tool erfordert, können Sie die dbisqlc-Programmdatei anstelle von Interactive SQL (dbisql) bereitstellen. Allerdings ist dbisqlc veraltet und wird nicht mehr weiterentwickelt. Beachten Sie auch, dass dbisqlc nicht alle Funktionen von Interactive SQL enthält und dass die Kompatibilität zwischen beiden Systemen nicht gewährleistet ist.

Das dbisqlc-Programm erfordert die clientseitigen Embedded SQL-Standardbibliotheken.

Siehe auch

- „[dbisqlc-Dienstprogramm \(nicht mehr empfohlen\)](#)“ [*SQL Anywhere Server - Datenbankadministration*]
- „[Interactive SQL-Dienstprogramm \(dbisql\)](#)“ [*SQL Anywhere Server - Datenbankadministration*]

Deployment der Dokumentation

In der Beschreibung weiter unten wird die Bezeichnung **[LL]** verwendet. Die Dokumentation steht in verschiedenen Sprachen zur Verfügung. Ersetzen Sie **[LL]** durch den Sprachcode für die Dokumentation, die Sie bereitstellen (z.B. **en**, **de**, **jp** usw.).

Für Windows stehen zwei Dokumentationsoptionen zur Verfügung. Entweder verwenden Sie die DCX-Hilfe – in diesem Fall müssen keine Hilfedateien per Deployment bereitgestellt werden. (Die Hilfe steht auf der dcx.sybase.com-Website zur Verfügung.) Oder Sie stellen die Hilfe im HTML-Format bereit.

Die HTML-basierte Hilfedokumentation wird bei der Installation unter `%SQLANY16%\Documentation` installiert.

```
Documentation\sqlanywhere_[LL]16.map
Documentation\[LL]\htmlhelp\dbadmin16.chm
Documentation\[LL]\htmlhelp\dbprogramming16.chm
Documentation\[LL]\htmlhelp\dbreference16.chm
Documentation\[LL]\htmlhelp\dbspatial16.chm
Documentation\[LL]\htmlhelp\dbusage16.chm
Documentation\[LL]\htmlhelp\mlclient16.chm
Documentation\[LL]\htmlhelp\mlserver16.chm
Documentation\[LL]\htmlhelp\mlsisync16.chm
Documentation\[LL]\htmlhelp\mlstart16.chm
Documentation\[LL]\htmlhelp\relayserver16.chm
Documentation\[LL]\htmlhelp\sachanges16.chm
Documentation\[LL]\htmlhelp\sacshelp16.chm
Documentation\[LL]\htmlhelp\saerrors16.chm
Documentation\[LL]\htmlhelp\saintrol16.chm
Documentation\[LL]\htmlhelp\sausinghelp16.chm
```

```
Documentation\[LL]\htmlhelp\sqlanywhere_[LL]16.chm
Documentation\[LL]\htmlhelp\sqlremotel6.chm
Documentation\[LL]\htmlhelp\uladmin16.chm
Documentation\[LL]\htmlhelp\ulcl6.chm
Documentation\[LL]\htmlhelp\uldotnet16.chm
Documentation\[LL]\htmlhelp\ulj16.chm
```

Bei Linux, Unix und Mac OS X stehen zwei Dokumentationsoptionen zur Verfügung. Entweder verwenden Sie die DCX-Hilfe – in diesem Fall müssen keine Hilfedateien per Deployment bereitgestellt werden. (Die Hilfe steht auf der dcx.sybase.com-Website zur Verfügung.) Oder Sie stellen die Eclipse-Hilfe bereit.

Die Eclipse-Hilfe kann lokal mit dem Dokumentations-Installationsprogramm installiert werden (*sa16_doc_[LL]_linux_x86+x64.[Version].[build].tar.gz*). Es gibt Versionen für verschiedene Sprachen.

Wenn Sie zum Beispiel *sa16_doc_en_linux_x86+x64.1600.2406.tar.gz* herunterladen, kann der folgende Befehl zum Installieren der Dokumentation verwendet werden.

```
/setup -ss -sqlany-dir $SQLANY16
```

Dadurch wird die Hilfe im Hintergrund an dem durch die Umgebungsvariable angegebenen Speicherort installiert.

Deployment des Datenbankservers

Sie können das Deployment eines Datenbankservers vornehmen, indem Sie den Endbenutzern den SQL Anywhere-Installer bereitstellen. Wenn der Endbenutzer die richtigen Installationsoptionen wählt, verfügt er garantiert über alle erforderlichen Dateien.

Die einfachste Möglichkeit zum Deployment eines Personal Datenbankservers oder eines Netzwerk-Datenbankservers ist die Verwendung des **Deployment-Assistenten**. Weitere Hinweise finden Sie unter „[Deployment-Assistent](#)“ auf Seite 952.

Um einen Datenbankserver zu betreiben, müssen Sie eine Gruppe von Dateien installieren. Die Dateien werden in der folgenden Tabelle angeführt. Die Weitergabe dieser Dateien unterliegt den Bestimmungen der Lizenzvereinbarung. Sie müssen vorher feststellen, ob Sie das Recht haben, die Dateien des Datenbankservers weiterzugeben.

Windows	Linux/Unix	Mac OS X
<i>dbeng16.exe</i>	<i>dbeng16</i>	<i>dbeng16</i>
<i>dbeng16.lic</i>	<i>dbeng16.lic</i>	<i>dbeng16.lic</i>
<i>dbsrv16.exe</i>	<i>dbsrv16</i>	<i>dbsrv16</i>
<i>dbsrv16.lic</i>	<i>dbsrv16.lic</i>	<i>dbsrv16.lic</i>
<i>dbserv16.dll</i>	<i>libdbserv16_r.so</i> , <i>libdb-tasks16_r.so</i>	<i>libdbserv16_r.dylib</i> , <i>libdb-tasks16_r.dylib</i>

Windows	Linux/Unix	Mac OS X
<i>dbscript16.dll</i>	<i>libdbscript16_r.so</i>	<i>libdbscript16_r.dylib</i>
<i>dblg[LL]16.dll</i>	<i>dblg[LL]16.res</i>	<i>dblg[LL]16.res</i>
<i>dbghelp.dll</i>	k.A.	k.A.
<i>dbctr16.dll</i>	k.A.	k.A.
<i>dbextf.dll</i> ¹	<i>libdbextf.so</i> ¹	<i>libdbextf.dylib</i> ¹
<i>dbicu16.dll</i> ²	<i>libdbicu16_r.so</i> ²	<i>libdbicu16_r.dylib</i> ²
<i>dbicudt16.dll</i> ^{2, 3}	<i>libdbicudt16.so</i> ²	<i>libdbicudt16.dylib</i> ²
<i>sqlany.cvf</i>	<i>sqlany.cvf</i>	<i>sqlany.cvf</i>
<i>dbrsakup16.dll</i> ⁴	<i>libdbrsakup16_r.so</i> ⁴	<i>libdbrsakup16_r.dylib</i> ⁴
<i>dbodbc16.dll</i> ⁵	<i>libdbodbc16.so</i> ⁵	<i>libdbodbc16.dylib</i> ⁵
<i>dbjodbc16.dll</i> ⁵	<i>libdbjodbc16.so</i> ⁵	<i>libdbjodbc16.dylib</i> ⁵
k.A.	<i>libdbodbc16_n.so</i> ⁵	<i>libdbodbc16_n.dylib</i> ⁵
k.A.	<i>libdbodbc16_r.so</i> ⁵	<i>libdbodbc16_r.dylib</i> ⁵
<i>dbjdbc16.dll</i> ⁶	<i>libdbjdbc16.so</i> ⁶	<i>libdbjdbc16.dylib</i> ⁶
<i>java\sajdbc4.jar</i> ⁶	<i>java/sajdbc4.jar</i> ⁶	<i>java/sajdbc4.jar</i> ⁶
<i>java\sajvm.jar</i> ⁶	<i>java/sajvm.jar</i> ⁶	<i>java/sajvm.jar</i> ⁶
<i>dbcis16.dll</i> ⁷	<i>libdbcis16.so</i> ⁷	<i>libdbcis16.dylib</i> ⁷
<i>libsybbr.dll</i> ⁸	<i>libsybbr.so</i> ⁸	<i>libsybbr.dylib</i> ⁸

¹ Nur erforderlich, wenn Sie die nicht mehr empfohlene Version der erweiterten Systemprozeduren und Funktionen benutzen (xp_*). Das *use_old_dbextf.sql*-Skript muss für Datenbanken verwendet werden, die die nicht mehr empfohlene Version verwenden.

² Nur erforderlich, wenn der Datenbankzeichensatz ein Mehrbyte-Zeichensatz ist oder wenn die UCA-Kollationssequenz verwendet wird.

³ Unter Windows Mobile hat die bereitzustellende Datei den Namen *dbicudt16.dat*.

⁴ Nur für verschlüsselte TDS-Verbindungen erforderlich.

⁵ Nur erforderlich, wenn Sie Java in der Datenbank mit Datenbanken der Version 10 verwenden.

⁶ Nur erforderlich, wenn Sie Java in der Datenbank verwenden.

⁷ Nur für Ferndatenzugriff erforderlich.

⁸ Nur für Archivsicherungen erforderlich.

Hinweise

- Je nach Konfiguration müssen Sie das Deployment des Personal Datenbankservers (dbeng16) oder des Netzwerk-Datenbankservers (dbsrv16) vornehmen.
- Sie müssen die separate zugehörige Lizenzdatei (*dbeng16.lic* oder *dbsrv16.lic*) einbeziehen, wenn Sie das Deployment eines Datenbankservers durchführen. Die Lizenzdateien befinden sich im gleichen Verzeichnis wie die Server-Programmdateien.
- Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).
- Die Java VM-JAR-Datei (*sajvm.jar*) ist nur erforderlich, wenn der Datenbankserver die Funktionalität von Java in der Datenbank verwenden soll.
- Die Tabelle enthält keine Dateien, die für die Ausführung von Dienstprogrammen wie dbbackup erforderlich sind.

Hinweise zum Deployment von Datenbank-Dienstprogrammen im System finden Sie unter [„Deployment von Administrationstools“ auf Seite 992](#).

- Die Datenbankserveroption **xd** ist in bereitgestellten Anwendungen sinnvoll, da sie verhindert, dass der Datenbankserver der Standarddatenbankserver wird. Verbindungen werden nur akzeptiert, wenn der Servername in der Zeichenfolge für die Verbindung enthalten ist.

Weitere Hinweise zur Option **xd** finden Sie unter [„Datenbankserveroption -xd“ \[SQL Anywhere Server - Datenbankadministration\]](#).

Weitere Hinweise zum Angeben des Servernamens finden Sie unter [„Verbindungsparameter ServerName \(Server\)“ \[SQL Anywhere Server - Datenbankadministration\]](#).

Robustheit bei Intel-Speichertreibern verbessern

Verbessern Sie die Robustheit gegen Stromausfälle bei Systemen, die bestimmte Intel-Speichertreiber verwenden, indem Sie den EnableFlush-Parameter in der Registrierung angeben. Wenn dieser Parameter nicht vorhanden ist, kann Datenverlust eintreten und bei Stromausfall können Datenbanken beschädigt werden.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Überprüfen Sie, ob der folgende Registrierungseintrag vorhanden ist:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\iastor\Parameters\
```

2. Wenn der Registrierungseintrag vorhanden ist, fügen Sie einen REG_DWORD-Wert namens EnableFlush im Schlüssel hinzu und ordnen Sie ihm den Datenwert 1 zu.

3. Überprüfen Sie, ob der folgende Registrierungseintrag vorhanden ist:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\iastorv\Parameters\
```

4. Wenn der Registrierungseintrag vorhanden ist, fügen Sie einen REG_DWORD-Wert namens EnableFlush im Schlüssel hinzu und ordnen Sie ihm den Datenwert 1 zu.

Ergebnisse

Der EnableFlush-Parameter ist angegeben und dies verhindert Datenverlust bei einem Stromausfall.

Beispiel

Das folgende Beispiel zeigt eine Registrierungsdatei, die den EnableFlush-Parameter an beiden möglichen Positionen angibt:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iastor\Parameters]
"EnableFlush"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iastorv\Parameters]
"EnableFlush"=dword:00000001
```

Ereignisprotokollmeldungen formatieren

Stellen Sie sicher, dass Meldungen, die unter Windows vom Server in das Ereignisprotokoll geschrieben werden, richtig formatiert sind, indem Sie einen Registrierungsschlüssel erstellen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Erstellen Sie den Registrierungsschlüssel.

Für die 32-Bit-Version des Servers müssen Sie den folgenden Registrierungsschlüssel erstellen:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 16.0
```

Für die 64-Bit-Version des Servers müssen Sie den folgenden Registrierungsschlüssel erstellen:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY64 16.0
```

2. In diesem Schlüssel fügen Sie einen REG_SZ-Wert mit der Bezeichnung EventMessageFile hinzu und ordnen ihm den Datenwert des voll qualifizierten Speicherorts von *dbngen16.dll*, beispielsweise *C:\Program Files\SQL Anywhere 16\Bin32\dbngen16.dll* zu.

Im folgenden Beispiel wird der EventMessageFile-Schlüssel für das Verzeichnis *c:\sa16\bin32* angegeben:

```
"EventMessageFile"="c:\sa16\bin32\dbngen16.dll"
```

Die englische Sprachen-DLL, *dbngen16.dll*, kann unabhängig von der Deploymentsprache angegeben werden.

3. Erstellen Sie den Registrierungsschlüssel, um zu sichern, dass Meldungen, die von MESSAGE...TO EVENT LOG-Anweisungen in das Ereignisprotokoll geschrieben werden, richtig formatiert werden.

Für die 32-Bit-Version des Servers müssen Sie den folgenden Registrierungsschlüssel erstellen:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQLANY 16.0 Admin
```

Für die 64-Bit-Version des Servers müssen Sie den folgenden Registrierungsschlüssel erstellen:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQLANY64 16.0 Admin
```

4. In diesem Schlüssel fügen Sie einen REG_SZ-Wert mit der Bezeichnung EventMessageFile hinzu und ordnen ihm den Datenwert des voll qualifizierten Speicherorts von *dbngen16.dll*, beispielsweise *C:\Program Files\SQL Anywhere 16\Bin32\dbngen16.dll*, zu.

Im folgenden Beispiel wird der EventMessageFile-Schlüssel für das Verzeichnis *c:\sa16\bin32* angegeben:

```
"EventMessageFile"="c:\sa16\bin32\dbngen16.dll"
```

Die englische Sprachen-DLL, *dbngen16.dll*, kann unabhängig von der Deploymentsprache angegeben werden.

5. Erstellen Sie den folgenden Registrierungsschlüssel, um die Unterdrückung von Logeinträgen zu steuern:

```
Software\Sybase\SQL Anywhere\16.0\EventLogMask
```

Der Schlüssel kann im Eintrag HKEY_CURRENT_USER oder HKEY_LOCAL_MACHINE platziert werden.

6. Erstellen Sie einen REG_DWORD-Wert namens EventLogMask und weisen Sie ihm einen Bitmaskenwert zu, der die internen Bitwerte für die verschiedenen Windows-Ereignistypen enthält.

Die folgenden Bittypen werden vom SQL Anywhere-Datenbankserver verwendet:

EVENTLOG_ERROR_TYPE	0x0001
EVENTLOG_WARNING_TYPE	0x0002
EVENTLOG_INFORMATION_TYPE	0x0004

Wenn beispielsweise der EventLogMask-Schlüssel auf Null gesetzt wurde, werden überhaupt keine Meldungen angezeigt. Eine bessere Einstellung ist 1, damit keine Informations- und Warnmeldungen, wohl aber Fehler angezeigt werden. Die Standardeinstellung (kein Eintrag vorhanden) lautet, dass alle Meldungstypen angezeigt werden. Hier ein Beispiel für den Registrierungseintrag:

```
"EventLogMask"=dword:00000007
```

Ergebnisse

Ereignisprotokollmeldungen werden in der gewünschten Sprache formatiert und entsprechend dem Bitmaskenwert angezeigt, der dem EventLogMask-Registrierungsschlüssel zugeordnet ist.

Beispiel

Das folgende Beispiel veranschaulicht eine Beispiel-Registrierungsdatei, die von der 32-Bit-Version des Servers gesendete Ereignisprotokollmeldungen formatiert:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 16.0]
"EventMessageFile"="c:\sa16\bin32\dbngen16.dll"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 16.0 Admin]
"EventMessageFile"="c:\sa16\bin32\dbngen16.dll"

[HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\16.0]
"EventLogMask"=dword:00000003
```

In diesem Beispiel wird die englische Version der DLL, die unter *c:\sa 16\bin32\dbngen16.dll* gespeichert ist, verwendet, um Ereignisprotokollmeldungen zu formatieren. Der Bitmaskenwert der EventLogMask gibt an, dass nur Fehler und Warnungen erfasst werden sollen, keine Informationsmeldungen.

Hinweise zu Mac OS X

Unter Mac OS X ermitteln DBLauncher und SyncConsole anhand der Benutzerstandardwerte die SQL Anywhere-Installation. Das SQL Anywhere-Installationsprogramm schreibt den Speicherort der Installation in das Repository für die Standardwerte des Benutzers, der die Installation ausführt. Andere Benutzer werden aufgefordert, das Installationsverzeichnis anzugeben, wenn DBLauncher das erste Mal verwendet wird. Außerdem können diese Einstellung im Fensterausschnitt "Voreinstellungen" geändert werden.

Deployment von Datenbanken

Das Deployment einer Datenbankdatei erfolgt, indem Sie die Datenbankdatei auf der Festplatte des Endbenutzers installieren.

Wenn der Datenbankserver sauber herunterfährt, müssen Sie zum Deployment der Datenbankdatei kein Transaktionslog hinzufügen. Wenn der Endbenutzer die Datenbank startet, wird ein neues Transaktionslog erstellt.

Für Anwendungen mit SQL Remote muss die Datenbank in einem einwandfrei synchronisierten Zustand erstellt werden, sodass kein Transaktionslog benötigt wird. Sie können dafür das Extraktionsdienstprogramm verwenden.

Siehe auch

- „Extraktion von entfernten Datenbanken“ [[SQL Remote](#)]

Hinweise für internationale Deployments

Wenn Sie ein weltweites Deployment einer Datenbank vornehmen, müssen Sie die Sprachumgebungen berücksichtigen, in denen die Datenbank verwendet werden soll. Verschiedene Sprachumgebungen können unterschiedliche Sortierreihenfolgen und Regeln für den Textvergleich verwenden. Die bereitgestellte Datenbank könnte z.B. mit der 1252LATIN1-Kollation erstellt worden sein, die für einige Umgebungen, in denen die Datenbank verwendet werden soll, möglicherweise nicht geeignet ist.

Da die Kollation einer Datenbank nach ihrer Erstellung nicht mehr geändert werden kann, empfiehlt es sich, die Datenbank während der Installationsphase zu erstellen und sie anschließend mit dem Schema und den erforderlichen Daten zu füllen. Die Datenbank kann während der Installation erstellt werden, indem Sie entweder mithilfe des Dienstprogramms dbinit verwenden oder den Datenbankserver mit der Dienstprogrammdatei dbinit starten und eine CREATE DATABASE-Anweisung absetzen. Anschließend können Sie SQL-Anweisungen verwenden, um das Schema zu erstellen und alle weiteren erforderlichen Aktionen zum Einrichten der neu angelegten Datenbank auszuführen.

Wenn Sie entscheiden, die UCA-Kollation zu verwenden, können Sie mit dem Dienstprogramm dbinit oder der CREATE DATABASE-Anweisung zusätzliche Optionen der Kollationsanpassung festlegen, um eine bessere Kontrolle über die Sortierung und den Zeichenvergleich zu ermöglichen. Diese Optionen werden in der Form von *Schlüsselwort=Wert*-Paaren, die in Klammern gesetzt werden, hinter dem Kollationsnamen angegeben. Mit der CREATE DATABASE-Anweisung können Sie z.B. mit folgender Syntax eine Kollationsanpassung festlegen:

```
CHAR COLLATION 'UCA( locale=es;case=respect;accent=respect )'
```

Alternativ dazu können Sie mehrere Datenbankvorlagen erstellen, eine für jede Sprachumgebung, in der die Datenbank verwendet werden soll. Dies ist sinnvoll, wenn die Gruppe der Sprachumgebungen, in denen ein Deployment der Datenbank vorgenommen wird, relativ klein ist. Sie können das Installationsprogramm auswählen lassen, welche Datenbank installiert wird.

Siehe auch

- „CREATE DATABASE-Anweisung“ [[SQL Anywhere Server - SQL-Referenzhandbuch](#)]
- „Dienstprogramm Initialisierung (dbinit)“ [[SQL Anywhere Server - Datenbankadministration](#)]
- „Hinweise zur Kollation“ [[SQL Anywhere Server - Datenbankadministration](#)]

Datenbank-Deployment auf schreibgeschützten Datenträgern

Sie können Datenbanken auf schreibgeschützten Datenträgern verteilen, wie etwa CD-ROM, sofern Sie sie im schreibgeschützten Modus ausführen.

Weitere Hinweise zum Ausführen von Datenbanken im schreibgeschützten Modus finden Sie unter [„Datenbankserveroption -r“ \[SQL Anywhere Server - Datenbankadministration\]](#).

Wenn Sie die Datenbank ändern müssen, kopieren Sie die Datenbank von der CD-ROM an einen Speicherort, an dem sie geändert werden kann, z.B. auf eine Festplatte.

DLL-Registrierung unter Windows

Einige DLL -Dateien erfordern eine Registrierung, wenn sie für die Verwendung mit SQL Anywhere bereitgestellt werden. Bei Windows Vista oder späteren Versionen von Windows müssen Sie den SQL Anywhere-Agenten für Vorgänge mit erweiterten Berechtigungen einbeziehen (*dbelevate16.exe*), der die erforderlichen Privilegien für die Registrierung bzw. Deregistrierung von DLLs unterstützt. Außerdem erfordert der ODBC-Treiber unter Windows Mobile keine Registrierung.

Es gibt verschiedene Möglichkeiten, diese DLLs zu registrieren, darunter ein Installationsskript oder die Verwendung des Dienstprogramms *regsvr32* unter Windows bzw. des Dienstprogramms *regsvrce* unter Windows Mobile. Sie können auch einen Befehl in eine Batchdatei einbeziehen.

In der folgenden Tabelle werden die DLLs aufgelistet, die beim Deployment unter Windows eine Registrierung erfordern:

Datei	Beschreibung
<i>dbctrs16.dll</i>	Die Zähler für den SQL Anywhere-Systemmonitor.
<i>dbodbc16.dll</i>	Der SQL Anywhere-ODBC-Treiber.
<i>dboledb16.dll</i>	Der SQL Anywhere OLE DB-Provider.
<i>dboledba16.dll</i>	Das Schema-Assist-Modul des SQL Anywhere OLE DB-Providers (nur Windows).

Beispiel

Führen Sie den folgenden Befehl aus, um den ODBC-Treiber mit dem Dienstprogramm *regsvr32* unter Windows zu registrieren:

```
regsvr32 dbodbc16.dll
```

Deployment der Unterstützung für externe Umgebungen

Die folgenden Tabellen fassen die Komponenten zusammen, deren Deployment vorgenommen werden muss, um Unterstützung für externe Aufrufen in SQL Anywhere zu erhalten.

Externe ESQL/ODBC-Aufrufe

Komponente	Windows	Linux/Unix	Mac OS X
ESQL- und ODBC-Startprogramm	<i>dbexternc16.exe</i>	<i>dbexternc16</i>	<i>dbexternc16</i>
Bridge	<i>dbextenv16.dll</i>	<i>libdbextenv16_r.so</i>	<i>libdbextenv16_r.dylib</i>
SQL Anywhere C-API-Laufzeitumgebung	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB	<i>dblib16.dll</i>	<i>libdblib16_r.so</i>	<i>libdblib16_r.dylib</i>

Hinweise zu weiteren Dateien, die von Embedded SQL-Anwendungen benötigt werden, finden Sie unter [„Deployment von Embedded SQL-Clients“ auf Seite 980](#).

Hinweise zu weiteren Dateien, die von ODBC-Anwendungen benötigt werden, finden Sie unter [„Deployment von ODBC-Clients“ auf Seite 970](#).

Externe Java-Aufrufe

Komponente	Windows	Linux/Unix	Mac OS X
Java-Installation (Drittanbieter)	<i>java.exe</i>	<i>java</i>	<i>java</i>
Startprogramm	<i>sajvm.jar</i>	<i>sajvm.jar</i>	<i>sajvm.jar</i>
SQL Anywhere-JDBC-Treiber (serverseitige Aufrufe)	<i>dbjdbc16.dll</i>	<i>libdbjdbc16.so</i>	<i>libdbjdbc16.dylib</i>

Hinweise zu weiteren Dateien, die von JDBC-Anwendungen benötigt werden, finden Sie unter [„Deployment von JDBC-Clients“ auf Seite 982](#).

Externe .NET CLR-Aufrufe

Komponente	Windows	Linux/Unix	Mac OS X
.NET 3.5 oder später	(von Microsoft)	k.A.	k.A.
SQL Anywhere .NET 3.5-oder 4.0-Provider	(von Sybase)	k.A.	k.A.
.NET CLR-Bridge	<i>dbextclr16.exe</i>	k.A.	k.A.
Bridge	<i>dbextenv16.dll</i>	k.A.	k.A.
.NET CLR-Unterstützung	<i>dbclrenv16.dll</i>	k.A.	k.A.

Komponente	Windows	Linux/Unix	Mac OS X
DBLIB	<i>dblib16.dll</i>	k.A.	k.A.

Externe Perl-Aufrufe

Komponente	Windows	Linux/Unix	Mac OS X
Perl-Installation (Drittanbieter)	<i>perl.exe</i>	<i>perl</i>	<i>perl</i>
Perl-Startprogramm	<i>perlenv.pl</i>	<i>perlenv.pl</i>	<i>perlenv.pl</i>
Bridge	<i>dbxenv16.dll</i>	<i>libdbxenv16_r.so</i>	<i>libdbxenv16_r.dylib</i>
SQL Anywhere C-API-Laufzeitumgebung	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB	<i>dblib16.dll</i>	<i>libdblib16_r.so</i>	<i>libdblib16_r.dylib</i>

Hinweise zu weiteren Dateien, die von Perl-Anwendungen benötigt werden, finden Sie unter „DBD::SQLAnywhere“ auf Seite 651.

Externe PHP-Aufrufe

Komponente	Windows	Linux/Unix	Mac OS X
PHP-Installation (Drittanbieter)	<i>php.exe</i>	<i>php</i>	<i>php</i>
PHP-Startprogramm	<i>phpenv.php</i>	<i>phpenv.php</i>	<i>phpenv.php</i>
Bridge	<i>dbxenv16.dll</i>	<i>libdbxenv16_r.so</i>	<i>libdbxenv16_r.dylib</i>
Externe PHP 5.1.x-Aufrufe	<i>php-5.1.[1-*)_sqlanywhere_extenv16.dll</i>	<i>php-5.1.[1-*)_sqlanywhere_extenv16_r.so</i> oder Build aus Quellcode	Build aus Quellcode
Externe PHP 5.2.x-Aufrufe	<i>php-5.2.[0-*)_sqlanywhere_extenv16.dll</i>	<i>php-5.2.[0-*)_sqlanywhere_extenv16_r.so</i> oder Build aus Quellcode	Build aus Quellcode
Externe PHP 5.3.x-Aufrufe	<i>php-5.3.[0-*)_sqlanywhere_extenv16.dll</i>	<i>php-5.3.[0-*)_sqlanywhere_extenv16_r.so</i> oder Build aus Quellcode	Build aus Quellcode

Komponente	Windows	Linux/Unix	Mac OS X
Externe PHP 5.4.x-Aufrufe	<i>php-5.4.[0-*)_sqlanywhere_extenv16.dll</i>	<i>php-5.4.[0-*)_sqlanywhere_extenv16_r.so</i> oder Build aus Quellcode	Build aus Quellcode
DBLIB (mit Threading)	<i>dblib16.dll</i>	<i>libdblib16_r.so</i>	<i>libdblib16_r.dylib</i>
Thread-Unterstützungsbibliothek	k.A.	<i>libdbtasks16_r.so</i>	<i>libdbtasks16_r.dylib</i>

Die Windows-Versionen der Unterstützungsdateien für die externe PHP-Umgebung sind nur für 32 Bit. Um diese Dateien auf Ihrem Windows-Computer installieren zu können, müssen Sie 32-Bit-Serverkomponenten als Teil der SQL Anywhere-Installation auswählen.

Hinweise zu weiteren Dateien, die von PHP-Anwendungen benötigt werden, finden Sie unter „[SQL Anywhere-PHP-Unterstützung](#)“ auf Seite 669.

Siehe auch

- „Die externen ESQL- und ODBC-Umgebungen“ auf Seite 625
- „Die externe Java-Umgebung“ auf Seite 635
- „Die externe CLR-Umgebung“ auf Seite 622
- „Die externe PERL-Umgebung“ auf Seite 640
- „Die externe PHP-Umgebung“ auf Seite 644

Verschlüsselungs-Deployment

Die folgende Tabelle gibt einen Überblick über die Komponenten, die Verschlüsselungsfunktionen in SQL Anywhere unterstützen.

Verschlüsselungsoption	Verschlüsselungstyp	Im Modul enthalten	Lizenzierbar
Datenbankverschlüsselung	AES	<i>dbserv16.dll</i> <i>libdbserv16_r.so</i>	enthalten ¹
Datenbankverschlüsselung	FIPS-zertifizierte AES	<i>dbfips16.dll</i>	getrennt lizenziert ²
Transportschichtssicherheit	RSA	<i>dbrsa16.dll</i> <i>libdbrsa16.so</i> <i>libdbrsa16.dylib</i> <i>libdbrsa16_r.dylib</i>	enthalten ¹

Verschlüsselungsoption	Verschlüsselungstyp	Im Modul enthalten	Lizenzierbar
Transportschichtssicherheit	FIPS-zertifizierte RSA	<i>dbfips16.dll, sbgse2.dll</i>	getrennt lizenziert ²

¹ Verschlüsselung mit AES und RSA sind in SQL Anywhere enthalten und erfordern keine getrennte Lizenz, die Bibliotheken sind aber nicht FIPS-zertifiziert.

² Die Software für die Verschlüsselung mit FIPS-zertifizierter Technologie muss separat bestellt werden.

Deployment von Anwendungen mit eingebetteten Datenbanken

In diesem Abschnitt finden Sie Informationen zum Deployment von eingebetteten Datenbankanwendungen, bei denen sich Anwendung und Datenbank auf demselben Computer befinden.

Eine eingebettete Datenbankanwendung enthält folgende Komponenten:

- **Clientanwendung** Dazu gehören die Komponenten des SQL Anywhere-Clients.
- **Datenbankserver** Der SQL Anywhere-Personal-Datenbankserver.
- **Datenbankdatei** Sie müssen eine Datenbankdatei bereitstellen, die die von der Anwendung verwendeten Daten enthält.
- **SQL Remote** Wenn Ihre Anwendung die Replikation mit SQL Remote benutzt, müssen Sie das Deployment des SQL Remote-Nachrichtenagenten vornehmen.

Siehe auch

- [„Deployment des Datenbankservers“ auf Seite 1016](#)
- [„Anforderungen für das Deployment von Clientanwendungen“ auf Seite 960](#)

Deployment von Personal Server

Wenn Sie das Deployment einer Anwendung vornehmen, die den Personal Server benutzt, müssen Sie sowohl das Deployment der Komponenten der Clientanwendung vornehmen, als auch der Komponenten des Datenbankservers.

Die Sprachen-Ressourcenbibliothek (*dblggen16.dll* oder *dblgde11.dll*) wird von Client und Server gemeinsam genutzt. Sie brauchen nur ein Exemplar dieser Datei.

Es wird empfohlen, die Installationsvorgaben von SQL Anywhere einzuhalten und die Dateien für Client und Server in demselben Verzeichnis zu installieren.

Deployment von Datenbanken

Wenn Sie beim Deployment Ihrer Anwendung der Datenbank auch Dienstprogramme (wie z.B. dbbackup) bereitstellen müssen, benötigen Sie die Programmdatei des Dienstprogramms mit folgenden zusätzlichen Dateien:

Beschreibung	Windows	Linux/Unix	Mac OS X
Bibliothek der Datenbanktools	<i>dbtool16.dll</i>	<i>libdbtool16_r.so, libdbtasks16_r.so</i>	<i>libdbtool16_r.dylib, libdbtasks16_r.dylib</i>
Schnittstellenbibliothek	<i>dblib16.dll</i>	<i>libdblib16_r.so</i>	<i>libdblib16_r.dylib</i>
Sprachen-Ressourcenbibliothek	<i>dblg[LL]16.dll</i>	<i>dblg[LL]16.res</i>	<i>dblg16.res</i>
Fenster "Verbinden"	<i>dbcon16.dll</i>		
Physische Speicherbibliothek der Version 9 oder früher	<i>dboftsp.dll</i>	<i>libdboftsp_r.so</i>	k.A.

Hinweise

- Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en, de, jp** usw.).
- Bei Anwendungen ohne Multi-Threading unter Linux und Unix können Sie *libdbtasks16.so* und *libdblib16.so* verwenden.
- Bei Anwendungen ohne Multi-Threading unter Mac OS X können Sie *libdbtasks16.dylib* und *libdblib16.dylib* verwenden.
- Unter Windows wird der Unterstützungscode für die Fenster **ODBC-Konfiguration für SQL Anywhere** und **Mit SQL Anywhere verbinden** (*dbcon16.dll*) benötigt, wenn Ihre Endbenutzer eigene Datenquellen erstellen oder beim Herstellen der Verbindung mit der Datenbank Benutzer-IDs und Kennwörter eingeben müssen sowie wenn das Fenster "Verbinden" aus anderen Gründen angezeigt werden muss.
- Manche Dienstprogramme benötigen die physische Speicherbibliothek der Version 9 oder früher (dblog, dbtran, dberase) für den Zugriff auf Logdateien, die von Softwareversionen vor Version 10.0.0 erstellt wurden. Wenn Sie diese Dienstprogramme nicht bereitstellen, dann ist diese Bibliothek nicht erforderlich.
- Wenn Sie Unterstützung beim Entladen von Datenbanken der Version 9.0 oder früher benötigen, müssen Sie das dbunload-Dienstprogramm zum Entladen von Datenbanken mit den unter [„Deployment](#)

von Unterstützung des Entladens für Datenbanken der Version 9 und früher“ beschriebenen Dateien verwenden.

- Der Personal Datenbankserver (dbeng16) ist für das Erstellen von Datenbanken mit dem Dienstprogramm dbinit erforderlich. Er ist auch erforderlich, wenn Sie Datenbanken von Sybase Central aus auf dem lokalen Computer erstellen und keine anderen Datenbankserver ausgeführt werden.

Siehe auch

- „Deployment des Datenbankservers“ auf Seite 1016

Deployment von Unterstützung des Entladens für Datenbanken der Version 9 und früher

Wenn Ihre Anwendung Datenbanken der Version 9.0 oder früher in das aktuelle Format konvertieren soll, benötigen Sie das Dienstprogramm zum Entladen von Datenbanken (dbunload) sowie die folgenden zusätzlichen Dateien:

Beschreibung	Windows	Linux/Unix	Mac OS X
Entladeunterstützung für Datenbankversionen vor Version 10.0	<i>dbunlspt.exe</i>	<i>dbunlspt</i>	<i>dbunlspt</i>
Meldungs-Ressourcenbibliothek	<i>dbus[LL].dll</i>	<i>dbus[LL].res</i>	<i>dbus[LL].res</i>
Entladen-Skriptdatei	<i>optdeflt.sql</i>	<i>optdeflt.sql</i>	<i>optdeflt.sql</i>
Entladen-Skriptdatei	<i>opttemp.sql</i>	<i>opttemp.sql</i>	<i>opttemp.sql</i>
Entladen-Skriptdatei	<i>unloadold.sql</i>	<i>unloadold.sql</i>	<i>unloadold.sql</i>

Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung [LL]. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen hinzufügen. Die Meldungsdateien werden nachstehend aufgelistet.

Windows-Meldungsdateien

<i>dbusde.dll</i>	Deutsch
<i>dbusen.dll</i>	Englisch
<i>dbuses.dll</i>	Spanisch
<i>dbusfr.dll</i>	Französisch

<i>dbusit.dll</i>	Italienisch
<i>dbusja.dll</i>	Japanisch
<i>dbusko.dll</i>	Koreanisch
<i>dbuslt.dll</i>	Litauisch
<i>dbuspl.dll</i>	Polnisch
<i>dbuspt.dll</i>	Portugiesisch
<i>dbusru.dll</i>	Russisch
<i>dbustw.dll</i>	Traditionelles Chinesisch
<i>dbusuk.dll</i>	Ukrainisch
<i>dbuszh.dll</i>	Vereinfachtes Chinesisch

Linux-Meldungsdateien

<i>dbusde_iso_1.res, dbusde_utf8.res, dbusen.res</i>	Deutsch
<i>dbusen.res</i>	Englisch
<i>dbusja_eucjis.res, dbusja_sjis.res, dbusja_utf8.res</i>	Japanisch
<i>dbuszh_cp936.res, dbuszh_eucgb.res, dbuszh_utf8.res</i>	Chinesisch

Diese Dateien sind in den lokalisierten Versionen von SQL Anywhere enthalten.

Zusätzlich zu diesen Dateien benötigen Sie auch die unter [„Deployment von Datenbanken“](#) auf Seite 1028 beschriebenen Dateien.

Deployment von SQL Remote

Wenn Sie das Deployment des Nachrichtenagenten von SQL Remote vornehmen, müssen folgende Dateien installiert werden:

Beschreibung	Windows	Linux/Solaris	Mac OS X
Nachrichten-agent	<i>dbremote.exe</i>	<i>dbremote</i>	<i>dbremote</i>

Beschreibung	Windows	Linux/Solaris	Mac OS X
Kodierungs-/ Dekodierungs- bibliothek	<i>dbencod16.dll</i>	<i>libdbencod16_r.so.1</i>	<i>libdbencod16_r.dylib</i>
Bibliothek für FILE-Nachricht- tenverbindun- gen ¹	<i>dbfile16.dll</i>	<i>libdbfile16_r.so.1</i>	<i>libdbfile16_r.dylib</i>
Bibliothek für FTP-Nachricht- tenverbindun- gen ¹	<i>dbftp16.dll</i>	<i>libdbftp16_r.so.1</i>	<i>libdbftp16_r.dylib</i>
Sprachen-Res- ourcenbiblio- thek	<i>dblg[LL]16.dll</i>	<i>dblg[LL]16.res</i>	<i>dblg[LL]16.res</i>
Schnittstellen- bibliothek	<i>dblib16.dll</i>	<i>libdblib16_r.so.1</i>	<i>libdblib16_r.dylib</i>
Bibliothek für SMTP-Nach- richtenverbin- dungen ¹	<i>dbsmtp16.dll</i>	<i>libdbsmtp16_r.so.1</i>	<i>libdbsmtp16_r.dylib</i>
Bibliothek der Datenbanktools	<i>dbtool16.dll</i>	<i>libdbtool16_r.so.1</i>	<i>libdbtool16_r.dylib</i>
Thread-Unter- stützungsbiblio- thek	k.A.	<i>libdbtasks16_r.so.1</i>	<i>libdbtasks16_r.dylib</i>

¹ Nehmen Sie ein Deployment der Bibliothek nur für die Nachrichtenverbindung vor, die Sie benutzen.

Eine Sprachen-Ressourcenbibliotheksdatei muss ebenfalls einbezogen werden. Die oben stehende Tabelle zeigt Dateien mit der Bezeichnung **[LL]**. Es gibt mehrere Meldungsdateien, die jeweils eine andere Sprache unterstützen. Um die Unterstützung für verschiedene Sprachen zu installieren, müssen Sie die Ressourcendateien für die betreffenden Sprachen einbeziehen. Ersetzen Sie **[LL]** durch den Sprachcode (z.B. **en**, **de**, **jp** usw.).

Es wird empfohlen, die Installationsvorgaben von SQL Anywhere einzuhalten und die Dateien für SQL Remote in demselben Verzeichnis zu installieren wie SQL Anywhere.

Index

Symbole

- install-Option
 - SetupVSPackage,960
- salocation Option
 - SetupVSPackage,960
- uninstall-Option
 - SetupVSPackage,963
- .NET
 - Datensteuerung,88
 - Deployment,960
 - SQL Anywhere .NET-Datenprovider verwenden,41
- .NET, Datenbank-Programmierschnittstellen
 - praktische Einführung,87
- .NET-API
 - Info,41
- .NET-Datenprovider
 - Aktualisieren von Daten,47
 - Beispielprojekte ausführen,43
 - dbdata.dll,74
 - Deployment,72
 - Einfügen von Daten,47
 - Entity Framework-Unterstützung,67
 - Fehlerbehandlung,66
 - Funktionen,42
 - für das Deployment erforderliche Dateien,72
 - gespeicherte Prozeduren ausführen,63
 - iAnywhere.Data.SQLAnywhere-Provider,42
 - INFO,41
 - Löschen von Daten,47
 - mit einer Datenbank verbinden,45
 - POOLING-Option,46
 - Protokollierungsunterstützung,75
 - Referenzen auf Providerklassen im Quellcode,44
 - Referenzen hinzufügen,44
 - registrieren,74
 - Systemanforderungen,72
 - Transaktionsverarbeitung,64
 - unterstützte Sprachen,41
 - unterstützte Versionen,41
 - Verarbeitungsroutine bei Ausnahmefehlern,66
 - Verbindungspools,46
 - Verwendung des Codebeispiels "Simple",79
 - Verwendung des Codebeispiels "Table Viewer",83

- Zeitwerte erhalten,63
- Zugriff auf Daten,47
- 64 Bit
 - ODBC,382
- @HttpMethod
 - Zugriff auf HTTP-Header,775
- @HttpQueryString
 - Zugriff auf HTTP-Header,775
- @HttpStatus
 - Zugriff auf HTTP-Header,775
- @HttpURI
 - Zugriff auf HTTP-Header,775
- @HttpVersion
 - Zugriff auf HTTP-Header,775

A

- a_backup_db-Struktur [Datenbanktools-API]
 - Beschreibung,894
- a_change_log-Struktur [Datenbanktools-API]
 - Beschreibung,897
- a_create_db-Struktur [Datenbanktools-API]
 - Beschreibung,899
- a_db_info-Struktur [Datenbanktools-API]
 - Beschreibung,903
- a_db_version_info-Struktur [Datenbanktools-API]
 - Beschreibung,905
- a_dblic_info-Struktur [Datenbanktools-API]
 - Beschreibung,906
- a_dbtools_info-Struktur [Datenbanktools-API]
 - Beschreibung,907
- a_log_file_info-Struktur [Datenbanktools-API]
 - Beschreibung,907
- a_name-Struktur [Datenbanktools-API]
 - Beschreibung,908
- a_remote_sql-Struktur [Datenbanktools-API]
 - Beschreibung,908
- a_sqlany_bind_param-Struktur [SQL Anywhere-C-API]
 - Beschreibung,593
- a_sqlany_bind_param_info-Struktur [SQL Anywhere-C-API]
 - Beschreibung,594
- a_sqlany_column_info-Struktur [SQL Anywhere-C-API]
 - Beschreibung,594
- a_sqlany_data_direction-Enumeration [SQL Anywhere-C-API]

- Beschreibung,590
- a_sqlany_data_info-Struktur [SQL Anywhere-C-API]
 - Beschreibung,595
- a_sqlany_data_type-Enumeration [SQL Anywhere-C-API]
 - Beschreibung,591
- a_sqlany_data_value-Struktur [SQL Anywhere-C-API]
 - Beschreibung,596
- a_sqlany_native_type-Enumeration [SQL Anywhere-C-API]
 - Beschreibung,591
- a_sync_db-Struktur [Datenbanktools-API]
 - Beschreibung,917
- a_syncpub-Struktur [Datenbanktools-API]
 - Beschreibung,927
- a_sysinfo-Struktur [Datenbanktools-API]
 - Beschreibung,928
- a_table_info-Struktur [Datenbanktools-API]
 - Beschreibung,928
- a_translate_log-Struktur [Datenbanktools-API]
 - Beschreibung,929
- a_truncate_log-Struktur [Datenbanktools-API]
 - Beschreibung,933
- a_validate_db-Struktur [Datenbanktools-API]
 - Beschreibung,934
- Abbruchverarbeitung
 - in externen Funktionen,614
- Abfragen
 - ADO-Recordset-Objekt,341
 - ADO-Recordset-Objekt und Cursor,342
 - einzelne Zeilen,515
- Abort-Eigenschaft
 - SARowsCopiedEventArgs-Klasse [SQL Anywhere .NET-API],307
- Abrufe
 - Array-Abrufe,518
 - mehrzeilige Abrufe,518
- Abrufen
 - ODBC,390
 - SQLDA,513
- Accept
 - Zugriff auf HTTP-Header,774
- Accept-Charset
 - Zugriff auf HTTP-Header,774
- Accept-Encoding
 - Zugriff auf HTTP-Header,774
- Accept-Language
 - Zugriff auf HTTP-Header,774
- AcceptCharset-Option
 - Beispiel,787
- access-bridge.jar
 - Deployment von Java-basierten Administrationstools,1000
- accessibility.properties
 - Deployment von Java-basierten Administrationstools,1000
- ActiveX-Datenobjekte
 - Info,338
- Add-Methode
 - SABulkCopyColumnMappingCollection-Klasse [SQL Anywhere .NET-API],123
 - SAPparameterCollection-Klasse [SQL Anywhere .NET-API],289
- addBatch
 - PreparedStatement-Klasse,442
 - Statement-Klasse,437
- AddRange-Methode
 - SAPparameterCollection-Klasse [SQL Anywhere .NET-API],293
- addShutdownHook
 - Java VM-Shutdown-Hooks,417
- AddWithValue-Methode
 - SAPparameterCollection-Klasse [SQL Anywhere .NET-API],294
- Administrationstools
 - dbtools,871
 - Deployment,992
- ADO
 - Abfragen,341
 - Aktualisierungen,343
 - Befehle,339
 - Befehlsobjekt,339
 - Cursor,33
 - Cursortypen,15
 - Daten mit einem Cursor aktualisieren,343
 - Einführung in die Programmierung,337
 - Info,338
 - Recordset-Objekt,341
 - Recordset-Objekt und Cursortypen,342
 - SQL-Anweisungen in Anwendungen verwenden,1
 - Transaktionen,344
 - Verbindungen,338
 - Verbindungsobjekt,338
- ADO-Befehlsobjekt
 - ADO,339

ADO-Recordset-Objekt
 ADO,341
 Daten aktualisieren,343
 ADO-Verbindungsobjekt
 ADO,338
 ADO.NET
 Autocommit-Modus,37
 Autocommit-Verhalten steuern,37
 Cursorunterstützung,33
 Deployment,960
 Info,41
 SQL-Anweisungen in Anwendungen verwenden,1
 vorbereitete Anweisungen,4
 ADO.NET-API
 Info,41
 Aktualisierungen
 Cursor,343
 All-Eigenschaft
 SACommLinksOptionsBuilder-Klasse [SQL
 Anywhere .NET-API],166
 alloc_sqllda-Funktion
 Info,532
 alloc_sqllda_noind-Funktion
 Info,533
 allowPasswordsInFavorites
 konfigurierbare Option,1011
 ALTER EXTERNAL ENVIRONMENT JAVA
 Beispiel,405
 an_erase_db-Struktur [Datenbanktools-API]
 Beschreibung,936
 an_extfn_api
 externer Funktionsaufruf, Schnittstelle,606
 an_extfn_result_set_column_data
 externer Funktionsaufruf, Schnittstelle,611
 an_extfn_result_set_column_info
 externer Funktionsaufruf, Schnittstelle,610
 an_extfn_result_set_info
 externer Funktionsaufruf, Schnittstelle,609
 an_extfn_value
 externer Funktionsaufruf, Schnittstelle,608
 an_unload_db-Struktur [Datenbanktools-API]
 Beschreibung,937
 an_upgrade_db-Struktur [Datenbanktools-API]
 Beschreibung,943
 Ändern
 Webdienste,762
 Anforderungen
 abbrechen,538
 Open Client-Anwendungen,749
 Anforderungen abbrechen
 Embedded SQL,531
 Anforderungsverarbeitung
 Embedded SQL,531
 Angeführte Bezeichner
 sql_needs_quotes-Funktion,559
 ANSI-ODBC-Treiber
 Unix,366
 Anweisungen
 INSERT,2
 Anweisungs-Handle
 ODBC,368
 Anwendungen
 Deployment,947
 Deployment von .NET-Clients,960
 Deployment von Clientanwendungen,960
 Deployment von JDBC-Clients,982
 Deployment von ODBC,970
 Deployment von OLE DB,964
 Deployment von Open Client,992
 Deployment von PHP-Clients,984
 SQL,1
 Anwendungen mit einem Thread
 Unix,363
 Anwendungen mit mehreren Threads
 Unix,363
 Anwendungen mit Threadverarbeitung
 Unix,949
 Apache
 installieren,984
 PHP-Erweiterung auswählen,986
 apache_files.txt
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 apache_license_1.1.txt
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 apache_license_2.0.txt
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 APIs
 ADO-API,337
 ADO.NET,41
 C-API,565

- JDBC-API,419
 - ODBC-API,359
 - OLE DB-API,337
 - Perl DBD::SQLAnywhere-API,651
 - PHP,684
 - Python-Datenbank-API,661
 - Ruby-APIs,717
 - Sybase Open Client-API,747
 - AppInfo-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],196
 - Arbeitstabellen
 - Cursor-Performance,28
 - Architektur
 - OData Server,455
 - Array-Abrufe
 - ESQL,518
 - Info,518
 - ARRAY-Klausel
 - FETCH-Anweisung verwenden,518
 - Asensitive Cursor
 - Beispiele für Aktualisieren,20
 - Beispiele für Löschen,18
 - Einführung,18
 - Info,25
 - ASP.NET
 - Provider, Info,97
 - Providerschema der Datenbank hinzufügen,98
 - SQL Anywhere ASP.NET-Provider registrieren,100
 - SQL Anywhere ASP.NET-Provider verwenden,97
 - Verbindungszeichenfolge registrieren,99
 - Whitepaper unter Sybase.com,97
 - Aufrufen, externe Bibliotheken aus Prozeduren
 - Info,601
 - Ausführen, SQL-Anweisungen
 - in Anwendungen,1
 - Ausnahmebedingungen
 - SQL Anywhere-.NET-Datenprovider,66
 - Ausnahmen
 - Java,403
 - Ausrichtung von Daten
 - ODBC,386
 - Autocommit
 - Einstellungen für Transaktionen,37
 - ODBC,370
 - steuern,37
 - autocommit
 - JDBC,434
 - Autocommit-Modus
 - Implementierung,39
 - autoCommit-Option
 - konfigurierbare Option,1011
 - AUTOINCREMENT
 - Suche nach der zuletzt eingefügten Zeile,12
 - autoRefetch
 - konfigurierbare Option,1011
 - AutoStart-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],196
 - AutoStop-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],196
 - Autotune-Enumeration [Datenbanktools-API]
 - Beschreibung,888
- ## B
- Batch-Einfügungen
 - JDBC,442
 - BatchSize-Eigenschaft
 - SABulkCopy-Klasse [SQL Anywhere .NET-API],112
 - BatchUpdateException
 - JDBC,437
 - batik-awt-util.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-bridge.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-codec.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-css.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-dom.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-ext.jar

-
- Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-extension.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-gui-util.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-gvt.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-parser.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-script.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-svg-dom.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-svggen.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-swing.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-transcoder.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-util.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - batik-xml.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - Befehle
 - ADO-Befehlsobjekt,339
 - Befehlszeilen-Dienstprogramme
 - Syntax für den SQL-Präprozessor (sqlpp),469
 - BeginExecuteNonQuery-Methode
 - SACommand-Klasse [SQL Anywhere .NET-API],134
 - BeginExecuteReader-Methode
 - SACommand-Klasse [SQL Anywhere .NET-API],135
 - BeginTransaction-Methode
 - SAConnection-Klasse [SQL Anywhere .NET-API],171
 - Behälter
 - OData-Metadaten,463
 - Beispiele
 - .NET-Datenprovider,79
 - DBTools-Programm,876
 - dynamische Cursor in Embedded SQL,480
 - Embedded SQL,479,481
 - Embedded SQL-Anwendungen,477
 - OData Server und Clients,464
 - ODBC,366
 - SimpleViewer,87
 - statische Cursor in Embedded SQL,478
 - Benutzerdatenquellen
 - Windows,977
 - Benutzerdefinierte Funktionen
 - CLR,622
 - Berechtigungen
 - Prozeduren, die externe Funktionen aufrufen,601
 - Beschädigte Datenbanken
 - EnableFlush-Registrierungseintrag,1018
 - Beschreiben
 - Ergebnismengen,35
 - Beschreibung
 - NCHAR-Spalten in Embedded SQL,508
 - Betriebssysteme
 - Dateinamen,950
 - Bezeichner
 - Anführungszeichen nötig,559
 - Bibliothek
 - dblib16.lib,474
 - dblibtm.lib,474
 - libdblib16.so,474
 - libdblib16_r.so,474
 - libdbtasks16.so,474
 - libdbtasks16_r.so,474
 - Bibliotheken

- dbtlstm.lib,872
- dbtool16.lib,872
- Embedded SQL,474
 - externe Bibliotheken aus gespeicherten Prozeduren oder Funktionen aufrufen,601
 - Importbibliotheken verwenden,872
- Bibliotheksfunktionen
 - Embedded SQL,532
- BIGINT-Datentyp
 - Embedded SQL,487
- Binärdatentypen
 - Embedded SQL,487
- Binden von Parametern
 - ODBC,379
 - SQLBindParameter-Funktion,380
- Bindevariablen
 - Info,501
- Bindungsparameter
 - vorbereitete Anweisungen,3
- BIT-Datentyp
 - Embedded SQL,487
- Bitfelder
 - verwenden,876
- BLOBs
 - Embedded SQL,522
 - in Embedded SQL abrufen,524
 - in Embedded SQL senden,526,527
- Block-Cursor
 - Info,11
 - ODBC,16
- Broadcast-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],318
- BroadcastListener-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],318
- Bulk-Library
 - Info,748
- BulkCopyTimeout-Eigenschaft
 - SABulkCopy-Klasse [SQL Anywhere .NET-API],112
- Bytecode
 - Java-Klassen,402

C

C#

- Unterstützung im .NET-Datenprovider,41

- C++-Anwendungen
 - dbtools,871
 - Embedded SQL,467
- C, Programmiersprache
 - Datentypen,487
 - Embedded SQL-Anwendungen,467
- C-API
 - Einführung in die Programmierung,565
- C_ESQL32-Schlüsselwort
 - externe Umgebung,626
- C_ESQL64-Schlüsselwort
 - externe Umgebung,626
- C_ODBC32-Schlüsselwort
 - externe Umgebung,626
- C_ODBC64-Schlüsselwort
 - externe Umgebung,626
- CALL-Anweisung
 - Embedded SQL,528
- Callback
 - JDBC,445
- Callback-Funktionen
 - Embedded SQL,531
 - registrieren,546
- Callbacks
 - DB_CALLBACK_CONN_DROPPED,547
 - DB_CALLBACK_DEBUG_MESSAGE,547
 - DB_CALLBACK_START,547
 - DB_CALLBACK_VALIDATE_FILE_TRANSFERR,549
- Cancel-Methode
 - SACommand-Klasse [SQL Anywhere .NET-API],139
- CanCreateDataSourceEnumerator-Eigenschaft
 - SAFactory-Klasse [SQL Anywhere .NET-API],264
- CD-ROM
 - Deployment von Datenbanken,1022
- Chained, Modus
 - Implementierung,39
 - steuern,37
 - Transaktionen,37
- chained-Option
 - JDBC,434
- ChangeDatabase-Methode
 - SAConnection-Klasse [SQL Anywhere .NET-API],174
- ChangePassword-Methode
 - SAConnection-Klasse [SQL Anywhere .NET-API],174

Charset-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],196
 CharSetConversion-Option
 Beispiel,787
 checkForUpdates
 konfigurierbare Option,1011
 Checkpoint-Enumeration [Datenbanktools-API]
 Beschreibung,889
 Class.forName-Methode
 SQL Anywhere JDBC 4.0-Treiber laden,423
 CLASSPATH-Umgebungsvariable
 einrichten,430
 Java in der Datenbank verwenden,404
 jConnect,424
 Clear-Methode
 SAPParameterCollection-Klasse [SQL Anywhere .NET-API],294
 ClearAllPools-Methode
 SAConnection-Klasse [SQL Anywhere .NET-API],175
 clearBatch
 Statement-Klasse,437
 ClearPool-Methode
 SAConnection-Klasse [SQL Anywhere .NET-API],175
 ClickOnce
 Installation, .NET Datenprovider, nicht verwaltete DLLs,963
 Client
 Zeitänderung,555
 Client-Library
 Sybase Open Client,748
 Clientdateien
 ESQL-Client-API-Callback-Funktion,546
 ODBC-Client-API, Callback-Funktion,375
 ClientPort-Eigenschaft
 SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],318
 CLIENTPORT-Klausel
 angeben,802
 Clients
 Web,794
 Clientseitiges Autocommit
 Info zu,39
 CLOSE-Anweisung
 Cursor in Embedded SQL,516
 Close-Methode
 SABulkCopy-Klasse [SQL Anywhere .NET-API],109
 SAConnection-Klasse [SQL Anywhere .NET-API],175
 SADataReader-Klasse [SQL Anywhere .NET-API],227
 close-Methode
 Python,664
 CLR
 externe Umgebung,622
 CLR-Schlüsselwort
 externe Umgebung,622
 CodeXchange
 Beispiele,757
 ColumnMappings-Eigenschaft
 SABulkCopy-Klasse [SQL Anywhere .NET-API],113
 Columns-Feld
 SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],269
 Command-Eigenschaft
 SARowUpdatedEventArgs-Klasse [SQL Anywhere .NET-API],309
 SARowUpdatingEventArgs-Klasse [SQL Anywhere .NET-API],312
 CommandText-Eigenschaft
 SACommand-Klasse [SQL Anywhere .NET-API],148
 CommandTimeout-Eigenschaft
 SACommand-Klasse [SQL Anywhere .NET-API],149
 CommandType-Eigenschaft
 SACommand-Klasse [SQL Anywhere .NET-API],149
 CommBufferSize-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],196
 COMMIT-Anweisung
 Cursor,40
 JDBC,434
 Commit-Methode
 SATransaction-Klasse [SQL Anywhere .NET-API],322
 commit-Methode
 Python,665
 commitOnExit
 konfigurierbare Option,1011
 CommitTrans ADO-Methode

- ADO-Programmierung,344
- Daten aktualisieren,344
- CommLinks-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],197
- Compiler
 - verwendet mit sqlpp,473
- Compress-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],197
- CompressionThreshold-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],197
- connect-Methode
 - Python,664
- Connection ADO-Objekt
 - ADO-Programmierung,344
- Connection-Eigenschaft
 - SACCommand-Klasse [SQL Anywhere .NET-API],150
 - SAServerSideConnection-Klasse [SQL Anywhere-.NET-API],312
 - SATransaction-Klasse [SQL Anywhere .NET-API],324
- CONNECTION_PROPERTY-Funktion
 - Beispiel,785
- ConnectionLifetime-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],197
- ConnectionName-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],198
- ConnectionPool-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],198
- ConnectionReset-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],198
- ConnectionString-Eigenschaft
 - SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],167
 - SACConnection-Klasse [SQL Anywhere .NET-API],185
- ConnectionTimeout-Eigenschaft
 - SACConnection-Klasse [SQL Anywhere .NET-API],186
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],198
- Contains-Methode
 - SABulkCopyColumnMappingCollection-Klasse [SQL Anywhere .NET-API],127
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],294
- ContainsKey-Methode
 - SACConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],209
- Cookies
 - erstellen,780
 - Sitzungsverwaltung,782
- CopyTo-Methode
 - SABulkCopyColumnMappingCollection-Klasse [SQL Anywhere .NET-API],127
 - SAErrorCollection-Klasse [SQL Anywhere .NET-API],255
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],296
- Count-Eigenschaft
 - SAErrorCollection-Klasse [SQL Anywhere .NET-API],256
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],299
- CREATE PROCEDURE-Anweisung
 - Embedded SQL-Syntax,528
- CreateCommand-Methode
 - SACConnection-Klasse [SQL Anywhere .NET-API],176
 - SAFactory-Klasse [SQL Anywhere .NET-API],261
- CreateCommandBuilder-Methode
 - SAFactory-Klasse [SQL Anywhere .NET-API],262
- CreateConnection-Methode
 - SAFactory-Klasse [SQL Anywhere .NET-API],262
- CreateConnectionStringBuilder-Methode
 - SAFactory-Klasse [SQL Anywhere .NET-API],262
- CreateDataAdapter-Methode
 - SAFactory-Klasse [SQL Anywhere .NET-API],263
- CreateDataSourceEnumerator-Methode
 - SAFactory-Klasse [SQL Anywhere .NET-API],263
- CreateParameter-Methode
 - SACCommand-Klasse [SQL Anywhere .NET-API],140
 - SAFactory-Klasse [SQL Anywhere .NET-API],263
 - verwenden,4
- CreatePermission-Methode
 - SAFactory-Klasse [SQL Anywhere .NET-API],264
 - SAPermissionAttribute-Klasse [SQL Anywhere .NET-API],306

<p>Cross-Site-Scripting Webdienste,787</p> <p>CS-Library Info,748</p> <p>CS_CSR_ABS nicht unterstützt von Open Client,754</p> <p>CS_CSR_FIRST nicht unterstützt von Open Client,754</p> <p>CS_CSR_LAST nicht unterstützt von Open Client,754</p> <p>CS_CSR_PREV nicht unterstützt von Open Client,754</p> <p>CS_CSR_REL nicht unterstützt von Open Client,754</p> <p>CS_DATA_BOUNDARY nicht unterstützt von Open Client,754</p> <p>CS_DATA_SENSITIVITY nicht unterstützt von Open Client,754</p> <p>CS_PROTO_DYNPROC nicht unterstützt von Open Client,754</p> <p>CS_REG_NOTIF nicht unterstützt von Open Client,754</p> <p>CS_REQ_BCP nicht unterstützt von Open Client,754</p> <p>ct_command-Funktion Anweisungen in Open Client ausführen,751 Ergebnisse in Open Client,753</p> <p>CT_CURSOR-Funktion Open Client,752</p> <p>ct_dynamic-Funktion Open Client,751</p> <p>ct_results-Funktion Open Client,753</p> <p>ct_send-Funktion Open Client,753</p> <p>Cursor abbrechen,14 Abrollfähigkeit,15 ADO,33 ADO.NET,33 Aktualisierbarkeit,15 aktualisieren,343,753 anfordern,33 Arbeitstabellen,28 asensitiv,25 Beispiel C-Code,477 bestimmen, welche Cursor für eine Verbindung vorhanden sind,5</p>	<p>Block-Cursor,16</p> <p>db_cancel_request-Funktion,538</p> <p>durch Keyset gesteuert,26</p> <p>DYNAMIC SCROLL und asensitive Cursor,25</p> <p>DYNAMIC SCROLL und Cursorpositionierung,10</p> <p>dynamisch,23</p> <p>Eigenschaften,15</p> <p>Eindeutigkeit,15</p> <p>Einschränkungen,9</p> <p>Embedded SQL, Verwendungszweck,516</p> <p>Ergebnismengen,5</p> <p>Ergebnismengen beschreiben,35</p> <p>Fat,11</p> <p>Fetch-Vorgänge für mehrere Zeilen,11</p> <p>Fetch-Vorgänge für Zeilen,10</p> <p>gespeicherte Prozeduren,529</p> <p>Info,6</p> <p>Inhalt der Cursor für eine Verbindung anzeigen,5</p> <p>insensitiv,22</p> <p>Interna,16</p> <p>Isolationsstufe (isolation level),10</p> <p>löschen,753</p> <p>mehrere Zeilen einfügen,12</p> <p>Mitgliedschaft,17</p> <p>ODBC,388</p> <p>ODBC- und SQL Anywhere-Typen,33</p> <p>ODBC-Aktualisierungen,392</p> <p>ODBC-Cursor-Merkmale wählen,389</p> <p>ODBC-Ergebnismengen,390</p> <p>ODBC-Lesezeichen,392</p> <p>ODBC-Löschungen,392</p> <p>OLE DB,33</p> <p>Open Client,752</p> <p>Plattformen,15</p> <p>positionieren,9</p> <p>Prefetch-Performance,28</p> <p>Python,665</p> <p>Reihenfolge,17</p> <p>Savepoints,40</p> <p>Schreibschutz,22</p> <p>SCROLL,26</p> <p>scrollfähig,12</p> <p>sensitiv,23</p> <p>Sensitivität,15</p> <p>Sensitivität in SQL Anywhere,16</p> <p>Sensitivität und Aktualisieren, Beispiel,20</p> <p>Sensitivität und Isolationsstufen,32</p> <p>Sensitivität und Löschen, Beispiel,18</p>
---	--

- Sensitivität und Performance,28
- Sensitivität, Überblick,18
- sichtbare Änderungen,17
- statisch,22
- Transaktionen,40
- unbestimmte Sensitivität,25
- Verfügbarkeit,15
- verwenden,6,8
- von Embedded SQL unterstützte Typen,34
- vorbereitete Anweisungen,8
- Vorteile,7
- Werte,17
- wertsensitiv,26
- Zeilen aktualisieren und löschen,12
- Zeilen einfügen,12
- Cursor positionieren
 - Fehlerbehandlung,10
- Cursor und Lesezeichen
 - Info,16
- Cursor, Sensitivität und Performance
 - Info,28

D

- DataAdapter
 - Aktualisieren von Daten,53
 - Daten abrufen,55,56
 - Einfügen von Daten,53
 - Info,47
 - Löschen von Daten,53
 - Primärschlüsselwerte abrufen,60
- DataAdapter-Eigenschaft
 - SACommandBuilder-Klasse [SQL Anywhere .NET-API],162
- Database-Eigenschaft
 - SAConnection-Klasse [SQL Anywhere .NET-API],187
- DatabaseFile-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],199
- DatabaseKey-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],199
- DatabaseName-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],199
- DatabaseSwitches-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],199
- DataSet
 - SQL Anywhere-.NET-Datenprovider,53
- DataSource-Eigenschaft
 - SAConnection-Klasse [SQL Anywhere .NET-API],187
- DataSourceInformation-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],270
- DataSourceName-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],200
- DataTypes-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],270
- Dateidatenquellen
 - Windows,977
- Dateien
 - Namenskonventionen,950
 - Standorte für das Deployment,948
- Dateinamen
 - .db-Dateierweiterung,951
 - .log-Dateierweiterung,951
 - Konventionen,950
 - Sprache,950
 - SQL Anywhere,951
 - Versionsnummer,950
- Dateinamen, Konventionen
 - Info,950
- Dateiübertragung
 - Callback,549
- Daten
 - Verarbeitung mit dem .NET-Datenprovider,47
 - Zugriff über den .NET-Datenprovider,47
- Daten abrufen
 - Embedded SQL,514
- Daten einfügen
 - mehrzeilig,518
 - weite Einfügungen,518
- Datenbank erstellen, Assistent
 - Hinweise zum Deployment,993
- Datenbank-Upgrade, Assistent
 - jConnect-Metadaten-Unterstützung installieren,425
- Datenbankeigenschaften
 - db_get_property-Funktion,541
- Datenbanken
 - Deployment,1021

- Java-Klassen speichern,402
- jConnect-Metadaten-Unterstützung installieren,425
- URL,426
- Datenbankmanagement
 - dbtools,871
- Datenbankoptionen
 - für jConnect eingestellt,427
- Datenbankserver
 - Deployment,1016
 - Funktionen,553
- Datenbanktools, Bibliothek
 - Info,871
- Datenbanktools-API
 - a_backup_db-Struktur,894
 - a_change_log-Struktur,897
 - a_create_db-Struktur,899
 - a_db_info-Struktur,903
 - a_db_version_info-Struktur,905
 - a_dblic_info-Struktur,906
 - a_dbtools_info-Struktur,907
 - a_log_file_info-Struktur,907
 - a_name-Struktur,908
 - a_remote_sql-Struktur,908
 - a_sync_db-Struktur,917
 - a_syncpub-Struktur,927
 - a_sysinfo-Struktur,928
 - a_table_info-Struktur,928
 - a_translate_log-Struktur,929
 - a_truncate_log-Struktur,933
 - a_validate_db-Struktur,934
 - an_erase_db-Struktur,936
 - an_unload_db-Struktur,937
 - an_upgrade_db-Struktur,943
 - Autotune-Enumeration,888
 - Checkpoint-Enumeration,889
 - DBBackup-Methode,879
 - DBChangeLogName-Methode,879
 - DBCreate-Methode,880
 - DBCreatedVersion-Methode,880
 - DBErase-Methode,881
 - DBInfo-Methode,881
 - DBInfoDump-Methode,882
 - DBInfoFree-Methode,882
 - DBLicense-Methode,883
 - DBLogFileInfo-Methode,883
 - DBRemoteSQL-Methode,884
 - DBSynchronizeLog-Methode,884
 - DBToolsFinis-Methode,885
 - DBToolsInit-Methode,885
 - DBToolsVersion-Methode,886
 - DBTranslateLog-Methode,886
 - DBTruncateLog-Methode,886
 - DBUnload-Methode,887
 - DBUpgrade-Methode,887
 - DBValidate-Methode,888
 - History-Enumeration,890
 - Padding-Enumeration,890
 - Unit-Enumeration,891
 - Unload-Enumeration,891
 - UserList-Enumeration,892
 - Validation-Enumeration,892
 - Verbosity-Enumeration,893
 - Version-Enumeration,894
- Datenbanktools-Schnittstelle
 - Info,871
- Datenquellen
 - Benutzer-DSN,977
 - Datei-DSN,977
 - DefaultDSNDir für FileDSN,977
 - Deployment,977
 - System-DSN,977
- Datenraster-Steuerelement
 - Visual Studio,92
- Datentypen
 - C-Datentypen,487
 - Dynamic SQL,504
 - Embedded SQL,482
 - Hostvariablen,487
 - in Webdienst-Handlern,817
 - Open Client-Bereiche,750
 - Open Client-Zuordnungen,749
 - SQL und C,616
 - SQLDA,506
- Datentypkonvertierungen
 - Indikatorvariablen,494
- Datenverbindung
 - Visual Studio,88
- DATETIME-Datentyp
 - Embedded SQL,487
 - Open Client-Konvertierung,750
- DB-Library
 - Info,748
- DB_ACTIVE_CONNECTION
 - db_find_engine-Funktion,540
- db_backup-Funktion
 - dbbackup-Dienstprogramm,531

- Info,533
- DB_BACKUP_CLOSE_FILE-Parameter
 - Info,533
- DB_BACKUP_END-Parameter
 - Info,533
- DB_BACKUP_INFO-Parameter
 - Info,533
- DB_BACKUP_INFO_CHKPT_LOG-Parameter
 - Info,533
- DB_BACKUP_INFO_PAGES_IN_BLOCK-Parameter
 - Info,533
- DB_BACKUP_OPEN_FILE-Parameter
 - Info,533
- DB_BACKUP_PARALLEL_READ-Parameter
 - Info,533
- DB_BACKUP_PARALLEL_START-Parameter
 - Info,533
- DB_BACKUP_READ_PAGE-Parameter
 - Info,533
- DB_BACKUP_READ_RENAME_LOG-Parameter
 - Info,533
- DB_BACKUP_START-Parameter
 - Info,533
- DB_CALLBACK_CONN_DROPPED-Callbackparameter
 - Info,547
- DB_CALLBACK_DEBUG_MESSAGE-Callbackparameter
 - Info,547
- DB_CALLBACK_FINISH-Callbackparameter
 - Info,547
- DB_CALLBACK_MESSAGE-Callbackparameter
 - Info,548
- DB_CALLBACK_START-Callbackparameter
 - Info,547
- DB_CALLBACK_VALIDATE_FILE_TRANSFER-Callbackparameter
 - Info,549
- DB_CALLBACK_WAIT-Callbackparameter
 - Info,548
- DB_CAN_MULTI_CONNECT
 - db_find_engine-Funktion,540
- DB_CAN_MULTI_DB_NAME
 - db_find_engine-Funktion,540
- db_cancel_request-Funktion
 - Anforderungsverwaltung,531
 - Info,538
- db_change_char_charset-Funktion
 - Info,538
- db_change_nchar_charset-Funktion
 - Info,539
- DB_CLIENT
 - db_find_engine-Funktion,540
- DB_CONNECTION_DIRTY
 - db_find_engine-Funktion,540
- DB_DATABASE_SPECIFIED
 - db_find_engine-Funktion,540
- DB_ENGINE
 - db_find_engine-Funktion,540
- db_find_engine-Funktion
 - Info,540
- db_fini-Funktion
 - Info,541
- db_fini_dll
 - aufrufen,476
- db_get_property-Funktion
 - Info,541
- db_init-Funktion
 - Info,542
- db_init_dll
 - aufrufen,476
- db_is_working-Funktion
 - Anforderungsverwaltung,531
 - Info,543
- db_locate_servers-Funktion
 - Info,544
- db_locate_servers_ex-Funktion
 - Info,545
- DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
 - Info,545
- DB_LOOKUP_FLAG_DATABASES
 - Info,545
- DB_LOOKUP_FLAG_NUMERIC
 - Info,545
- DB_NO_DATABASES
 - db_find_engine-Funktion,540
- DB_PROP_CLIENT_CHARSET
 - Verwendungszweck,542
- DB_PROP_DBLIB_VERSION
 - Verwendungszweck,542
- DB_PROP_SERVER_ADDRESS
 - Verwendungszweck,542
- db_register_a_callback-Funktion
 - Anforderungsverwaltung,531

Info,546
 db_start_database-Funktion
 Info,550
 db_start_engine-Funktion
 Info,550
 db_stop_database-Funktion
 Info,552
 db_stop_engine-Funktion
 Info,552
 db_string_connect-Funktion
 Info,553
 db_string_disconnect-Funktion
 Info,554
 db_string_ping_server-Funktion
 Info,555
 db_time_change-Funktion
 Info,555
 DBBackup-Methode [Datenbanktools-API]
 Beschreibung,879
 dbcapi.dll
 Deployment von,1024
 Deployment von PHP-Clients,984
 PHP-Unterstützungsmodul,647
 DBChangeLogName-Methode [Datenbanktools-API]
 Beschreibung,879
 dbcis16.dll
 Deployment von Datenbankservern,1016
 dbclrenv16.dll
 Deployment,1024
 dbcon16.dll
 Deployment unter Windows,996
 Deployment von Datenbank-Dienstprogrammen,1028
 Deployment von Embedded SQL-Clients,981
 Deployment von ODBC-Clients,971
 Deployment von OLE DB-Clients,964
 Deployment von PHP-Clients,984
 dbconsole-Dienstprogramm
 Deployment,992
 Deployment unter Linux und Unix,1003
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,994
 Deployment von Administrationstools unter Linux/Unix/Mac OS X,1008
 Deployment von Administrationstools unter Windows,999
 dbconsole.exe
 Deployment unter Windows,996
 dbconsole.ini
 Deployment von Administrationstools,992
 DBConsole.jar
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 DBCreate-Methode [Datenbanktools-API]
 Beschreibung,880
 DBCreatedVersion-Methode [Datenbanktools-API]
 Beschreibung,880
 dbctrs16.dll
 Deployment von Datenbankservern,1016
 Deployment von SQL Anywhere,1023
 DBD::SQLAnywhere
 Ausführen von SQL-Anweisungen,656
 Behandeln mehrerer Ergebnismengen,657
 Info,651
 Perl-Skripten schreiben,654
 Verbindung mit einer Datenbank,655
 Zeilen einfügen,658
 dbdata.dll
 SQL Anywhere-.NET-Datenprovider,74
 dbelevate16.exe
 Deployment unter Windows,996
 DLLs registrieren,1023
 dbencod16.dll
 Deployment von SQL Remote,1030
 dbeng16
 Deployment von Datenbankservern,1016
 dbeng16.lic
 Deployment von Datenbankservern,1016
 DBErase-Methode [Datenbanktools-API]
 Beschreibung,881
 dbextclr16.exe
 Deployment,1024
 dbextenv16.dll
 Deployment,1024
 PHP-Unterstützungsmodul,647
 dbexternc16
 Deployment,1024
 externe Aufrufe,627
 dbextf.dll
 Deployment von Datenbankservern,1016
 dbfile16.dll
 Deployment von SQL Remote,1030
 dbfips16.dll
 Deployment von Embedded SQL-Clients,981

- Deployment von PHP-Clients,984
- FIPS-zertifizierte AES-Verschlüsselung,1026
- FIPS-zertifizierte RSA-Verschlüsselung,1026
- dbftp16.dll
 - Deployment von SQL Remote,1030
- dbghelp.dll
 - Deployment unter Windows,996
 - Deployment von Datenbankservern,1016
- dbicu16.dll
 - Deployment unter Windows,996
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
 - Deployment von PHP-Clients,984
- dbicudt16.dat
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
- dbicudt16.dll
 - Deployment unter Windows,996
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
 - Deployment von PHP-Clients,984
- DBInfo-Methode [Datenbanktools-API]
 - Beschreibung,881
- DBInfoDump-Methode [Datenbanktools-API]
 - Beschreibung,882
- DBInfoFree-Methode [Datenbanktools-API]
 - Beschreibung,882
- dbinit, Dienstprogramm
 - Hinweise zum Deployment,1028
- dbinit-Dienstprogramm
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
- dbinit.exe
 - Deployment unter Windows,996
- dbisql-Dienstprogramm
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment von Administrationstools unter Windows,996
 - von Unix unterstützte Deployment-Plattformen,1003
- dbisql.com
 - Deployment unter Windows,996
- dbisql.exe
 - Deployment unter Windows,996
- dbisql.ini
 - Deployment von Administrationstools,992
- dbisqlc-Dienstprogramm
 - begrenzte Funktionalität,1015
 - Deployment,1015
- dbjdbc16.dll
 - Deployment,1024
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
- dbjodbc16.dll
 - Deployment unter Windows,996
 - Deployment von Datenbankservern,1016
- dblgen16.dll
 - Deployment unter Windows,996
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von Datenbankservern,1016
 - Deployment von Embedded SQL-Clients,981
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
 - Deployment von OLE DB-Clients,964
 - Deployment von PHP-Clients,984
 - Deployment von SQL Remote,1030
- dblgen16.dll-Registrierungseintrag
 - Info,1019
- dblgen16.res
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
 - Deployment von SQL Remote,1030
- DBLIB
 - dynamisch laden,476
 - Schnittstellenbibliothek,467
- dblib16.dll
 - Deployment unter Windows,996
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von Embedded SQL-Clients,981
 - Deployment von PHP-Clients,984
 - Deployment von SQL Remote,1030
- DBLicense-Methode [Datenbanktools-API]
 - Beschreibung,883
- DBLogFileInfo-Methode [Datenbanktools-API]
 - Beschreibung,883

dbodbc16.bundle
 Deployment von ODBC-Clients,971
 Mac OS X-ODBC-Treiber,364
 dbodbc16.dll
 Deployment unter Windows,996
 Deployment von Datenbankservern,1016
 Deployment von ODBC-Clients,971
 Deployment von SQL Anywhere,1023
 linken,361
 dbodbc16.lib
 Windows Mobile-ODBC-Importbibliothek,362
 dbodbc16_r.bundle
 Deployment von ODBC-Clients,971
 dboftsp.dll
 Deployment von Datenbank-Dienstprogrammen,1028
 dboledb16.dll
 Deployment von OLE DB-Clients,964
 Deployment von SQL Anywhere,1023
 dboledba16.dll
 Deployment von OLE DB-Clients,964
 Deployment von SQL Anywhere,1023
 dbosrv16.exe, Dienstprogramm
 Syntax,464
 dbostop.exe, Dienstprogramm
 Syntax,465
 DBProviderFactory
 registrieren,74
 dbput16.dll
 Deployment unter Windows,996
 dbremote-Dienstprogramm
 Deployment von SQL Remote,1030
 DBRemoteSQL-Methode [Datenbanktools-API]
 Beschreibung,884
 dbrmt.h
 Info,871
 dbrsa16.dll
 Deployment von PHP-Clients,984
 RSA-Verschlüsselung,1026
 dbrsakp16.dll
 Deployment von Datenbankservern,1016
 Deployment von JDBC-Clients,984
 Deployment von Open Client,992
 dbscript16.dll
 Deployment von Datenbankservern,1016
 dbserv16.dll
 AES-Verschlüsselung,1026
 Deployment von Datenbankservern,1016
 dbsmtp16.dll
 Deployment von SQL Remote,1030
 dbsrv16
 Deployment von Datenbankservern,1016
 dbsrv16.lic
 Deployment von Datenbankservern,1016
 DBSynchronizeLog-Methode [Datenbanktools-API]
 Beschreibung,884
 dbtool16.dll
 Deployment unter Windows,996
 Deployment von Datenbank-Dienstprogrammen,1028
 Deployment von SQL Remote,1030
 Info,871
 Windows Mobile,871
 DBTools-Schnittstelle
 beenden,872
 Beispielprogramm,876
 DBTools-Funktionen aufrufen,873
 Einführung,871
 Info,871
 initialisieren,872
 Rückgabecodes,945
 starten,872
 Verwendung,871
 dbtools.h
 Info,871
 dbtools.h-Headerdatei
 SQL Anywhere-Datenbanktools, C-API-Referenz,878
 DBToolsFini-Methode [Datenbanktools-API]
 Beschreibung,885
 DBToolsInit-Methode [Datenbanktools-API]
 Beschreibung,885
 DBToolsVersion-Methode [Datenbanktools-API]
 Beschreibung,886
 DBTranslateLog-Methode [Datenbanktools-API]
 Beschreibung,886
 DBTruncateLog-Methode [Datenbanktools-API]
 Beschreibung,886
 DbType-Eigenschaft
 SAPparameter-Klasse [SQL Anywhere .NET-API],282
 dbunload, Dienstprogramm
 Hinweise zum Deployment,1028
 dbunload-Dienstprogramm
 Deployment für Datenbanken der Version 9 und früher,1029

- DBUnload-Methode [Datenbanktools-API]
 - Beschreibung,887
- dbunlspt
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbupgrad-Dienstprogramm
 - jConnect-Metadaten-Unterstützung installieren,425
- DBUpgrade-Methode [Datenbanktools-API]
 - Beschreibung,887
- dbusde.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusde.res
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusen.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusen.res
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbuses.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusfr.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusfr.res
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusit.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusja.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusja.res
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusko.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbuslt.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbuspl.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbuspt.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusr.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbustw.dll
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbusuk.res
 - Deployment für Datenbanken der Version 9 und früher,1029
- dbuszh.res
 - Deployment für Datenbanken der Version 9 und früher,1029
- DBValidate-Methode [Datenbanktools-API]
 - Beschreibung,888
- debugger.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- DECIMAL-Datentyp
 - Embedded SQL,487
- DECL_BIGINT-Makro
 - Info,487
- DECL_BINARY-Makro
 - Info,487
- DECL_BIT-Makro
 - Info,487
- DECL_DATETIME-Makro
 - Info,487
- DECL_DECIMAL-Makro
 - Info,487
- DECL_FIXCHAR-Makro
 - Info,487
- DECL_LONGBINARY-Makro
 - Info,487
- DECL_LONGNVARCHAR-Makro
 - Info,487
- DECL_LONGVARCHAR-Makro
 - Info,487
- DECL_NCHAR-Makro
 - Info,487
- DECL_NFIXCHAR-Makro
 - Info,487
- DECL_NVARCHAR-Makro
 - Info,487
- DECL_UNSIGNED_BIGINT-Makro

- Info,487
- DECL_VARCHAR-Makro
 - Info,487
- DECLARE-Anweisung
 - Cursor in Embedded SQL,516
 - Info,486
- DefaultDSNDir
 - ODBC-FileDSN,977
- defaultShowMultipleResultSets
 - konfigurierbare Option,1011
- defaultShowResultsForAllStatements
 - konfigurierbare Option,1011
- Deinstallationsoption
 - SetupVSPackage,963
- Deinstallieren
 - .NET-Datenprovider,963
- Deklarationsabschnitt
 - Info,486
- Deklarieren
 - Embedded SQL-Datentypen,482
 - Hostvariablen,486
- DELETE-Anweisung
 - JDBC,437
 - positionsbasiert,12
- DeleteCommand-Eigenschaft
 - SADaAdapter-Klasse [SQL Anywhere .NET-API],219
- DeleteDynamic-Methode
 - JDBCExample,441
- DeleteStatic-Methode
 - JDBCExample,438
- Deployment
 - .NET-Datenprovider,960
 - Administrationtools,992
 - Administrationtools unter Linux und Unix,1003
 - Administrationtools unter Windows,994
 - ADO.NET-Datenprovider,960
 - Assistent,952
 - Clientanwendungen,960
 - Dateistandorte,948
 - Datenbanken,1021
 - Datenbanken auf CD-ROM,1022
 - Datenbanken lokalisieren,1022
 - Datenbanken und Anwendungen,947
 - Datenbankserver,1016
 - Deployment,1024
 - Deployment-Assistent,952
 - dialogfreie Installation,956
 - DLLs registrieren,1023
 - eingebettete Datenbanken,1027
 - Embedded SQL,980
 - externe CLR-Umgebung,1024
 - externe ESQL-Umgebung,1024
 - externe Java-Umgebung,1024
 - externe ODBC-Umgebung,1024
 - externe Perl-Umgebung,1025
 - externe PHP-Umgebung,1025
 - externe Umgebung,1024
 - Info,947
 - Interactive SQL,992
 - Interactive SQL (dbisqlc),1015
 - Interactive SQL unter Linux und Unix,1003
 - Interactive SQL unter Windows,994
 - international, Hinweise,1022
 - Java in der Datenbank,1016
 - jConnect,982
 - JDBC-Clients,982
 - Konsolendienstprogramm (dbconsole),992
 - Konsolendienstprogramm (dbconsole) unter Linux und Unix,1003
 - Konsolendienstprogramm (dbconsole) unter Windows,994
 - MobiLink-Plug-In,994
 - ODa Producer,461
 - ODBC,970
 - ODBC-Datenquellen,976
 - OLE DB-Provider,964
 - Open Client,992
 - Personal Datenbankserver,1027
 - PHP-Clients,984
 - PHP-Erweiterung,984
 - Registrierungseinstellungen,1018
 - schreibgeschützte Datenbanken,1022
 - SQL Anywhere,947
 - SQL Anywhere JDBC-Treiber,982
 - SQL Anywhere, Komponenten unter Windows,952
 - SQL Anywhere-.NET-Datenprovider,72
 - SQL Anywhere-Plug-In,994
 - SQL Remote,1030
 - Sybase Central,992
 - Sybase Central unter Linux und Unix,1003
 - Sybase Central unter Windows,994
 - Überblick,947
 - UltraLite-Plug-In,994
 - Unix-Probleme,949
 - Deployment unter Linux und Unix

- SQL Anywhere-Konsolendienstprogramm (dbconsole),1003
- Deployment von Anwendungen
 - Embedded SQL,980
- Deployment, SQL Anywhere .NET-Datenprovider
 - Info,72
- Deployment-Assistent
 - ausführen,953
 - Info,952
- DeployUtility
 - .NET-Datenprovider-Beispielprojekt.NET-Datenprovider-Beispielprojekt,43
 - Installation, .NET-Datenprovider, nicht verwaltete DLLs,963
- Depth-Eigenschaft
 - SADatReader-Klasse [SQL Anywhere .NET-API],247
- DeriveParameters-Methode
 - SACommandBuilder-Klasse [SQL Anywhere .NET-API],155
- DESCRIBE SELECT LIST-Anweisung
 - dynamische SELECT-Anweisung,503
- DESCRIBE-Anweisung
 - DT_HAS_USERTYPE_INFO,507
 - in dynamischen SELECT-Anweisungen verwenden,503
 - mehrere Ergebnismengen,530
 - SQLDA-Felder,506
 - sqlen-Felder,508
 - sqltype-Feld,508
- DesignTimeVisible-Eigenschaft
 - SACommand-Klasse [SQL Anywhere .NET-API],150
- Deskriptoren
 - Ergebnismengen beschreiben,35
- DestinationColumn-Eigenschaft
 - SABulkCopyColumnMapping-Klasse [SQL Anywhere .NET-API],119
- DestinationOrdinal-Eigenschaft
 - SABulkCopyColumnMapping-Klasse [SQL Anywhere .NET-API],119
- DestinationTableName-Eigenschaft
 - SABulkCopy-Klasse [SQL Anywhere .NET-API],113
- Dialogfreie Deinstallation
 - msiexec,954
- Dialogfreie Installation
 - Info,954
- Dialogfreie Installationen
 - Info,956
- Dienst
 - OData Producer-Modell,463
- Dienste
 - Datentypen,817
 - Web,755
- Dienstprogramm für den SQL-Präprozessor (sqlpp)
 - ausführen,469
- Dienstprogramm Initialisierung (dbinit)
 - Hinweise zum Deployment,1028
- Dienstprogramm zum Entladen (dbunload)
 - Deployment für Datenbanken der Version 9 und früher,1029
 - Hinweise zum Deployment,1028
- Dienstprogramme
 - Deployment,1028
 - Deployment von Datenbank-Dienstprogrammen,1028
 - OData Server (dbosrv16.exe), Syntax,464
 - OData Serverstopp (dbostop.exe), Syntax,465
 - Rückgabecode,945
 - Syntax für den SQL-Präprozessor (sqlpp),469
- Direction-Eigenschaft
 - SAParameter-Klasse [SQL Anywhere .NET-API],282
- disableExecuteAll
 - konfigurierbare Option,1011
- DisableMultiRowFetch-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],200
- disableResultsEditing
 - konfigurierbare Option,1011
- DISH-Dienste
 - .NET, praktische Einführung,848
 - erstellen,764,765
 - Info,760
 - JAX-WS, praktische Einführung,856
 - kommentieren,766
 - löschen,766
 - SQL Anywhere-Webclient, praktische Einführung,839
- Dispose-Methode
 - SABulkCopy-Klasse [SQL Anywhere .NET-API],109
- Distributed Transaction Coordinator
 - dreischichtige Datenverarbeitung,867
- DLL, Eintrittspunkte

Info,532	DESCRIBE-Anweisung,507
DllMain	DT_INT, Embedded SQL-Datentyp
Aufruf von db_fini,541	Info,482
Aufruf von ODBC-Funktionen,377	DT_LONGBINARY, Embedded SQL-Datentyp
DLLs	Info,484
aus gespeicherten Prozeduren aufrufen,601	DT_LONGNVARCHAR, Embedded SQL-Datentyp
Deployment,1023	Info,484
externe Funktionsaufrufe,603	DT_LONGVARCHAR, Embedded SQL-Datentyp
für Deployment registrieren,1023	Info,483
mehrere SQLCAs,500	DT_NFIXCHAR, Embedded SQL-Datentyp
DoBroadcast-Eigenschaft	Info,483
SATcpOptionsBuilder-Klasse [SQL	DT_NSTRING, Embedded SQL-Datentyp
Anywhere .NET-API],318	Info,483
Dokumentation	DT_NVARCHAR, Embedded SQL-Datentyp
Deployment unter Linux,1016	Info,483
Deployment unter Mac OS X,1016	DT_PROCEDURE_IN
Deployment unter Unix,1016	Verwendung,530
Deployment unter Windows,1015	DT_PROCEDURE_OUT
Doppelpunkt, Hostvariablenname	Verwendung,530
SQLBindParameter-Funktion,380	DT_SMALLINT, Embedded SQL-Datentyp
Dreischichtige Datenverarbeitung	Info,482
Architektur,865	DT_STRING, Embedded SQL-Datentyp
Distributed Transaction Coordinator,867	Info,483
EAServer,867	DT_TIME, Embedded SQL-Datentyp
Info,865	Info,483
Microsoft Transaction Server,867	DT_TIMESTAMP, Embedded SQL-Datentyp
Ressourcen-Manager,867	Info,483
Ressourcen-Verteiler,867	DT_TIMESTAMP_STRUCT, Embedded SQL-
verteilte Transaktionen,866	Datentyp
DSN-freie Verbindungen	Info,484
ODBC verwenden,980	DT_TINYINT, Embedded SQL-Datentyp
DT_BIGINT, Embedded SQL-Datentyp	Info,482
Info,482	DT_UNSBIGINT, Embedded SQL-Datentyp
DT_BINARY, Embedded SQL-Datentyp	Info,482
Info,484	DT_UNSINT, Embedded SQL-Datentyp
DT_BIT, Embedded SQL-Datentyp	Info,482
Info,482	DT_UNSSMALLINT, Embedded SQL-Datentyp
DT_DATE, Embedded SQL-Datentyp	Info,482
Info,483	DT_VARCHAR, Embedded SQL-Datentyp
DT_DECIMAL, Embedded SQL-Datentyp	Info,483
Info,483	DT_VARIABLE, Embedded SQL-Datentyp
DT_DOUBLE, Embedded SQL-Datentyp	Info,485
Info,482	DTC
DT_FIXCHAR, Embedded SQL-Datentyp	dreischichtige Datenverarbeitung,867
Info,483	Isolationsstufen,869
DT_FLOAT, Embedded SQL-Datentyp	DTC-Isolationsstufen
Info,482	Info,869
DT_HAS_USERTYPE_INFO	DYNAMIC SCROLL-Cursor

- asensitive Cursor,25
- Embedded SQL,34
- Fehlerbehandlung,10
- Dynamic SQL
 - Info,501
 - SQLDA,504
- Dynamische Cursor
 - Beispiel,480
 - Info,23
 - ODBC,33
- dynamische SELECT-Anweisung
 - DESCRIBE SELECT LIST-Anweisung,503

E

- EAServer
 - dreischichtige Datenverarbeitung,867
- Eigenschaften
 - db_get_property-Funktion,541
- Einbeziehung
 - verteilte Transaktionen,867
- Eindeutige Cursor
 - Info,15
- Einfügungen
 - JDBC,442
- Eingabehilfen
 - Deployment von Java-basierten Administrationstools,994
- Eingebettete Datenbanken
 - Deployment,1027
- Eingebetteter HTTP-Server
 - Info,455
- Einschränkungen
 - OData Server,456
- Einstellen
 - Werte mit SQLDA,511
- Elevate-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],200
- Embedded SQL
 - Anweisungszusammenfassung,561
 - Aufruf von db_fini aus DIIMain,541
 - Autocommit-Modus,37
 - Autocommit-Verhalten steuern,37
 - Autorisierung,469
 - Beispielprogramm,475
 - Cursor,34
 - Cursor verwenden,516
 - Cursorbeispiele,477
 - Cursortypen,15
 - Daten abrufen,514
 - Deployment von Clients,980
 - dynamische Anweisungen,501
 - dynamische Cursor,480
 - Entwicklung,467
 - externe Umgebung,625
 - FETCH FOR UPDATE,32
 - Funktionen,532
 - Header-Dateien,473
 - Hostvariablen,485
 - Importbibliotheken,474
 - Info,467
 - Kompilieren- und Verknüpfen-Prozess,468
 - SQL-Anweisungen,1
 - statische Anweisungen,501
 - Zeichenfolgen,469
 - Zeilennummern,469
- EnableFlush-Registrierungseintrag
 - Beschädigung der Datenbank verhindern,1018
- EncryptedPassword-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],200
- Encryption-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],201
- EndExecuteNonQuery-Methode
 - SACCommand-Klasse [SQL Anywhere .NET-API],140
- EndExecuteReader-Methode
 - SACCommand-Klasse [SQL Anywhere .NET-API],142
- Enlist-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],201
- EnlistDistributedTransaction-Methode
 - SACConnection-Klasse [SQL Anywhere .NET-API],176
- EnlistTransaction-Methode
 - SACConnection-Klasse [SQL Anywhere .NET-API],177
- Entfernen
 - .NET-Datenprovider,963
- Entity Framework
 - Verwendung,67
- Entity Framework-Unterstützung
 - iAnywhere.Data.SQLAnywhere-Provider,42

Entwickeln, Anwendung mit dem .NET-Datenprovider
 Info,41

Entwicklung von Anwendungen mit den ASP.NET-Providern
 Info,97

Ereignisprotokoll
 Registrierungseintrag,1019

Ergebnismengen
 , gespeicherte Prozeduren,529
 ADO-Recordset-Objekt verwenden,342
 Cursor,5
 Info ADO-Recordset-Objekt,341
 Java in der Datenbank, Gespeicherte Prozeduren,415
 JDBC,443
 Metadaten,35
 ODBC,388
 ODBC abrufen,390
 ODBC, gespeicherte Prozeduren,393
 Open Client,753
 verwenden,8
 von einem Webdienst abrufen,816
 Zugriff von einem WebClient aus,814

Erhalten, Zeitwerte
 Info,63

Errors-Eigenschaft
 SAException-Klasse [SQL Anywhere .NET-API],258
 SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API],266

Erstellen
 Prozeduren und Funktionen mit externen Aufrufen,601
 Webdienste,762

Erstellen, Java-Klassen, Assistent
 verwenden,411

Erstellung von Java-Klassen, Assistent
 verwenden,433

Escape-Syntax
 in Interactive SQL,449
 ODBC,394

esqldll.c
 Info,476

EXEC SQL
 Entwicklung in Embedded SQL,475

EXECUTE-Anweisung
 gespeicherte Prozeduren in Embedded SQL,528
 verwenden,501

execute-Methode
 Python,665

executeBatch
 PreparedStatement-Klasse,442
 Statement-Klasse,437

executemany-Methode
 Python,666

ExecuteNonQuery-Methode
 SACommand verwenden,50
 SACommand-Klasse [SQL Anywhere .NET-API],144

ExecuteReader-Methode
 ADO.NET, vorbereitete Anweisungen,4
 SACommand verwenden,48
 SACommand-Klasse [SQL Anywhere .NET-API],145
 SACommand-Klasse, Simple (Beispiel),82
 SACommand-Klasse, TableView (Beispiel),87

ExecuteScalar-Methode
 SACommand verwenden,48
 SACommand-Klasse [SQL Anywhere .NET-API],147

executeToolBarButtonSemantics
 konfigurierbare Option,1011

executeUpdate
 Statement-Klasse,437

executeUpdate, JDBC-Methode
 verwenden,4

Exit-Codes
 Info,945

Exportdatei
 dblib.def,474

Externe Aufrufe
 Prozeduren und Funktionen erstellen,601

Externe Funktionen
 abbrechen,614
 Bibliotheken entladen,618
 Parameter weiterleiten,612
 Prototypen,603
 Rückgabetypen,618

Externe Funktionsaufrufe
 Info,601
 Unterstützung von externen Umgebungen,621

Externe Objekte
 Info,621
 Kommentar,621

Externe Prozeduraufrufe

- Info,601
- Unterstützung von externen Umgebungen,621
- Externe Umgebungen
 - C-API,625
 - CLR,622
 - Embedded SQL,625
 - Info,621
 - JAVA,635
 - ODBC,625
 - PERL,640
 - PHP,644
- Externer Funktionsaufruf, Schnittstelle
 - an_extfn_api,606
 - an_extfn_result_set_column_data,611
 - an_extfn_result_set_column_info,610
 - an_extfn_result_set_info,609
 - an_extfn_value,608
 - extfn_cancel-Methode,604
 - extfn_post_load_library-Methode,605
 - extfn_pre_unload_library-Methode,605
 - extfn_use_new_api-Methode,604
 - get_piece-Callback,612
 - get_value-Callback,612
 - Prototyp,603
 - set_cancel-Callback,614
 - set_value-Callback,614
- extfn_cancel
 - externer Funktionsaufruf, Schnittstelle,604
- extfn_post_load_library
 - externer Funktionsaufruf, Schnittstelle,605
- extfn_pre_unload_library
 - externer Funktionsaufruf, Schnittstelle,605
- extfn_use_new_api
 - externer Funktionsaufruf, Schnittstelle,604
- F**
- fastLauncherEnabled
 - konfigurierbare Option,1011
- Favoritenliste
 - konfigurierbare Optionen,1011
- Fehler
 - HTTP-Codes,832
 - SOAP-Fehler,832
 - sqlcode, SQLCA-Feld,496
- Fehlerbehandlung
 - Cursor positionieren,10
 - Java,403
 - Java in der Datenbank-Methoden,414
 - ODBC,397
 - SQL Anywhere-.NET-Datenprovider,66
- Fehlercodes
 - SQL Anywhere-Exit-Codes,945
- Fehlermeldungen
 - Embedded SQL-Funktion,561
- Festschreiben
 - Transaktionen aus ODBC,370
- Fetch
 - ODBC,390
- FETCH FOR UPDATE
 - Embedded SQL,32
 - ODBC,32
- Fetch, Vorgänge
 - Beschränkungen,9
 - Cursor,10
 - Mehrere Zeilen,11
 - scrollfähige Cursor,12
- FETCH-Anweisung
 - Cursor in Embedded SQL,516
 - dynamische Abfragen,503
 - mehrzeilig,518
 - verwenden,514
 - weit,518
- fetchall-Methode
 - Python,665
- Fette Cursor
 - Info,11
- FieldCount-Eigenschaft
 - SADatReader-Klasse [SQL Anywhere .NET-API],247
- FileDataSourceName-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],201
- fill_s_sqlda-Funktion
 - Info,556
- fill_sqlda-Funktion
 - Info,557
- fill_sqlda_ex-Funktion
 - Info,557
- ForceStart, Verbindungsparameter
 - db_start_engine,551
- ForceStart-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],201
- ForeignKeys-Feld

SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],270
free_filled_sqlda-Funktion
 Info,558
free_sqlda-Funktion
 Info,558
free_sqlda_noind-Funktion
 Info,559
Funktion WRITE_CLIENT_FILE
 ODBC-Client-API, Callback-Funktion,375
Funktionen
 Anforderungen für Webclients,799
 CLR,622
 DBTools-Funktionen aufrufen,873
 Embedded SQL,532
 externe,601
 SQL Anywhere PHP-Modul,684
 Webclients,798

G

Gebundene Parameter
 Datenausrichtung,386
 ODBC,379
 Prozedurparameter,393
 SQLBindParameter-Funktion,380
 vorbereitete Anweisungen,3
Gebundene Spalten
 Datenausrichtung,386
 ODBC,390
 Prozedur-Ergebnismengen,393
Gemischte Cursor
 ODBC,33
Gespeicherte Funktionen
 externe Funktionen aufrufen,601
Gespeicherte Prozeduren
 CLR,622
 Ergebnismengen in ESQL,529
 externe Funktionen aufrufen,601
 in dynamischem SQL ausführen,528
 in dynamischem SQL erstellen,528
 INOUT-Parameter und Java,416
 Java in der Datenbank,415
 OUT-Parameter und Java,416
 SQL Anywhere-.NET-Datenprovider,63
get_piece
 Info,612
get_value-Funktion
 Info,612
 verwenden,617
getAutoCommit-Methode
 JDBC,434
GetBoolean-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],227
GetByte-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],228
GetBytes-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],228
 verwenden,62
GetChar-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],229
GetChars-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],230
 verwenden,62
GetConnection-Methode
 Instanzen,434
GetData-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],231
GetDataSources-Methode
 SADatSourceEnumerator-Klasse [SQL Anywhere .NET-API],251
GetDataTypeName-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],232
GetDateTime-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],232
GetDateTimeOffset-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],233
GetDecimal-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],233
GetDeleteCommand-Methode
 SACommandBuilder-Klasse [SQL Anywhere .NET-API],156
GetDouble-Methode
 SADatReader-Klasse [SQL Anywhere .NET-API],234
GetEnumerator-Methode

- SADaReader-Klasse [SQL Anywhere .NET-API],234
 - SAErrorCollection-Klasse [SQL Anywhere .NET-API],255
 - SAParameCollection-Klasse [SQL Anywhere .NET-API],296
 - GetFieldType-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],235
 - GetFillParameters-Methode
 - SADaAdapter-Klasse [SQL Anywhere .NET-API],218
 - GetFloat-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],235
 - GetGuid-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],236
 - GetInsertCommand-Methode
 - SACommandBuilder-Klasse [SQL Anywhere .NET-API],158
 - GetInt16-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],236
 - GetInt32-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],237
 - GetInt64-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],237
 - GetKeyword-Methode
 - SAConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],210
 - GetName-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],238
 - GetObjectData-Methode
 - SAException-Klasse [SQL Anywhere .NET-API],258
 - GetOrdinal-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],238
 - GetSchema-Methode
 - SAConnection-Klasse [SQL Anywhere .NET-API],177
 - GetSchemaTable-Methode
 - SADaReader verwenden,50
 - SADaReader-Klasse [SQL Anywhere .NET-API],239
 - GetString-Methode
 - SADaReader-Klasse,82
 - SADaReader-Klasse [SQL Anywhere .NET-API],241
 - GetTimeSpan-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],241
 - verwenden,63
 - GetUInt16-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],242
 - GetUInt32-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],243
 - GetUInt64-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],243
 - GetUpdateCommand-Methode
 - SACommandBuilder-Klasse [SQL Anywhere .NET-API],160
 - getUpdateCounts
 - BatchUpdateException,437
 - GetUseLongNameAsKeyword-Methode
 - SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],165
 - SAConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],210
 - GetValue-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],243
 - GetValues-Methode
 - SADaReader-Klasse [SQL Anywhere .NET-API],245
 - GNU-Compiler
 - Unterstützung von Embedded SQL,473
 - GRANT-Anweisung
 - JDBC,445
- ## H
- Handles
 - ODBC zuweisen,369
 - über ODBC,368
 - HasRows-Eigenschaft
 - SADaReader-Klasse [SQL Anywhere .NET-API],248

Header
 in SOAP-Webdiensten,776
 Zugriff in HTTP-Webdiensten,772

Header-Dateien
 Embedded SQL,473
 ODBC,361

HEADER-Klausel
 verwalten,803

Hintergrundverarbeitung
 Callback-Funktionen,531

Hinweise
 OData Server,456

Hinzufügen
 JAR-Dateien,411
 Klassen für Java in der Datenbank,411

History-Enumeration [Datenbanktools-API]
 Beschreibung,890

HKEY_CURRENT_USER
 Benutzerdatenquellen,977

HKEY_LOCAL_MACHINE
 Administrationstools,1001
 Ereignisprotokoll,1019
 Intel-Speichertreiber, Konfiguration,1018
 ODBC-Treiber,974
 Systemdatenquellen,977
 Systempfad,1002

Host
 Zugriff auf HTTP-Header,774

Host-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],201
 SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],319

Hostvariablen
 Beispiel,773
 Datentypen,487
 deklarieren,486
 in Batches nicht unterstützt,486
 Info,485
 SQLDA,506
 Verwendung,491

HTML-Dienste
 erstellen,763
 Info,760
 kommentieren,766
 löschen,766
 Schnellstart,796
 Schnellstart für Webclients,794

 Schnellstart für Webserver,755

HTTP-Anforderungen
 Strukturen,830

HTTP-Anforderungs-Header
 Verwaltung,803

HTTP-Anforderungsheader
 Zugriff,774

HTTP-Header
 Zugriff,774

HTTP-Protokoll
 aktivieren,757
 konfigurieren,758

HTTP-Systemprozeduren
 alphabetische Liste,788

HTTP_HEADER-Funktion
 Beispiel,773

HTTP_VARIABLE-Funktion
 Beispiel,773

HttpMethod
 Zugriff auf HTTP-Header,775

HttpQueryString
 Zugriff auf HTTP-Header,775

HTTPS-Protokoll
 aktivieren,757
 konfigurieren,758

HttpStatus
 Zugriff auf HTTP-Header,775

HttpURI
 Zugriff auf HTTP-Header,775

HttpVersion
 Zugriff auf HTTP-Header,775

I

iAnywhere ODBC, Treibermanager
 Unix,364

iAnywhere.Data.SQLAnywhere
 Entity Framework-Unterstützung,42

iAnywhere.Data.SQLAnywhere-Namespace
 SQL Anywhere-.NET-API-Referenz,105

iAnywhere.Data.SQLAnywhere.dll
 Deployment von .NET-Clients,960

iAnywhere.Data.SQLAnywhere.dll.config
 Deployment von .NET-Clients,960

iAnywhere.Data.SQLAnywhere.gac
 Deployment von .NET-Clients,960

iAnywhere.Data.SQLAnywhere.v3.5.dll
 Deployment von .NET-Clients,960

- iAnywhere.Data.SQLAnywhere.v4.0.dll
 - Deployment von .NET-Clients,960
- iAnywhere.SQLAnywhere.Server-Namespace
 - SQL Anywhere-.NET-API-Referenz,105
- iastor-Registrierungseintrag
 - Beschädigung der Datenbank verhindern,1018
- iastorv-Registrierungseintrag
 - Beschädigung der Datenbank verhindern,1018
- IdleTimeout-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],202
- Import-Anweisung
 - jConnect,424
- Importbibliotheken
 - Alternativen,476
 - DBTools,872
 - Einführung,468
 - Embedded SQL,474
 - ODBC,361
 - Windows Mobile ODBC,362
- INCLUDE-Anweisung
 - SQLCA,495
- IndexColumns-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],271
- Indexes-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],271
- IndexOf-Methode
 - SABulkCopyColumnMappingCollection-Klasse [SQL Anywhere .NET-API],128
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],297
- Indikator
 - weiter Abruf,518
- Indikatorvariablen
 - Datentypkonvertierung,494
 - Info,493
 - kürzen,494
 - NULL,493
 - ODBC,382
 - SQLDA,506
 - Zusammenfassungen der Werte,494
- InfoMessage-Ereignis
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],189
- InitString-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],187
- SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],202
- INOUT-Parameter
 - Java in der Datenbank,416
- InProcess-Option
 - Verbindungsserver,349,351
- Insensitive Cursor
 - Beispiele für Aktualisieren,20
 - Beispiele für Löschen,18
 - Cursor-Eigenschaften,15
 - Einführung,18
 - Embedded SQL,34
 - Info,22
- INSERT-Anweisung
 - JDBC,437
 - Performance,2
 - Python-Skripten erstellen,665
- Insert-Methode
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],298
- InsertCommand-Eigenschaft
 - SADDataAdapter-Klasse [SQL Anywhere .NET-API],219
- InsertDynamic-Methode
 - JDBCExample,440,441
- InsertStatic-Methode
 - JDBCExample,438
- INSTALL JAVA-Anweisung
 - zum Installieren einer JAR,411
 - zum Installieren von Java-Klassen,411
- Installation
 - dialogfreie Installation,956
 - Produktcodes,960
 - Registrierungsschlüssel,956
- Installationsoption
 - SetupVSPackage,960
- Installationsprogramme
 - Deployment,948
- Installationsschlüssel
 - dialogfreie Installation,956
- Installer
 - dialogfreie Installation,956
- Installieren
 - JAR-Dateien in einer Datenbank,411
 - Java-Klassen in einer Datenbank,410
 - Java-Klassen in einer Datenbank mit SQL,411

Java-Klassen in einer Datenbank mit Sybase Central,411	SAPparameter-Klasse [SQL Anywhere .NET-API],283
jConnect-Metadaten-Unterstützung,425	IsolationLevel-Eigenschaft
Instance-Eigenschaft	SATransaction-Klasse [SQL Anywhere .NET-API],324
SADataSourceEnumerator-Klasse [SQL Anywhere .NET-API],251	ISOLATIONLEVEL_BROWSE
Instance-Feld	Info,869
SAFactory-Klasse [SQL Anywhere .NET-API],265	ISOLATIONLEVEL_CHAOS
Integrated-Eigenschaft	Info,869
SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],202	ISOLATIONLEVEL_CURSORSTABILITY
Intel-Speichertreiber	Info,869
Beschädigung der Datenbank verhindern,1018	ISOLATIONLEVEL_ISOLATED
Interactive SQL	Info,869
Deployment,992,1011	ISOLATIONLEVEL_READCOMMITTED
Deployment unter Linux und Unix,1003	Info,869
Deployment unter Windows,994	ISOLATIONLEVEL_READUNCOMMITTED
Deployment von dbisqlc,1015	Info,869
JDBC-Escape-Syntax,449	ISOLATIONLEVEL_REPEATABLE_READ
OEM-Konfigurationen,1011	Info,869
Optionseinstellungen in bereitgestellten Anwendungen,1014	ISOLATIONLEVEL_SERIALIZABLE
von Unix unterstützte Deployment-Plattformen,1003	Info,869
Interactive SQL-Dienstprogramm (dbisql),	ISOLATIONLEVEL_UNSPECIFIED
Unix-unterstützte Deployment-Plattformen,1003	Info,869
Interactive SQL-Optionen	Isolationsstufe
Einstellungen in bereitgestellten Anwendungen sperren,1014	JDBC,434
interop	readonly-statement-snapshot,40
Webdienste,825	snapshot,40
iODBC	statement-snapshot,40
Treibermanager,366	Isolationsstufen
IPV6-Eigenschaft	ADO-Programmierung,344
SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],319	Anwendungen,39
iSchnittstellenbibliotheken	Cursor,10
SQL Anywhere C-API,566	Cursor-Sensitivität,32
IsClosed-Eigenschaft	DTC,869
SADataReader-Klasse [SQL Anywhere .NET-API],248	Einstellungen für das SATransaction-Objekt,65
IsDBNull-Methode	SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT,388
SADataReader-Klasse [SQL Anywhere .NET-API],245	SA_SQL_TXN_SNAPSHOT,388
IsFixedSize-Eigenschaft	SA_SQL_TXN_STATEMENT_SNAPSHOT,388
SAPparameterCollection-Klasse [SQL Anywhere .NET-API],300	SQL_TXN_READ_COMMITTED,388
IsNullable-Eigenschaft	SQL_TXN_READ_UNCOMMITTED,388
	SQL_TXN_REPEATABLE_READ,388
	SQL_TXN_SERIALIZABLE,388
	Verlorene Aktualisierungen,30
	isql.jar
	Deployment unter Mac OS X,1005
	Deployment unter Unix,1005
	Deployment unter Windows,996

- IsReadOnly-Eigenschaft
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],300
- IsSynchronized-Eigenschaft
 - SAPParameterCollection-Klasse [SQL Anywhere .NET-API],300

J

- jaccess.jar
 - Deployment von Java-basierten Administrationstools,1000
- JAR- und ZIP-Dateierstellung
 - verwenden,411
- JAR-Dateien
 - aktualisieren,412
 - hinzufügen,411
 - installieren,410
 - mit SQL installieren,411
 - über Sybase Central installieren,411
 - Versionen,412
- Java
 - externe Umgebung,635
 - in der Datenbank,401
 - JDBC,419
- Java in der Datenbank
 - Deployment,1016
 - die wichtigsten Funktionen,401
 - Fehlerbehandlung,403
 - Info,401
 - Java VM ,403
 - Java VM wählen,405
 - Klassen installieren,410
 - Klassen speichern,402
 - main-Methode,413
 - NoSuchMethodException,414
 - Rückgabe von Ergebnismengen,415
 - Sicherheitsmanagement,417
 - VM starten,417
 - VM stoppen,417
 - VM-Shutdown-Hooks,417
- Java VM
 - auswählen,405
 - Shutdown-Hooks,417
 - starten,417
 - stoppen,417
 - Umgebungsvariablen,405
- Java, gespeicherte Prozeduren
 - Beispiel,415
 - Info,415
- Java-Klassen
 - Beispielinstallation,407
 - hinzufügen,411
 - installieren,411
- Java-Laufzeitumgebung
 - Java in der Datenbank verwenden,405
- Java-Methoden aufrufen
 - praktische Einführung,408
- Java-Methoden, Zugriff
 - praktische Einführung,408
- JAVA-Schlüsselwort
 - externe Umgebung,635
- Java-Signaturen
 - CREATE PROCEDURE-Anwendung [benutzerdefiniert],637
- JAVA_HOME-Umgebungsvariable
 - Java VM starten,405
- java_vm_options-Option
 - verwenden,405
- JavaAccessBridge.dll
 - Deployment von Java-basierten Administrationstools,1000
- JAVAHOME-Umgebungsvariable
 - Java VM starten,405
- JAWTAccessBridge.dll
 - Deployment von Java-basierten Administrationstools,1000
- JAX-WS
 - installieren,860
 - praktische Einführung,856
 - Versionen,860
- JComponents1600.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- jconn4.jar
 - Deployment von Datenbankservern,1016
- jConnect laden,425
- jConnect-JDBC-Treiber laden,430
- jConnect
 - CLASSPATH-Umgebungsvariable,424
 - Deployment von JDBC-Clients,982
 - Einrichtung der Datenbank,425
 - externe Verbindungen,427
 - gelieferte Versionen,424
 - herunterladen,424

- Info,424
- JDBC-Treiber wählen,420
- laden,425
- Metadaten-Unterstützung installieren,425
- Pakete,424
- serverseitige Verbindungen,431
- Systemobjekte,425
- URL,426
- verbinden,426
- JDBC
 - Arten der Benutzung,420
 - autocommit,434
 - Autocommit-Modus,37
 - Autocommit-Verhalten steuern,37
 - Batch-Einfügungen,438
 - Batch-Einfügungen, Beispiel,442
 - Beispiel-Quellcode,420
 - Beispielverbindung,427
 - Client-Verbindungen,427
 - clientseitig,422
 - Cursortypen,15
 - Datenzugriff,436
 - DELETE-Anweisung,437
 - Deployment von JDBC-Clients,982
 - Einführung in die Programmierung,419
 - Ergebnismengen,443
 - Escape-Syntax in Interactive SQL,449
 - Info,419
 - INSERT-Anweisung,437
 - jConnect,424
 - Privilegien,445
 - serverseitig,422
 - serverseitige Verbindungen,431
 - SQL Anywhere-JDBC-Treiber,423
 - SQL-Anweisungen,1
 - Standardwerte für die Verbindung,434
 - Transaktion, Isolationsstufe,434
 - Überblick über die Anwendungen,422
 - UPDATE-Anweisung,437
 - Verbindung mit einer Datenbank,426
 - Verbindungen,422
 - Verbindungscode,428
 - Voraussetzungen,420
 - vorbereitete Anweisungen,440
- JDBC-Callback
 - Beispiel,445
- JDBC-Escape-Syntax
 - in Interactive SQL,449
- JDBC-ODBC-Brücke
 - SQL Anywhere-JDBC-Treiber,420
- JDBC-Treiber
 - Kompatibilität,420
 - OSGi Bundle,420
 - Performance,420
 - wählen,420
- JDBCExample-Klasse
 - Info,436
- JDBCExample.java
 - Info,436
- jlogon.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- jodbc4.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- JRE
 - Java in der Datenbank verwenden,405
 - Version prüfen unter Mac OS X,1004
- jre170
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - Java Runtime Environment,995
- jre_170
 - Deployment unter Mac OS X,1005
- js.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- JSON-Dienste
 - erstellen,763
 - Info,760
 - kommentieren,766
 - löschen,766
 - Schnellstart,796
 - Schnellstart für Webclients,794
 - Schnellstart für Webserver,755
- jsyblib1600.dll
 - Deployment unter Windows,996
- jsyblib1600.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996

K

- Kapazitäten
 - unterstützte,754
- Kennwörter
 - in jConnect verschlüsseln,425
 - in Open Client verschlüsseln,754
 - Interactive SQL,1011
- Kerberos-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],202
- Keys-Eigenschaft
 - SACConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],212
- Keyset-gesteuerte Cursor
 - ODBC,33
 - wertsensitiv,26
- Klassen
 - aktualisieren,412
 - erstellen,410
 - installieren,410
 - Versionen,412
- Klassendatei
 - in einer Datenbank installieren,411
- Klauseln
 - WITH HOLD,10
- Kommentieren
 - Webdienste,766
- Kompilieren und Verknüpfen, Prozess
 - Info,468
- Konfigurationsdateien
 - OData Server,457
- Konfigurieren
 - Administrationstools für Deployments,1011
 - Interactive SQL für Deployments,1011
 - Sybase Central für Deployments,1011
- Konsolendienstprogramm (dbconsole)
 - Deployment,992
 - Deployment unter Linux und Unix,1003
 - Deployment unter Windows,994
- Konventionen
 - Dateinamen,950
- Konvertierung
 - Datentypen,494
- Kürzen
 - bei FETCH,494
 - FETCH-Anweisung,494
 - Indikatorvariablen,494

L

- Language-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],203
- LazyClose-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],203
- LD_LIBRARY_PATH-Umgebungsvariable
 - Deployment,950
 - Deployment-Einstellung,1009
- LDAP-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],319
- length, SQLDA-Feld
 - Info,506
 - Werte,508
- Lesezeichen
 - Info,16
 - ODBC-Cursor,392
- libdbcapi.dylib
 - PHP-Unterstützungsmodul,647
- libdbcapi.so
 - PHP-Unterstützungsmodul,647
- libdbcapi_r.dylib
 - Deployment von,1024
 - Deployment von PHP-Clients,984
 - PHP-Unterstützungsmodul,647
- libdbcapi_r.so
 - Deployment von,1024
 - Deployment von PHP-Clients,984
 - PHP-Unterstützungsmodul,647
- libdbcis16.dylib
 - Deployment von Datenbankservern,1016
- libdbcis16.so
 - Deployment von Datenbankservern,1016
- libdbencod16_r
 - Deployment von SQL Remote,1030
- libdbextenv16_r.dylib
 - Deployment,1024
 - PHP-Unterstützungsmodul,647
- libdbextenv16_r.so
 - Deployment,1024
 - PHP-Unterstützungsmodul,647
- libdbextf.dylib
 - Deployment von Datenbankservern,1016
- libdbextf.so
 - Deployment von Datenbankservern,1016

libdbfile16_r	libdbjodbc16.so
Deployment von SQL Remote,1030	Deployment von Datenbankservern,1016
libdbfips16_r.dylib	libdbjodbc16.so.1
Deployment von PHP-Clients,984	Deployment unter Unix,1005
libdbfips16_r.so	libdblib16.dylib
Deployment von PHP-Clients,984	Deployment von Datenbank-
libdbftp16_r	Dienstprogrammen,1028
Deployment von SQL Remote,1030	Deployment von PHP-Clients,984
libdbicu16.dylib	libdblib16.so
Deployment von JDBC-Clients,971	Deployment von Datenbank-
Deployment von ODBC-Clients,971	Dienstprogrammen,1028
libdbicu16.so	Deployment von Embedded SQL-Clients,981
Deployment von ODBC-Clients,971	libdblib16_r
libdbicu16_r.dylib	Deployment von SQL Remote,1030
Deployment von Datenbankservern,1016	libdblib16_r.dylib
Deployment von JDBC-Clients,983	Deployment unter Mac OS X,1005
Deployment von ODBC-Clients,971	Deployment von Datenbank-
Deployment von PHP-Clients,984	Dienstprogrammen,1028
libdbicu16_r.so	libdblib16_r.so
Deployment unter Mac OS X,1005	Deployment von Datenbank-
Deployment unter Unix,1005	Dienstprogrammen,1028
Deployment von Datenbankservern,1016	Deployment von PHP-Clients,984
Deployment von JDBC-Clients,983	libdblib16_r.so.1
Deployment von ODBC-Clients,971	Deployment unter Unix,1005
Deployment von PHP-Clients,984	libdbodbc16
libdbicudt16.dylib	Unix-ODBC-Treiber,363
Deployment von Datenbankservern,1016	libdbodbc16.dylib
Deployment von JDBC-Clients,971,983	Deployment von Datenbankservern,1016
Deployment von ODBC-Clients,971	Deployment von ODBC-Clients,971
Deployment von PHP-Clients,984	libdbodbc16.so
libdbicudt16.so	Deployment von Datenbankservern,1016
Deployment unter Mac OS X,1005	Deployment von ODBC-Clients,971
Deployment unter Unix,1005	libdbodbc16_n.dylib
Deployment von Datenbankservern,1016	Deployment von Datenbankservern,1016
Deployment von JDBC-Clients,983	Deployment von ODBC-Clients,971
Deployment von ODBC-Clients,971	libdbodbc16_n.so
Deployment von PHP-Clients,984	Deployment von Datenbankservern,1016
libdbjdbc16.dylib	Deployment von ODBC-Clients,971
Deployment,1024	libdbodbc16_r.dylib
Deployment von Datenbankservern,1016	Deployment unter Mac OS X,1005
Deployment von JDBC-Clients,983	Deployment von Datenbankservern,1016
libdbjdbc16.so	Deployment von ODBC-Clients,971
Deployment,1024	libdbodbc16_r.so
Deployment von Datenbankservern,1016	Deployment von Datenbankservern,1016
Deployment von JDBC-Clients,983	Deployment von ODBC-Clients,971
libdbjodbc16.dylib	libdbodbc16_r.so.1
Deployment unter Mac OS X,1005	Deployment unter Unix,1005
Deployment von Datenbankservern,1016	libdbodbcansi16_r

- ANSI-ODBC-Treiber,366
- libdbodm16
 - Info,364
 - Unix-ODBC-Treibermanager,364
- libdbodm16.dylib
 - Deployment unter Mac OS X,1005
 - Deployment von ODBC-Clients,971
- libdbodm16.so
 - Deployment von ODBC-Clients,971
- libdbodm16.so.1
 - Deployment unter Unix,1005
- libdbsoftsp_r.so
 - Deployment von Datenbank-Dienstprogrammen,1028
- libdbput16_r.dylib
 - Deployment unter Mac OS X,1005
- libdbput16_r.so.1
 - Deployment unter Unix,1005
- libdbrsa16.dylib
 - RSA-Verschlüsselung,1026
- libdbrsa16.so
 - RSA-Verschlüsselung,1026
- libdbrsa16_r.dylib
 - Deployment von PHP-Clients,984
 - RSA-Verschlüsselung,1026
- libdbrsa16_r.so
 - Deployment von PHP-Clients,984
- libdbrsakp16.dylib
 - Deployment von JDBC-Clients,984
 - Deployment von Open Client,992
- libdbrsakp16.so
 - Deployment von JDBC-Clients,984
 - Deployment von Open Client,992
- libdbrsakp16_r.dll
 - Deployment von Datenbankservern,1016
- libdbrsakp16_r.dylib
 - Deployment von Datenbankservern,1016
- libdbscript16_r.dylib
 - Deployment von Datenbankservern,1016
- libdbscript16_r.so
 - Deployment von Datenbankservern,1016
- libdbserv16_r.dylib
 - Deployment von Datenbankservern,1016
- libdbserv16_r.so
 - AES-Verschlüsselung,1026
 - Deployment von Datenbankservern,1016
- libdbsmtp16_r
 - Deployment von SQL Remote,1030
- libdbtasks16.dylib
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von JDBC-Clients,971
- libdbtasks16.so
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von Embedded SQL-Clients,981
 - Deployment von ODBC-Clients,971
 - Deployment von PHP-Clients,984
- libdbtasks16_r
 - Deployment von SQL Remote,1030
- libdbtasks16_r.dylib
 - Deployment unter Mac OS X,1005
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
 - Deployment von PHP-Clients,984
- libdbtasks16_r.so
 - Deployment von Datenbank-Dienstprogrammen,1028
 - Deployment von Datenbankservern,1016
 - Deployment von JDBC-Clients,983
 - Deployment von ODBC-Clients,971
- libdbtasks16_r.so.1
 - Deployment unter Unix,1005
- libdbtool16_r
 - Deployment von SQL Remote,1030
 - Info,871
- libdbtool16_r.dylib
 - Deployment unter Mac OS X,1005
 - Deployment von Datenbank-Dienstprogrammen,1028
- libdbtool16_r.so
 - Deployment von Datenbank-Dienstprogrammen,1028
- libdbtool16_r.so.1
 - Deployment unter Unix,1005
- libjsyblib1600_r.dylib
 - Deployment unter Mac OS X,1005
- libjsyblib1600_r.so.1
 - Deployment unter Unix,1005
- libmljodbc16.dylib
 - Deployment unter Mac OS X,1005
- libmljodbc16.so.1
 - Deployment unter Unix,1005

libsybbr.dll
 Deployment von Datenbankservern,1016
libsybbr.dylib
 Deployment von Datenbankservern,1016
libsybbr.so
 Deployment von Datenbankservern,1016
libulscutil16.so.1
 Deployment unter Unix,1005
LINQ-Unterstützung
 .NET-Datenprovider-Funktionen,42
 LinqSample, .NET-Datenprovider-
 Beispielprojekt,43
LinqSample
 .NET-Datenprovider-Beispielprojekt,43
Linux
 Hinweise zu Deploymentmethoden,949
 Verzeichnisstruktur,949
LivenessTimeout-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL
 Anywhere .NET-API],203
Lizenzierung
 Webclients,779
LocalOnly-Eigenschaft
 SATcpOptionsBuilder-Klasse [SQL
 Anywhere .NET-API],319
lockedPreferences
 konfigurierbare Option,1011
LogFile-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL
 Anywhere .NET-API],203
LONG BINARY-Datentyp
 Embedded SQL,522
 in Embedded SQL abrufen,524
 in Embedded SQL senden,526,527
LONG NVARCHAR-Datentyp
 Embedded SQL,522
 in Embedded SQL abrufen,524
 in Embedded SQL senden,526,527
LONG VARCHAR-Datentyp
 Embedded SQL,522
 in Embedded SQL abrufen,524
 in Embedded SQL senden,526,527
LONGBINARY-Datentyp
 Embedded SQL,487
LONGNVARCHAR-Datentyp
 Embedded SQL,487
LONGVARCHAR-Datentyp
 Embedded SQL,487

Löschen
 Webdienste,766

M

Mac OS X
 Hinweise zu Deploymentmethoden,949
 JRE-Versionen prüfen,1004
 Verzeichnisstruktur,949
main-Methode
 Java in der Datenbank,413
Makros
 _SQL_OS_WINDOWS,476
Managementtools
 dbtools,871
manual commit, Modus
 Implementierung,39
 steuern,37
 Transaktionen,37
maximumDisplayedRows
 konfigurierbare Option,1011
MaxPoolSize-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL
 Anywhere .NET-API],204
MDAC
 Deployment,965
 Version,965
Mehrere Ergebnismengen
 DESCRIBE-Anweisung,530
mehrere Ergebnismengen
 OEM-konfigurierbare Option ,1011
Mehrere Threads in Anwendungen
 Embedded SQL,498
Mehrere Threads, Anwendungen
 Java in der Datenbank,414
 mehrere SQLCA-Bereiche in Embedded SQL,500
 ODBC-Funktionsunterstützung,360
 ODBC-Verbindungen,374
Mehrfache Ergebnismengen
 ODBC,393
Mehrzeilenabrufe
 ESQL,518
Mehrzeilige Abfragen
 Cursor,516
Mehrzeilige Abrufe
 ESQL,518
 Info,11,518
Mehrzeilige Einfügungen

- ESQL,518
 - Mehrzeiliges Speichern
 - ESQL,518
 - Meldungen
 - callback,548
 - Server,548
 - MergeModule.CABinet
 - Deployment-Assistent,952
 - Message-Eigenschaft
 - SAError-Klasse [SQL Anywhere .NET-API],253
 - SAException-Klasse [SQL Anywhere .NET-API],259
 - SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API],267
 - MessageType-Eigenschaft
 - SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API],267
 - MetaDataCollections-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],272
 - Metadaten-Unterstützung
 - installieren für jConnect,425
 - Methodensignaturen
 - Java,637
 - Microsoft SQL Server Management Studio
 - Verbindungsserver,349,351
 - Microsoft Transaction Server
 - dreischichtige Datenverarbeitung,867
 - Microsoft Visual C++
 - Unterstützung von Embedded SQL,473
 - MIME
 - Typen einstellen,771
 - MIME-Typen
 - Webdienste, praktische Einführung,834
 - MinPoolSize-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],204
 - Mit Leerzeichen aufgefüllt
 - Zeichenfolgen in Embedded SQL,483
 - Mitgliedschaft
 - Ergebnismengen,17
 - mldesign.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - mljdbc16.dll
 - Deployment unter Windows,996
 - mlmon.ini
 - Deployment von Administrationstools,992
 - mlplugin.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - mlstream.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - mobilink.jpr
 - Deployment der Administrationstools unter Windows,998
 - Deployment von Administrationstools unter Linux/Unix/Mac OS X,1007
 - Modell
 - OData Producer,463
 - MONEY-Datentyp
 - Open Client-Konvertierung,750
 - MSDASQL
 - OLE DB-Provider,337
 - msiexec
 - Deployment-Assistent,952
 - dialogfreie Installation,956
 - Info,954
 - MSSqlToSA.xml
 - Deployment von .NET-Clients,960
 - myDispose-Methode
 - SADataReader-Klasse [SQL Anywhere .NET-API],246
 - MyIP-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],319
- ## N
- Nachrichtenagent (dbremote)
 - Deployment von SQL Remote,1030
 - name, SQLDA-Feld
 - Info,506
 - NAMESPACE-Klausel
 - verwalten,809
 - namespaces
 - Webdienste,824
 - NativeError-Eigenschaft
 - SAError-Klasse [SQL Anywhere .NET-API],253
 - SAException-Klasse [SQL Anywhere .NET-API],259

SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API],267
 NCHAR-Datentyp
 Embedded SQL,487
 Netzwerkdienst
 OLE DB,348
 NewPassword-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],204
 NEXT_CONNECTION-Funktion
 Beispiel,785
 NEXT_HTTP_HEADER-Funktion
 Beispiel,773
 NEXT_HTTP_VARIABLE-Funktion
 Beispiel,773
 NextResult-Methode
 SADataReader-Klasse [SQL Anywhere .NET-API],246
 ngdbc.jar
 Deployment unter Unix,1005
 Deployment unter Windows,996
 Nicht verwalteter Code
 .NET-Datenprovider,963
 nicht verwalteter Code
 dbdata.dll,74
 NO_SCROLL-Cursor
 Embedded SQL,34
 Info,22
 NodeType-Eigenschaft
 SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],204
 NotifyAfter-Eigenschaft
 SABulkCopy-Klasse [SQL Anywhere .NET-API],114
 ntodbc.h
 Info,361
 Kompilierungsplattform,382
 NuGet
 Entity Framework 4,67
 NULL
 Dynamic SQL,504
 Indikatorvariablen,493
 Nullabschlusszeichen für Zeichenfolge
 Embedded SQL-Datentyp,483
 NVARCHAR-Datentyp
 Embedded SQL,487

O

Objekte
 Speicherformat,412
 OData
 Info,455
 OData Producer
 Dienstmodelle,463
 Info,455
 Optionen,457
 OData Server
 Einschränkungen,456
 Info,455
 Optionen,457
 Sicherheitshinweise,456
 OData Server (dbosrv16.exe)
 Syntax,464
 OData Serverstopp (dbostop.exe)
 Syntax,465
 ODBC
 64-Bit-Hinweise,382
 Anwendungen mit mehreren Threads,360
 Aufruf von ODBC-Funktionen aus DllMain,377
 Autocommit-Modus,37
 Autocommit-Verhalten steuern,37
 Beispielanwendung,370
 Beispielprogramm,366
 Benutzerdatenquelle,977
 Cursor,33
 Cursortypen,15
 Cursorunterstützung und Ergebnismengen,388
 Datenausrichtung,386
 Datenquellentypen,977
 Deployment des Treibers,970
 Ergebnismengen,393
 Escape-Syntax,394
 externe Umgebung,625
 Fehlerprüfung,397
 FETCH FOR UPDATE,32
 gespeicherte Prozeduren,393
 Handles,368
 Header-Dateien,361
 Importbibliotheken,361
 Info,359
 linken,361
 mehrfache Ergebnismengen,393
 Optionen,374
 Registrierungseinträge,977

- SQL-Anweisungen,1
- SQLSetConnectAttr,375
- Systemdatenquellen,977
- Treibermanager,360
- Treibername für SQL Anywhere,370
- Übereinstimmung,360
- Unix-Entwicklung,363
- unterstützte Version,360
- Verbindung ohne Datenquelle,980
- Verbindungen für Anwendungen mit mehreren Threads,374
- Voraussetzungen,359
- vorbereitete Anweisungen,380
- Windows Mobile,362
- ODBC-Client
 - Deployment,970
 - Installation,970
- ODBC-Datenquellen
 - Deployment,976
- ODBC-Treiber
 - Unix,363
- ODBC-Treibermanager
 - iODBC,366
 - Unix,364
 - unixODBC,365
 - UTF-32,366
 - Windows-Performance,977
- odbc.h
 - Info,361
- OEM.ini
 - Administrationstools ,1011
- Offset-Eigenschaft
 - SAPparameter-Klasse [SQL Anywhere .NET-API],283
- OLE DB
 - Aktualisierungen,343
 - Autocommit-Modus,37
 - Autocommit-Verhalten steuern,37
 - Cursor,33
 - Cursortypen,15
 - Daten mit einem Cursor aktualisieren,343
 - Deployment,964
 - Deployment des Providers,964
 - Info,337
 - Microsoft-Verbindungsserver einrichten,348
 - ODBC,337
 - unterstützte Plattformen,338
 - unterstützte Schnittstellen,352
 - Verbindungsparameter,345
 - Verbindungspooling,347
- OLE DB- und ADO-Entwicklung
 - Info,337
- OLE DB- und ADO-Programmierschnittstelle
 - Einführung,337
 - Info,337
- OLE-Transaktionen
 - dreischichtige Datenverarbeitung,866,867
- Online-Sicherungen
 - Embedded SQL-Funktionen,531
- Open Client
 - Anforderungen,749
 - Architektur,748
 - Autocommit-Modus,37
 - Autocommit-Verhalten steuern,37
 - Cursortypen,15
 - Datentypbereiche,750
 - Datentypkompatibilität,749
 - Deployment von Open Client-Anwendungen,992
 - Einführung,747
 - Einschränkungen,753
 - Kennwörter verschlüsseln,754
 - Schnittstelle,747
 - SQL,751
 - SQL Anywhere-Einschränkungen,753
 - SQL-Anweisungen,1
- OPEN-Anweisung
 - Cursor in Embedded SQL,516
- Open-Methode
 - SACONNECTION-Klasse [SQL Anywhere .NET-API],185
- OPENQUERY
 - Verbindungsserver,348
- Option -d
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- Option -e
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- Option -gn
 - Threads,414
- Option -h
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- Option -k
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -m
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -n
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -q
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -r
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -s
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -u
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -w
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -x
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Option -z
Dienstprogramm für den SQL-Präprozessor (sqlpp),469

Optionen
OData,457
ODBC-Anwendungen,374
Webdienste,789

OSGi-Deployment-Bundle
JDBC 4.0-Treiber,420

OUT-Parameter
Java in der Datenbank,416

OWASP
Webdienste,787

P

Padding-Enumeration [Datenbanktools-API]
Beschreibung,890

Pakete
installieren,411
jConnect,424

Parallele Sicherungen
db_backup-Funktion,533

Parallelitätswerte

SQL_CONCUR_LOCK,389
SQL_CONCUR_READ_ONLY,389
SQL_CONCUR_ROWVER,389
SQL_CONCUR_VALUES,389

Parameter
Ersetzung,829

Parameter übergeben
an externe Funktionen,612

Parametergröße, Hinweise
ODBC,382

ParameterName-Eigenschaft
SAPparameter-Klasse [SQL Anywhere .NET-API],283

Parameters-Eigenschaft
SACommand-Klasse [SQL Anywhere .NET-API],150

Password-Eigenschaft
SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],204

pdf-transcoder.jar
Deployment unter Mac OS X,1005
Deployment unter Unix,1005
Deployment unter Windows,996

Performance
Cursor,28
Cursor und Prefetch-Zeilen,28
JDBC,440
JDBC-Treiber,420
vorbereitete Anweisungen,2

Perl
Behandeln mehrerer Ergebnismengen,657
DBD::SQLAnywhere,651
DBD::SQLAnywhere-Skripten schreiben,654
externe Umgebungen,640
Perl/DBI-Unterstützung unter Unix und Mac OS X installieren,653
Perl/DBI-Unterstützung unter Windows installieren,651
Verbindung mit einer Datenbank,655
Zeilen einfügen,658

Perl DBD::SQLAnywhere
Einführung in die Programmierung,651
Info,651

Perl DBI, Modul
Info,651

PERL, Schlüsselwort
Externe Umgebung,640

perlenv.pl

- Deployment von, 1025
- PersistSecurityInfo-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API], 205
- Personal Server
 - Deployment, 1027
- PHP
 - API-Referenz, 684
 - Datenzugriff, 669
 - Deployment von PHP-Clients, 984
 - Erweiterung, 669
 - externe Umgebung, 644
 - Fehlerbehandlung für die externe Umgebung, 646
 - in der Datenbank, 644
 - Info, 669
 - konfigurieren, 990
 - PHP Unterstützung unter Linux/Solaris installieren, 988
 - PHP-Skripten aus gespeicherten Prozeduren aufrufen, 646
 - PHP-Skripten in Webseiten ausführen, 670
 - PHP-Unterstützung unter Windows installieren, 987
 - Skripten erstellen, 672
 - Unterstützung für die externe Umgebung installieren, 645
 - Versionen, 986
- PHP, Hypertext-Präprozessor
 - Info, 669
- php-5.1.[1-*]_sqlanywhere.dll
 - Deployment von PHP-Clients, 984
- php-5.1.[1-*]_sqlanywhere.so
 - Deployment von PHP-Clients, 984
- php-5.1.[1-*]_sqlanywhere_extenv16.dll
 - Deployment, 1025
- php-5.1.[1-*]_sqlanywhere_extenv16.so
 - Deployment, 1025
- php-5.2.[0-*]_sqlanywhere.dll
 - Deployment von PHP-Clients, 984
- php-5.2.[0-*]_sqlanywhere.so
 - Deployment von PHP-Clients, 984
- php-5.2.[0-*]_sqlanywhere_extenv16.dll
 - Deployment, 1025
- php-5.2.[0-*]_sqlanywhere_extenv16.so
 - Deployment, 1025
- php-5.3.[0-*]_sqlanywhere.dll
 - Deployment von PHP-Clients, 984
- php-5.3.[0-*]_sqlanywhere.so
 - Deployment von PHP-Clients, 984
- php-5.3.[0-*]_sqlanywhere_extenv16.dll
 - Deployment, 1025
- php-5.3.[0-*]_sqlanywhere_extenv16.so
 - Deployment, 1025
- php-5.4.[0-*]_sqlanywhere.dll
 - Deployment von PHP-Clients, 984
- php-5.4.[0-*]_sqlanywhere.so
 - Deployment von PHP-Clients, 984
- php-5.4.[0-*]_sqlanywhere_extenv16.dll
 - Deployment, 1025
- php-5.4.[0-*]_sqlanywhere_extenv16.so
 - Deployment, 1025
- PHP-Erweiterung
 - Auswahl, 986
 - Einführung in die Programmierung, 669
 - Linux/Solaris, 988
 - testen, 669
 - Versionen, 986
 - Windows, 987
- PHP-Funktionen
 - sasql_affected_rows, 685
 - sasql_close, 686
 - sasql_commit, 686
 - sasql_connect, 687
 - sasql_data_seek, 687
 - sasql_disconnect, 688
 - sasql_error, 688
 - sasql_errorcode, 689
 - sasql_escape_string, 690
 - sasql_fetch_array, 690
 - sasql_fetch_assoc, 691
 - sasql_fetch_field, 691
 - sasql_fetch_object, 692
 - sasql_fetch_row, 693
 - sasql_field_count, 693
 - sasql_field_seek, 694
 - sasql_free_result, 694
 - sasql_get_client_info, 694
 - sasql_insert_id, 695
 - sasql_message, 695
 - sasql_multi_query, 696
 - sasql_next_result, 696
 - sasql_num_fields, 697
 - sasql_num_rows, 697
 - sasql_pconnect, 698
 - sasql_prepare, 698
 - sasql_query, 699
 - sasql_real_escape_string, 700

-
- sasql_real_query,700
 - sasql_result_all,701
 - sasql_rollback,702
 - sasql_set_option,702
 - sasql_sqlstate,714
 - sasql_stmt_affected_rows,703
 - sasql_stmt_bind_param,704
 - sasql_stmt_bind_param_ex,704
 - sasql_stmt_bind_result,705
 - sasql_stmt_close,706
 - sasql_stmt_data_seek,706
 - sasql_stmt_errno,706
 - sasql_stmt_error,707
 - sasql_stmt_execute,708
 - sasql_stmt_fetch,708
 - sasql_stmt_field_count,709
 - sasql_stmt_free_result,709
 - sasql_stmt_insert_id,709
 - sasql_stmt_next_result,710
 - sasql_stmt_num_rows,710
 - sasql_stmt_param_count,711
 - sasql_stmt_reset,711
 - sasql_stmt_result_metadata,712
 - sasql_stmt_send_long_data,712
 - sasql_stmt_store_result,713
 - sasql_store_result,713
 - sasql_use_result,714
 - PHP-Schlüsselwort
 - Externe Umgebung,646
 - php.ini
 - PHP konfigurieren,990
 - phpenv.php
 - Deployment von,1025
 - PHP-Unterstützungsmodul,647
 - PHPRC
 - Umgebungsvariable,647
 - Plattformen
 - Cursor,15
 - Platzhalter
 - Dynamic SQL,501
 - Plug-Ins
 - Deployment,994
 - policy.16.0.iAnywhere.Data.SQLAnywhere.dll
 - Deployment von .NET-Clients,960
 - policy.16.0.iAnywhere.Data.SQLAnywhere.v3.5.dll
 - Deployment von .NET-Clients,960
 - policy.16.0.iAnywhere.Data.SQLAnywhere.v4.0.dll
 - Deployment von .NET-Clients,960
 - Pooling
 - Webdienste,769
 - POOLING, Option
 - .NET-Datenprovider,46
 - Pooling-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],205
 - Pools
 - Verbindungen mit .NET-Datenprovider,46
 - Positionsbasierte DELETE-Anweisung
 - Info,12
 - Positionsbasierte UPDATE-Anweisung
 - Info,12
 - Positionsbasierte Updates
 - Info,10
 - Praktische Einführungen
 - einfache .NET-Datenbankanwendung entwickeln,87
 - Java in der Datenbank verwenden,404
 - Verwenden von JAX-WS für den Zugriff auf einen SOAP/DISH-Webdienst,856
 - Verwenden von SQL Anywhere für den Zugriff auf SOAP-Dienst,839
 - Verwenden von Visual C# für den Zugriff auf einen SOAP/DISH-Webdienst,848
 - Verwendung des .NET-Datenprovider-Codebeispiels "Simple",79
 - Verwendung des .NET-Datenprovider-Codebeispiels "Table Viewer",83
 - Webserver erstellen und über einen Webclient darauf zugreifen,834
 - Präprozessor
 - ausführen,469
 - Info,467
 - Precision-Eigenschaft
 - SAPParameter-Klasse [SQL Anywhere .NET-API],284
 - Prefetch
 - Cursor,28
 - Cursor-Performance,28
 - Prefetch-Option
 - Cursor,28
 - Prefetch-Vorgänge
 - mehrere Zeilen abrufen,11
 - PrefetchBuffer-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],205
 - PrefetchRows-Eigenschaft

- SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],205
- PREPARE TRANSACTION-Anweisung und Open Client,753
- PREPARE-Anweisung verwenden,501
- Prepare-Methode
 - SACommand-Klasse [SQL Anywhere .NET-API],147
 - verwenden,4
- PreparedStatement-Schnittstelle Info,440
- PrepareStatement-Methode
 - JDBC,4
- Primärschlüssel
 - mit SACommand abrufen,52
 - mit SDataAdapter abrufen,60
- Privilegien
 - JDBC,445
- ProcedureParameters-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],272
- Procedures-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],273
- Produktcodes
 - SQL Anywhere,960
- Programmeintrittspunkte
 - DBTools-Funktionen aufrufen,873
- Programmierschnittstellen
 - C-API,565
 - JDBC-API,419
 - ODBC-API,359
 - Perl DBD::SQLAnywhere-API,651
 - Python-Datenbank-API,661
 - Ruby-APIs,717
 - SQL Anywhere Embedded SQL,467
 - SQL Anywhere OLE DB- und ADO-API,337
 - SQL Anywhere PHP DBI,669
 - SQL Anywhere-.NET-API,41
 - Sybase Open Client-API,747
- Programmstruktur
 - Embedded SQL,475
- Protokolle
 - Webdienste aktivieren,757
 - Webdienste konfigurieren,758
- Protokollieren
 - Webdienst-Clientinformationen,831
- Protokollierung
 - .NET -Unterstützung,75
- Prototyp
 - externe Funktionen,603
- Provider
 - SQL Anywhere ASP.NET-Mitgliedschaftsprovider,97
 - SQL Anywhere ASP.NET-Profilprovider,97
 - SQL Anywhere ASP.NET-Provider für Systemüberwachung,97
 - SQL Anywhere ASP.NET-Provider für Webseiten-Personalisierung,97
 - SQL Anywhere ASP.NET-Rollenprovider,97
 - unterstützt in .NET,42
 - unterstützt in ASP.NET,97
- Prozeduren
 - Anforderungen für Webclients,799
 - CLR,622
 - Embedded SQL,528
 - Ergebnismengen in ESQL,529
 - ODBC,393
 - Webclients,798
- Pufferlänge
 - ODBC,382
- PUT-Anweisung
 - mehrzeilig,518
 - weit,518
 - Zeilen durch einen Cursor ändern,12
- Python
 - Ausführen von SQL-Anweisungen,656
 - commit-Methode,665
 - Cursor erstellen,665
 - Datenbanktypen,667
 - in Tabellen einfügen,665
 - Kontrolle über die Typkonvertierung,667
 - mehrere Einfügungen,666
 - Python-Unterstützung unter Unix und Mac OS X installieren,662
 - Python-Unterstützung unter Windows installieren,662
 - SQL-Anweisungen ausführen,665
 - sqlanydb,661
 - sqlanydb-Skripten erstellen,663
 - Verbindungen erstellen,664
 - Verbindungen schließen,664
- Python-Datenbank-API
 - Einführung ins Programmieren,661
- Python-Datenbankunterstützung

Info,661

Q

QuoteIdentifier-Methode

SACommandBuilder-Klasse [SQL
Anywhere .NET-API],161

R

Rails

ActiveRecord-Adapter installieren,717
Info,717
praktische Einführung,720

RAW-Dienste

erstellen,763
Info,760
kommentieren,766
löschen,766
Schnellstart,796
Schnellstart für Webclients,794
Schnellstart für Webserver,755

Read-Methode

SADataReader-Klasse [SQL Anywhere .NET-
API],246

READ_CLIENT_FILE-Funktion

ESQL-Client-API-Callback-Funktion,546
ODBC-Client-API, Callback-Funktion,375

ReceiveBufferSize-Eigenschaft

SATcpOptionsBuilder-Klasse [SQL
Anywhere .NET-API],320

RecordsAffected-Eigenschaft

SADataReader-Klasse [SQL Anywhere .NET-
API],248
SARowUpdatedEventArgs-Klasse [SQL
Anywhere .NET-API],310

Recordset ADO-Objekt

ADO-Programmierung,344

Recordset-Objekt

ADO,342

RecordSets

ADO-Programmierung,343

Reentrant-Code

Embedded SQL-Beispiel mit mehreren
Threads,498

Registrieren

DLLs für Deployment,1023
SQL Anywhere ASP.NET-Provider,100

SQL Anywhere ASP.NET-
Verbindungszeichenfolge,99

SQL Anywhere-.NET-Datenprovider,74

Registrierung

Deployment,1018
Deployment der Administrationstools unter
Windows,1001
ODBC,977
Wow6432Node,1001

Registrierungseinstellungen

SQL Anywhere-ODBC-Treiber,974

Registrierungsschlüssel

dialogfreie Installation,956

relayserver.jpr

Deployment von Administrationstools unter Linux/
Unix/Mac OS X,1008
Deployment von Administrationstools unter
Windows,998

REMOTEPWD

verwenden,427

Remove-Methode

SABulkCopyColumnMappingCollection-Klasse
[SQL Anywhere .NET-API],128
SACConnectionStringBuilderBase-Klasse [SQL
Anywhere .NET-API],211
SAParameterCollection-Klasse [SQL
Anywhere .NET-API],298

RemoveAt-Methode

SABulkCopyColumnMappingCollection-Klasse
[SQL Anywhere .NET-API],128
SAParameterCollection-Klasse [SQL
Anywhere .NET-API],298

removeShutdownHook

Java VM-Shutdown-Hooks,417

reportErrors

konfigurierbare Option,1011

ReservedWords-Feld

SAMetaDataCollectionNames-Klasse [SQL
Anywhere .NET-API],273

ResetCommandTimeout-Methode

SACommand-Klasse [SQL Anywhere .NET-
API],148

ResetDbType-Methode

SAParameter-Klasse [SQL Anywhere .NET-
API],281

Ressourcen-Manager

dreischichtige Datenverarbeitung,867
Info,865

- Ressourcen-Verteiler
 - dreischichtige Datenverarbeitung, 867
- Restrictions-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API], 273
- Results-Methode
 - JDBCExample, 443, 444
- RetryConnectionTimeout-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API], 206
- ROLLBACK TO SAVEPOINT-Anweisung
 - Cursor, 40
- ROLLBACK-Anweisung
 - Cursor, 40
- Rollback-Methode
 - SATransaction-Klasse [SQL Anywhere .NET-API], 322
- RollbackTrans ADO-Methode
 - ADO-Programmierung, 344
 - Daten aktualisieren, 344
- Root-Webdienste
 - im Web browsen, 790
- root-Webdienste
 - Info, 767
- RowsCopied-Eigenschaft
 - SARowsCopiedEventArgs-Klasse [SQL Anywhere .NET-API], 307
- RowUpdated-Ereignis
 - SADataAdapter-Klasse [SQL Anywhere .NET-API], 222
- RowUpdating-Ereignis
 - SADataAdapter-Klasse [SQL Anywhere .NET-API], 222
- RPC Out-Option
 - Verbindungsserver, 349, 351
- RPC-Option
 - Verbindungsserver, 349, 351
- Ruby
 - ActiveRecord-Adapter installieren, 717
 - Info, 717
 - nativen Ruby-Treiber installieren, 717
 - Ruby/DBI-Unterstützung installieren, 718
- Ruby on Rails
 - ActiveRecord-Adapter installieren, 717
 - Info, 717
 - praktische Einführung, 720
- Ruby-API
 - Info, 725
- sqlany_affected_rows-Funktion, 726
- sqlany_bind_param-Funktion, 726
- sqlany_clear_error-Funktion, 727
- sqlany_client_version-Funktion, 728
- sqlany_commit-Funktion, 728
- sqlany_connect-Funktion, 728
- sqlany_describe_bind_param-Funktion, 729
- sqlany_disconnect-Funktion, 730
- sqlany_error-Funktion, 730
- sqlany_execute-Funktion, 731
- sqlany_execute_direct-Funktion, 731
- sqlany_execute_immediate-Funktion, 732
- sqlany_fetch_absolute-Funktion, 733
- sqlany_fetch_next-Funktion, 733
- sqlany_fini-Funktion, 734
- sqlany_free_connection-Funktion, 735
- sqlany_free_stmt-Funktion, 735
- sqlany_get_bind_param_info-Funktion, 736
- sqlany_get_column-Funktion, 736
- sqlany_get_column_info-Funktion, 737
- sqlany_get_next_result-Funktion, 738
- sqlany_init-Funktion, 739
- sqlany_new_connection-Funktion, 739
- sqlany_num_cols-Funktion, 740
- sqlany_num_params-Funktion, 741
- sqlany_num_rows-Funktion, 741
- sqlany_prepare-Funktion, 742
- sqlany_rollback-Funktion, 743
- sqlany_sqlstate-Funktion, 743
- Ruby-APIs
 - Einführung in die Programmierung, 717
- Ruby-DBI
 - dbd-sqlanywhere installieren, 718
 - Info, 721
 - Verbindungsbeispiele, 722
- RubyGems
 - installieren, 717
- Rückgabecodes
 - Info, 945
 - ODBC, 397
- Rückgabetypen
 - externe Funktionen, 618
- Rückgabewerte
 - Webclients, 814
- Rückrufe
 - DB_CALLBACK_FINISH, 547
 - DB_CALLBACK_MESSAGE, 548
 - DB_CALLBACK_WAIT, 548

S

- sa_config.csh
 - Deployment, 950
- sa_config.sh
 - Deployment, 950
- sa_external_library_unload-Systemprozedur verwenden, 618
- SA_GET_MESSAGE_CALLBACK_PARM
 - SQLSetConnectAttr, 375
- SA_REGISTER_MESSAGE_CALLBACK
 - SQLSetConnectAttr, 375
- SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK
 - SQLSetConnectAttr, 375
- sa_set_http_header-Systemprozedur
 - Beispiel, 771
- SA_SQL_ATTR_TXN_ISOLATION
 - SQLSetConnectAttr, 375
- SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
 - Isolationsstufe, 388
- SA_SQL_TXN_SNAPSHOT
 - Isolationsstufe, 388
- SA_SQL_TXN_STATEMENT_SNAPSHOT
 - Isolationsstufe, 388
- SA_TRANSACTION_SNAPSHOT
 - Transaktion, Isolationsstufe, 434
- SA_TRANSACTION_STATEMENT_READONLY_SNAPSHOT
 - Transaktion, Isolationsstufe, 434
- SABulkCopy-Klasse [SQL Anywhere .NET-API]
 - BatchSize-Eigenschaft, 112
 - Beschreibung, 105
 - BulkCopyTimeout-Eigenschaft, 112
 - Close-Methode, 109
 - ColumnMappings-Eigenschaft, 113
 - DestinationTableName-Eigenschaft, 113
 - Dispose-Methode, 109
 - NotifyAfter-Eigenschaft, 114
 - SABulkCopy-Konstruktor, 106
 - SARowsCopied-Ereignis, 114
 - WriteToServer-Methode, 109
- SABulkCopy-Konstruktor
 - SABulkCopy-Klasse [SQL Anywhere .NET-API], 106
- SABulkCopyColumnMapping-Klasse [SQL Anywhere .NET-API]
 - Beschreibung, 115
 - DestinationColumn-Eigenschaft, 119
 - DestinationOrdinal-Eigenschaft, 119
 - SABulkCopyColumnMapping-Konstruktor, 116
 - SourceColumn-Eigenschaft, 120
 - SourceOrdinal-Eigenschaft, 120
- SABulkCopyColumnMapping-Konstruktor
 - SABulkCopyColumnMapping-Klasse [SQL Anywhere .NET-API], 116
- SABulkCopyColumnMappingCollection-Klasse [SQL Anywhere .NET-API]
 - Add-Methode, 123
 - Beschreibung, 121
 - Contains-Methode, 127
 - CopyTo-Methode, 127
 - IndexOf-Methode, 128
 - Remove-Methode, 128
 - RemoveAt-Methode, 128
 - this-Eigenschaft, 129
- SABulkCopyOptions-Enumeration [SQL Anywhere .NET-API]
 - Beschreibung, 327
- sacapi.h-Headerdatei
 - SQL Anywhere-C-API-Referenz, 566
- sacapidll.h
 - Schnittstellenbibliothek, 566
- sacapidll.h-Headerdatei
 - SQL Anywhere-C-API-Referenz, 566
- SACCommand
 - Primärschlüsselwerte abrufen, 52
- SACCommand, Klasse
 - in einem Visual Studio-Projekt verwenden, 82
- SACCommand-Klasse
 - Aktualisieren von Daten, 50
 - Beispiel, 86
 - Daten abrufen, 48
 - Einfügen von Daten, 50
 - Info, 47
 - Löschen von Daten, 50
 - vorbereitete Anweisungen verwenden, 4
- SACCommand-Klasse [SQL Anywhere .NET-API]
 - BeginExecuteNonQuery-Methode, 134
 - BeginExecuteReader-Methode, 135
 - Beschreibung, 129
 - Cancel-Methode, 139
 - CommandText-Eigenschaft, 148

- CommandTimeout-Eigenschaft, 149
- CommandType-Eigenschaft, 149
- Connection-Eigenschaft, 150
- CreateParameter-Methode, 140
- DesignTimeVisible-Eigenschaft, 150
- EndExecuteNonQuery-Methode, 140
- EndExecuteReader-Methode, 142
- ExecuteNonQuery-Methode, 144
- ExecuteReader-Methode, 145
- ExecuteScalar-Methode, 147
- Parameters-Eigenschaft, 150
- Prepare-Methode, 147
- ResetCommandTimeout-Methode, 148
- SACommand-Konstruktor, 132
- Transaction-Eigenschaft, 151
- UpdatedRowSource-Eigenschaft, 151
- SACommand-Konstruktor
 - SACommand-Klasse [SQL Anywhere .NET-API], 132
- SACommandBuilder-Klasse [SQL Anywhere .NET-API]
 - Beschreibung, 152
 - DataAdapter-Eigenschaft, 162
 - DeriveParameters-Methode, 155
 - GetDeleteCommand-Methode, 156
 - GetInsertCommand-Methode, 158
 - GetUpdateCommand-Methode, 160
 - QuoteIdentifier-Methode, 161
 - SACommandBuilder-Konstruktor, 155
 - UnquoteIdentifier-Methode, 162
- SACommandBuilder-Konstruktor
 - SACommandBuilder-Klasse [SQL Anywhere .NET-API], 155
- SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API]
 - All-Eigenschaft, 166
 - Beschreibung, 163
 - ConnectionString-Eigenschaft, 167
 - GetUseLongNameAsKeyword-Methode, 165
 - SACommLinksOptionsBuilder-Konstruktor, 164
 - SetUseLongNameAsKeyword-Methode, 166
 - SharedMemory-Eigenschaft, 167
 - TcpOptionsBuilder-Eigenschaft, 167
 - TcpOptionsString-Eigenschaft, 168
 - ToString-Methode, 166
- SACommLinksOptionsBuilder-Konstruktor
 - SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API], 164
- SAConnection, Funktion
 - in einem Visual Studio-Projekt verwenden, 86
- SAConnection, Klasse
 - in einem Visual Studio-Projekt verwenden, 81
- SAConnection-Klasse
 - Beispiel, 86
 - mit einer Datenbank verbinden, 45
- SAConnection-Klasse [SQL Anywhere .NET-API]
 - BeginTransaction-Methode, 171
 - Beschreibung, 168
 - ChangeDatabase-Methode, 174
 - ChangePassword-Methode, 174
 - ClearAllPools-Methode, 175
 - ClearPool-Methode, 175
 - Close-Methode, 175
 - ConnectionString-Eigenschaft, 185
 - ConnectionTimeout-Eigenschaft, 186
 - CreateCommand-Methode, 176
 - Database-Eigenschaft, 187
 - DataSource-Eigenschaft, 187
 - EnlistDistributedTransaction-Methode, 176
 - EnlistTransaction-Methode, 177
 - GetSchema-Methode, 177
 - InfoMessage-Ereignis, 189
 - InitString-Eigenschaft, 187
 - Open-Methode, 185
 - SAConnection-Konstruktor, 170
 - ServerVersion-Eigenschaft, 188
 - State-Eigenschaft, 188
 - StateChange-Ereignis, 189
- SAConnection-Konstruktor
 - SAConnection-Klasse [SQL Anywhere .NET-API], 170
- SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API]
 - AppInfo-Eigenschaft, 196
 - AutoStart-Eigenschaft, 196
 - AutoStop-Eigenschaft, 196
 - Beschreibung, 189
 - Charset-Eigenschaft, 196
 - CommBufferSize-Eigenschaft, 196
 - CommLinks-Eigenschaft, 197
 - Compress-Eigenschaft, 197
 - CompressionThreshold-Eigenschaft, 197
 - ConnectionLifetime-Eigenschaft, 197
 - ConnectionName-Eigenschaft, 198
 - ConnectionPool-Eigenschaft, 198
 - ConnectionReset-Eigenschaft, 198

ConnectionTimeout-Eigenschaft,198
 DatabaseFile-Eigenschaft,199
 DatabaseKey-Eigenschaft,199
 DatabaseName-Eigenschaft,199
 DatabaseSwitches-Eigenschaft,199
 DataSourceName-Eigenschaft,200
 DisableMultiRowFetch-Eigenschaft,200
 Elevate-Eigenschaft,200
 EncryptedPassword-Eigenschaft,200
 Encryption-Eigenschaft,201
 Enlist-Eigenschaft,201
 FileDataSourceName-Eigenschaft,201
 ForceStart-Eigenschaft,201
 Host-Eigenschaft,201
 IdleTimeout-Eigenschaft,202
 InitString-Eigenschaft,202
 Integrated-Eigenschaft,202
 Kerberos-Eigenschaft,202
 Language-Eigenschaft,203
 LazyClose-Eigenschaft,203
 LivenessTimeout-Eigenschaft,203
 LogFile-Eigenschaft,203
 MaxPoolSize-Eigenschaft,204
 MinPoolSize-Eigenschaft,204
 NewPassword-Eigenschaft,204
 NodeType-Eigenschaft,204
 Password-Eigenschaft,204
 PersistSecurityInfo-Eigenschaft,205
 Pooling-Eigenschaft,205
 PrefetchBuffer-Eigenschaft,205
 PrefetchRows-Eigenschaft,205
 RetryConnectionTimeout-Eigenschaft,206
 SAConnectionStringBuilder-Konstruktor,194
 ServerName-Eigenschaft,206
 StartLine-Eigenschaft,206
 Unconditional-Eigenschaft,206
 UserID-Eigenschaft,207
 SAConnectionStringBuilder-Konstruktor
 SAConnectionStringBuilder-Klasse [SQL
 Anywhere .NET-API],194
 SAConnectionStringBuilderBase-Klasse [SQL
 Anywhere .NET-API]
 Beschreibung,207
 ContainsKey-Methode,209
 GetKeyword-Methode,210
 GetUseLongNameAsKeyword-Methode,210
 Keys-Eigenschaft,212
 Remove-Methode,211
 SetUseLongNameAsKeyword-Methode,211
 ShouldSerialize-Methode,212
 this-Eigenschaft,213
 TryGetValue-Methode,212
 sacshelp16.chm
 Deployment von ODBC-Clients,971
 SAdaptAdapter
 Primärschlüsselwerte abrufen,60
 SAdaptAdapter-Klasse
 Aktualisieren von Daten,53
 Daten abrufen,55,56
 Einfügen von Daten,53
 Info,47
 Löschen von Daten,53
 verwenden,48
 SAdaptAdapter-Klasse [SQL Anywhere .NET-API]
 Beschreibung,213
 DeleteCommand-Eigenschaft,219
 GetFillParameters-Methode,218
 InsertCommand-Eigenschaft,219
 RowUpdated-Ereignis,222
 RowUpdating-Ereignis,222
 SAdaptAdapter-Konstruktor,216
 SelectCommand-Eigenschaft,220
 TableMappings-Eigenschaft,220
 UpdateBatchSize-Eigenschaft,220
 UpdateCommand-Eigenschaft,221
 SAdaptAdapter-Konstruktor
 SAdaptAdapter-Klasse [SQL Anywhere .NET-
 API],216
 SAdaptReader, Klasse
 in einem Visual Studio-Projekt verwenden,82
 SAdaptReader-Klasse
 Beispiel,87
 SAdaptReader-Klasse [SQL Anywhere .NET-API]
 Beschreibung,223
 Close-Methode,227
 Depth-Eigenschaft,247
 FieldCount-Eigenschaft,247
 GetBoolean-Methode,227
 GetByte-Methode,228
 GetBytes-Methode,228
 GetChar-Methode,229
 GetChars-Methode,230
 GetData-Methode,231
 GetDataTypeName-Methode,232
 GetDateTime-Methode,232
 GetDateTimeOffset-Methode,233

- GetDecimal-Methode,233
- GetDouble-Methode,234
- GetEnumerator-Methode,234
- GetFieldType-Methode,235
- GetFloat-Methode,235
- GetGuid-Methode,236
- GetInt16-Methode,236
- GetInt32-Methode,237
- GetInt64-Methode,237
- GetName-Methode,238
- GetOrdinal-Methode,238
- GetSchemaTable-Methode,239
- GetString-Methode,241
- GetTimeSpan-Methode,241
- GetUInt16-Methode,242
- GetUInt32-Methode,243
- GetUInt64-Methode,243
- GetValue-Methode,243
- GetValues-Methode,245
- HasRows-Eigenschaft,248
- IsClosed-Eigenschaft,248
- IsDBNull-Methode,245
- myDispose-Methode,246
- NextResult-Methode,246
- Read-Methode,246
- RecordsAffected-Eigenschaft,248
- this-Eigenschaft,249
- SADataSourceEnumerator-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,250
 - GetDataSources-Methode,251
 - Instance-Eigenschaft,251
- SADbType-Eigenschaft
 - SAPParameter-Klasse [SQL Anywhere .NET-API],284
- SADbType-Enumeration [SQL Anywhere .NET-API]
 - Beschreibung,328
- SADefault-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,251
 - Value-Feld,252
- SAError-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,252
 - Message-Eigenschaft,253
 - NativeError-Eigenschaft,253
 - Source-Eigenschaft,254
 - SqlState-Eigenschaft,254
 - ToString-Methode,253
- SAErrorCollection-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,254
 - CopyTo-Methode,255
 - Count-Eigenschaft,256
 - GetEnumerator-Methode,255
 - this-Eigenschaft,256
- SAException, Klasse
 - in einem Visual Studio-Projekt verwenden,83
- SAException-Klasse
 - Beispiel,87
- SAException-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,256
 - Errors-Eigenschaft,258
 - GetObjectData-Methode,258
 - Message-Eigenschaft,259
 - NativeError-Eigenschaft,259
 - Source-Eigenschaft,259
- SAFactory-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,260
 - CanCreateDataSourceEnumerator-Eigenschaft,264
 - CreateCommand-Methode,261
 - CreateCommandBuilder-Methode,262
 - CreateConnection-Methode,262
 - CreateConnectionStringBuilder-Methode,262
 - CreateDataAdapter-Methode,263
 - CreateDataSourceEnumerator-Methode,263
 - CreateParameter-Methode,263
 - CreatePermission-Methode,264
 - Instance-Feld,265
- SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,265
 - Errors-Eigenschaft,266
 - Message-Eigenschaft,267
 - MessageType-Eigenschaft,267
 - NativeError-Eigenschaft,267
 - Source-Eigenschaft,267
 - ToString-Methode,266
- SAInfoMessageEventHandler-Delegat [SQL Anywhere .NET-API]
 - Beschreibung,325
- saip16.jar
 - Deployment unter Windows,996
- SAIsolationLevel-Eigenschaft
 - SATransaction-Klasse [SQL Anywhere .NET-API],325

SAIsolationLevel-Enumeration [SQL Anywhere .NET-API]
 Beschreibung,333
 sajdbc4.jar
 Deployment von Datenbankservern,1016
 Deployment von JDBC-Clients,982
 SQL Anywhere JDBC-Treiber laden,430
 sajvm.jar
 Deployment von Datenbankservern,1016
 salib.jar
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 salocation-Option
 SetupVSPackage,960
 SAMessageType-Enumeration [SQL Anywhere .NET-API]
 Beschreibung,336
 SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API]
 Beschreibung,268
 Columns-Feld,269
 DataSourceInformation-Feld,270
 DataTypes-Feld,270
 ForeignKeys-Feld,270
 IndexColumns-Feld,271
 Indexes-Feld,271
 MetaDataCollections-Feld,272
 ProcedureParameters-Feld,272
 Procedures-Feld,273
 ReservedWords-Feld,273
 Restrictions-Feld,273
 Tables-Feld,274
 UserDefinedTypes-Feld,274
 Users-Feld,275
 ViewColumns-Feld,275
 Views-Feld,275
 SAOLEDB
 OLE DB-Provider,337
 SAPParameter-Klasse [SQL Anywhere .NET-API]
 Beschreibung,276
 DbType-Eigenschaft,282
 Direction-Eigenschaft,282
 IsNullable-Eigenschaft,283
 Offset-Eigenschaft,283
 ParameterName-Eigenschaft,283
 Precision-Eigenschaft,284
 ResetDbType-Methode,281
 SADbType-Eigenschaft,284
 SAPParameter-Konstruktor,277
 Scale-Eigenschaft,284
 Size-Eigenschaft,285
 SourceColumn-Eigenschaft,285
 SourceColumnNullMapping-Eigenschaft,286
 SourceVersion-Eigenschaft,286
 ToString-Methode,281
 Value-Eigenschaft,286
 SAPParameter-Konstruktor
 SAPParameter-Klasse [SQL Anywhere .NET-API],277
 SAPParameterCollection-Klasse [SQL Anywhere .NET-API]
 Add-Methode,289
 AddRange-Methode,293
 AddWithValue-Methode,294
 Beschreibung,287
 Clear-Methode,294
 Contains-Methode,294
 CopyTo-Methode,296
 Count-Eigenschaft,299
 GetEnumerator-Methode,296
 IndexOf-Methode,297
 Insert-Methode,298
 IsFixedSize-Eigenschaft,300
 IsReadOnly-Eigenschaft,300
 IsSynchronized-Eigenschaft,300
 Remove-Methode,298
 RemoveAt-Methode,298
 SyncRoot-Eigenschaft,301
 this-Eigenschaft,301
 SAPPermission-Klasse [SQL Anywhere .NET-API]
 Beschreibung,303
 SAPPermission-Konstruktor,304
 SAPPermission-Konstruktor
 SAPPermission-Klasse [SQL Anywhere .NET-API],304
 SAPPermissionAttribute-Klasse [SQL Anywhere .NET-API]
 Beschreibung,304
 CreatePermission-Methode,306
 SAPPermissionAttribute-Konstruktor,305
 SAPPermissionAttribute-Konstruktor
 SAPPermissionAttribute-Klasse [SQL Anywhere .NET-API],305
 sapplugin.jar
 Deployment unter Mac OS X,1005

- Deployment unter Unix,1005
- Deployment unter Windows,996
- SARowsCopied-Ereignis
 - SABulkCopy-Klasse [SQL Anywhere .NET-API],114
- SARowsCopiedEventArgs-Klasse [SQL Anywhere .NET-API]
 - Abort-Eigenschaft,307
 - Beschreibung,306
 - RowsCopied-Eigenschaft,307
 - SARowsCopiedEventArgs-Konstruktor,307
- SARowsCopiedEventArgs-Konstruktor
 - SARowsCopiedEventArgs-Klasse [SQL Anywhere .NET-API],307
- SARowsCopiedEventHandler-Delegat [SQL Anywhere .NET-API]
 - Beschreibung,327
- SARowUpdatedEventArgs-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,308
 - Command-Eigenschaft,309
 - RecordsAffected-Eigenschaft,310
 - SARowUpdatedEventArgs-Konstruktor,309
- SARowUpdatedEventArgs-Konstruktor
 - SARowUpdatedEventArgs-Klasse [SQL Anywhere .NET-API],309
- SARowUpdatedEventHandler-Delegat [SQL Anywhere .NET-API]
 - Beschreibung,326
- SARowUpdatingEventArgs-Klasse [SQL Anywhere .NET-API]
 - Beschreibung,310
 - Command-Eigenschaft,312
 - SARowUpdatingEventArgs-Konstruktor,311
- SARowUpdatingEventArgs-Konstruktor
 - SARowUpdatingEventArgs-Klasse [SQL Anywhere .NET-API],311
- SARowUpdatingEventHandler-Delegat [SQL Anywhere .NET-API]
 - Beschreibung,326
- SAServerSideConnection-Klasse [SQL Anywhere-.NET-API]
 - Beschreibung,312
 - Connection-Eigenschaft,312
- sasql_affected_rows-Funktion (PHP)
 - Syntax,685
- sasql_close-Funktion (PHP)
 - Syntax,686
- sasql_commit-Funktion (PHP)
 - Syntax,686
- sasql_connect-Funktion (PHP)
 - Syntax,687
- sasql_data_seek-Funktion (PHP)
 - Syntax,687
- sasql_disconnect-Funktion (PHP)
 - Syntax,688
- sasql_error-Funktion (PHP)
 - Syntax,688
- sasql_errorcode-Funktion (PHP)
 - Syntax,689
- sasql_escape_string-Funktion (PHP)
 - Syntax,690
- sasql_fetch_array-Funktion (PHP)
 - Syntax,690
- sasql_fetch_assoc-Funktion (PHP)
 - Syntax,691
- sasql_fetch_field-Funktion (PHP)
 - Syntax,691
- sasql_fetch_object-Funktion (PHP)
 - Syntax,692
- sasql_fetch_row-Funktion (PHP)
 - Syntax,693
- sasql_field_count-Funktion (PHP)
 - Syntax,693
- sasql_field_seek-Funktion (PHP)
 - Syntax,694
- sasql_free_result-Funktion (PHP)
 - Syntax,694
- sasql_get_client_info-Funktion (PHP)
 - Syntax,694
- sasql_insert_id-Funktion (PHP)
 - Syntax,695
- sasql_message-Funktion (PHP)
 - Syntax,695
- sasql_multi_query-Funktion (PHP)
 - Syntax,696
- sasql_next_result-Funktion (PHP)
 - Syntax,696
- sasql_num_fields-Funktion (PHP)
 - Syntax,697
- sasql_num_rows-Funktion (PHP)
 - Syntax,697
- sasql_pconnect-Funktion (PHP)
 - Syntax,698
- sasql_prepare-Funktion (PHP)
 - Syntax,698

<code>sasql_query</code> -Funktion (PHP)	<code>sasql_stmt_result_metadata</code> -Funktion (PHP)
Syntax,699	Syntax,712
<code>sasql_real_escape_string</code> -Funktion (PHP)	<code>sasql_stmt_send_long_data</code> -Funktion (PHP)
Syntax,700	Syntax,712
<code>sasql_real_query</code> -Funktion (PHP)	<code>sasql_stmt_store_result</code> -Funktion (PHP)
Syntax,700	Syntax,713
<code>sasql_result_all</code> -Funktion (PHP)	<code>sasql_store_result</code> -Funktion (PHP)
Syntax,701	Syntax,713
<code>sasql_rollback</code> -Funktion (PHP)	<code>sasql_use_result</code> -Funktion (PHP)
Syntax,702	Syntax,714
<code>sasql_set_option</code> -Funktion (PHP)	SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API]
Syntax,702	Beschreibung,313
<code>sasql_sqlstate</code> -Funktion (PHP)	Broadcast-Eigenschaft,318
Syntax,714	BroadcastListener-Eigenschaft,318
<code>sasql_stmt_affected_rows</code> -Funktion (PHP)	ClientPort-Eigenschaft,318
Syntax,703	DoBroadcast-Eigenschaft,318
<code>sasql_stmt_bind_param</code> -Funktion (PHP)	Host-Eigenschaft,319
Syntax,704	IPV6-Eigenschaft,319
<code>sasql_stmt_bind_param_ex</code> -Funktion (PHP)	LDAP-Eigenschaft,319
Syntax,704	LocalOnly-Eigenschaft,319
<code>sasql_stmt_bind_result</code> -Funktion (PHP)	MyIP-Eigenschaft,319
Syntax,705	ReceiveBufferSize-Eigenschaft,320
<code>sasql_stmt_close</code> -Funktion (PHP)	SATcpOptionsBuilder-Konstruktor,316
Syntax,706	SendBufferSize-Eigenschaft,320
<code>sasql_stmt_data_seek</code> -Funktion (PHP)	ServerPort-Eigenschaft,320
Syntax,706	TDS-Eigenschaft,320
<code>sasql_stmt_errno</code> -Funktion (PHP)	Timeout-Eigenschaft,321
Syntax,706	ToString-Methode,317
<code>sasql_stmt_error</code> -Funktion (PHP)	VerifyServerName-Eigenschaft,321
Syntax,707	SATcpOptionsBuilder-Konstruktor
<code>sasql_stmt_execute</code> -Funktion (PHP)	SATcpOptionsBuilder-Klasse [SQL
Syntax,708	Anywhere .NET-API],316
<code>sasql_stmt_fetch</code> -Funktion (PHP)	SAToMSSql10.xml
Syntax,708	Deployment von .NET-Clients,960
<code>sasql_stmt_field_count</code> -Funktion (PHP)	SATransaction-Klasse
Syntax,709	verwenden,64
<code>sasql_stmt_free_result</code> -Funktion (PHP)	SATransaction-Klasse [SQL Anywhere .NET-API]
Syntax,709	Beschreibung,321
<code>sasql_stmt_insert_id</code> -Funktion (PHP)	Commit-Methode,322
Syntax,709	Connection-Eigenschaft,324
<code>sasql_stmt_next_result</code> -Funktion (PHP)	IsolationLevel-Eigenschaft,324
Syntax,710	Rollback-Methode,322
<code>sasql_stmt_num_rows</code> -Funktion (PHP)	SAIsolationLevel-Eigenschaft,325
Syntax,710	Save-Methode,323
<code>sasql_stmt_param_count</code> -Funktion (PHP)	Save-Methode
Syntax,711	SATransaction-Klasse [SQL Anywhere .NET-API],323
<code>sasql_stmt_reset</code> -Funktion (PHP)	
Syntax,711	

- Savepoints
 - Cursor,40
- sbgse2.dll
 - FIPS-zertifizierte RSA-Verschlüsselung,1026
- Scale-Eigenschaft
 - SAPParameter-Klasse [SQL Anywhere .NET-API],284
- SCEditor1600.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- Schema
 - Mitgliedschaftsprovider,102
 - Profilprovider,103
 - Provider für Webseiten-Personalisierung,104
 - Rollenprovider,102
 - SQL Anywhere ASP.NET-Provider hinzufügen,98
 - Systemüberwachungsprovider,104
- Schnelllader
 - OEM-konfigurierbare Option,1011
- Schnellstart
 - SQL Anywhere-Webserver,755
 - SQL Anywhere-Webclient,794
 - Zugriff auf SQL Anywhere-Webserver,796
- Schnittstellen
 - SQL Anywhere Embedded SQL,467
- Schnittstellenbibliotheken
 - DBLIB,467
 - dynamisch laden,476
- Schreibgeschützte Cursor
 - Info,15
- Schreibschutz
 - Deployment von Datenbanken,1022
- scjview
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
- scjview.exe
 - Deployment unter Windows,996
- scjview.ini
 - Deployment von Administrationstools,992
- SCROLL-Cursor
 - Embedded SQL,34
 - wertsensitiv,26
- Scrollfähige Cursor
 - Info,12
 - JDBC-Unterstützung,420
- Selbstregistrierende DLLs
 - Deployment von SQL Anywhere,1023
- SELECT-Anweisung
 - dynamische SELECT-Anweisungen verwenden,503
 - einzelne Zeilen,515
- SelectCommand-Eigenschaft
 - SADDataAdapter-Klasse [SQL Anywhere .NET-API],220
- SendBufferSize-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],320
- Sensitive Cursor
 - Beispiele für Aktualisieren,20
 - Beispiele für Löschen,18
 - Cursor-Eigenschaften,15
 - Einführung,18
 - Embedded SQL,34
 - Info,23
- Sensitivität
 - Beispiele für Aktualisieren,20
 - Beispiele für Löschen,18
 - Cursor,18
 - Isolationsstufen,32
 - SQL Anywhere-Cursor,16
- Serialisierung
 - Objekte in Tabellen,412
- Server
 - über ESQL finden,555
 - Web,755
- Server-Explorer
 - Visual Studio,88
- Serveradressen
 - Embedded SQL-Funktion,541
- ServerName-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],206
- ServerPort-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],320
- Serverseitiges Autocommit
 - Info,39
- ServerVersion-Eigenschaft
 - SACConnection-Klasse [SQL Anywhere .NET-API],188
- SessionCreateTime-Eigenschaft
 - Info,783
- SessionID-Eigenschaft
 - Beispiel,780
- SessionLastTime-Eigenschaft

- Info,783
- set_cancel
 - Info,614
- set_value-Funktion
 - Info,614
 - verwenden,617
- setAutocommit-Methode
 - Info,434
- setTransactionIsolation-Methode
 - JDBC,434
- SetupVSPackage
 - .NET-Datenprovider deinstallieren ,963
 - Deployment von .NET-Clients,960
- SetUseLongNameAsKeyword-Methode
 - SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],166
 - SAConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],211
- Shared Objects
 - aus gespeicherten Prozeduren aufrufen,601
 - externe Prozeduraufrufe,603
- SharedMemory-Eigenschaft
 - SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],167
- ShouldSerialize-Methode
 - SAConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],212
- showMultipleResultSets
 - konfigurierbare Option,1011
- showPerformanceDataUI
 - konfigurierbare Option,1011
- showResultsForAllStatements
 - konfigurierbare Option,1011
- Sicherheit
 - Interactive SQL,1011
 - Java in der Datenbank,417
 - OData Server,456
- Sicherheitskontext
 - Verbindungsserver,349,351
- Sicherheitsmanager
 - Info,417
- Sicherungen
 - DBTools-Beispiel,876
 - Embedded SQL-Funktionen,531
- Sichtbare Änderungen
 - Cursor,17
- Signaturen
 - Java-Methoden,637
- SimpleCE
 - .NET-Datenprovider, Beispielpjekt,43
- SimpleViewer
 - .NET-Datenprovider-Beispielpjekt,43
 - .NET-Projekt,87
- SimpleWin32
 - .NET-Datenprovider, Beispielpjekt,43
- SimpleXML
 - .NET-Datenprovider, Beispielpjekt,43
- Sitzungen
 - administrieren,785
 - erkennen,783
 - erstellen,780
 - Fehler,786
 - Info,779
 - löschen,784
- Size-Eigenschaft
 - SAPparameter-Klasse [SQL Anywhere .NET-API],285
- SMALLDATETIME-Datentyp
 - Open Client-Konvertierung,750
- SMALLMONEY-Datentyp
 - Open Client-Konvertierung,750
- Snapshot-Isolation
 - SQL Anywhere .NET-Datenprovider,65
 - verlorene Aktualisierungen,30
- SOAP
 - Variablen in einem Rahmen übergeben,813
- SOAP-Anforderungen
 - Strukturen,830
- SOAP-Dienste
 - .NET, praktische Einführung,848
 - Datentypen,817
 - erstellen,763
 - Fehler,832
 - Info,760
 - JAX-WS, praktische Einführung,856
 - kommentieren,766
 - löschen,766
 - SQL Anywhere-Webclient, praktische Einführung,839
- SOAP-Header
 - verwalten,805
- SOAP-Namespace
 - Verwaltung,809
- SOAP-Systemprozeduren
 - alphabetische Liste,788
- SOAPHEADER-Klausel

- verwalten,805
- Software
 - Rückgabecode,945
- Source-Eigenschaft
 - SAError-Klasse [SQL Anywhere .NET-API],254
 - SAException-Klasse [SQL Anywhere .NET-API],259
 - SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API],267
- SourceColumn-Eigenschaft
 - SABulkCopyColumnMapping-Klasse [SQL Anywhere .NET-API],120
 - SAParameter-Klasse [SQL Anywhere .NET-API],285
- SourceColumnNullMapping-Eigenschaft
 - SAParameter-Klasse [SQL Anywhere .NET-API],286
- SourceOrdinal-Eigenschaft
 - SABulkCopyColumnMapping-Klasse [SQL Anywhere .NET-API],120
- SourceVersion-Eigenschaft
 - SAParameter-Klasse [SQL Anywhere .NET-API],286
- sp_tsql_environment-Systemprozedur
 - Optionen für jConnect einstellen,427
- Sprachen
 - Dateinamen,950
- SQL
 - ADO-Anwendungen,1
 - Anwendungen,1
 - Embedded SQL-Anwendungen,1
 - JDBC-Anwendungen,1
 - ODBC-Anwendungen,1
 - Open Client-Anwendungen,1
- SQL Anywhere
 - C-API,565
 - Einführung,565
- SQL Anywhere .NET-API
 - SABulkCopy-Klasse,105
 - SABulkCopyColumnMapping-Klasse,115
 - SABulkCopyColumnMappingCollection-Klasse,121
 - SABulkCopyOptions-Enumeration,327
 - SACommand-Klasse,129
 - SACommandBuilder-Klasse,152
 - SACommLinksOptionsBuilder-Klasse,163
 - SAConnection-Klasse,168
 - SAConnectionStringBuilder-Klasse,189
 - SAConnectionStringBuilderBase-Klasse,207
 - SADDataAdapter-Klasse,213
 - SADDataReader-Klasse,223
 - SADDataSourceEnumerator-Klasse,250
 - SADbType-Enumeration,328
 - SADefault-Klasse,251
 - SAError-Klasse,252
 - SAErrorCollection-Klasse,254
 - SAException-Klasse,256
 - SAFactory-Klasse,260
 - SAInfoMessageEventArgs-Klasse,265
 - SAInfoMessageEventHandler-Delegat,325
 - SAIsolationLevel-Enumeration,333
 - SAMessageType-Enumeration,336
 - SAMetaDataCollectionNames-Klasse,268
 - SAParameter-Klasse,276
 - SAParameterCollection-Klasse,287
 - SAPermission-Klasse,303
 - SAPermissionAttribute-Klasse,304
 - SARowsCopiedEventArgs-Klasse,306
 - SARowsCopiedEventHandler-Delegat,327
 - SARowUpdatedEventArgs-Klasse,308
 - SARowUpdatedEventHandler-Delegat,326
 - SARowUpdatingEventArgs-Klasse,310
 - SARowUpdatingEventHandler-Delegat,326
 - SATcpOptionsBuilder-Klasse,313
 - SATransaction-Klasse,321
- SQL Anywhere .NET-Datenprovider
 - Beispiele,79
 - Info,41
- SQL Anywhere 16 Demo.dsn
 - Windows Mobile-ODBC,362
- SQL Anywhere ASP.NET-Datenprovider
 - Info,97
- SQL Anywhere C-API
 - Schnittstellenbibliothek,566
- SQL Anywhere JDBC-Treiber
 - Deployment von JDBC-Clients,982
- SQL Anywhere ODBC-Treiber
 - Deployment,970
- SQL Anywhere Perl DBD::SQLAnywhere DBI-Modul
 - Info,651
- SQL Anywhere Ruby-API
 - Funktionen,725
- SQL Anywhere, ODBC-Treiber
 - unter Windows linken,361
- SQL Anywhere-.NET-API

- Info,41
- SAServerSideConnection-Klasse,312
- SQL Anywhere-.NET-API-Referenz
- Info,105
- SQL Anywhere-C-API
 - a_sqlany_bind_param-Struktur,593
 - a_sqlany_bind_param_info-Struktur,594
 - a_sqlany_column_info-Struktur,594
 - a_sqlany_data_direction-Enumeration,590
 - a_sqlany_data_info-Struktur,595
 - a_sqlany_data_type-Enumeration,591
 - a_sqlany_data_value-Struktur,596
 - a_sqlany_native_type-Enumeration,591
 - C-API,565
 - externe Umgebung,625
 - sqlany_affected_rows-Methode,566
 - sqlany_bind_param-Methode,567
 - sqlany_cancel-Methode,567
 - sqlany_clear_error-Methode,568
 - sqlany_client_version-Methode,568
 - sqlany_client_version_ex-Methode,568
 - sqlany_commit-Methode,569
 - sqlany_connect-Methode,569
 - sqlany_describe_bind_param-Methode,570
 - sqlany_disconnect-Methode,571
 - sqlany_error-Methode,572
 - sqlany_execute-Methode,572
 - sqlany_execute_direct-Methode,573
 - sqlany_execute_immediate-Methode,574
 - sqlany_fetch_absolute-Methode,575
 - sqlany_fetch_next-Methode,575
 - sqlany_finalize_interface-Methode,576
 - sqlany_fini-Methode,576
 - sqlany_fini_ex-Methode,576
 - sqlany_free_connection-Methode,577
 - sqlany_free_stmt-Methode,577
 - sqlany_get_bind_param_info-Methode,577
 - sqlany_get_column-Methode,578
 - sqlany_get_column_info-Methode,579
 - sqlany_get_data-Methode,580
 - sqlany_get_data_info-Methode,580
 - sqlany_get_next_result-Methode,581
 - sqlany_init-Methode,582
 - sqlany_init_ex-Methode,583
 - sqlany_initialize_interface-Methode,583
 - sqlany_make_connection-Methode,584
 - sqlany_make_connection_ex-Methode,584
 - sqlany_new_connection-Methode,585
 - sqlany_new_connection_ex-Methode,585
 - sqlany_num_cols-Methode,586
 - sqlany_num_params-Methode,586
 - sqlany_num_rows-Methode,587
 - sqlany_prepare-Methode,587
 - sqlany_reset-Methode,588
 - sqlany_rollback-Methode,588
 - sqlany_send_param_data-Methode,589
 - sqlany_sqlstate-Methode,590
 - SQLAnywhereInterface-Struktur,597
- SQL Anywhere-C-API-Referenz
 - sacapi.h-Headerdatei,566
 - sacapidll.h-Headerdatei,566
- SQL Anywhere-Datenbanktools, C-API-Referenz
 - dbtools.h-Headerdatei,878
- SQL Anywhere-JDBC-Treiber
 - 4.0 laden,423
 - erforderliche Dateien,423
 - Info,419
 - JDBC-Treiber wählen,420
 - Komponenten,982
 - URL,423
 - Verbindung herstellen,423
 - verwenden,423
- SQL Anywhere-ODBC-Treiber
 - ANSI,366
 - Datenquellen definieren,976
 - installieren,970
 - Komponenten,971
 - Namen für das Deployment anpassen,974
 - Registrierungseinstellungen,974
 - Treibername,370
- SQL Anywhere-PHP-API
 - Info,684
- SQL Anywhere-PHP-Erweiterung
 - Info,669
 - konfigurieren,990
- SQL Anywhere-PHP-Modul
 - API-Referenz,684
- SQL Anywhere-Plug-In
 - Hinweise zum Deployment,993
- SQL Anywhere-Python-Datenbankunterstützung
 - Info,661
- SQL Anywhere-Webdienste
 - Info,755
- SQL Remote
 - Deployment,1030
- SQL Server

- Deployment von Zuordnungsdateien,960
- SQL-Anweisungen
 - ausführen,751
 - Webclients,811
 - Webdienste,769
- SQL-Anwendungen
 - SQL-Anweisungen ausführen,1
- SQL-Kommunikationsbereich
 - Info,495
- SQL-Präprozessor-Dienstprogramm (sqlpp)
 - Info,469
 - Syntax,469
- SQL/1992
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- SQL/1999
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- SQL/2003
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- SQL/2008
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- SQL_ATTR_CONCURRENCY-Attribut
 - Info,389
- SQL_ATTR_CONNECTION_DEAD
 - SQLGetConnectAttr,373
- SQL_ATTR_CURSOR_SCROLLABLE-Attribut
 - Info,389
- SQL_ATTR_KEYSET_SIZE
 - ODBC-Attribut,33
- SQL_ATTR_MAX_LENGTH-Attribut
 - Info,390
- SQL_ATTR_ROW_ARRAY_SIZE
 - mehrere Zeilen abrufen,11
 - ODBC-Attribut,33
- SQL_CALLBACK, Typendeklaration
 - Info,546
- SQL_CALLBACK_PARM, Typendeklaration
 - Info,546
- SQL_CONCUR_LOCK
 - Parallelitätswert,389
- SQL_CONCUR_READ_ONLY
 - Parallelitätswert,389
- SQL_CONCUR_ROWVER
 - Parallelitätswert,389
- SQL_CONCUR_VALUES
 - Parallelitätswert,389
- SQL_CURSOR_KEYSET_DRIVEN
 - ODBC-Cursorattribut,33
- SQL_ERROR
 - ODBC-Rückgabewert,397
- SQL_INVALID_HANDLE
 - ODBC-Rückgabewert,397
- SQL_NEED_DATA
 - ODBC-Rückgabewert,397
- sql_needs_quotes-Funktion
 - Info,559
- SQL_NO_DATA_FOUND
 - ODBC-Rückgabewert,397
- SQL_ROWSET_SIZE
 - mehrere Zeilen abrufen,11
- SQL_SUCCESS
 - ODBC-Rückgabewert,397
- SQL_SUCCESS_WITH_INFO
 - ODBC-Rückgabewert,397
- SQL_TXN_READ_COMMITTED
 - Isolationsstufe,388
- SQL_TXN_READ_UNCOMMITTED
 - Isolationsstufe,388
- SQL_TXN_REPEATABLE_READ
 - Isolationsstufe,388
- SQL_TXN_SERIALIZABLE
 - Isolationsstufe,388
- SQLAllocHandle, ODBC-Funktion
 - Anweisungen ausführen,378
 - Info,368
 - Parameter binden,379
 - verwenden,369
- sqlany.cvf
 - Deployment von Datenbankservern,1016
- SQLANY16-Umgebungsvariable
 - Deployment,950
 - Deployment-Einstellung,1009
- sqlany_affected_rows-Funktion [Ruby-API]
 - Beschreibung,726
- sqlany_affected_rows-Methode [SQL Anywhere-C-API]
 - Beschreibung,566
- sqlany_bind_param-Funktion [Ruby-API]
 - Beschreibung,726
- sqlany_bind_param-Methode [SQL Anywhere-C-API]
 - Beschreibung,567
- sqlany_cancel-Methode [SQL Anywhere-C-API]

Beschreibung,567
 sqlany_clear_error-Funktion [Ruby-API]
 Beschreibung,727
 sqlany_clear_error-Methode [SQL Anywhere-C-API]
 Beschreibung,568
 sqlany_client_version-Funktion [Ruby-API]
 Beschreibung,728
 sqlany_client_version-Methode [SQL Anywhere-C-API]
 Beschreibung,568
 sqlany_client_version_ex-Methode [SQL Anywhere-C-API]
 Beschreibung,568
 sqlany_commit-Funktion [Ruby-API]
 Beschreibung,728
 sqlany_commit-Methode [SQL Anywhere-C-API]
 Beschreibung,569
 sqlany_connect-Funktion [Ruby-API]
 Beschreibung,728
 sqlany_connect-Methode [SQL Anywhere-C-API]
 Beschreibung,569
 sqlany_describe_bind_param-Funktion [Ruby-API]
 Beschreibung,729
 sqlany_describe_bind_param-Methode [SQL Anywhere-C-API]
 Beschreibung,570
 sqlany_disconnect-Funktion [Ruby-API]
 Beschreibung,730
 sqlany_disconnect-Methode [SQL Anywhere-C-API]
 Beschreibung,571
 sqlany_error-Funktion [Ruby-API]
 Beschreibung,730
 sqlany_error-Methode [SQL Anywhere-C-API]
 Beschreibung,572
 sqlany_execute-Funktion [Ruby-API]
 Beschreibung,731
 sqlany_execute-Methode [SQL Anywhere-C-API]
 Beschreibung,572
 sqlany_execute_direct-Funktion [Ruby-API]
 Beschreibung,731
 sqlany_execute_direct-Methode [SQL Anywhere-C-API]
 Beschreibung,573
 sqlany_execute_immediate-Funktion [Ruby-API]
 Beschreibung,732
 sqlany_execute_immediate-Methode [SQL Anywhere-C-API]
 Beschreibung,574
 sqlany_fetch_absolute-Funktion [Ruby-API]
 Beschreibung,733
 sqlany_fetch_absolute-Methode [SQL Anywhere-C-API]
 Beschreibung,575
 sqlany_fetch_next-Funktion
 Info,733
 sqlany_fetch_next-Methode [SQL Anywhere-C-API]
 Beschreibung,575
 sqlany_finalize_interface-Methode [SQL Anywhere-C-API]
 Beschreibung,576
 sqlany_fini-Funktion [Ruby-API]
 Beschreibung,734
 sqlany_fini-Methode [SQL Anywhere-C-API]
 Beschreibung,576
 sqlany_fini_ex-Methode [SQL Anywhere-C-API]
 Beschreibung,576
 sqlany_free_connection-Funktion [Ruby-API]
 Beschreibung,735
 sqlany_free_connection-Methode [SQL Anywhere-C-API]
 Beschreibung,577
 sqlany_free_stmt-Funktion [Ruby-API]
 Beschreibung,735
 sqlany_free_stmt-Methode [SQL Anywhere-C-API]
 Beschreibung,577
 sqlany_get_bind_param_info-Funktion [Ruby-API]
 Beschreibung,736
 sqlany_get_bind_param_info-Methode [SQL Anywhere-C-API]
 Beschreibung,577
 sqlany_get_column-Funktion [Ruby-API]
 Beschreibung,736
 sqlany_get_column-Methode [SQL Anywhere-C-API]
 Beschreibung,578
 sqlany_get_column_info-Funktion [Ruby-API]
 Beschreibung,737
 sqlany_get_column_info-Methode [SQL Anywhere-C-API]
 Beschreibung,579
 sqlany_get_data-Methode [SQL Anywhere-C-API]
 Beschreibung,580
 sqlany_get_data_info-Methode [SQL Anywhere-C-API]
 Beschreibung,580
 sqlany_get_next_result-Funktion [Ruby-API]
 Beschreibung,738

- sqlany_get_next_result-Methode [SQL Anywhere-C-API]
 - Beschreibung,581
- sqlany_init-Funktion [Ruby-API]
 - Beschreibung,739
- sqlany_init-Methode [SQL Anywhere-C-API]
 - Beschreibung,582
- sqlany_init_ex-Methode [SQL Anywhere-C-API]
 - Beschreibung,583
- sqlany_initialize_interface-Methode [SQL Anywhere-C-API]
 - Beschreibung,583
- sqlany_make_connection-Methode [SQL Anywhere-C-API]
 - Beschreibung,584
- sqlany_make_connection_ex-Methode [SQL Anywhere-C-API]
 - Beschreibung,584
- sqlany_new_connection-Funktion [Ruby-API]
 - Beschreibung,739
- sqlany_new_connection-Methode [SQL Anywhere-C-API]
 - Beschreibung,585
- sqlany_new_connection_ex-Methode [SQL Anywhere-C-API]
 - Beschreibung,585
- sqlany_num_cols-Funktion [Ruby-API]
 - Beschreibung,740
- sqlany_num_cols-Methode [SQL Anywhere-C-API]
 - Beschreibung,586
- sqlany_num_params-Funktion [Ruby-API]
 - Beschreibung,741
- sqlany_num_params-Methode [SQL Anywhere-C-API]
 - Beschreibung,586
- sqlany_num_rows-Funktion [Ruby-API]
 - Beschreibung,741
- sqlany_num_rows-Methode [SQL Anywhere-C-API]
 - Beschreibung,587
- sqlany_prepare-Funktion [Ruby-API]
 - Beschreibung,742
- sqlany_prepare-Methode [SQL Anywhere-C-API]
 - Beschreibung,587
- sqlany_reset-Methode [SQL Anywhere-C-API]
 - Beschreibung,588
- sqlany_rollback-Funktion [Ruby-API]
 - Beschreibung,743
- sqlany_rollback-Methode [SQL Anywhere-C-API]
 - Beschreibung,588
- sqlany_send_param_data-Methode [SQL Anywhere-C-API]
 - Beschreibung,589
- sqlany_sqlstate-Funktion [Ruby-API]
 - Beschreibung,743
- sqlany_sqlstate-Methode [SQL Anywhere-C-API]
 - Beschreibung,590
- sqlanydb
 - Info,661
 - Python-Datenbank-API,661
 - Python-Skripten erstellen,663
 - unter Unix und Mac OS X installieren,662
 - unter Windows installieren,662
- SQLANYDIR
 - msiexec,954
- sqlanywhere.jpr
 - Deployment von Administrationstools unter Linux/Unix/Mac OS X,1007
 - Deployment von Administrationstools unter Windows,997
- sqlanywhere_en16.chm
 - Deployment unter Windows,1015
- sqlanywhere_en16.map
 - Deployment unter Windows,1015
- SQLAnywhereInterface-Struktur [SQL Anywhere-C-API]
 - Beschreibung,597
- SQLBindCol, ODBC-Funktion
 - Info,390
 - Parametergröße,382
 - Speicherausrichtung,386
- SQLBindParam, ODBC-Funktion
 - Parametergröße,382
- SQLBindParameter, ODBC-Funktion
 - gespeicherte Prozeduren,393
 - Info,379
 - Parametergröße,382
 - Speicherausrichtung,386
- SQLBindParameter-Funktion
 - vorbereitete Anweisungen,380
 - vorbereitete ODBC-Anweisungen,4
- SQLBrowseConnect, ODBC-Funktion
 - Info,370
- SQLBulkOperations
 - ODBC-Funktion,12
- SQLCA
 - ändern,498

- Felder,495
- Info,495
- Länge von,496
- mehrere Bereiche verwalten,500
- Threads,498
- sqlcabc, SQLCA-Feld
 - Info,496
- sqlcaid
 - SQLCA-Feld,496
- sqlcode, SQLCA-Feld
 - Info,496
- SQLColAttribute, ODBC-Funktion
 - Parametergröße,382
- SQLColAttributes, ODBC-Funktion
 - Parametergröße,382
- SQLConnect, ODBC-Funktion
 - Info,370
- SQLConnect-ODBC-Funktion
 - Windows-Performance,977
- SQLCOUNT
 - sqlerror, SQLCA-Feldelement,496
- sqld, SQLDA-Feld
 - Info,505
- SQLDA
 - Deskriptoren,36
 - Dynamic SQL,501
 - Felder,505
 - freigeben,556
 - Hostvariablen,506
 - Info,504
 - sqlen-Felder,508
 - über fill_sqlda ausfüllen,557
 - über fill_sqlda_ex ausfüllen,557
 - Zeichenfolgen und fill_sqlda,557
 - Zeichenfolgen und fill_sqlda_ex,557
 - zuweisen,532
- sqlda_storage-Funktion
 - Info,560
- sqlda_string_length-Funktion
 - Info,560
- sqldabc, SQLDA-Feld
 - Info,505
- sqldaaid-SQLDA-Feld
 - Info,505
- sqldata, SQLDA-Feld
 - Info,506
- SQLDATETIME-Datentyp
 - Embedded SQL,487
- sqldef.h
 - Datentypen,482
 - Software-Exit-Codes, Speicherort,946
- SQLDescribeCol, ODBC-Funktion
 - Parametergröße,382
- SQLDescribeParam, ODBC-Funktion
 - Parametergröße,382
- SQLDriverConnect, ODBC-Funktion
 - Info,370
- sqlerrd, SQLCA-Feld
 - Info,496
- sqlerrmc, SQLCA-Feld
 - Info,496
- sqlerrml, SQLCA-Feld
 - Info,496
- SQLError, ODBC-Funktion
 - Info,397
- sqlerror, SQLCA-Feld
 - Elemente,496
 - SQLCOUNT,496
 - SQLIOCOUNT,496
 - SQLIOESTIMATE,497
- sqlerror_message-Funktion
 - Info,561
- sqlerrp, SQLCA-Feld
 - Info,496
- SQLExecDirect, ODBC-Funktion
 - gebundene Parameter,379
 - Info,378
- SQLExecute-Funktion
 - vorbereitete ODBC-Anweisungen,4
- SQLExtendedFetch
 - mehrere Zeilen abrufen,11
 - ODBC-Funktion,10
- SQLExtendedFetch, ODBC-Funktion
 - gespeicherte Prozeduren,393
 - Info,390
 - Parametergröße,382
- SQLFetch
 - ODBC-Funktion,10
- SQLFetch, ODBC-Funktion
 - gespeicherte Prozeduren,393
 - Info,390
- SQLFetchScroll
 - mehrere Zeilen abrufen,11
 - ODBC-Funktion,10
- SQLFetchScroll, ODBC-Funktion
 - Info,390

- Parametergröße,382
- SQLFreeHandle, ODBC-Funktion
 - verwenden,369
- SQLFreeStmt-Funktion
 - vorbereitete ODBC-Anweisungen,4
- SQLGetConnectAttr, ODBC-Funktion
 - Info,373
- SQLGetData, ODBC-Funktion
 - Info,390
 - Parametergröße,382
 - Speicherausrichtung,386
- SQLGetDescRec, ODBC-Funktion
 - Parametergröße,382
- sqlind, SQLDA-Feld
 - Info,506
- SQLIOCOUNT
 - sqlerror, SQLCA-Feldelement,496
- SQLIOESTIMATE
 - sqlerror, SQLCA-Feldelement,497
- SQLJ-Standard
 - Info,401
- SQLLEN versus SQLINTEGER
 - ODBC,382
- sqlllen, SQLDA-Feld
 - DESCRIBE-Anweisung,508
 - Info,506
 - Werte,508
 - Werte abrufen,513
 - Werte beschreiben,508
 - Werte senden,511
- sqlname, SQLDA-Feld
 - Info,506
- SQLNumResultCols, ODBC-Funktion
 - gespeicherte Prozeduren,393
- SQLParamOptions, ODBC-Funktion
 - Parametergröße,382
- sqlpp-Dienstprogramm
 - Info,469
 - Präprozessor Optionen,469
 - Syntax,469
 - unterstützte Compiler,473
- SQLPrepare-Funktion
 - Info,380
 - vorbereitete ODBC-Anweisungen,4
- SQLPutData, ODBC-Funktion
 - Parametergröße,382
- SQLRETURN
 - Typ des ODBC-Rückgabewertes,397
- SQLRowCount, ODBC-Funktion
 - Parametergröße,382
- SQLSetConnectAttr
 - ODBC-Anwendungen,375
- SQLSetConnectAttr, ODBC-Funktion
 - Info,373
 - Transaktions-Isolationsstufen,388
- SQLSetConnectOption, ODBC-Funktion
 - Parametergröße,382
- SQLSetDescRec, ODBC-Funktion
 - Parametergröße,382
- SQLSetParam, ODBC-Funktion
 - Parametergröße,382
- SQLSetPos, ODBC-Funktion
 - Info,392
 - Parametergröße,382
- SQLSetScrollOptions, ODBC-Funktion
 - Parametergröße,382
- SQLSetStmtAttr, ODBC-Funktion
 - Cursor-Merkmale,389
- SQLSetStmtOption, ODBC-Funktion
 - Parametergröße,382
- sqlstate, SQLCA-Feld
 - Info,496
- SqlState-Eigenschaft
 - SAError-Klasse [SQL Anywhere .NET-API],254
- SQLtransact, ODBC-Funktion
 - Info,370
- sqltype, SQLDA-Feld
 - DESCRIBE-Anweisung,508
 - Info,506
- SQLULEN versus SQLINTEGER
 - ODBC,382
- sqlvar, SQLDA-Feld
 - Info,505
 - Inhalt,506
- sqlwarn, SQLCA-Feld
 - Info,496
- SSDLToSA16.tt
 - Deployment von .NET 4.x-Clients,960
- SSIS
 - Deployment von Zuordnungsdateien,960
- Standards
 - SQLJ,401
- Starten
 - Datenbanken unter Verwendung von jConnect,427
- StartLine-Eigenschaft

SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],206
 State, Eigenschaft
 .NET-Datenprovider,47
 State-Eigenschaft
 SAConnection-Klasse [SQL Anywhere .NET-API],188
 StateChange-Ereignis
 SAConnection-Klasse [SQL Anywhere .NET-API],189
 Static SQL
 Info,501
 Statischer Cursor
 Info,22
 ODBC,33
 stax-api-1.0.jar
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 strlen_or_ind
 ODBC,382
 Stromausfall
 EnableFlush-Registrierungseintrag,1018
 Strukturverdichten
 Header-Dateien,473
 Sybase Central
 Deployment,992
 Deployment unter Linux und Unix,1003
 Deployment unter Windows,994
 Deployment von Administrationstools unter Linux/Unix/Mac OS X,1006
 Deployment von Administrationstools unter Windows,997
 Hinweise zum Deployment,993
 JAR-Dateien hinzufügen,411
 Java-Klassen hinzufügen,411
 jConnect-Metadaten-Unterstützung installieren,425
 Konfigurieren für Deployments,1011
 ZIP-Dateien hinzufügen,411
 Sybase Open Client-Unterstützung
 Info,747
 sybasecentral1600.jar
 Deployment unter Mac OS X,1005
 Deployment unter Unix,1005
 Deployment unter Windows,996
 symlink
 Deployment unter Unix,949
 SyncRoot-Eigenschaft

SAPParameterCollection-Klasse [SQL Anywhere .NET-API],301
 System.Transactions
 verwenden,65
 Systemanforderungen
 SQL Anywhere-.NET-Datenprovider,72
 Systemdatenquellen
 Windows,977
 Systemprozeduren
 HTTP,788
 SOAP,788

T

Tabellen-Adapter
 Visual Studio,92
 TableMappings-Eigenschaft
 SADDataAdapter-Klasse [SQL Anywhere .NET-API],220
 Tables-Feld
 SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],274
 TableViewer
 .NET-Datenprovider-Beispielprojekt,43
 TcpOptionsBuilder-Eigenschaft
 SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],167
 TcpOptionsString-Eigenschaft
 SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],168
 TDS-Eigenschaft
 SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],320
 Temporäre Optionen
 in ODBC-Anwendungen setzen,374
 this-Eigenschaft
 SABulkCopyColumnMappingCollection-Klasse [SQL Anywhere .NET-API],129
 SAConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],213
 SADDataReader-Klasse [SQL Anywhere .NET-API],249
 SAErrorCollection-Klasse [SQL Anywhere .NET-API],256
 SAPParameterCollection-Klasse [SQL Anywhere .NET-API],301
 Threads
 Java in der Datenbank,414

- mehrere SQLCAs,500
 - ODBC,360
 - ODBC-Anwendungen,374
 - Unix-Entwicklung,363
 - Verwaltung mehrerer Threads in Embedded SQL,498
 - Timeout-Eigenschaft
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],321
 - TimeSpan
 - SQL Anywhere-.NET-Datenprovider,63
 - TIMESTAMP-Datentyp
 - Open Client-Konvertierung,750
 - ToString-Methode
 - SACommLinksOptionsBuilder-Klasse [SQL Anywhere .NET-API],166
 - SAError-Klasse [SQL Anywhere .NET-API],253
 - SAInfoMessageEventArgs-Klasse [SQL Anywhere .NET-API],266
 - SAParameter-Klasse [SQL Anywhere .NET-API],281
 - SATcpOptionsBuilder-Klasse [SQL Anywhere .NET-API],317
 - Transaction-Eigenschaft
 - SACommand-Klasse [SQL Anywhere .NET-API],151
 - TransactionScope, Klasse
 - verwenden,65
 - Transaktionen
 - ADO,344
 - Anwendungsentwicklung,37
 - Autocommit-Modus,37
 - Autocommit-Verhalten steuern,37
 - Cursor,40
 - Isolationsstufe,39
 - Isolationsstufe für ODBC-Transaktionen wählen,388
 - ODBC,370
 - OLE DB,344
 - verteilt,865
 - verteilte verwenden,869
 - Transaktionsverarbeitung
 - SQL Anywhere-.NET-Datenprovider verwenden,64
 - Treiber
 - Deployment des SQL Anywhere ODBC-Treibers,970
 - die SQL Anywhere ODBC-Treiber unter Windows linken,361
 - jConnect-JDBC-Treiber,420
 - SQL Anywhere JDBC-Treiber,420
 - Treiber-Eintrag in der Registrierung ODBC,977
 - TryGetValue-Methode
 - SACConnectionStringBuilderBase-Klasse [SQL Anywhere .NET-API],212
 - TYPE-Klausel
 - angeben,801
 - Beispiel,812
- ## U
- Übereinstimmung
 - ODBC,360
 - Überlauffehler
 - Open Client-Datentypkonvertierung,750
 - ulplugin.jar
 - Deployment unter Windows,996
 - ulscutil16.dll
 - Deployment unter Windows,996
 - ultralite.jpr
 - Deployment von Administrationstools unter Linux/ Unix/Mac OS X,1008
 - Deployment von Administrationstools unter Windows,998
 - Umgebungs-Handle
 - ODBC,368
 - Umgebungsvariablen
 - CLASSPATH,404
 - für Java in der Datenbank,405
 - JAVA_HOME,405
 - JAVAHOME,405
 - Umleitung
 - URLs validieren,787
 - Unchained, Modus
 - Implementierung,39
 - steuern,37
 - Transaktionen,37
 - Unconditional-Eigenschaft
 - SACConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],206
 - Ungültige Zeichenfolgen- oder Pufferlänge
 - Datenausrichtung,386
 - Unicode

-
- ODBC-Anwendungen für Windows Mobile
 - linken,363
 - uninstallall-Option
 - SetupVSPackage,963
 - Unit-Enumeration [Datenbanktools-API]
 - Beschreibung,891
 - Unix
 - Anwendungen mit mehreren Threads,949
 - Hinweise zu Deploymentmethoden,949
 - ODBC,363
 - ODBC-Treibermanager,364
 - Verzeichnisstruktur,949
 - unixODBC
 - Treibermanager,365
 - unixodbc.h
 - Info,361
 - Kompilierungsplattform,382
 - Unload-Enumeration [Datenbanktools-API]
 - Beschreibung,891
 - UnquoteIdentifier-Methode
 - SACommandBuilder-Klasse [SQL Anywhere .NET-API],162
 - UNSIGNED BIGINT-Datentyp
 - Embedded SQL,487
 - Unterstützte Plattformen
 - OLE DB,338
 - UPDATE-Anweisung
 - JDBC,437
 - positionsbasiert,12
 - UpdateBatch ADO-Methode
 - ADO-Programmierung,344
 - Daten aktualisieren,344
 - UpdateBatchSize-Eigenschaft
 - SADDataAdapter-Klasse [SQL Anywhere .NET-API],220
 - UpdateCommand-Eigenschaft
 - SADDataAdapter-Klasse [SQL Anywhere .NET-API],221
 - UpdatedRowSource-Eigenschaft
 - SACommand-Klasse [SQL Anywhere .NET-API],151
 - Upgrade-Dienstprogramm [dbupgrad]
 - jConnect-Metadatenunterstützung installieren,425
 - URL-Klausel
 - angeben,799
 - URLs
 - Datenbank,426
 - interpretieren,790
 - jConnect,426
 - Sitzungsverwaltung,782
 - SQL Anywhere-JDBC-Treiber,423
 - Variablen übergeben,812
 - User-Agent
 - Zugriff auf HTTP-Header,774
 - UserDefinedTypes-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],274
 - UserID-Eigenschaft
 - SAConnectionStringBuilder-Klasse [SQL Anywhere .NET-API],207
 - UserList-Enumeration [Datenbanktools-API]
 - Beschreibung,892
 - Users-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],275
 - UTF-32-
 - ODBC-Treibermanager,366
- ## V
- Validation-Enumeration [Datenbanktools-API]
 - Beschreibung,892
 - Value-Eigenschaft
 - SAPParameter-Klasse [SQL Anywhere .NET-API],286
 - Value-Feld
 - SADefault-Klasse [SQL Anywhere .NET-API],252
 - VARCHAR-Datentyp
 - Embedded SQL,487
 - Variablen
 - an HTTP-Webdienste übergeben,811
 - in SOAP-Webdiensten,776
 - Zugriff in HTTP-Webdiensten,772
 - velocity-dep.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - velocity.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
 - Verarbeitungsfortschritts-Meldungen
 - ODBC,375
 - Verbindung
 - Zugriff auf HTTP-Header,774
 - Verbindungen

- ADO-Verbindungsobjekt,338
- Funktionen,553
- connect,427
- connect-URL,426
- JDBC,422
- JDBC im Server,431
- JDBC-Beispiel,427
- JDBC-Beispiel, serverseitig,431
- JDBC-Client-Anwendungen,427
- JDBC-Standardwerte,434
- JDBC-Transaktion, Isolationsstufe,434
- ODBC, Attribute abrufen,373
- ODBC, Attribute einstellen,373
- ODBC-Funktionen,370
- ODBC-Programmierung,371
- SQL Anywhere-JDBC-Treiber, URL,423
- Verbindung zu einer Datenbank mit dem .NET-Datenprovider,45
- Webanwendungen lizenzieren,779
- Verbindungs-Handle
 - ODBC,368
- Verbindungseigenschaften
 - Webdienste,789
- Verbindungsparameter
 - OLE DB,345
- Verbindungspooling
 - OLE DB,347
 - Webdienste,769
- Verbindungspools
 - .NET-Datenprovider,46
- Verbindungsprofile
 - Deployment von Administrationstools unter Linux, Solaris und Mac OS X,1010
 - Deployment von Administrationstools unter Windows,1002
- Verbindungsserver
 - 4-teilige Syntax,348
 - InProcess-Option,349,351
 - OLE DB,348
 - OPENQUERY,348
 - RPC Out-Option,349,351
 - RPC-Option,349,351
 - Sicherheitskontext,349,351
 - vierteilige Syntax,348
- Verbindungsstatus
 - .NET-Datenprovider,47
- Verbosity-Enumeration [Datenbanktools-API]
 - Beschreibung,893
- Verfügbarkeit
 - Timeout-Callback,547
- VerifyServerName-Eigenschaft
 - SAOptionsBuilder-Klasse [SQL Anywhere .NET-API],321
- Verlorene Aktualisierungen
 - Info,30
- Version-Enumeration [Datenbanktools-API]
 - Beschreibung,894
- Versionsnummer
 - Dateinamen,950
- Verteilte Transaktionen
 - Architektur,868
 - dreischichtige Datenverarbeitung,866
 - Einbeziehung,867
 - Einschränkungen,869
 - Info,865
 - Wiederherstellung,870
- Verteilte Transaktionsverarbeitung
 - SQL Anywhere-.NET-Datenprovider verwenden,65
- Verzeichnisstruktur
 - Unix,949
- ViewColumns-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],275
- Views-Feld
 - SAMetaDataCollectionNames-Klasse [SQL Anywhere .NET-API],275
- Visual Basic
 - praktische Einführung,88
 - Unterstützung im .NET-Datenprovider,41
- Visual C#
 - praktische Einführung,88
- Visual C++
 - Unterstützung von Embedded SQL,473
- VM
 - Java starten,417
 - Java stoppen,417
 - Java VM,403
 - Shutdown-Hooks,417
- Vorbereiten
 - Anweisungen,2
 - zum Festschreiben,867
- Vorbereitete Anweisungen
 - ADO.NET-Überblick,4
 - Bindungsparameter,3
 - Cursor,8

- JDBC,440
- löschen,3
- ODBC,380
- Open Client,751
- verwenden,2

W

- WebClientLogFile-Eigenschaft
 - Serveroption,831

- WebClientLogging-Eigenschaft
 - Serveroption,831

- Webclients
 - Anforderungen für Funktionen und Prozeduren,799
 - Anforderungstypen angeben,801
 - auf Ergebnismengen zugreifen,814
 - Ergebnismengen abrufen,816
 - Ersetzungsparameter,829
 - Funktionen und Prozeduren,798
 - HTTP-Anforderungs-Header verwalten,803
 - Info,794
 - Ports angeben,802
 - Schnellstart,794
 - SOAP-Header verwalten,805
 - SOAP-Namespace verwalten,809
 - SQL-Anweisungen,811
 - URL-Klausel,799
 - Variablen übergeben,811

- Webdienste
 - alphabetische Liste der Systemprozeduren,788
 - ändern,762
 - Array-Typen,825
 - auf Ergebnismengen zugreifen,814
 - auf HTTP-Variablen und -Header zugreifen,772
 - auf SOAP-Header zugreifen,776
 - Client-Logdatei, Info,831
 - Cross-Site-Scripting,787
 - Datentypen,817
 - entwickeln,770
 - Ergebnismengen abrufen,816
 - erstellen,762
 - Fehler,832
 - Hostvariablen,773
 - HTTP_HEADER-Beispiel,773
 - HTTP_VARIABLE-Beispiel,773
 - Info,755
 - kommentieren,766
 - Liste der Systemprozeduren für Webdienste,788

- löschen,766
- MIME-Typen, praktische Einführung,834
- NEXT_HTTP_HEADER-Beispiel,773
- NEXT_HTTP_VARIABLE-Beispiel,773
- Optionen,789
- Pooling,769
- Protokolle aktivieren,757
- Protokolle konfigurieren,758
- protokollieren,831
- root,767
- Sitzungen verwalten,779
- SOAP-Datentypen,824
- SOAP/DISH, praktische Einführung,839
- SQL-Anweisungen,769
- Strukturtypen,825
- Typen,760
- URLs interpretieren,790
- Verbindungseigenschaften,789
- verwalten,760,762
- Zeichensätze,787
- Zugriff,794

- Webseiten
 - anpassen,771
 - PHP-Skripten ausführen,670

- Webserver
 - Anwendungsentwicklung,770
 - Info,755
 - mehrere starten,759
 - PHP-API,669
 - Protokolle aktivieren,757
 - Protokolle konfigurieren,758
 - Schnellstart,755

- Weite Einfügungen
 - ESQL,518
 - JDBC,442

- Weites Speichern
 - ESQL,518

- Wertsensitive Cursor
 - Beispiele für Aktualisieren,20
 - Beispiele für Löschen,18
 - Einführung,18
 - Info,26

- Wiederherstellung
 - verteilte Transaktionen,870

- Windows
 - Deployment von Administrationstools,994
 - dialogfreie Installation,954
 - OLE DB-Unterstützung,337

- Windows Mobile
 - dbtool16.dll,871
 - ODBC,362
 - OLE DB-Unterstützung,337
- WindowsAccessBridge.dll
 - Deployment von Java-basierten Administrationstools,1000
- WITH HOLD-Klausel
 - Cursor,10
- Wow6432Node
 - Registrierungseinträge,977
- WRITE_CLIENT_FILE-Funktion
 - ESQL-Client-API-Callback-Funktion,546
- WriteToServer-Methode
 - SABulkCopy-Klasse [SQL Anywhere .NET-API],109
- wsimport
 - JAX-WS und Webdienste,860
- wstx-asl-3.2.6.jar
 - Deployment unter Mac OS X,1005
 - Deployment unter Unix,1005
 - Deployment unter Windows,996
- DT_STRING wird mit Leerzeichen aufgefüllt,483
- Embedded SQL,469
- Zeichensätze
 - CHAR-Zeichensatz festlegen,538
 - NCHAR-Zeichensatz festlegen,539
 - Webdienste,787
- Zeilenlänge
 - SQL-Präprozessor-Ausgabe,469
- Zeilennummern
 - Dienstprogramm für den SQL-Präprozessor (sqlpp),469
- Zeitstruktur
 - Zeitwerte in .NET-Datenprovider,63
- Zeitwerte
 - mit .NET-Datenprovider erhalten,63
- Zuordnungsdateien
 - Deployment für SQL Server,960
- Zwei-Phasen-Commit
 - dreischichtige Datenverarbeitung,866
 - und Open Client,753
 - verteilte Transaktionen verwalten,867

X

- XML-Dienste
 - erstellen,763
 - Info,760
 - kommentieren,766
 - löschen,766
 - Schnellstart,796
 - Schnellstart für Webclients,794
 - Schnellstart für Webserver,755
- XMLCONCAT-Funktion
 - Beispiel,772
- XMLELEMENT-Funktion
 - Beispiel,772
- XSS
 - Webdienste,787

Z

- Zeichendaten
 - Länge in Embedded SQL,490
 - Zeichensätze in Embedded SQL,490
- Zeichenfolgen
 - Datentyp,560
 - DT_NSTRING wird mit Leerzeichen aufgefüllt,483