PUBLIC
SQL Anywhere Server
Document Version: 17.01.0 – 2023-09-12

SQL Anywhere Programming

© 2025 SAP SE or an SAP affiliate company. All rights reserved.

Content

1 SQL Anywhere Server - Programming. 7
1.1 Programming Interfaces. 9
Interface Library Communication Protocols. 10
1.2 Application Development Using SQL. 11
SQL Statement Execution in Applications. 12
Prepared Statements. 13
Cursor Usage. 16
Cursor Principles. 19
Cursor Types. 25
Cursor Attributes 28
Result Set Descriptors. 46
Transactions in Applications. 47
1.3 Appendix - .NET Framework. 53
SQL Anywhere .NET Data Provider. 54
SQL Anywhere .NET Core Data Provider. 104
.NET Data Provider Tutorials. 105
SQL Anywhere ASP.NET Providers. 123
SQL Anywhere .NET API Reference. 131
1.4 OLE DB and ADO Development. 131
OLE DB. 132
ADO Programming with the OLE DB Provider. 133
OLE DB Connection Parameters. 141
OLE DB Connection Pooling. 144
OLE DB Isolation Levels. 145
Microsoft Linked Servers. 145
Supported OLE DB Interfaces. 149
OLE DB Provider Registration. 154
1.5 ODBC Support. 154
Requirements for Developing ODBC Applications. 156
ODBC Application Development. 158
Sample ODBC Programs. 163
ODBC Handles. 166
ODBC Connection Functions. 168
Server Options Changed by ODBC. 172
SQLSetConnectAttr Extended Connection Attributes. 173
SQL_ATTR_CURRENT_CATALOG ODBC API Connection Attribute. 175

Considerations for the Windows DllMain Function. . . . 176
Ways to Execute SQL Statements. . . . 177
64-bit ODBC Considerations. . . . 182
Data Alignment Requirements. . . . 185
Result Sets in ODBC Applications. . . . 186
Stored Procedure Considerations. . . . 192
ODBC Escape Syntax. . . . 194
Error Handling in ODBC. . . . 196
1.6 Java in the Database. . . . 199
Introduction to Java in the Database. . . . 200
Java Error Handling. . . . 201
Tutorial: Using Java in the Database. . . . 202
How to Install Java Classes into a Database. . . . 209
Special Features of Java Classes in the Database. . . . 213
How to Start and Stop the Java VM. . . . 218
Shutdown Hooks in the Java VM. . . . 218
1.7 XS JavaScript Application Programming (Deprecated). . . . 219
1.8 JDBC Support. . . . 221
JDBC Applications. . . . 223
JDBC Drivers. . . . 224
SQL Anywhere JDBC Driver. . . . 225
The jConnect JDBC Driver. . . . 226
JDBC Program Structure. . . . 232
Differences Between Client- and Server-side JDBC Applications. . . . 233
A Sample Client-side JDBC Application. . . . 234
A Sample Server-side JDBC Application. . . . 238
Notes on JDBC Connections. . . . 241
Server-side Data Access Using JDBC. . . . 242
JDBC Callbacks. . . . 253
JDBC Escape Syntax. . . . 256
SQL Anywhere JDBC API Support. . . . 259
1.9 Node.js Application Programming. . . . 260
1.10 Embedded SQL. . . . 260
Development Process Overview. . . . 263
The Embedded SQL Preprocessor. . . . 264
Supported Compilers. . . . 268
Embedded SQL Header Files. . . . 269
Import Libraries. . . . 269
Embedded SQL Program Example. . . . 270
Structure of Embedded SQL Programs. . . . 271
Loading DBLIB Dynamically under Windows. . . . 271

Sample Embedded SQL Programs. . . . 272
Embedded SQL Data Types. . . . 277
Host Variables in Embedded SQL. . . . 281
The SQL Communication Area (SQLCA). . . . 290
Static and Dynamic SQL. . . . 296
The SQL Descriptor Area (SQLDA). . . . 300
How to Fetch Data Using Embedded SQL. . . . 311
Wide Inserts Using Embedded SQL. . . . 318
Wide Deletes Using Embedded SQL. . . . 322
Wide Merges Using Embedded SQL. . . . 325
How to Send and Retrieve Long Values Using Embedded SQL. . . . 329
Simple Stored Procedures in Embedded SQL. . . . 336
Request Management with Embedded SQL. . . . 339
Database Backup with Embedded SQL. . . . 339
Library Function Reference. . . . 340
Embedded SQL Statement Summary. . . . 386
1.11 SQL Anywhere Database API for C/C++. . . . 388
SQL Anywhere C API Support. . . . 389
SQL Anywhere C API Reference. . . . 390
1.12 External Call Interface. . . . 391
Procedures and Functions That Use External Calls. . . . 392
External Function Prototypes. . . . 393
External Function Call Interface Methods. . . . 407
Data Type Handling. . . . 411
How to Unload an External Library. . . . 413
1.13 External Environment Support. . . . 413
The CLR External Environment. . . . 415
The ESQL and ODBC External Environments. . . . 419
The Java External Environment. . . . 427
The JavaScript External Environment (Deprecated). . . . 432
The Perl External Environment. . . . 438
The PHP External Environment (Deprecated). . . . 442
1.14 Perl DBI Support. . . . 448
DBD::SQLAnywhere. . . . 448
Installing DBD::SQLAnywhere on Windows. . . . 449
Installing DBD::SQLAnywhere on UNIX/Linux. . . . 451
Perl Scripts That Use DBD::SQLAnywhere. . . . 453
1.15 Python and Database Access. . . . 458
Installing sqlanydb on Windows. . . . 459
Installing sqlanydb on UNIX/Linux. . . . 460
Installing the Django Driver (sqlany-django). . . . 462

Installing the SQLAlchemy Dialect (sqlalchemy-sqlany). 462
Python Scripts That Use sqlanydb. 462
1.16 PHP Support (Deprecated). 468
SQL Anywhere PHP Extension (Deprecated). 469
SQL Anywhere PHP API Reference (Deprecated). 478
1.17 Ruby Support. 529
Ruby Programming. 529
SQL Anywhere Ruby API Reference. 539
1.18 SAP Open Client Support. 571
Open Client Architecture. 573
What You Need to Build Open Client Applications. 574
Open Client Data Type Mappings. 574
SQL in Open Client Applications. 575
Known Open Client Limitations of SQL Anywhere 578
System Requirements for Using SQL Anywhere as an Open Server. 579
Database Server as an Open Server Startup. 580
1.19 OData Support. 580
OData Server Architecture. 581
OData Protocol Limitations. 583
OData Server Security Considerations. 584
How to Set Up an OData Server. 586
OData Server Samples. 588
How to Set Up Repeatable Requests. 589
How to Protect Against Cross-site Request Forgery Attacks. 590
How to Create an OData Producer Service Model. 591
How to Configure OData Producers for a Third-party HTTP server. 592
OSDL Statement Reference. 596
1.20 HTTP Web Services. 603
The Database Server as an HTTP Web Server. 603
Access to Web Services Using Web Clients. 647
Web Service Error Code Reference. 682
HTTP Web Service Examples. 684
1.21 Three-tier Computing and Distributed Transactions. 716
Three-tier Computing Architecture. 717
Distributed Transactions. 719
1.22 Database Tools Programming. 721
Database Tools Interface (DBTools). 722
SQL Anywhere Database Tools C API Reference. 729
Software Component Exit Codes. 806
1.23 Database and Application Deployment. 807
Types of Deployment. 808

Ways to Distribute Files. 809
Installation Directories and File Names. 810
The *Deployment Wizard* for Windows 813
Silent Installs on Microsoft Windows. 817
The Deployment Wizard for UNIX/Linux. 822
Silent Installs on UNIX/Linux. 823
Requirements for Deploying Client Applications. 826
Administration Tool Deployment. 857
Documentation Deployment. 880
Database Server Deployment. 881
DLL Registration on Windows. 888
External Environment Support Deployment. 889
Encryption Deployment. 892
LDAP Deployment. 894
Embedded Database Application Deployment. 894

1 SQL Anywhere Server - Programming

This book describes how to build and deploy database applications using the C, C++, Java, Perl, PHP, Python, Ruby, and Microsoft .NET programming languages (such as Microsoft Visual Basic and Microsoft Visual C#). A variety of programming interfaces such as Microsoft ADO.NET, OLE DB, and ODBC are described.

In this section:

Programming Interfaces [page 9]
Several data access programming interfaces provide flexibility in the kinds of applications and application development environments you can use.

Application Development Using SQL [page 11]
SQL can be executed from applications by using a variety of application programming interfaces such as ADO.NET, JDBC, ODBC, Embedded SQL, OLE DB, and Open Client.

Appendix - .NET Framework [page 53]
Use SQL Anywhere with .NET, including the API for the SQL Anywhere .NET Data Provider.

OLE DB and ADO Development [page 131]
An OLE DB provider for OLE DB and ADO applications is included with the software.

ODBC Support [page 154]
ODBC (Open Database Connectivity) is a standard call level interface (CLI) developed by Microsoft Corporation. It is based on the SQL Access Group CLI specification.

Java in the Database [page 199]
The database server supports a mechanism for executing Java classes from within the database environment. Using Java methods from the database server provides powerful ways of adding programming logic to a database.

XS JavaScript Application Programming (Deprecated) [page 219]
The SQL Anywhere XS JavaScript driver can be used to connect to SQL Anywhere databases, issue SQL queries, and obtain result sets.

JDBC Support [page 221]
JDBC is a call-level interface for Java applications. JDBC provides you with a uniform interface to a wide range of relational databases, and provides a common base on which higher level tools and interfaces can be built.

Node.js Application Programming [page 260]
The Node.js API can be used to connect to SQL Anywhere databases, issue SQL queries, and obtain result sets.

Embedded SQL [page 260]
SQL statements embedded in a C or C++ source file are referred to as Embedded SQL. A preprocessor translates these statements into calls to a runtime library. Embedded SQL is an ISO/ANSI and IBM standard.

SQL Anywhere Database API for C/C++ [page 388]
The SQL Anywhere C application programming interface (API) is a data access API for the C / C++ languages.

External Call Interface [page 391]
You can call a function in an external library from a stored procedure or function.

External Environment Support [page 413]
Seven external runtime environments are supported. These include Embedded SQL and ODBC applications written in C/C++, and applications written in Java, JavaScript (Deprecated), Perl, PHP (Deprecated), or languages such as Microsoft C# and Microsoft Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Perl DBI Support [page 448]
DBD::SQLAnywhere is a database driver for the Perl Database Independent Interface (DBI), which is a data access API for the Perl language.

Python and Database Access [page 458]
The Python extension modules sqlanydb, sqlany-django, and sqlalchemy-sqlany support the use of Python when connecting to databases, issuing SQL queries, and obtaining result sets.

PHP Support (Deprecated) [page 468]
You can use PHP to develop database applications.

Ruby Support [page 529]
You can use Ruby to develop database applications.

SAP Open Client Support [page 571]
SAP Open Client provides customer applications, third-party products, and other SAP products with the interfaces needed to communicate with Open Servers.

OData Support [page 580]
OData (Open Data Protocol) enables data services over RESTful HTTP. It allows you to perform operations through URIs (Uniform Resource Identifiers) to access and modify information.

HTTP Web Services [page 603]
You can create HTTP web servers using SQL Anywhere as well as send requests and receive replies from other web servers.

Three-tier Computing and Distributed Transactions [page 716]
You can use the database server as a **resource manager**, participating in distributed transactions coordinated by a transaction server.

Database Tools Programming [page 721]
You can write applications in C or C++ that implement common database tasks such as backup and unload.

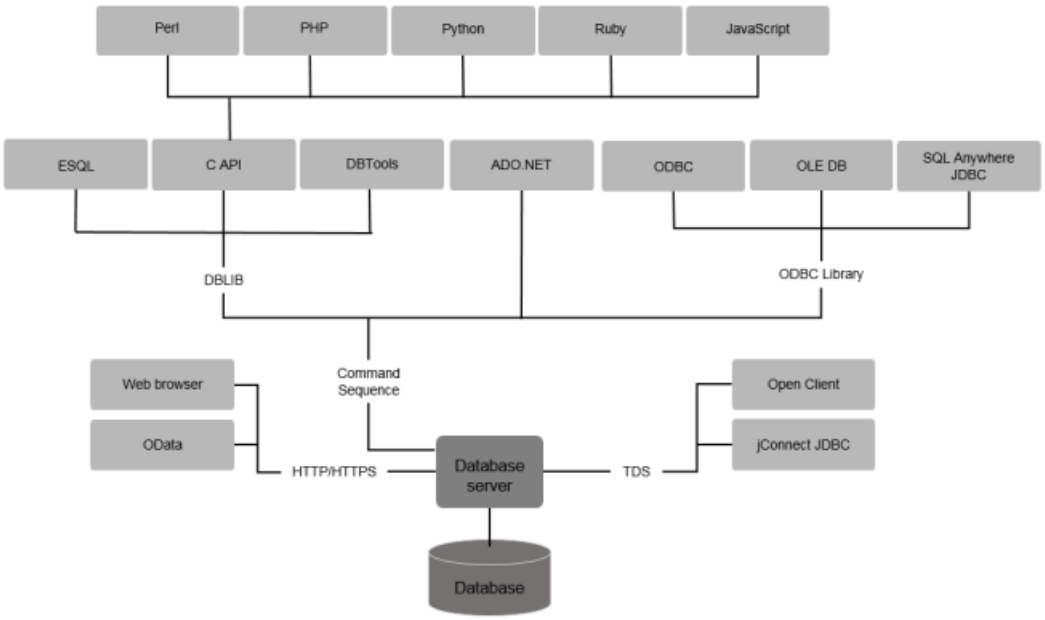
Database and Application Deployment [page 807]
When you have completed a database application, you must deploy the application to your end users.

1.1 Programming Interfaces

Several data access programming interfaces provide flexibility in the kinds of applications and application development environments you can use.

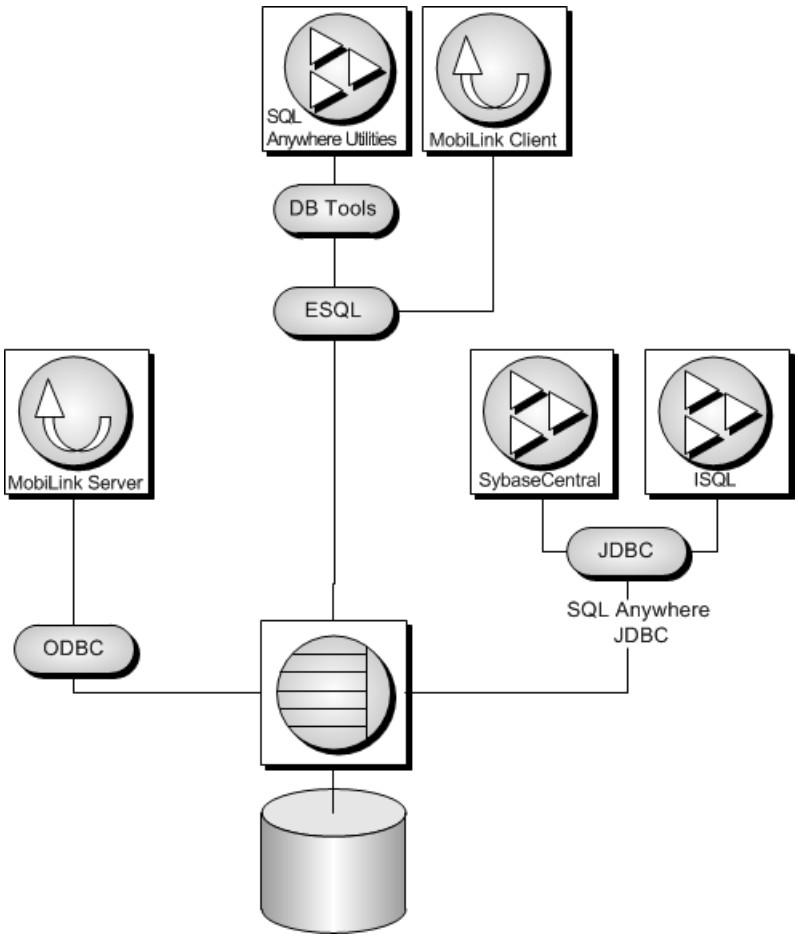
Supported SQL Anywhere Programming Interfaces and Protocols

The following diagram displays the supported interfaces, and the interface libraries used. The name of the interface library and the interface are usually the same.



SQL Anywhere applications

The applications supplied with SQL Anywhere use several of these interfaces:



In this section:

Interface Library Communication Protocols [page 10]
Each interface library communicates with the database server using a communication protocol.

1.1.1 Interface Library Communication Protocols

Each interface library communicates with the database server using a communication protocol.

SQL Anywhere supports two communication protocols, **Command Sequence** (**CmdSeq**) and **Tabular Data Stream** (**TDS**). These protocols are internal, and for most purposes it does not matter which one you are using. Your choice of development environment is governed by your available tools.

The major differences are visible when connecting to the database. Command Sequence applications and TDS applications use different methods to identify a database and database server, and so connection parameters are different.

Command Sequence

This protocol is used by SQL Anywhere, the SQL Anywhere JDBC driver, and the Embedded SQL, ODBC, OLE DB, and ADO.NET APIs. Client-side data transfers are supported by the **CmdSeq** protocol.

TDS

This protocol is used by SAP Adaptive Server Enterprise, the jConnect JDBC driver, and SAP Open Client applications. Client-side data transfers are not supported by the **TDS** protocol.

1.2 Application Development Using SQL

SQL can be executed from applications by using a variety of application programming interfaces such as ADO.NET, JDBC, ODBC, Embedded SQL, OLE DB, and Open Client.

In this section:

SQL Statement Execution in Applications [page 12]
The way you include SQL statements in your application depends on the application development tool and programming interface you use.

Prepared Statements [page 13]
Each time a statement is sent to a database, the database server must perform a series of steps.

Cursor Usage [page 16]
When you execute a query in an application, the result set consists of several rows. In general, you do not know how many rows the application is going to receive before you execute the query.

Cursor Principles [page 19]
To use a cursor, there are some basic steps to follow.

Cursor Types [page 25]
The various programming interfaces do not support all aspects of database cursors. The terminology may also be different. The mappings between cursors and the options available to for each programming interface is explained in the following material.

Cursor Attributes [page 28]
Any cursor, once opened, has an associated result set. The cursor is kept open for a length of time. During that time, the result set associated with the cursor may be changed, either through the cursor itself or, subject to isolation level requirements, by other transactions.

Result Set Descriptors [page 46]
Some applications build SQL statements that cannot be completely specified in the application. Sometimes statements are dependent on a user response before the application knows exactly what information to retrieve, such as when a reporting application allows a user to select which columns to display.

Transactions in Applications [page 47]
Transactions are sets of atomic SQL statements. Either all statements in the transaction are executed, or none.

1.2.1 SQL Statement Execution in Applications

The way you include SQL statements in your application depends on the application development tool and programming interface you use.

ADO.NET

You can execute SQL statements using various ADO.NET objects. The SACommand object is one example:

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

ODBC

If you are writing directly to the ODBC programming interface, your SQL statements appear in function calls. For example, the following C function call executes a DELETE statement:

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

JDBC

If you are using the JDBC programming interface, you can execute SQL statements by invoking methods of the statement object. For example:

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

Embedded SQL

If you are using Embedded SQL, you prefix your C language SQL statements with the keyword EXEC SQL. The code is then run through a preprocessor before compiling. For example:

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

SAP Open Client

If you use the SAP Open Client interface, your SQL statements appear in function calls. For example, the following pair of calls executes a DELETE statement:

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

For more details about including SQL in your application, see your development tool documentation. If you are using ODBC or JDBC, consult the software development kit for those interfaces.

Applications Inside the Database Server

In many ways, stored procedures and triggers act as applications or parts of applications running inside the database server. You can also use many of the techniques here in stored procedures.

Java classes in the database can use the JDBC interface in the same way as Java applications outside the server.

Related Information

SQL Anywhere .NET Data Provider [page 54]
ODBC Support [page 154]
JDBC Support [page 221]
Embedded SQL [page 260]
SAP Open Client Support [page 571]
Stored Procedures, Triggers, Batches, and User-defined Functions
JDBC Support [page 221]

1.2.2 Prepared Statements

Each time a statement is sent to a database, the database server must perform a series of steps.

- It must parse the statement and transform it into an internal form. This process is sometimes called **preparing** the statement.
- It must verify the correctness of all references to database objects by checking, for example, that columns named in a query actually exist.
- If the statement involves joins or subqueries, then the query optimizer generates an access plan.
- It executes the statement after all these steps have been carried out.

Reusing Prepared Statements Can Improve Performance

If you use the same statement repeatedly, for example inserting many rows into a table, repeatedly preparing the statement causes a significant and unnecessary overhead. To remove this overhead, some database programming interfaces provide ways of using prepared statements. A **prepared statement** is a statement containing a series of placeholders. When you want to execute the statement, assign values to the placeholders, rather than prepare the entire statement over again.

Using prepared statements is useful when carrying out many similar actions, such as inserting many rows.

Generally, using prepared statements requires the following steps:

Prepare the statement

In this step, you generally provide the statement with some placeholder character instead of the values.

Repeatedly execute the prepared statement

In this step, you supply values to be used each time the statement is executed. The statement does not have to be prepared each time.

Drop the statement

In this step, you free the resources associated with the prepared statement. Some programming interfaces handle this step automatically.

Do Not Prepare Statements That Are Used Only Once

In general, do not prepare statements if they are only executed once. There is a slight performance penalty for separate preparation and execution, and it introduces unnecessary complexity into your application.

In some interfaces, however, you must prepare a statement to associate it with a cursor.

The calls for preparing and executing statements are not a part of SQL, and they differ from interface to interface. Each of the application programming interfaces provides a method for using prepared statements.

In this section:

Prepared Statements Overview [page 14]
The general procedure for using prepared statements is consistent, but the details vary from interface to interface.

Related Information

Cursor Usage [page 16]

1.2.2.1 Prepared Statements Overview

The general procedure for using prepared statements is consistent, but the details vary from interface to interface.

Comparing how to use prepared statements in different interfaces illustrates this point.

You typically perform the following tasks to use a prepared statement:

1. Prepare the statement.
2. Bind the parameters that will hold values in the statement.
3. Assign values to the bound parameters in the statement.
4. Execute the statement.
5. Repeat steps 3 and 4 as needed.
6. Drop the statement when finished. In JDBC, the Java garbage collection mechanism drops the statement.

Use a Prepared Statement in ADO.NET

You typically perform the following tasks to use a prepared statement in ADO.NET:

1. Create an SACommand object holding the statement:

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

2. Declare data types for any parameters in the statement.
   Use the SACommand.CreateParameter method.

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

3. Prepare the statement using the Prepare method.
4. Execute the statement:

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

For an example of preparing statements using ADO.NET, see the source code in *%SQLANYSAMP17%* `\SQLAnywhere\ADO.NET\SimpleWin32`.

Use a Prepared Statement in ODBC

You typically perform the following tasks to use a prepared statement in ODBC:

1. Prepare the statement using SQLPrepare.
2. Bind the statement parameters using SQLBindParameter.
3. Execute the statement using SQLExecute.
4. Drop the statement using SQLFreeStmt.

For an example of preparing statements using ODBC, see the source code in *%SQLANYSAMP17%* `\SQLAnywhere\ODBCPrepare`.

For more information about ODBC prepared statements, see the ODBC SDK documentation.

Use a Prepared Statement in JDBC

You typically perform the following tasks to use a prepared statement in JDBC:

1. Prepare the statement using the prepareStatement method of the connection object. This returns a prepared statement object.
2. Set the statement parameters using the appropriate *set*`Type` methods of the prepared statement object. Here, `Type` is the data type assigned.
3. Execute the statement using the appropriate method of the prepared statement object. For inserts, updates, and deletes this is the executeUpdate method.

For an example of preparing statements using JDBC, see the source code file *%SQLANYSAMP17%* `\SQLAnywhere\JDBC\JDBCExample.java`.

Use a Prepared Statement in Embedded SQL

You typically perform the following tasks to use a prepared statement in Embedded SQL:

1. Prepare the statement using the EXEC SQL PREPARE statement.
2. Assign values to the parameters in the statement.
3. Execute the statement using the EXEC SQL EXECUTE statement.
4. Free the resources associated with the statement using the EXEC SQL DROP statement.

Use a Prepared Statement in Open Client

You typically perform the following tasks to use a prepared statement in Open Client:

1. Prepare the statement using the ct_dynamic function, with a CS_PREPARE type parameter.
2. Set statement parameters using ct_param.
3. Execute the statement using ct_dynamic with a CS_EXECUTE type parameter.
4. Free the resources associated with the statement using ct_dynamic with a CS_DEALLOC type parameter.

Related Information

SQL in Open Client Applications [page 575]
How to Use Prepared Statements for More Efficient Access [page 247]
Executing Prepared Statements [page 180]
PREPARE Statement [ESQL]

1.2.3 Cursor Usage

When you execute a query in an application, the result set consists of several rows. In general, you do not know how many rows the application is going to receive before you execute the query.

Cursors provide a way of handling query result sets in applications. The way you use cursors and the kinds of cursors available to you depend on the programming interface you use.

Several system procedures are provided to help determine what cursors are in use for a connection, and what they contain:

- sa_list_cursors system procedure
- sa_describe_cursor system procedure

- sa_copy_cursor_to_temp_table system procedure

With cursors, you can perform the following tasks within any programming interface:

- Loop over the results of a query.
- Perform inserts, updates, and deletes on the underlying data at any point within a result set.

In addition, some programming interfaces allow you to use special features to tune the way result sets return to your application, providing substantial performance benefits for your application.

In this section:

Cursors [page 17]
A **cursor** is a name associated with a result set. The result set is obtained from a SELECT statement or stored procedure call.

Benefits of Using Cursors [page 18]
Although server-side cursors are not required in database applications, they do provide several benefits.

Related Information

Availability of Cursors [page 26]
sa_list_cursors System Procedure
sa_describe_cursor System Procedure
sa_copy_cursor_to_temp_table System Procedure

1.2.3.1 Cursors

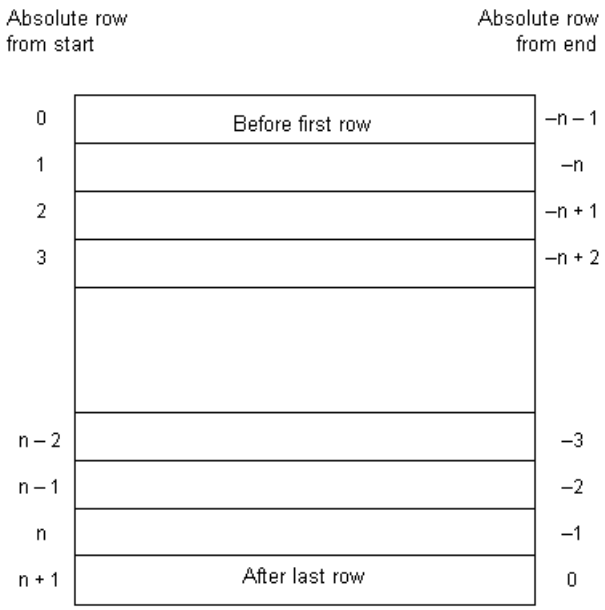
A **cursor** is a name associated with a result set. The result set is obtained from a SELECT statement or stored procedure call.

A cursor is a handle on the result set. At any time, the cursor has a well-defined position within the result set. With a cursor, you can examine and possibly manipulate the data one row at a time. With a cursor, you can move forward and backward through the query results.

Cursor Positions

Cursors can be positioned in the following places:

- Before the first row of the result set.
- On a row in the result set.
- After the last row of the result set.



The cursor position and result set are maintained in the database server. Rows are **fetched** by the client for display and processing either one at a time or a few at a time. The entire result set does not need to be delivered to the client.

1.2.3.2 Benefits of Using Cursors

Although server-side cursors are not required in database applications, they do provide several benefits.

A server-side cursor is preferable to a client-side cursor for the following reasons:

Response time

Server-side cursors do not require that the whole result set be assembled before the first row is fetched by the client. A client-side cursor requires that the entire result set be obtained and transferred to the client before the first row is fetched by the client.

Client-side memory

For large result sets, obtaining the entire result set on the client side can lead to demanding memory requirements.

Concurrency control

If you make updates to your data and do not use server-side cursors in your application, you must send separate SQL statements like UPDATE, INSERT, or DELETE to the database server to apply changes. This raises the possibility of concurrency problems if any corresponding rows in the database have changed since the result set was delivered to the client. As a consequence, updates by other clients may be lost.

Server-side cursors can act as pointers to the underlying data, permitting you to impose proper concurrency constraints on any changes made by the client by setting an appropriate isolation level.

1.2.4 Cursor Principles

To use a cursor, there are some basic steps to follow.

1. Prepare and execute a statement.
   Execute a statement using the usual method for the interface. You can prepare and then execute the statement, or you can execute the statement directly.
   With ADO.NET, only the SACommand.ExecuteReader method returns a cursor. It provides a read-only, forward-only cursor.
2. Test to see if the statement returns a result set.
   A cursor is implicitly opened when a statement that creates a result set is executed. When the cursor is opened, it is positioned before the first row of the result set.
3. Fetch results.
   Although simple fetch operations move the cursor to the next row in the result set, more complicated movement around the result set is also possible.
4. Close the cursor.
   When you have finished with the cursor, close it to free associated resources.
5. Free the statement.
   If you used a prepared statement, free it to reclaim memory.

The approach for using a cursor in Embedded SQL differs from the approach used in other interfaces. Follow these general steps to use a cursor in Embedded SQL:

1. Prepare a statement.
   Cursors generally use a statement handle rather than a string. Prepare a statement to have a handle available.
2. Declare the cursor.
   Each cursor refers to a single SELECT or CALL statement. When you declare a cursor, you state the name of the cursor and the statement it refers to.
3. Open the cursor.
   For a CALL statement, opening the cursor executes the procedure up to the point where the first row is about to be obtained.
4. Fetch results.
   Although simple fetch operations move the cursor to the next row in the result set, more complicated movement around the result set is also possible. How you declare the cursor determines which fetch operations are available to you.
5. Close the cursor.
   When you have finished with the cursor, close it. This frees any resources associated with the cursor.
6. Drop the statement.
   To free the memory associated with the statement, you must drop the statement.

In this section:

Cursor Positioning [page 20]
When a cursor is opened, it is positioned before the first row. You can move the cursor position to an absolute position from the start or the end of the query results, or to a position relative to the current cursor position.

Cursor Behavior When Opening Cursors [page 21]
You can configure the isolation level and duration of a cursor when you open it.

Row Fetching Through a Cursor [page 21]

The simplest way of processing the result set of a query using a cursor is to loop through all the rows of the result set until there are no more rows.

Multiple-row Fetching [page 22]
Multiple-row fetching should not be confused with prefetching rows. Multiple row fetching is performed by the application, while prefetching is transparent to the application, and provides a similar performance gain.

Scrollable Cursors [page 23]
ODBC and Embedded SQL provide methods for using scrollable cursors and dynamic scrollable cursors. These methods allow you to move several rows forward at a time, or to move backward through the result set.

Cursors Used to Modify Rows [page 23]
Cursors can do more than just read result sets from a query. You can also modify data in the database while processing a cursor.

Updatable Statements [page 24]
Clauses in the SELECT statement can affect updatable statements and cursors in various ways.

Cursor Operations That Are Canceled [page 25]
You can cancel a request through an interface function.

Related Information

Prepared Statements [page 13]
How to Fetch Data Using Embedded SQL [page 311]
DECLARE CURSOR Statement [ESQL] [SP]
OPEN Statement [ESQL] [SP]
FETCH Statement [ESQL] [SP]
CLOSE statement [ESQL] [SP]
DROP STATEMENT Statement [ESQL]

1.2.4.1 Cursor Positioning

When a cursor is opened, it is positioned before the first row. You can move the cursor position to an absolute position from the start or the end of the query results, or to a position relative to the current cursor position.

The specifics of how you change cursor position, and what operations are possible, are governed by the programming interface.

The number of row positions you can fetch in a cursor is governed by the size of an integer. You can fetch rows numbered up to number 2147483646, which is one less than the value that can be held in an integer. When using negative numbers (rows from the end) you can fetch down to one more than the largest negative value that can be held in an integer.

You can use special positioned update and delete operations to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, an error is returned indicating that there is no corresponding cursor row.

> **Note**
>
> Inserts and some updates to asensitive cursors can cause problems with cursor positioning. Inserted rows are not put at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear at all until the cursor is closed and opened again. This occurs if a work table had to be created to open the cursor.
>
> The UPDATE statement may cause a row to move in the cursor. This happens if the cursor has an ORDER BY clause that uses an existing index (a work table is not created). Using STATIC SCROLL cursors alleviates these problems but requires more memory and processing.

1.2.4.2 Cursor Behavior When Opening Cursors

You can configure the isolation level and duration of a cursor when you open it.

Isolation level

You can explicitly set the isolation level of operations on a cursor to be different from the current isolation level of the transaction. To do this, set the isolation_level option.

Duration

By default, cursors in Embedded SQL close at the end of a transaction. Opening a cursor WITH HOLD allows you to keep it open until the end of a connection, or until you explicitly close it. ADO.NET, ODBC, JDBC, and Open Client leave cursors open at the end of transactions by default.

Related Information

isolation_level Option

1.2.4.3 Row Fetching Through a Cursor

The simplest way of processing the result set of a query using a cursor is to loop through all the rows of the result set until there are no more rows.

You can accomplish this task by performing these steps:

1. Declare and open the cursor (Embedded SQL), or execute a statement that returns a result set (ODBC, JDBC, Open Client) or SADataReader object (ADO.NET).
2. Continue to fetch the next row until you get a `Row Not Found` error.
3. Close the cursor.

The technique used to fetch the next row is dependent on the interface you use. For example:

ADO.NET

Use the SADataReader.Read method.

ODBC

SQLFetch, SQLExtendedFetch, or SQLFetchScroll advances the cursor to the next row and returns the data.

JDBC

The next method of the ResultSet object advances the cursor and returns the data.

Embedded SQL

The FETCH statement carries out the same operation.

Open Client

The ct_fetch function advances the cursor to the next row and returns the data.

Related Information

Result Sets in ODBC Applications [page 186]
How to Return Result Sets from Java [page 251]
Cursors in Embedded SQL [page 312]
Open Client Cursor Management [page 577]

1.2.4.4 Multiple-row Fetching

Multiple-row fetching should not be confused with prefetching rows. Multiple row fetching is performed by the application, while prefetching is transparent to the application, and provides a similar performance gain.

Fetching multiple rows at a time can improve performance.

Some interfaces provide methods for fetching more than one row at a time into the next several fields in an array. Generally, the fewer separate fetch operations you execute, the fewer individual requests the server must respond to, and the better the performance. A modified FETCH statement that retrieves multiple rows is also sometimes called a **wide fetch**. Cursors that use multiple-row fetches are sometimes called **block cursors** or **fat cursors**.

Using Multiple-Row Fetching

- In ODBC, you can set the number of rows that are returned on each call to SQLFetchScroll or SQLExtendedFetch by setting the SQL_ATTR_ROW_ARRAY_SIZE or SQL_ROWSET_SIZE attribute.
- In Embedded SQL, the FETCH statement uses an ARRAY clause to control the number of rows fetched at a time.
- Open Client and JDBC do not support multi-row fetches. They do use prefetching.

1.2.4.5 Scrollable Cursors

ODBC and Embedded SQL provide methods for using scrollable cursors and dynamic scrollable cursors. These methods allow you to move several rows forward at a time, or to move backward through the result set.

The JDBC and Open Client interfaces do not support scrollable cursors.

Prefetching does not apply to scrollable operations. For example, fetching a row in the reverse direction does not prefetch several previous rows.

1.2.4.6 Cursors Used to Modify Rows

Cursors can do more than just read result sets from a query. You can also modify data in the database while processing a cursor.

These operations are commonly called **positioned** insert, update, and delete operations, or PUT operations if the action is an insert.

Not all query result sets allow positioned updates and deletes. If you perform a query on a non-updatable view, then no changes occur to the underlying tables. Also, if the query involves a join, then you must specify which table you want to delete from, or which columns you want to update, when you perform the operations.

Inserts through a cursor can only be executed if any non-inserted columns in the table allow NULL or have defaults.

If multiple rows are inserted into a value-sensitive (keyset driven) cursor, they appear at the end of the cursor result set. The rows appear at the end, even if they do not match the WHERE clause of the query or if an ORDER BY clause would normally have placed them at another location in the result set. This behavior is independent of programming interface. For example, it applies when using the Embedded SQL PUT statement or the ODBC SQLBulkOperations function. The value of an AUTOINCREMENT column for the most recent row inserted can be found by selecting the last row in the cursor. For example, in Embedded SQL the value could be obtained using `FETCH ABSOLUTE -1 cursor-name`. As a result of this behavior, the first multiple-row insert for a value-sensitive cursor may be expensive.

ODBC, JDBC, Embedded SQL, and Open Client permit data manipulation using cursors, but ADO.NET does not. With Open Client, you can delete and update rows, but you can only insert rows on a single-table query.

Which Table Are Rows Deleted From?

If you attempt a positioned delete through a cursor, the table from which rows are deleted is determined as follows:

1. If no FROM clause is included in the DELETE statement, the cursor must be on a single table only.
2. If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.
3. If a FROM clause is included, and no table owner is specified, the table-spec value is first matched against any correlation names.

4. If a correlation name exists, the table-spec value is identified with the correlation name.
5. If a correlation name does not exist, the table-spec value must be unambiguously identifiable as a table name in the cursor.
6. If a FROM clause is included, and a table owner is specified, the table-spec value must be unambiguously identifiable as a table name in the cursor.
7. The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

Related Information

FROM Clause

1.2.4.7 Updatable Statements

Clauses in the SELECT statement can affect updatable statements and cursors in various ways.

Updatability of Read-Only Statements

Specifying FOR READ ONLY in the cursor declaration, or including a FOR READ ONLY clause in the statement, renders the statement read-only. In other words, a FOR READ ONLY clause, or the appropriate read-only cursor declaration when using a client API, overrides any other updatability specification.

If the outermost block of a SELECT statement contains an ORDER BY clause, and the statement does not specify FOR UPDATE, then the cursor is read-only. If the SQL SELECT statement specifies FOR XML, then the cursor is read-only. Otherwise, the cursor is updatable.

Updatable Statements and Concurrency Control

For updatable statements, both optimistic and pessimistic concurrency control mechanisms on cursors are provided to ensure that a result set stays consistent during scrolling operations. These mechanisms are alternatives to using INSENSITIVE cursors or snapshot isolation, although they have different semantics and tradeoffs.

The specification of FOR UPDATE can affect whether a cursor is updatable. However, the FOR UPDATE syntax has no other effect on concurrency control. If FOR UPDATE is specified with additional parameters, the processing of the statement is altered to incorporate one of two concurrency control options as follows:

Pessimistic

For all rows fetched in the cursor's result set, the database server acquires intent row locks to prevent the rows from being updated by any other transaction.

Optimistic

The cursor type used by the database server is changed to a keyset-driven cursor (insensitive row membership, value-sensitive) so that the application can be informed when a row in the result has been modified or deleted by this, or any other transaction.

Pessimistic or optimistic concurrency is specified at the cursor level either through options with DECLARE CURSOR or FOR statements, or though the concurrency setting API for a specific programming interface. If a statement is updatable and the cursor does not specify a concurrency control mechanism, the statement's specification is used. The syntax is as follows:

FOR UPDATE BY LOCK

The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until transaction COMMIT or ROLLBACK.

FOR UPDATE BY { VALUES | TIMESTAMP }

The database server uses a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

Restricting Updatable Statements

FOR UPDATE (`column-list`) enforces the restriction that only named result set attributes can be modified in a subsequent UPDATE WHERE CURRENT OF statement.

Related Information

DECLARE Statement
FOR Statement

1.2.4.8 Cursor Operations That Are Canceled

You can cancel a request through an interface function.

If you cancel a request that is carrying out a cursor operation, the position of the cursor is indeterminate. After canceling the request, you must locate the cursor by its absolute position, or close it.

1.2.5 Cursor Types

The various programming interfaces do not support all aspects of database cursors. The terminology may also be different. The mappings between cursors and the options available to for each programming interface is explained in the following material.

In this section:

Availability of Cursors [page 26]
Not all interfaces provide support for all types of cursors.

Cursor Properties [page 27]
You request a cursor type, either explicitly or implicitly, from the programming interface. Different interface libraries offer different choices of cursor types.

Bookmarks and Cursors [page 27]
ODBC provides **bookmarks**, or values, used to identify rows in a cursor.

Block Cursors [page 27]
ODBC provides a cursor type called a block cursor. When you use a BLOCK cursor, you can use SQLFetchScroll or SQLExtendedFetch to fetch a block of rows, rather than a single row.

Related Information

Cursor Attributes [page 28]

1.2.5.1 Availability of Cursors

Not all interfaces provide support for all types of cursors.

- ADO.NET supports only forward-only, read-only cursors.
- ADO/OLE DB and ODBC support all types of cursors.
- Embedded SQL supports all types of cursors.
- For JDBC:
  - The SQL Anywhere JDBC driver permits the declaration of insensitive, sensitive, and forward-only asensitive cursors.
  - jConnect supports the declaration of insensitive, sensitive, and forward-only asensitive cursors in the same manner as the SQL Anywhere JDBC driver driver. However, the underlying implementation of jConnect only supports asensitive cursor semantics.
- SAP Open Client supports only asensitive cursors. Also, a severe performance penalty results when using updatable, non-unique cursors.

Related Information

Result Sets in ODBC Applications [page 186]
Requests for Cursors [page 43]

1.2.5.2 Cursor Properties

You request a cursor type, either explicitly or implicitly, from the programming interface. Different interface libraries offer different choices of cursor types.

For example, JDBC and ODBC specify different cursor types.

Each cursor type is defined by several characteristics:

Uniqueness

Declaring a cursor to be unique forces the query to return all the columns required to uniquely identify each row. Often this means returning all the columns in the primary key. Any columns required but not specified are added to the result set. The default cursor type is non-unique.

Updatability

A cursor declared as read-only cannot be used in a positioned update or delete operation. The default cursor type is updatable.

Scrollability

You can declare cursors to behave different ways as you move through the result set. Some cursors can fetch only the current row or the following row. Others can move backward and forward through the result set.

Sensitivity

Changes to the database may or may not be visible through a cursor.

These characteristics may have significant side effects on performance and on database server memory usage.

Cursors with a variety of mixes of these characteristics are possible. When you request a cursor of a given type, a match to those characteristics is attempted.

There are some occasions when not all characteristics can be supplied. For example, insensitive cursors must be read-only. If your application requests an updatable insensitive cursor, a different cursor type (value-sensitive) is supplied instead.

1.2.5.3 Bookmarks and Cursors

ODBC provides **bookmarks**, or values, used to identify rows in a cursor.

Bookmarks for value-sensitive and insensitive cursors are supported. For example, the ODBC cursor types SQL_CURSOR_STATIC and SQL_CURSOR_KEYSET_DRIVEN support bookmarks while cursor types SQL_CURSOR_DYNAMIC and SQL_CURSOR_FORWARD_ONLY do not.

1.2.5.4 Block Cursors

ODBC provides a cursor type called a block cursor. When you use a BLOCK cursor, you can use SQLFetchScroll or SQLExtendedFetch to fetch a block of rows, rather than a single row.

1.2.6 Cursor Attributes

Any cursor, once opened, has an associated result set. The cursor is kept open for a length of time. During that time, the result set associated with the cursor may be changed, either through the cursor itself or, subject to isolation level requirements, by other transactions.

Some cursors permit changes to the underlying data to be visible, while others do not reflect these changes. A sensitivity to changes to the underlying data causes different cursor behavior, or **cursor sensitivity**.

Cursors with a variety of sensitivity characteristics are provided.

Membership, Order, and Value Changes

Changes to the underlying data can affect the result set of a cursor in the following ways:

Membership

The set of rows in the result set, as identified by their primary key values.

Order

The order of the rows in the result set.

Value

The values of the rows in the result set.

For example, consider the following simple table with employee information (EmployeeID is the primary key column):

EmployeeID	Surname
1	Whitney
2	Cobb
3	Chin

A cursor on the following query returns all results from the table in primary key order:

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

The membership of the result set could be changed by adding a new row or deleting a row. The values could be changed by changing one of the names in the table. The order could be changed by changing the primary key value of one of the employees.

Visible and Invisible Changes

Subject to isolation level requirements, the membership, order, and values of the result set of a cursor can be changed after the cursor is opened. Depending on the type of cursor in use, the result set as seen by the application may or may not change to reflect these changes.

Changes to the underlying data may be **visible** or **invisible** through the cursor. A visible change is a change that is reflected in the result set of the cursor. Changes to the underlying data that are not reflected in the result set seen by the cursor are invisible.

In this section:

Cursor Sensitivity [page 30]
Cursors are classified by their sensitivity to changes in the underlying data. In other words, cursor sensitivity is defined by the changes that are visible.

Cursor Sensitivity Example: A Deleted Row [page 30]
This example uses a simple query to illustrate how different cursors respond to a row in the result set being deleted.

Cursor Sensitivity Example: An Updated Row [page 32]
This example uses a simple query to illustrate how different cursor types respond to a row in the result set being updated in such a way that the order of the result set is changed.

Insensitive Cursors [page 33]
These cursors have insensitive membership, order, and values. No changes made after cursor open time are visible.

Sensitive Cursors [page 34]
Sensitive cursors can be used for read-only or updatable cursor types.

Asensitive Cursors [page 36]
These cursors do not have well-defined sensitivity in their membership, order, or values. The flexibility that is allowed in the sensitivity permits asensitive cursors to be optimized for performance.

Value-sensitive Cursors [page 37]
For value-sensitive cursors, membership is insensitive, and the order and value of the result set is sensitive.

Cursor Sensitivity and Performance [page 39]
There is a trade-off between performance and other cursor properties. In particular, making a cursor updatable places restrictions on the cursor query processing and delivery that constrain performance. Also, putting requirements on cursor sensitivity may constrain cursor performance.

Cursor Sensitivity and Isolation Levels [page 42]
Both cursor sensitivity and isolation levels address the problem of concurrency control, but in different ways, and with different sets of tradeoffs.

Requests for Cursors [page 43]
When you request a cursor type from your client application, a cursor is provided. Cursors are defined, not by the type as specified in the programming interface, but by the sensitivity of the result set to changes in the underlying data.

Related Information

Cursors [page 17]

1.2.6.1 Cursor Sensitivity

Cursors are classified by their sensitivity to changes in the underlying data. In other words, cursor sensitivity is defined by the changes that are visible.

Insensitive cursors

The result set is fixed when the cursor is opened. No changes to the underlying data are visible.

Sensitive cursors

The result set can change after the cursor is opened. All changes to the underlying data are visible.

Asensitive cursors

Changes may be reflected in the membership, order, or values of the result set seen through the cursor, or may not be reflected at all.

Value-sensitive cursors

Changes to the order or values of the underlying data are visible. The membership of the result set is fixed when the cursor is opened.

The differing requirements on cursors place different constraints on execution and therefore affect performance.

Related Information

Insensitive Cursors [page 33]
Sensitive Cursors [page 34]
Asensitive Cursors [page 36]
Value-sensitive Cursors [page 37]
Cursor Sensitivity and Performance [page 39]

1.2.6.2 Cursor Sensitivity Example: A Deleted Row

This example uses a simple query to illustrate how different cursors respond to a row in the result set being deleted.

Consider the following sequence of events:

1. An application opens a cursor on the following query against the sample database.

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

EmployeeID	Surname
102	Whitney

EmployeeID	Surname
105	Cobb
129	Chin
...	...

2. The application fetches the first row through the cursor (102).
3. The application fetches the next row through the cursor (105).
4. A separate transaction deletes employee 102 (Whitney) and commits the change.

The results of cursor actions in this situation depend on the cursor sensitivity:

Insensitive cursors

The DELETE is not reflected in either the membership or values of the results as seen through the cursor:

Action	Result
Fetch previous row	Returns the original copy of the row (102).
Fetch the first row (absolute fetch)	Returns the original copy of the row (102).
Fetch the second row (absolute fetch)	Returns the unchanged row (105).

Sensitive cursors

The membership of the result set has changed so that row 105 is now the first row in the result set:

Action	Result
Fetch previous row	Returns `Row Not Found`. There is no previous row.
Fetch the first row (absolute fetch)	Returns row 105.
Fetch the second row (absolute fetch)	Returns row 129.

Value-sensitive cursors

The membership of the result set is fixed, and so row 105 is still the second row of the result set. The DELETE is reflected in the values of the cursor, and creates an effective hole in the result set.

Action	Result
Fetch previous row	Returns `No current row of cursor`. There is a hole in the cursor where the first row used to be.
Fetch the first row (absolute fetch)	Returns `No current row of cursor`. There is a hole in the cursor where the first row used to be.
Fetch the second row (absolute fetch)	Returns row 105.

Asensitive cursors

For changes, the membership and values of the result set are indeterminate. The response to a fetch of the previous row, the first row, or the second row depends on the particular optimization method for the query, whether that method involved the formation of a work table, and whether the row being fetched was prefetched from the client.

The benefit of asensitive cursors is that for many applications, sensitivity is unimportant. In particular, if you are using a forward-only, read-only cursor, no underlying changes are seen. Also, if you are running at a high isolation level, underlying changes are disallowed.

1.2.6.3 Cursor Sensitivity Example: An Updated Row

This example uses a simple query to illustrate how different cursor types respond to a row in the result set being updated in such a way that the order of the result set is changed.

Consider the following sequence of events:

1. An application opens a cursor on the following query against the sample database.

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

EmployeeID	Surname
102	Whitney
105	Cobb
129	Chin
...	...

2. The application fetches the first row through the cursor (102).
3. The application fetches the next row through the cursor (105).
4. A separate transaction updates the employee ID of employee 102 (Whitney) to 165 and commits the change.

The results of the cursor actions in this situation depend on the cursor sensitivity:

Insensitive cursors

The UPDATE is not reflected in either the membership or values of the results as seen through the cursor:

Action	Result
Fetch previous row	Returns the original copy of the row (102).
Fetch the first row (absolute fetch)	Returns the original copy of the row (102).
Fetch the second row (absolute fetch)	Returns the unchanged row (105).

Sensitive cursors

The membership of the result set has changed so that row 105 is now the first row in the result set:

Action	Result
Fetch previous row	Returns SQLCODE 100. The membership of the result set has changed so that 105 is now the first row. The cursor is moved to the position before the first row.

Action	Result
Fetch the first row (absolute fetch)	Returns row 105.
Fetch the second row (absolute fetch)	Returns row 129.

In addition, a fetch on a sensitive cursor returns a SQLE_ROW_UPDATED_WARNING warning if the row has changed since the last reading. The warning is given only once. Subsequent fetches of the same row do not produce the warning.

Similarly, a positioned update or delete through the cursor on a row since it was last fetched returns the SQLE_ROW_UPDATED_SINCE_READ error. An application must fetch the row again for an update or delete on a sensitive cursor to work.

An update to any column causes the warning/error, even if the column is not referenced by the cursor. For example, a cursor on a query returning Surname would report the update even if only the Salary column was modified.

Value-sensitive cursors

The membership of the result set is fixed, and so row 105 is still the second row of the result set. The UPDATE is reflected in the values of the cursor, and creates an effective "hole" in the result set.

Action	Result
Fetch previous row	Returns SQLCODE 100. The membership of the result set has changed so that 105 is now the first row: The cursor is positioned on the hole: it is before row 105.
Fetch the first row (absolute fetch)	Returns SQLCODE -197. The membership of the result set has changed so that 105 is now the first row: The cursor is positioned on the hole: it is before row 105.
Fetch the second row (absolute fetch)	Returns row 105.

Asensitive cursors

For changes, the membership and values of the result set are indeterminate. The response to a fetch of the previous row, the first row, or the second row depends on the particular optimization method for the query, whether that method involved the formation of a work table, and whether the row being fetched was prefetched from the client.

> **Note**
>
> Update warning and error conditions do not occur in bulk operations mode (-b database server option).

1.2.6.4 Insensitive Cursors

These cursors have insensitive membership, order, and values. No changes made after cursor open time are visible.

Insensitive cursors are used only for read-only cursor types.

Standards

Insensitive cursors correspond to the ISO/ANSI standard definition of insensitive cursors, and to ODBC static cursors.

Programming Interfaces

Interface	Cursor Type	Comment
ODBC, ADO/OLE DB	Static	If an updatable static cursor is requested, a value-sensitive cursor is used instead.
Embedded SQL	INSENSITIVE	
JDBC	INSENSITIVE	
Open Client	Unsupported	

Description

Insensitive cursors always return rows that match the query's selection criteria, in the order specified by any ORDER BY clause.

The result set of an insensitive cursor is fully materialized as a work table when the cursor is opened. This has the following consequences:

- If the result set is very large, the disk space and memory requirements for managing the result set may be significant.
- No row is returned to the application before the entire result set is assembled as a work table. For complex queries, this may lead to a delay before the first row is returned to the application.
- Subsequent rows can be fetched directly from the work table, and so are returned quickly. The client library may prefetch several rows at a time, further improving performance.
- Insensitive cursors are not affected by ROLLBACK or ROLLBACK TO SAVEPOINT.

1.2.6.5 Sensitive Cursors

Sensitive cursors can be used for read-only or updatable cursor types.

These cursors have sensitive membership, order, and values.

Standards

Sensitive cursors correspond to the ISO/ANSI standard definition of sensitive cursors, and to ODBC dynamic cursors.

Programming Onterfaces

Interface	Cursor Type	Comment
ODBC, ADO/OLE DB	Dynamic	
Embedded SQL	SENSITIVE	Also supplied in response to a request for a DYNAMIC SCROLL cursor when no work table is required and the pre-fetch option is set to Off.
JDBC	SENSITIVE	Sensitive cursors are fully supported by the SQL Anywhere JDBC driver.

Description

Prefetching is disabled for sensitive cursors. All changes are visible through the cursor, including changes through the cursor and from other transactions. Higher isolation levels may hide some changes made in other transactions because of locking.

Changes to cursor membership, order, and all column values are all visible. For example, if a sensitive cursor contains a join, and one of the values of one of the underlying tables is modified, then all result rows composed from that base row show the new value. Result set membership and order may change at each fetch.

Sensitive cursors always return rows that match the query's selection criteria, and are in the order specified by any ORDER BY clause. Updates may affect the membership, order, and values of the result set.

The requirements of sensitive cursors place restrictions on the implementation of sensitive cursors:

- Rows cannot be prefetched, as changes to the prefetched rows would not be visible through the cursor. This may impact performance.
- Sensitive cursors must be implemented without any work tables being constructed, as changes to those rows stored as work tables would not be visible through the cursor.
- The no work table limitation restricts the choice of join method by the optimizer and therefore may impact performance.
- For some queries, the optimizer is unable to construct a plan that does not include a work table that would make a cursor sensitive.
  Work tables are commonly used for sorting and grouping intermediate results. A work table is not needed for sorting if the rows can be accessed through an index. It is not possible to state exactly which queries employ work tables, but the following queries do employ them:
  - UNION queries, although UNION ALL queries do not necessarily use work tables.
  - Statements with an ORDER BY clause, if there is no index on the ORDER BY column.

- Any query that is optimized using a hash join.
- Many queries involving DISTINCT or GROUP BY clauses.

In these cases, either an error is returned to the application, or the cursor type is changed to an asensitive cursor and a warning is returned.

1.2.6.6 Asensitive Cursors

These cursors do not have well-defined sensitivity in their membership, order, or values. The flexibility that is allowed in the sensitivity permits asensitive cursors to be optimized for performance.

Asensitive cursors are used only for read-only cursor types.

Standards

Asensitive cursors correspond to the ISO/ANSI standard definition of asensitive cursors, and to ODBC cursors with unspecific sensitivity.

Programming Interfaces

Interface	Cursor Type
ODBC, ADO/OLE DB	Unspecified sensitivity
Embedded SQL	DYNAMIC SCROLL

Description

A request for an asensitive cursor places few restrictions on the methods used to optimize the query and return rows to the application. For these reasons, asensitive cursors provide the best performance. In particular, the optimizer is free to employ any measure of materialization of intermediate results as work tables, and rows can be prefetched by the client.

There are no guarantees about the visibility of changes to base underlying rows. Some changes may be visible, others not. Membership and order may change at each fetch. In particular, updates to base rows may result in only some of the updated columns being reflected in the cursor's result.

Asensitive cursors do not guarantee to return rows that match the query's selection and order. The row membership is fixed at cursor open time, but subsequent changes to the underlying values are reflected in the results.

Asensitive cursors always return rows that matched the customer's WHERE and ORDER BY clauses at the time the cursor membership is established. If column values change after the cursor is opened, rows may be returned that no longer match WHERE and ORDER BY clauses.

1.2.6.7 Value-sensitive Cursors

For value-sensitive cursors, membership is insensitive, and the order and value of the result set is sensitive.

Value-sensitive cursors can be used for read-only or updatable cursor types.

Standards

Value-sensitive cursors do not correspond to an ISO/ANSI standard definition. They correspond to ODBC keyset-driven cursors.

Programming Interfaces

Interface	Cursor Type	Comment
ODBC, ADO/OLE DB	Keyset-driven	
Embedded SQL	SCROLL	
JDBC	INSENSITIVE and CONCUR_UPDATABLE	With the SQL Anywhere JDBC driver, a request for an updatable INSENSITIVE cursor is answered with a value-sensitive cursor.
Open Client and jConnect	Not supported	

Description

If the application fetches a row composed of a base underlying row that has changed, then the application must be presented with the updated value, and the SQL_ROW_UPDATED status must be issued to the application. If the application attempts to fetch a row that was composed of a base underlying row that was deleted, a SQL_ROW_DELETED status must be issued to the application.

Changes to primary key values remove the row from the result set (treated as a delete, followed by an insert). A special case occurs when a row in the result set is deleted (either from cursor or outside) and a new row with the same key value is inserted. This will result in the new row replacing the old row where it appeared.

There is no guarantee that rows in the result set match the query's selection or order specification. Since row membership is fixed at open time, subsequent changes that make a row not match the WHERE clause or ORDER BY do not change a row's membership nor position.

All values are sensitive to changes made through the cursor. The sensitivity of membership to changes made through the cursor is controlled by the ODBC option SQL_STATIC_SENSITIVITY. If this option is on, then inserts through the cursor add the row to the cursor. Otherwise, they are not part of the result set. Deletes through the cursor remove the row from the result set, preventing a hole returning the SQL_ROW_DELETED status.

Value-sensitive cursors use a **key set table**. When the cursor is opened, a work table is populated with identifying information for each row contributing to the result set. When scrolling through the result set, the key set table is used to identify the membership of the result set, but values are obtained, if necessary, from the underlying tables.

The fixed membership property of value-sensitive cursors allows your application to remember row positions within a cursor and be assured that these positions will not change.

- If a row was updated or may have been updated since the cursor was opened, a SQLE_ROW_UPDATED_WARNING is returned when the row is fetched. The warning is generated only once: fetching the same row again does not produce the warning.
  An update to any column of the row causes the warning, even if the updated column is not referenced by the cursor. For example, a cursor on Surname and GivenName would report the update even if only the Birthdate column was modified. These update warning and error conditions do not occur in bulk operations mode (-b database server option) when row locking is disabled.
- An attempt to execute a positioned update or delete on a row that has been modified since it was last fetched returns a SQLE_ROW_UPDATED_SINCE_READ error and cancels the statement. An application must FETCH the row again before the UPDATE or DELETE is permitted.
  An update to any column of the row causes the error, even if the updated column is not referenced by the cursor. The error does not occur in bulk operations mode.
- If a row has been deleted after the cursor is opened, either through the cursor or from another transaction, a **hole** is created in the cursor. The membership of the cursor is fixed, so a row position is reserved, but the DELETE operation is reflected in the changed value of the row. If you fetch the row at this hole, you receive a -197 SQLCODE error, indicating that there is no current row, and the cursor is left positioned on the hole. You can avoid holes by using sensitive cursors, as their membership changes along with the values.

Rows cannot be prefetched for value-sensitive cursors. This requirement may affect performance.

Inserting Multiple Rows

When inserting multiple rows through a value-sensitive cursor, the new rows appear at the end of the result set.

Related Information

Performance Aspects of Bulk Operations
Cursors Used to Modify Rows [page 23]
Cursor Sensitivity Example: A Deleted Row [page 30]

1.2.6.8 Cursor Sensitivity and Performance

There is a trade-off between performance and other cursor properties. In particular, making a cursor updatable places restrictions on the cursor query processing and delivery that constrain performance. Also, putting requirements on cursor sensitivity may constrain cursor performance.

To understand how the updatability and sensitivity of cursors affects performance, you must understand how the results that are visible through a cursor are transmitted from the database to the client application.

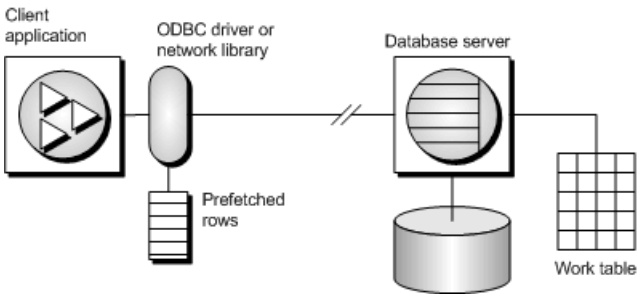
In particular, results may be stored at two intermediate locations for performance reasons:

Work tables

Either intermediate or final results may be stored as work tables. Value-sensitive cursors employ a work table of primary key values. Query characteristics may also lead the optimizer to use work tables in its chosen execution plan.

Prefetching

The client side of the communication may retrieve rows into a buffer on the client side to avoid separate requests to the database server for each row.



Sensitivity and updatability limit the use of intermediate locations.

In this section:

Prefetches [page 39]
Prefetching assists performance by cutting down on client/server round trips, and increases throughput by making many rows available without a separate request to the server for each row or block of rows.

Lost Updates [page 41]
When using an updatable cursor, it is important to guard against lost updates.

1.2.6.8.1 Prefetches

Prefetching assists performance by cutting down on client/server round trips, and increases throughput by making many rows available without a separate request to the server for each row or block of rows.

Prefetches and multiple-row fetches are different. Prefetches can be carried out without explicit instructions from the client application. Prefetching retrieves rows from the server into a buffer on the client side, but does not make those rows available to the client application until the application fetches the appropriate row.

By default, the client library prefetches multiple rows whenever an application fetches a single row. The client library stores the additional rows in a buffer.

Controlling Prefetching from an Application

- The prefetch option controls whether prefetching occurs. You can set the prefetch option to Always, Conditional, or Off for a single connection. By default, it is set to Conditional.
- In Embedded SQL, you can control prefetching on a per-cursor basis when you open a cursor on an individual FETCH operation using the BLOCK clause.
  The application can specify a maximum number of rows contained in a single fetch from the server by specifying the BLOCK clause. For example, if you are fetching and displaying 5 rows at a time, you could use BLOCK 5. Specifying BLOCK 0 fetches 1 record at a time and also causes a FETCH RELATIVE 0 to always fetch the row from the server again.
  Although you can also turn off prefetch by setting a connection parameter on the application, it is more efficient to specify BLOCK 0 than to set the prefetch option to Off.
- Prefetch is disabled by default for value sensitive cursor types.
- In Open Client, you can control prefetching behavior using ct_cursor with CS_CURSOR_ROWS after the cursor is declared, but before it is opened.

Prefetch dynamically increases the number of prefetch rows when improvements in performance could be achieved. This includes cursors that meet the following conditions:

- They use one of the supported cursor types:

  ODBC and OLE DB

  FORWARD-ONLY and READ-ONLY (default) cursors

  Embedded SQL

  DYNAMIC SCROLL (default), NO SCROLL, and INSENSITIVE cursors

  ADO.NET

  all cursors
- They perform only FETCH NEXT operations (no absolute, relative, or backward fetching).
- The application does not change the host variable type between fetches and does not use a GET DATA statement to get column data in chunks (using *one* GET DATA statement to get the value is supported).

Related Information

prefetch Option

1.2.6.8.2 Lost Updates

When using an updatable cursor, it is important to guard against lost updates.

A lost update is a scenario in which two or more transactions update the same row, but neither transaction is aware of the modification made by the other transaction, and the second change overwrites the first modification.

The following example illustrates this problem:

1. An application opens a cursor on the following query against the sample database.

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

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

2. The application fetches the row with ID = 300 through the cursor.
3. A separate transaction updates the row using the following statement:

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

4. The application then updates the row through the cursor to a value of `(Quantity - 5)`.
5. The correct final value for the row would be 13. If the cursor had prefetched the row, the new value of the row would be 23. The update from the separate transaction is lost.

In a database application, the potential for a lost update exists at any isolation level if changes are made to rows without verification of their values beforehand. At higher isolation levels (2 and 3), locking (read, intent, and write locks) can be used to ensure that changes to rows cannot be made by another transaction once the row has been read by the application. However, at isolation levels 0 and 1, the potential for lost updates is greater: at isolation level 0, read locks are not acquired to prevent subsequent changes to the data, and isolation level 1 only locks the current row. Lost updates cannot occur when using snapshot isolation since any attempt to change an old value results in an update conflict. Also, the use of prefetching at isolation level 1 can also introduce the potential for lost updates, since the result set row that the application is positioned on, which is in the client's prefetch buffer, may not be the same as the current row that the server is positioned on in the cursor.

To prevent lost updates from occurring with cursors at isolation level 1, the database server supports three different concurrency control mechanisms that can be specified by an application:

1. The acquisition of intent row locks on each row in the cursor as it is fetched. Intent locks prevent other transactions from acquiring intent or write locks on the same row, preventing simultaneous updates. However, intent locks do not block read row locks, so they do not affect the concurrency of read-only statements.
2. The use of a value-sensitive cursor. Value-sensitive cursors can be used to track when an underlying row has changed, or has been deleted, so that the application can respond.

3. The use of FETCH FOR UPDATE, which acquires an intent row lock for that specific row.

How these alternatives are specified depends on the interface used by the application. For the first two alternatives that pertain to a SELECT statement:

- In ODBC, lost updates cannot occur because the application must specify a cursor concurrency parameter to the SQLSetStmtAttr function when declaring an updatable cursor. This parameter is one of SQL_CONCUR_LOCK, SQL_CONCUR_VALUES, SQL_CONCUR_READ_ONLY, or SQL_CONCUR_TIMESTAMP. For SQL_CONCUR_LOCK, the database server acquires row intent locks. For SQL_CONCUR_VALUES and SQL_CONCUR_TIMESTAMP, a value-sensitive cursor is used. SQL_CONCUR_READ_ONLY is used for read-only cursors, and is the default.
- In JDBC, the concurrency setting for a statement is similar to that of ODBC. The JDBC driver supports the JDBC concurrency values RESULTSET_CONCUR_READ_ONLY and RESULTSET_CONCUR_UPDATABLE. The first value corresponds to the ODBC concurrency setting SQL_CONCUR_READ_ONLY and specifies a read-only statement. The second value corresponds to the ODBC SQL_CONCUR_LOCK setting, so row intent locks are used to prevent lost updates. Value-sensitive cursors cannot be specified directly in the JDBC driver.
- In jConnect, updatable cursors are supported at the API level, but the underlying implementation (using TDS) does not support updates through a cursor. Instead, jConnect sends a separate UPDATE statement to the database server to update the specific row. To avoid lost updates, the application must run at isolation level 2 or higher. Alternatively, the application can issue separate UPDATE statements from the cursor, but you must ensure that the UPDATE statement verifies that the row values have not been altered since the row was read by placing appropriate conditions in the UPDATE statement's WHERE clause.
- In Embedded SQL, a concurrency specification can be set by including syntax within the SELECT statement itself, or in the cursor declaration. In the SELECT statement, the syntax SELECT...FOR UPDATE BY LOCK causes the database server to acquire intent row locks on the result set. Alternatively, SELECT...FOR UPDATE BY [VALUES | TIMESTAMP] causes the database server to change the cursor type to a value-sensitive cursor, so that if a specific row has been changed since the row was last read through the cursor, the application receives either a warning (SQLE_ROW_UPDATED_WARNING) on a FETCH statement, or an error (SQLE_ROW_UPDATED_SINCE_READ) on an UPDATE WHERE CURRENT OF statement. If the row was deleted, the application also receives an error (SQLE_NO_CURRENT_ROW).

FETCH FOR UPDATE functionality is also supported by the Embedded SQL and ODBC interfaces, although the details differ depending on the API that is used.

In Embedded SQL, the application uses FETCH FOR UPDATE, rather than FETCH, to cause an intent lock to be acquired on the row. In ODBC, the application uses the API call SQLSetPos with the operation argument SQL_POSITION or SQL_REFRESH, and the lock type argument SQL_LOCK_EXCLUSIVE, to acquire an intent lock on a row. These are long-term locks that are held until the transaction is committed or rolled back.

1.2.6.9 Cursor Sensitivity and Isolation Levels

Both cursor sensitivity and isolation levels address the problem of concurrency control, but in different ways, and with different sets of tradeoffs.

By choosing an isolation level for a transaction (typically at the connection level), you determine the type and locks to place, and when, on rows in the database. Locks prevent other transactions from accessing or modifying rows in the database. In general, the greater the number of locks held, the lower the expected level of concurrency across concurrent transactions.

However, locks do not prevent updates from other portions of the same transaction from occurring. So, a single transaction that maintains multiple updatable cursors cannot rely on locking to prevent such problems as lost updates.

Snapshot isolation is intended to eliminate the need for read locks by ensuring that each transaction sees a consistent view of the database. The obvious advantage is that a consistent view of the database can be queried without relying on fully serializable transactions (isolation level 3), and the loss of concurrency that comes with using isolation level 3. However, snapshot isolation comes with a significant cost because copies of modified rows must be maintained to satisfy the requirements of both concurrent snapshot transactions already executing, and snapshot transactions that have yet to start. Because of this copy maintenance, the use of snapshot isolation may be inappropriate for heavy-update workloads.

Cursor sensitivity, however, determines which changes are visible (or not) to the cursor's result. Because cursor sensitivity is specified on a cursor basis, cursor sensitivity applies to both the effects of other transactions and to update activity of the same transaction, although these effects depend entirely on the cursor type specified. By setting cursor sensitivity, you are not directly determining when locks are placed on rows in the database. However, it is the combination of cursor sensitivity and isolation level that controls the various concurrency scenarios that are possible with a particular application.

Related Information

Choosing a Snapshot Isolation Level

1.2.6.10 Requests for Cursors

When you request a cursor type from your client application, a cursor is provided. Cursors are defined, not by the type as specified in the programming interface, but by the sensitivity of the result set to changes in the underlying data.

Depending on the cursor type you ask for, a cursor is provided with behavior to match the type.

Cursor sensitivity is set in response to the client cursor type request.

In this section:

Cursor Requests in ADO.NET [page 44]
Forward-only, read-only cursors are available by using SACommand.ExecuteReader. The SADataAdapter object uses a client-side result set instead of cursors.

Cursor Requests in ADO/OLE DB and ODBC [page 44]
The following table illustrates the cursor sensitivity that is set in response to different ODBC scrollable cursor types.

Cursor Requests in JDBC [page 45]
The SQL Anywhere JDBC driver supports three types of cursors: insensitive, sensitive, and forward-only asensitive.

Cursor Requests in Embedded SQL [page 45]

To request a cursor from an Embedded SQL application, you specify the cursor type on the DECLARE statement. The following table illustrates the cursor sensitivity that is set in response to different requests:

Cursor Requests in Open Client [page 46]

The underlying protocol (TDS) for Open Client supports only forward-only, read-only, asensitive cursors.

1.2.6.10.1 Cursor Requests in ADO.NET

Forward-only, read-only cursors are available by using SACommand.ExecuteReader. The SADataAdapter object uses a client-side result set instead of cursors.

1.2.6.10.2 Cursor Requests in ADO/OLE DB and ODBC

The following table illustrates the cursor sensitivity that is set in response to different ODBC scrollable cursor types.

ODBC Scrollable Cursor Type	Cursor Sensitivity
STATIC	Insensitive
KEYSET-DRIVEN	Value-sensitive
DYNAMIC	Sensitive
MIXED	Value-sensitive

A MIXED cursor is obtained by setting the cursor type to SQL_CURSOR_KEYSET_DRIVEN, and then specifying the number of rows in the keyset for a keyset-driven cursor using SQL_ATTR_KEYSET_SIZE. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside the keyset). The default keyset size is 0. It is an error if the keyset size is greater than 0 and less than the rowset size (SQL_ATTR_ROW_ARRAY_SIZE).

ODBC cursor characteristics help you decide the curser type to request.

Exceptions

If a STATIC cursor is requested as updatable, a value-sensitive cursor is supplied instead and a warning is issued.

If a DYNAMIC or MIXED cursor is requested and the query cannot be executed without using work tables, a warning is issued and an asensitive cursor is supplied instead.

Related Information

Cursor Attributes [page 28]
ODBC Cursor Characteristics [page 188]

1.2.6.10.3 Cursor Requests in JDBC

The SQL Anywhere JDBC driver supports three types of cursors: insensitive, sensitive, and forward-only asensitive.

The SQL Anywhere JDBC driver supports these different cursor types for a JDBC ResultSet object. However, there are cases when the database server cannot construct an access plan with the required semantics for a given cursor type. In these cases, the database server either returns an error or substitutes a different cursor type.

Related Information

Sensitive Cursors [page 34]

1.2.6.10.4 Cursor Requests in Embedded SQL

To request a cursor from an Embedded SQL application, you specify the cursor type on the DECLARE statement. The following table illustrates the cursor sensitivity that is set in response to different requests:

Cursor Type	Cursor Sensitivity
NO SCROLL	Asensitive
DYNAMIC SCROLL	Asensitive
SCROLL	Value-sensitive
INSENSITIVE	Insensitive
SENSITIVE	Sensitive

Exceptions

If a DYNAMIC SCROLL or NO SCROLL cursor is requested as UPDATABLE, then a sensitive or value-sensitive cursor is supplied. It is not guaranteed which of the two is supplied. This uncertainty fits the definition of asensitive behavior.

If an INSENSITIVE cursor is requested as UPDATABLE, then a value-sensitive cursor is supplied.

If a DYNAMIC SCROLL cursor is requested, if the prefetch database option is set to Off, and if the query execution plan involves no work tables, then a sensitive cursor may be supplied. Again, this uncertainty fits the definition of asensitive behavior.

1.2.6.10.5 Cursor Requests in Open Client

The underlying protocol (TDS) for Open Client supports only forward-only, read-only, asensitive cursors.

1.2.7 Result Set Descriptors

Some applications build SQL statements that cannot be completely specified in the application. Sometimes statements are dependent on a user response before the application knows exactly what information to retrieve, such as when a reporting application allows a user to select which columns to display.

In such a case, the application needs a method for retrieving information about both the nature of the **result set** and the contents of the result set. The information about the nature of the result set, called a **descriptor**, identifies the data structure, including the number and type of columns expected to be returned. Once the application has determined the nature of the result set, retrieving the contents is straightforward.

This **result set metadata** (information about the nature and content of the data) is manipulated using descriptors. Obtaining and managing the result set metadata is called **describing**.

Since cursors generally produce result sets, descriptors and cursors are closely linked, although some interfaces hide the use of descriptors from the user. Typically, statements needing descriptors are either SELECT statements or stored procedures that return result sets.

A sequence for using a descriptor with a cursor-based operation is as follows:

1. Allocate the descriptor. This may be done implicitly, although some interfaces allow explicit allocation as well.
2. Prepare the statement.
3. Describe the statement. If the statement is a stored procedure call or batch, and the result set is not described by a RESULT clause in the procedure definition, then the describe should occur after opening the cursor.
4. Declare and open a cursor for the statement (Embedded SQL) or execute the statement.
5. Get the descriptor and modify the allocated area if necessary. This is often done implicitly.
6. Fetch and process the statement results.
7. Deallocate the descriptor.
8. Close the cursor.
9. Drop the statement. Some interfaces do this automatically.

Implementation Notes

- In Embedded SQL, a SQLDA (SQL Descriptor Area) structure holds the descriptor information.
- In ODBC, a descriptor handle allocated using SQLAllocHandle provides access to the fields of a descriptor. You can manipulate these fields using SQLSetDescRec, SQLSetDescField, SQLGetDescRec, and SQLGetDescField.
  Alternatively, you can use SQLDescribeCol, SQLColAttribute, or SQLColAttributes (ODBC 2.0) to obtain column information.
- In Open Client, you can use ct_dynamic to prepare a statement and ct_describe to describe the result set of the statement. However, you can also use ct_command to send a SQL statement without preparing it first and use ct_results to handle the returned rows one by one. This is the more common way of operating in Open Client application development.
- In JDBC, the java.sql.ResultSetMetaData class provides information about result sets.
- You can also use descriptors for sending data to the database server (for example, with the INSERT statement); however, this is a different kind of descriptor than for result sets.

Related Information

The SQL Descriptor Area (SQLDA) [page 300]
DESCRIBE Statement [ESQL]

1.2.8 Transactions in Applications

Transactions are sets of atomic SQL statements. Either all statements in the transaction are executed, or none.

In this section:

Autocommit and Manual Commit Mode [page 48]
Database programming interfaces can operate in either **manual commit** mode or **autocommit** mode.

Isolation Level Settings [page 52]
You can set the isolation level of a current connection using the isolation_level database option.

Cursors and Transactions [page 52]
In general, a cursor closes when a COMMIT is performed.

Related Information

Transactions and Isolation Levels

1.2.8.1 Autocommit and Manual Commit Mode

Database programming interfaces can operate in either **manual commit** mode or **autocommit** mode.

Manual commit mode

Operations are committed only when your application carries out an explicit commit operation or when the database server carries out an automatic commit, for example when executing an ALTER TABLE statement or other data definition statement. Manual commit mode is also sometimes called **chained mode**.

To use transactions in your application, including nested transactions and savepoints, you must operate in manual commit mode.

Autocommit mode

Each statement is treated as a separate transaction. Autocommit mode is equivalent to appending a COMMIT statement to the end of each of your SQL statements. Autocommit mode is also sometimes called **unchained mode**.

Autocommit mode can affect the performance and behavior of your application. Do not use autocommit if your application requires transactional integrity.

In this section:

How to Control Autocommit Behavior [page 48]
The way to control the commit behavior of your application depends on the programming interface you are using. The implementation of autocommit may be client-side or server-side, depending on the interface.

Autocommit Implementation Details [page 51]
Autocommit mode has slightly different behavior depending on the interface and provider that you are using and how you control the autocommit behavior.

Related Information

Tip: Turn Off Autocommit Mode

1.2.8.1.1 How to Control Autocommit Behavior

The way to control the commit behavior of your application depends on the programming interface you are using. The implementation of autocommit may be client-side or server-side, depending on the interface.

Control Autocommit Mode (ADO.NET)

By default, the ADO.NET data provider operates in autocommit mode. To use explicit transactions, use the SAConnection.BeginTransaction method. Automatic commit is handled on the client side by the provider.

Control Autocommit Mode (Embedded SQL)

By default, Embedded SQL applications operate in manual commit mode. To enable automatic commits temporarily, set the auto_commit database option (a server-side option) to On using a statement such as the following:

```
SET TEMPORARY OPTION auto_commit='On'
```

Control Autocommit Mode (JDBC)

By default, JDBC operates in autocommit mode. To turn off autocommit, use the setAutoCommit method of the connection object:

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

Control Autocommit Mode (ODBC)

By default, ODBC operates in autocommit mode. The way you turn off autocommit depends on whether you are using ODBC directly, or using an application development tool. If you are programming directly to the ODBC interface, set the SQL_ATTR_AUTOCOMMIT connection attribute. The following example disables autocommit.

```
SQLSetConnectAttr( hdbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
0 );
```

Control Autocommit Mode (OLE DB)

By default, the OLE DB provider operates in autocommit mode. To use explicit transactions, use the ITransactionLocal::StartTransaction, ITransaction::Commit, and ITransaction::Abort methods.

Control Autocommit Mode (Open Client)

By default, a connection made through Open Client operates in autocommit mode. You can change this behavior by setting the chained database option (a server-side option) to On in your application using a statement such as the following:

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

Control Autocommit Mode (Perl)

By default, Perl operates in autocommit mode. To disable autocommit, set the AutoCommit option:

```
my $dbh = DBI->connect( "DBI:SQLAnywhere:$connstr", '', '', {AutoCommit => 0} );
```

Control Autocommit Mode (PHP)

By default, PHP operates in autocommit mode. To disable autocommit, use the sasql_set_option function:

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

Control Autocommit Mode (Python)

By default, Python operates in manual commit mode. To enable autocommit, set the auto_commit database option (a server-side option) to On using a statement such as the following:

```
cursor.execute("SET TEMPORARY OPTION auto_commit='On'")
```

Control Aautocommit Mode (Ruby)

By default, Ruby operates in manual commit mode. To enable autocommit, set the auto_commit database option (a server-side option) to On using a statement such as the following:

```
rc = api.sqlany_execute_immediate( conn, "SET TEMPORARY OPTION
auto_commit='On'" )
```

Control Autocommit Mode (on the server)

By default, the database server operates in manual commit mode. To enable automatic commits temporarily, set the auto_commit database option (a server-side option) to On using a statement such as the following:

```
SET TEMPORARY OPTION auto_commit='On'
```

> **Note**
>
> Do not set the auto_commit server option directly when using an API such as ADO.NET, JDBC, ODBC, or OLE DB. Use the API-specific mechanism for enabling or disabling automatic commit. For example, in ODBC set the SQL_ATTR_AUTOCOMMIT connection attribute using SQLSetConnectAttr. When you use the API, the driver can track the current setting of automatic commit.

> **Note**
>
> The auto_commit option cannot be set in Interactive SQL using a SET TEMPORARY OPTION statement since this sets the Interactive SQL option of the same name. You could imbed the statement in an EXECUTE IMMEDIATE statement, however it is inadvisable to do so as unpredictable behavior can result. By default, Interactive SQL operates in manual commit mode (auto_commit = 'Off').

Related Information

Autocommit Implementation Details [page 51]
Transaction Processing [page 82]

1.2.8.1.2 Autocommit Implementation Details

Autocommit mode has slightly different behavior depending on the interface and provider that you are using and how you control the autocommit behavior.

Some application programming interfaces operate in manual commit mode by default. Others operate in automatic commit (or autocommit) mode by default. Consult the documentation for each API to determine which mode is default.

The way autocommit behavior is controlled depends on the application programming interface.

API method execution

To enable or disable autocommit, some application programming interfaces provide native callable methods. Examples are ADO.NET, ADO/OLE DB, JDBC, ODBC, Perl, and PHP.

For example, in ODBC applications you enable autocommit using an API call as follows:

```
SQLSetConnectAttr( hdbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_ON,
0 );
```

Statement option execution

To enable or disable autocommit, some application programming interfaces require the execution of a SQL SET OPTION statement. Examples are ESQL, Python, and Ruby. The following statement temporarily enables autocommit on the database server for this connection only.

```
SET TEMPORARY OPTION auto_commit='On'
```

When you use the SAP Open Client interface, you set the CHAINED option to manipulate autocommit behavior. The CHAINED option is provided for TDS application compatibility. If you are using the jConnect JDBC driver, then you must call the JDBC setAutoCommit method rather than set the CHAINED option.

> **Note**
>
> If the application programming interface provides a callable native method specifically for enabling or disabling autocommit, then that interface must be used.

1.2.8.2 Isolation Level Settings

You can set the isolation level of a current connection using the isolation_level database option.

Some interfaces, such as ODBC, allow you to set the isolation level for a connection at connection time. You can reset this level later using the isolation_level database option.

You can override any temporary or public settings for the isolation_level database option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

Related Information

isolation_level Option

1.2.8.3 Cursors and Transactions

In general, a cursor closes when a COMMIT is performed.

There are two exceptions to this behavior:

- The close_on_endtrans database option is set to Off.
- A cursor is opened WITH HOLD, which is the default with Open Client and JDBC.

If either of these two cases is true, the cursor stays open on a COMMIT.

ROLLBACK and Cursors

If a transaction rolls back, then cursors close except for those cursors opened WITH HOLD. However, don't rely on the contents of any cursor after a rollback.

The draft ISO SQL3 standard states that on a rollback, all cursors (even those cursors opened WITH HOLD) should close. You can obtain this behavior by setting the ansi_close_cursors_on_rollback option to On.

Savepoints

If a transaction rolls back to a savepoint, and if the ansi_close_cursors_on_rollback option is On, then all cursors (even those cursors opened WITH HOLD) opened after the SAVEPOINT close.

Cursors and Isolation Levels

You can change the isolation level of a connection during a transaction using the SET OPTION statement to alter the isolation_level option. However, this change does not affect open cursors.

A snapshot of all rows committed at the snapshot start time is visible when the WITH HOLD clause is used with the snapshot, statement-snapshot, and readonly-statement-snapshot isolation levels. Also visible are all modifications completed by the current connection since the start of the transaction within which the cursor was open.

Related Information

Isolation Levels and Consistency
isolation_level Option

1.3 Appendix - .NET Framework

Use SQL Anywhere with .NET, including the API for the SQL Anywhere .NET Data Provider.

> ⚠ Restriction
>
> This content is for .NET Framework users. It does not apply to .NET Core or .NET 5+ users.

In this section:

SQL Anywhere .NET Data Provider [page 54]
Microsoft ADO.NET is the latest data access API in the line of ODBC, OLE DB, and ADO. It is the preferred data access component for the Microsoft .NET Framework and allows you to access relational database systems.

SQL Anywhere .NET Core Data Provider [page 104]
The SQL Anywhere Provider for .NET Core is an ADO.NET driver that provides data access from .NET Core applications to SAP SQL Anywhere databases.

.NET Data Provider Tutorials [page 105]
The Simple and Table Viewer sample projects introduce you to .NET application programming using the .NET provider. A tutorial is included that takes you though the steps of building the Simple Viewer .NET database application using Microsoft Visual Studio.

SQL Anywhere ASP.NET Providers [page 123]
The SQL Anywhere ASP.NET providers replace the standard ASP.NET providers for Microsoft SQL Server, and allow you to run your website using SQL Anywhere.

SQL Anywhere .NET API Reference [page 131]
Use SQL Anywhere with .NET, including the API for the SQL Anywhere .NET Data Provider.

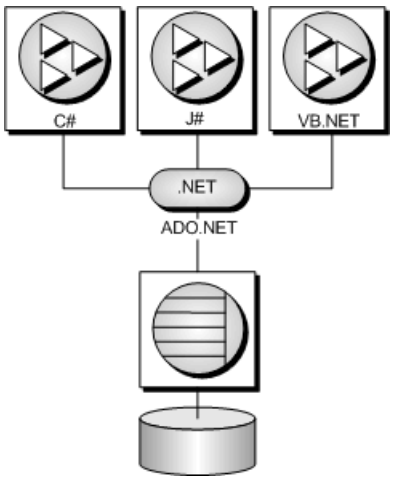
1.3.1 SQL Anywhere .NET Data Provider

Microsoft ADO.NET is the latest data access API in the line of ODBC, OLE DB, and ADO. It is the preferred data access component for the Microsoft .NET Framework and allows you to access relational database systems.

The SQL Anywhere .NET Data Provider implements the Sap.Data.SQLAnywhere namespace and allows you to write programs in any of the .NET supported languages, such as Microsoft C# and Microsoft Visual Basic .NET, and access data from SQL Anywhere databases.

Microsoft ADO.NET Applications

You can develop Internet and intranet applications using object-oriented languages, and then connect these applications to the database server using the SQL Anywhere.NET Data Provider.



In this section:

SQL Anywhere .NET Data Provider Features [page 55]
The Microsoft .NET Framework is supported through three distinct namespaces.

.NET Sample Projects [page 56]
There are several sample projects included with the SQL Anywhere .NET Data Provider.

Using the SQL Anywhere .NET Data Provider in a Microsoft Visual Studio Project [page 57]
Use the SQL Anywhere .NET Data Provider to develop Microsoft .NET applications with Microsoft Visual Studio by including both a reference to the SQL Anywhere .NET Data Provider, and a line in your source code referencing the SQL Anywhere .NET Data Provider classes.

Microsoft .NET Database Connections [page 58]
To connect to a database, an SAConnection object must be created. The connection string can be specified when creating the object or it can be established later by setting the ConnectionString property.

Data Access and Manipulation [page 66]
With the SQL Anywhere .NET Data Provider, there are two ways you can access data, using the SACommand class or the SADataAdapter class.

Stored Procedures [page 81]
You can use SQL stored procedures with the SQL Anywhere .NET Data Provider.

Transaction Processing [page 82]
You can use the SATransaction object to group statements together. Each transaction ends with a call to the Commit method, which either makes your changes to the database permanent, or the Rollback method, which cancels all the operations in the transaction.

Microsoft .NET Error Handling [page 84]
Your application should be designed to handle any errors that occur. An SAException object is created when an exception is thrown. Information about the exception is stored in the SAException object.

Entity Framework Support [page 85]
The SQL Anywhere .NET Data Provider supports Entity Framework 5.0 and 6.0, separate packages available from Microsoft.

.NET Tracing Support [page 98]
The .NET Data Provider supports tracing using the .NET tracing feature.

The SQL Anywhere .NET Data Provider Unmanaged Code [page 103]
The unmanaged code portion of the SQL Anywhere .NET Data Provider is contained in a shared object or DLL that is unloaded by the provider into a directory and subsequently loaded from there into memory.

1.3.1.1 SQL Anywhere .NET Data Provider Features

The Microsoft .NET Framework is supported through three distinct namespaces.

Sap.Data.SQLAnywhere

The ADO.NET object model is an all-purpose data access model. ADO.NET components were designed to factor data access from data manipulation. There are two central components of ADO.NET that do this: the DataSet, and the .NET Framework data provider, which is a set of components including the Connection, Command, DataReader, and DataAdapter objects. A .NET Entity Framework Data Provider is included that communicates directly with a database server without adding the overhead of OLE DB or ODBC. The .NET Data Provider is represented in the .NET namespace as Sap.Data.SQLAnywhere.

The SQL Anywhere .NET Data Provider namespace is described in this document.

System.Data.Oledb

This namespace supports OLE DB data sources. This namespace is an intrinsic part of the Microsoft .NET Framework. You can use System.Data.Oledb together with the OLE DB provider, SAOLEDB, to access databases.

System.Data.Odbc

This namespace supports ODBC data sources. This namespace is an intrinsic part of the Microsoft .NET Framework. You can use System.Data.Odbc together with the ODBC drivers to access databases.

There are some key benefits to using the .NET Data Provider:

- In the .NET environment, the .NET Data Provider provides native access to a database. Unlike the other supported providers, it communicates directly with a database server and does not require bridge technology.

- As a result, the .NET Data Provider is faster than the OLE DB and ODBC Data Providers. It is the recommended Data Provider for accessing SQL Anywhere databases.

1.3.1.2 .NET Sample Projects

There are several sample projects included with the SQL Anywhere .NET Data Provider.

DeployUtility

This is a code example to assist in the deployment of the unmanaged code portions of the SQL Anywhere .NET Data Provider in a ClickOnce deployment.

LinqSample

A .NET Framework sample project for Windows that demonstrates language-integrated query, set, and transform operations using the SQL Anywhere .NET Data Provider and C#.

SimpleWin32

A Microsoft .NET Framework sample project for Microsoft Windows that demonstrates a simple listbox that is filled with the names from the Employees table when you click *Connect*.

SimpleXML

A Microsoft .NET Framework sample project for Microsoft Windows that demonstrates how to obtain XML data from a database via Microsoft ADO.NET. Samples for Microsoft C#, Visual Basic, and Microsoft Visual C++ are provided.

SimpleViewer

A Microsoft .NET Framework sample project for Microsoft Windows.

TableViewer

A Microsoft .NET Framework sample project for Microsoft Windows that allows you to enter and execute SQL statements.

Related Information

ClickOnce and .NET Data Provider Unmanaged Code DLLs [page 829]
Tutorial: Using the Simple Code Sample in SimpleWin32 [page 105]
Tutorial: Developing a Simple .NET Database Application with Microsoft Visual Studio [page 114]
Tutorial: Using the Table Viewer Code Sample [page 110]

1.3.1.3 Using the SQL Anywhere .NET Data Provider in a Microsoft Visual Studio Project

Use the SQL Anywhere .NET Data Provider to develop Microsoft .NET applications with Microsoft Visual Studio by including both a reference to the SQL Anywhere .NET Data Provider, and a line in your source code referencing the SQL Anywhere .NET Data Provider classes.

Procedure

1. Start Microsoft Visual Studio and create or open your project.
2. In the *Solution Explorer* window, right-click *References* and click *Add Reference*.

   The reference indicates which provider to include and locates the code for the SQL Anywhere .NET Data Provider.
3. Click the *.NET* tab (or open *Assemblies/Extensions*), and scroll through the list to locate any of the following:
   - Sap.Data.SQLAnywhere for .NET 3.5
   - Sap.Data.SQLAnywhere for .NET 4
   - Sap.Data.SQLAnywhere for .NET 4.5
   - Sap.Data.SQLAnywhere for Entity Framework 6

   The choices for provider may be governed by your project's Target framework.
4. Click the desired provider, make sure the checkbox for the selected provider is checked if one is present, and then click *OK*.

   The provider is added to the *References* folder in the *Solution Explorer* window of your project.
5. Specify a directive to your source code to assist with the use of the SQL Anywhere .NET Data Provider namespace and the defined types.

   Add the following line to your project:

   - If you are using C#, add the following line to the list of `using` directives at the beginning of your source code:

     ```
     using Sap.Data.SQLAnywhere;
     ```
   - If you are using Visual Basic, add the following line at the beginning of source code:

     ```
     Imports Sap.Data.SQLAnywhere
     ```

Results

The SQL Anywhere .NET Data Provider is set up for use with your SQL Anywhere .NET application.

Example

The following C# example shows how to create a connection object when a *using* directive has been specified:

```
SAConnection conn = new SAConnection();
```

The following C# example shows how to create a connection object when a *using* directive has not been specified:

```
Sap.Data.SQLAnywhere.SAConnection conn =
    new Sap.Data.SQLAnywhere.SAConnection();
```

The following Microsoft Visual Basic example shows how to create a connection object when an *Imports* directive has been specified:

```
Dim conn As New SAConnection()
```

The following Microsoft Visual Basic example shows how to create a connection object when an *Imports* directive has not been specified:

```
Dim conn As New Sap.Data.SQLAnywhere.SAConnection()
```

1.3.1.4 Microsoft .NET Database Connections

To connect to a database, an SAConnection object must be created. The connection string can be specified when creating the object or it can be established later by setting the ConnectionString property.

A well-designed application should handle any errors that occur when attempting to connect to a database.

A connection to the database is created when the connection is opened and released when the connection is closed.

Microsoft C# SAConnection Example

The following Microsoft C# code creates a button click handler that opens a connection to the sample database and then closes it. An exception handler is included.

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=SQL Anywhere 17
Demo;Password="+pwd);
    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");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Failed to connect");
    }
}
```

Microsoft Visual Basic SAConnection Example

The following Microsoft Visual Basic code creates a button click handler that opens a connection to the sample database and then closes it. An exception handler is included.

```
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 17
Demo;Password="+pwd)
    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")
    Catch ex as Exception
        MessageBox.Show(ex.Message, "Failed to connect")
    End Try
End Sub
```

In this section:

.NET Connection Parameters [page 60]
Use connection parameters in connection strings to connect and authenticate to a database server.

.NET Connection Pooling [page 64]
The SQL Anywhere .NET Data Provider supports native .NET connection pooling. Connection pooling allows your application to reuse existing connections by saving the connection handle to a pool so it can be reused, rather than repeatedly creating a new connection to the database.

Connection State [page 65]
Once your application has established a connection to the database, you can check the connection state to ensure that the connection is still open before communicating a request to the database server.

Related Information

Alphabetical List of Connection Parameters

1.3.1.4.1 .NET Connection Parameters

Use connection parameters in connection strings to connect and authenticate to a database server.

Specify a connection string in a .NET application when the connection object is created or by setting the ConnectionString property of a connection object.

- ```
  SAConnection conn = new SAConnection("connection-string");
  ```
- ```
  SAConnection conn = new SAConnection();
  conn.ConnectionString = "connection-string";
  ```

Specify connection parameters as keyword=value pairs, separated by semicolons. For example:

```
SAConnection conn = new SAConnection(
       "Host=sqla-host:2638;Server=sqla-server;UserID=JSmith;Password=secret");
```

Connection parameter names are case insensitive. For example, `UserID` and `userid` are equivalent. If a connection parameter name contains spaces, then they must be preserved.

Connection parameter values can be case sensitive. For example, passwords are usually case sensitive.

Connection Parameters

Connection Parameter	Description
`Connection lifetime`	Specifies the maximum lifetime (in seconds) for a connection that is to be pooled. If the time between closing a connection and the time it was originally opened is longer than the maximum lifetime, then the connection is not pooled but closed. A connection may have been opened, closed, and pooled many times but when the total lifetime is exceeded, it is no longer returned to the pool. This can result in the pool of available connections shrinking over time. The default is 0, which means no maximum. `Connection lifetime=seconds`
`Connection timeout`	Specifies the length of time (in seconds) to wait for a connection to the database server before terminating the attempt and generating an error. The default is 15. The alternate form `Connect Timeout` can be used. `Connection timeout=seconds`

Connection Parameter	Description

`DatabaseName`	Identifies the database name. The alternate form **DBN** can be used.
	`DatabaseName=db-name`
	This parameter sets the read-only `Database` property of the connection object. The property can be queried as follows:
	`String database = conn.Database;`

`Data Source`	Identifies the data source name. The alternate forms `DataSourceName` and **DSN** can be used. `Data Source=datasource-name`
`Enlist`	Specifies whether transactions are enlisted with the Microsoft Distributed Transaction Coordinator. The default is **`true`**. `Enlist=boolean-value` When set **`false`**, transactions are not enlisted with the Microsoft Distributed Transaction Coordinator. A Distributed Transaction Coordinator (DTC) service must be running on each computer to operate distributed transactions. You can start or stop DTC from the Microsoft Windows *Services* window; the DTC service task is named MSDTC.
`FileDataSourceName`	Identifies the file data source name. The alternate form `FileDSN` can be used. `FileDataSourceName=file-name`
`Host`	Identifies the host computer name or IP address and port number. `Host=host-spec[:host-port]`
`InitString`	Specifies a SQL statement that is executed immediately after a connection is established to the database server. `InitString=sql-statement`

Connection Parameter	Description
`Max pool size`	Specifies the maximum size of the connection pool. The default is 100. `Max pool size=number` For example, if Max Pool Size=20, then up to 20 concurrent connections will be pooled when closed. Any concurrent connections beyond the first 20 closed connections will not be pooled (that is, at most 20 connections will be pooled and the rest will be closed). Note each pooled connection counts as a "real" connection to the database server. Max Pool Size does not prevent the application from making as many concurrent connections as it desires. An application that makes at most one connection to the database server at a time will not need to adjust the Min and Max Pool Size settings.
`Min pool size`	Specifies the minimum size of the connection pool. The default is 0. `Min pool size=number` For example, if Min Pool Size=5, then 5 connections are made to the database server at the time the first connection is made. The next 4 concurrent connections will come out of this pool. Any additional concurrent connections are added as needed. This option is useful for multithreaded applications that make several connections where you want a quick response to an "open" request (the speed is equivalent to unpooling a pooled connection). An application that makes at most one connection to the database server at a time will not need to adjust the Min and Max Pool Size settings.
`Password`	Specifies the database user password. The alternate form `PWD` can be used. `PWD=passcode`

Connection Parameter	Description
`Persist security info`	Indicates whether the `Password` (`PWD`) connection parameter must be retained in the `ConnectionString` property of the connection object. The default is **`false`**. `Persist security info=boolean-value` When set **`true`**, the application can obtain the user's password from the `ConnectionString` property if the `Password` (`PWD`) connection parameter was specified in the original connection string.
`Pooling`	Enables or disables connection pooling. The default is **`true`**. `Pooling=boolean-value`

`Server`	Identifies the host name and port of the database server. The alternate form `ServerName` can be used.
	`Server=sqla-server:port`
	This parameter sets the read-only `DataSource` property of the connection object. The property can be queried as follows:
	`String datasource = conn.DataSource;`

`User ID`	Specifies the database user name. The alternate forms `Username`, `UserID`, and `UID` can be used. `UID=username` **Note** Avoid the use of `User ID` in connection strings in application configuration files. `UserID` and `UID` can be used instead.

Not all connection parameters are described here. For information on other connection parameters, see the topic on database connection parameters.

Note that the ConnectionPool(CPOOL) option is ignored by the .NET data provider (it has no effect at all).

Connection String Examples

- Creates a connection object setting the ConnectionString property and then connects to the database server.

```
SAConnection conn = new SAConnection(
    "Host=sqla-host:2638;Server=sqla-server;UserID=JSmith;Password=secret" );
conn.Open();
```

- Creates a connection object, sets the ConnectionString property for the connection object, and then connects to the database server. A SET TEMPORARY OPTION statement is executed after connecting to the database server to verify the application signature against the database signature for authenticated applications. Since the SET TEMPORARY OPTION statement contains semicolons, it must be enclosed by quotation marks (").

```
SAConnection conn = new SAConnection();
conn.ConnectionString =
    "Host=sqla-host:2638;Server=sqla-server;Database=sqla-db;" +
    "UID=JSmith;PWD=secret;" +
    "InitString=\"SET TEMPORARY OPTION connection_authentication=" +
    "'Company=MyCo;" +
    "Application=MyApp;" +
    "Signature=0fa55157edb8e14d818e...'\""
conn.Open();
```

- Connects to a database server running on the computer identified by an IP address using a user ID and password that were obtained from the user.

```
SAConnection conn = new SAConnection();
conn.ConnectionString =
    "Host=10.7.185.43:2638;Server=sqla-server;Database=sqla-db;" +
    "UID=" + user_name + ";PWD=" + pass_code;
conn.Open();
```

Related Information

Alphabetical List of Connection Parameters

1.3.1.4.2 .NET Connection Pooling

The SQL Anywhere .NET Data Provider supports native .NET connection pooling. Connection pooling allows your application to reuse existing connections by saving the connection handle to a pool so it can be reused, rather than repeatedly creating a new connection to the database.

Connection pooling is enabled and disabled using the *Pooling* connection parameter. Connection pooling is enabled by default.

The maximum pool size is set in your connection string using the *Max Pool Size* parameter. The minimum or initial pool size is set in your connection string using the *Min Pool Size* parameter. The default maximum pool size is 100, while the default minimum pool size is 0.

The following is an example of a .NET connection string. Data source names are supported.

```
"Data Source=SQL Anywhere 17 Demo;Pooling=true;Max Pool Size=50;Min Pool
Size=5;Password="+pwd
```

Do not use the ConnectionPool (CPOOL) connection parameter to enable or disable .NET connection pooling. Use the Pooling connection parameter for this purpose.

When your application first attempts to connect to the database, it checks the pool for an existing connection that uses the same connection parameters you have specified. If a matching connection is found, that connection is used. Otherwise, a new connection is used. When you disconnect, the connection is returned to the pool so that it can be reused.

Multiple connection pools are supported by .NET and a different connection string creates a different pool. To reuse a pooled connection, the connection strings must be textually identical. For .NET applications, the following two connection strings are not considered equivalent for connection pooling; thus each string would create its own connection pool.

```
UID=DBA;PWD=passwd;ConnectionName=One
UserID=DBA;PWD=passwd;ConnectionName=One
```

If Min Pool Size=5, then 5 connections are made to the database server at the time the first connection is made. The next 4 concurrent connections will come out of this pool. Any additional concurrent connections are added as needed. This option is useful for multithreaded applications that make several connections where you want a quick response to an "open" request (the speed is equivalent to unpooling a pooled connection).

If Max Pool Size=20, then up to 20 concurrent connections will be pooled when closed. Any concurrent connections beyond the first 20 closed connections will not be pooled (that is, at most 20 connections will be pooled and the rest will be closed). Note each pooled connection counts as a "real" connection to the database server. Max Pool Size does not prevent the application from making as many concurrent connections as it desires.

A .NET application that makes at most one connection to the database server at a time will not need to adjust the Min and Max pool size settings.

Connection pooling is not supported for non-standard database authentication such as Integrated or Kerberos logins. Only user ID and password authentication is supported.

The database server also supports connection pooling. This feature is controlled using the ConnectionPool (CPOOL) connection parameter. However, the .NET Data Provider does not use this server feature and disables it (CPOOL=NO). All connection pooling is done in the .NET client application instead (client-side connection pooling).

1.3.1.4.3 Connection State

Once your application has established a connection to the database, you can check the connection state to ensure that the connection is still open before communicating a request to the database server.

If a connection is closed, you can return an appropriate message to the user and/or attempt to reopen the connection.

The SAConnection class has a State property that can be used to check the state of the connection. Possible state values are ConnectionState.Open and ConnectionState.Closed.

The following code checks whether the SAConnection object has been initialized, and if it has, it checks that the connection is open. A message is returned to the user if the connection is not open.

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

1.3.1.5 Data Access and Manipulation

With the SQL Anywhere .NET Data Provider, there are two ways you can access data, using the SACommand class or the SADataAdapter class.

SACommand object

The SACommand object is the recommended way of accessing and manipulating data in .NET.

The SACommand object allows you to execute SQL statements that retrieve or modify data directly from the database. Using the SACommand object, you can issue SQL statements and call stored procedures directly against the database.

Within an SACommand object, an SADataReader is used to return read-only result sets from a query or stored procedure. The SADataReader returns only one row at a time, but this does not degrade performance because the client-side libraries use prefetch buffering to prefetch several rows at a time.

Using the SACommand object allows you to group your changes into transactions rather than operating in autocommit mode. When you use the SATransaction object, locks are placed on the rows so that other users cannot modify them.

SADataAdapter object

The SADataAdapter object retrieves the entire result set into a DataSet. A DataSet is a disconnected store for data that is retrieved from a database. You can then edit the data in the DataSet and when you are finished, the SADataAdapter object updates the database with the changes made to the DataSet. When you use the SADataAdapter, there is no way to prevent other users from modifying the rows in your DataSet. You must include logic within your application to resolve any conflicts that may occur.

There is no performance impact from using the SADataReader within an SACommand object to fetch rows from the database rather than the SADataAdapter object.

In this section:

SACommand: Fetch Data Using ExecuteReader and ExecuteScalar [page 67]
The SACommand object allows you to execute a SQL statement or call a stored procedure against a database. You can use the ExecuteReader or ExecuteScalar methods to retrieve data from the database.

SACommand: Fetch Result Set Schema Using GetSchemaTable [page 69]
You can obtain schema information about columns in a result set using the GetSchemaTable method.

SACommand: Insert, Delete, and Update Rows Using ExecuteNonQuery [page 69]
To perform an insert, update, or delete with an SACommand object, use the ExecuteNonQuery method. The ExecuteNonQuery method issues a query (SQL statement or stored procedure) that does not return a result set.

SACommand: Retrieve Primary Key Values for Newly Inserted Rows [page 71]
If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain the primary key values generated by the data source.

SADataAdapter: Overview [page 72]
The SADataAdapter retrieves a result set into a DataTable. A DataSet is a collection of tables (DataTables) and the relationships and constraints between those tables. The DataSet is built into the .NET Framework, and is independent of the Data Provider used to connect to your database.

SADataAdapter: Fetch Data into a DataTable Using Fill [page 74]
The SADataAdapter allows you to view a result set by using the Fill method to fill a DataTable with the results from a query and then binding the DataTable to a display grid.

SADataAdapter: Format a DataTable Using FillSchema [page 75]
The SADataAdapter allows you to configure the schema of a DataTable to match that of a specific query using the FillSchema method. The attributes of the columns in the DataTable will match those of the SelectCommand of the SADataAdapter object.

SADataAdapter: Insert Rows Using Add and Update [page 76]
The SADataAdapter allows you to insert rows using the Add and Update methods.

SADataAdapter: Delete Rows Using Delete and Update [page 77]
The SADataAdapter allows you to delete rows using the Delete and Update methods.

SADataAdapter: Update Rows Using Update [page 78]
The SADataAdapter allows you to update rows using the Update method.

SADataAdapter: Retrieve Primary Key Values for Newly Inserted Rows [page 79]
If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain the primary key values generated by the data source.

BLOB Handling in .NET Applications [page 80]
When fetching long string values or binary data, there are methods that you can use to fetch the data in pieces. For binary data, use the GetBytes method, and for string data, use the GetChars method.

Time Values [page 81]
The .NET Framework does not have a Time structure. To fetch time values from a database, you must use the GetTimeSpan method.

1.3.1.5.1 SACommand: Fetch Data Using ExecuteReader and ExecuteScalar

The SACommand object allows you to execute a SQL statement or call a stored procedure against a database. You can use the ExecuteReader or ExecuteScalar methods to retrieve data from the database.

ExecuteReader

Issues a SQL query that returns a result set. This method uses a forward-only, read-only cursor. You can loop quickly through the rows of the result set in one direction.

ExecuteScalar

Issues a SQL query that returns a single value. This can be the first column in the first row of the result set, or a SQL statement that returns an aggregate value such as COUNT or AVG. This method uses a forward-only, read-only cursor.

When using the SACommand object, you can use the SADataReader to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

When using the SADataReader, there are several Get methods available that you can use to return the results in the specified data type.

Microsoft C# ExecuteReader Example

The following Microsoft C# code opens a connection to the sample database and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd);
conn.Open();
SACommand cmd = new SACommand("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();
```

Microsoft Visual Basic ExecuteReader Example

The following Microsoft Visual Basic code opens a connection to the sample database and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
Dim conn As New SAConnection("Data Source=SQL Anywhere 17 Demo;Password="+pwd)
conn.Open()
Dim cmd As New SACommand("SELECT Surname FROM Employees", conn)
Dim reader As SADataReader = cmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (reader.Read())
    ListEmployees.Items.Add(reader.GetString(0))
Loop
ListEmployees.EndUpdate()
reader.Close()
conn.Close()
```

Microsoft C# ExecuteScalar Example

The following Microsoft C# code opens a connection to the sample database and uses the ExecuteScalar method to obtain a count of the number of male employees in the Employees table:

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd);
conn.Open();
SACommand cmd = new SACommand(
    "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'", conn );
int count = (int) cmd.ExecuteScalar();
textBox1.Text = count.ToString();
conn.Close();
```

1.3.1.5.2 SACommand: Fetch Result Set Schema Using GetSchemaTable

You can obtain schema information about columns in a result set using the GetSchemaTable method.

The GetSchemaTable method of the SADataReader class obtains information about the current result set. The GetSchemaTable method returns the standard .NET DataTable object, which provides information about all the columns in the result set, including column properties.

C# Schema Information Example

The following example obtains information about a result set using the GetSchemaTable method and binds the DataTable object to the datagrid on the screen.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
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;
```

1.3.1.5.3 SACommand: Insert, Delete, and Update Rows Using ExecuteNonQuery

To perform an insert, update, or delete with an SACommand object, use the ExecuteNonQuery method. The ExecuteNonQuery method issues a query (SQL statement or stored procedure) that does not return a result set.

You can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. You must be connected to a database to use the SACommand object.

For information about obtaining primary key values for AUTOINCREMENT primary keys, see the documentation on retrieving primary key values for newly inserted rows.

To set the isolation level for a SQL statement, you must use the SACommand object as part of an SATransaction object. When you modify data without an SATransaction object, the provider operates in autocommit mode and any changes that you make are applied immediately.

C# ExecuteNonQuery DELETE and INSERT Example

The following example opens a connection to the sample database and uses the ExecuteNonQuery method to remove all departments whose ID is greater than or equal to 600 and then add two new rows to the Departments table. It displays the updated table in a datagrid.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd);
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# ExecuteNonQuery UPDATE Example

The following example opens a connection to the sample database and uses the ExecuteNonQuery method to update the DepartmentName column to "Engineering" in all rows of the Departments table where the DepartmentID is 100. It displays the updated table in a datagrid.

```
SAConnection conn = new SAConnection(
```

```
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd);
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();
```

Related Information

Transaction Processing [page 82]
SACommand: Retrieve Primary Key Values for Newly Inserted Rows [page 71]

1.3.1.5.4 SACommand: Retrieve Primary Key Values for Newly Inserted Rows

If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain the primary key values generated by the data source.

C# SACommand Primary Key Example

The following example shows how to obtain the primary key that is generated for a newly inserted row. The example uses an SACommand object to call a SQL stored procedure and an SAParameter object to retrieve the primary key that it returns. For demonstration purposes, the example creates a sample table (adodotnet_primarykey) and the stored procedure (sp_adodotnet_primarykey) that is used to insert rows and return primary key values.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
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;
SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add(parmId);
SAParameter parmName = new SAParameter();
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;
```

1.3.1.5.5 SADataAdapter: Overview

The SADataAdapter retrieves a result set into a DataTable. A DataSet is a collection of tables (DataTables) and the relationships and constraints between those tables. The DataSet is built into the .NET Framework, and is independent of the Data Provider used to connect to your database.

When you use the SADataAdapter, you must be connected to the database to fill a DataTable and to update the database with changes made to the DataTable. However, once the DataTable is filled, you can modify the DataTable while disconnected from the database.

If you do not want to apply your changes to the database right away, you can write the DataSet, including the data and/or the schema, to an XML file using the WriteXml method. Then, you can apply the changes at a later time by loading a DataSet with the ReadXml method. The following shows two examples.

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

For more information, see the .NET Framework documentation for WriteXml and ReadXml.

When you call the Update method to apply changes from the DataSet to the database, the SADataAdapter analyzes the changes that have been made and then invokes the appropriate statements, INSERT, UPDATE, or

DELETE, as necessary. When you use the DataSet, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. If another user has a lock on the row you are trying to update, an exception is thrown.

> ⚠ Caution
>
> Any changes you make to the DataSet are made while you are disconnected. Your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that may occur when changes from the DataSet are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

Resolving Conflicts when Using the SADataAdapter

When you use the SADataAdapter, no locks are placed on the rows in the database. This means there is the potential for conflicts to arise when you apply changes from the DataSet to the database. Your application should include logic to resolve or log conflicts that arise.

Some of the conflicts that your application logic should address include:

Unique primary keys

If two users insert new rows into a table, each row must have a unique primary key. For tables with AUTOINCREMENT primary keys, the values in the DataSet may become out of sync with the values in the data source.

It is possible to obtain the values for AUTOINCREMENT primary keys for newly inserted rows.

Updates made to the same value

If two users modify the same value, your application should include logic to determine which value is correct.

Schema changes

If a user modifies the schema of a table you have updated in the DataSet, the update will fail when you apply the changes to the database.

Data concurrency

Concurrent applications should see a consistent set of data. The SADataAdapter does not place a lock on rows that it fetches, so another user can update a value in the database once you have retrieved the DataSet and are working offline.

Many of these potential problems can be avoided by using the SACommand, SADataReader, and SATransaction objects to apply changes to the database. The SATransaction object is recommended because it allows you to set the isolation level for the transaction and it places locks on the rows so that other users cannot modify them.

To simplify the process of conflict resolution, you can design your INSERT, UPDATE, or DELETE statement to be a stored procedure call. By including INSERT, UPDATE, and DELETE statements in stored procedures, you can catch the error if the operation fails. In addition to the statement, you can add error handling logic to the stored procedure so that if the operation fails the appropriate action is taken, such as recording the error to a log file, or trying the operation again.

Related Information

SADataAdapter: Retrieve Primary Key Values for Newly Inserted Rows [page 79]
SACommand: Insert, Delete, and Update Rows Using ExecuteNonQuery [page 69]

1.3.1.5.6 SADataAdapter: Fetch Data into a DataTable Using Fill

The SADataAdapter allows you to view a result set by using the Fill method to fill a DataTable with the results from a query and then binding the DataTable to a display grid.

When setting up an SADataAdapter, you can specify a SQL statement that returns a result set. When Fill is called to populate a DataTable, all the rows are fetched in one operation using a forward-only, read-only cursor. Once all the rows in the result set have been read, the cursor is closed. Changes made to the rows in a DataTable can be reflected to the database using the Update method.

You can use the SADataAdapter object to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

> ⚠ Caution
>
> Any changes you make to a DataTable are made independently of the original database table. Your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that may occur when changes from the DataTable are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

C# SADataAdapter Fill Example Using a DataTable

The following example shows how to fill a DataTable using the SADataAdapter. It creates a new DataTable object named Results and a new SADataAdapter object. The SADataAdapter Fill method is used to fill the DataTable with the results of the query. The DataTable is then bound to the grid on the screen.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
conn.Open();
DataTable dt = new DataTable("Results");
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees", conn);
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

C# SADataAdapter Fill Example Using a DataSet

The following example shows how to fill a DataTable using the SADataAdapter. It creates a new DataSet object and a new SADataAdapter object. The SADataAdapter Fill method is used to create a DataTable table named

Results in the DataSet and then fill it with the results of the query. The Results DataTable is then bound to the grid on the screen.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
conn.Open();
DataSet ds = new DataSet();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees", conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

1.3.1.5.7 SADataAdapter: Format a DataTable Using FillSchema

The SADataAdapter allows you to configure the schema of a DataTable to match that of a specific query using the FillSchema method. The attributes of the columns in the DataTable will match those of the SelectCommand of the SADataAdapter object.

Unlike the Fill method, no rows are stored in the DataTable.

C# SADataAdapter FillSchema Example Using a DataTable

The following example shows how to use the FillSchema method to set up a new DataTable object with the same schema as a result set. The Additions DataTable is then bound to the grid on the screen.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 17
Demo;Password="+pwd );
conn.Open();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees", conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = dt;
```

C# SADataAdapter FillSchema Example Using a DataSet

The following example shows how to use the FillSchema method to set up a new DataTable object with the same schema as a result set. The DataTable is added to the DataSet using the Merge method. The Additions DataTable is then bound to the grid on the screen.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 17
Demo;Password="+pwd );
conn.Open();
SADataAdapter da = new SADataAdapter("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"];
```

1.3.1.5.8 SADataAdapter: Insert Rows Using Add and Update

The SADataAdapter allows you to insert rows using the Add and Update methods.

C# SADataAdapter Insert Example

The example shows how to use the Update method of SADataAdapter to add rows to a table. The example fetches the Departments table into a DataTable using the SelectCommand property and the Fill method of the SADataAdapter. It then adds two new rows to the DataTable and updates the Departments table from the DataTable using the InsertCommand property and the Update method of the SADataAdapter.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
conn.Open();
SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();
SADataAdapter da = new SADataAdapter();
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;
```

1.3.1.5.9 SADataAdapter: Delete Rows Using Delete and Update

The SADataAdapter allows you to delete rows using the Delete and Update methods.

C# SADataAdapter Delete Example

The following example shows how to use the Update method of SADataAdapter to delete rows from a table. The example adds two new rows to the Departments table and then fetches this table into a DataTable using the SelectCommand property and the Fill method of the SADataAdapter. It then deletes some rows from the DataTable and updates the Departments table from the DataTable using the DeleteCommand property and the Update method of the SADataAdapter.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
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;
```

1.3.1.5.10 SADataAdapter: Update Rows Using Update

The SADataAdapter allows you to update rows using the Update method.

C# SADataAdapter Update Example

The following example shows how to use the Update method of SADataAdapter to update rows in a table. The example adds two new rows to the Departments table and then fetches this table into a DataTable using the SelectCommand property and the Fill method of the SADataAdapter. It then modifies some values in the DataTable and updates the Departments table from the DataTable using the UpdateCommand property and the Update method of the SADataAdapter.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
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.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;
```

1.3.1.5.11 SADataAdapter: Retrieve Primary Key Values for Newly Inserted Rows

If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain the primary key values generated by the data source.

C# SADataAdapter Primary Key Example

The following example shows how to obtain the primary key that is generated for a newly inserted row. The example uses an SADataAdapter object to call a SQL stored procedure and an SAParameter object to retrieve the primary key that it returns. For demonstration purposes, the example creates a sample table (adodotnet_primarykey) and the stored procedure (sp_adodotnet_primarykey) that is used to insert rows and return primary key values.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
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();
SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
da.SelectCommand = new SACommand(
    "SELECT * FROM adodotnet_primarykey", conn);
da.InsertCommand = new SACommand(
    "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"];
```

1.3.1.5.12 BLOB Handling in .NET Applications

When fetching long string values or binary data, there are methods that you can use to fetch the data in pieces. For binary data, use the GetBytes method, and for string data, use the GetChars method.

Otherwise, BLOB data is treated in the same manner as any other data you fetch from the database.

C# GetChars BLOB Example

The following example reads three columns from a result set. The first two columns are integers, while the third column is a LONG VARCHAR. The length of the third column is computed by reading this column with the GetChars method in chunks of 100 characters.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
conn.Open();
SACommand cmd = new SACommand("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();
```

1.3.1.5.13 Time Values

The .NET Framework does not have a Time structure. To fetch time values from a database, you must use the GetTimeSpan method.

This method returns the data as a .NET Framework TimeSpan object.

C# TimeSpan Example

The following example uses the GetTimeSpan method to return the time as TimeSpan.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
conn.Open();
SACommand cmd = new SACommand("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();
```

1.3.1.6 Stored Procedures

You can use SQL stored procedures with the SQL Anywhere .NET Data Provider.

The ExecuteReader method is used to call stored procedures that return result sets, while the ExecuteNonQuery method is used to call stored procedures that do not return any result sets. The ExecuteScalar method is used to call stored procedures that return only a single value.

You can use SAParameter objects to pass parameters to a stored procedure.

C# Stored Procedure Call with Parameters Example

The following example shows two ways to call a stored procedure and pass it a parameter. The example uses an SADataReader to fetch the result set returned by the stored procedure.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
conn.Open();
bool method1 = true;
SACommand cmd = new SACommand("", 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();
```

Related Information

SACommand: Insert, Delete, and Update Rows Using ExecuteNonQuery [page 69]
SACommand: Fetch Data Using ExecuteReader and ExecuteScalar [page 67]

1.3.1.7 Transaction Processing

You can use the SATransaction object to group statements together. Each transaction ends with a call to the Commit method, which either makes your changes to the database permanent, or the Rollback method, which cancels all the operations in the transaction.

Once the transaction is complete, you must create a new SATransaction object to make further changes. This behavior is different from ODBC and Embedded SQL, where a transaction persists after you execute a COMMIT or ROLLBACK until the transaction is closed.

If you do not create a transaction, the .NET Data Provider operates in autocommit mode by default. There is an implicit COMMIT after each insert, update, or delete, and once an operation is completed, the change is made to the database. In this case, the changes cannot be rolled back.

Isolation Level Settings for Transactions

The database isolation level is used by default for transactions. You can choose to specify the isolation level for a transaction using the IsolationLevel property when you begin the transaction. The isolation level applies to all statements executed within the transaction. The .NET Data Provider supports snapshot isolation.

The locks that the database server uses when you execute a SQL statement depend on the transaction's isolation level.

Distributed Transaction Processing

The .NET 2.0 framework introduced a new namespace System.Transactions, which contains classes for writing transactional applications. Client applications can create and participate in distributed transactions with one or multiple participants. Client applications can implicitly create transactions using the TransactionScope class. The connection object can detect the existence of an ambient transaction created by the TransactionScope and automatically enlist. The client applications can also create a CommittableTransaction and call the EnlistTransaction method to enlist. This feature is supported by the .NET Data Provider. Distributed transaction has significant performance overhead. Use database transactions for non-distributed transactions.

C# SATransaction Example

The following example shows how to wrap an INSERT into a transaction so that it can be committed or rolled back. A transaction is created with an SATransaction object and linked to the execution of a SQL statement using an SACommand object. Isolation level 2 (RepeatableRead) is specified so that other database users cannot update the row. The lock on the row is released when the transaction is committed or rolled back. If you do not use a transaction, the .NET Data Provider operates in autocommit mode and you cannot roll back any changes that you make to the database.

```
SAConnection conn = new SAConnection(
    "Data Source=SQL Anywhere 17 Demo;Password="+pwd );
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();
```

Related Information

Isolation Levels and Consistency
Locks During Queries

1.3.1.8 Microsoft .NET Error Handling

Your application should be designed to handle any errors that occur. An SAException object is created when an exception is thrown. Information about the exception is stored in the SAException object.

The SQL Anywhere .NET Data Provider creates an SAException object and throws an exception whenever errors occur during execution. Each SAException object consists of a list of SAError objects, and these error objects include the error message and code.

Errors are different from conflicts. Conflicts arise when changes are applied to the database. Your application should include a process to compute correct values or to log conflicts when they arise.

Microsoft C# Error Handling Example

The following Microsoft C# code creates a button click handler that opens a connection to the sample database. If the connection cannot be made, the exception handler displays one or more messages.

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection(
        "Data Source=SQL Anywhere 17 Demo;Password="+pwd);
    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");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Failed to connect");
    }
}
```

Microsoft Visual Basic Error Handling Example

The following Microsoft Visual Basic code creates a button click handler that opens a connection to the sample database. If the connection cannot be made, then the exception handler displays one or more messages.

```
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 17
Demo;Password="+pwd)
    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
    Catch ex as Exception
        MessageBox.Show(ex.Message, "Failed to connect")
    End Try
End Sub
```

Related Information

The Simple Sample Project Explained [page 108]
The Table Viewer Sample Project Explained [page 112]

1.3.1.9 Entity Framework Support

The SQL Anywhere .NET Data Provider supports Entity Framework 5.0 and 6.0, separate packages available from Microsoft.

To use Entity Framework 5.0 or 6.0, you must add it to Microsoft Visual Studio using Microsoft's NuGet Package Manager.

One of the new features of Entity Framework is Code First. It enables a different development workflow: defining data model objects by simply writing Microsoft Visual C# .NET or Microsoft Visual Basic .NET classes mapping to database objects without ever having to open a designer or define an XML mapping file. Optionally, additional configuration can be performed by using data annotations or the Fluent API. Models can be used to generate a database schema or map to an existing database.

Here's an example which creates new database objects using the model:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using Sap.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 17 Demo;Password="+pwd ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}
```

To build and run this example, the following assembly references must be added:

```
EntityFramework
Sap.Data.SQLAnywhere.v4.5
System.ComponentModel.DataAnnotations
System.Data.Entity
```

Here is another example that maps to an existing database:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using Sap.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 17 Demo;Password="+pwd ) )
            {
                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 ) );
                    }
                }
            }
        }
    }
}
```

Be aware that there are some implementation detail differences between the Microsoft .NET Framework Data Provider for Microsoft SQL Server (SqlClient) and the .NET Data Provider.

1. A new class SAConnectionFactory (implements IDbConnectionFactory) is included. You set the Database.DefaultConnectionFactory to an instance of SAConnectionFactory before creating any data model as shown below:

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

2. The major principle of Entity Framework Code First is coding by conventions. The Entity Framework infers the data model by coding conventions. Entity Framework also does lots of things implicitly. Sometimes the developer might not realize all these Entity Framework conventions. But some code conventions do not make sense for database management systems like SQL Anywhere. There are some differences between Microsoft SQL Server and these database servers.
   - Microsoft SQL Server permits access to multiple databases with a single sign-on. SQL Anywhere permits a connection to one database at a time.
   - If the user creates a user-defined DbContext using the parameterless constructor, SqlClient will connect to Microsoft SQL Server Express on the local computer using integrated security. The .NET Data Provider connects to the default server using integrated login if the user has already created a login mapping.
   - SqlClient drops the existing database and creates a new database when the Entity Framework calls DbDeleteDatabase or DbCreateDatabase (Microsoft SQL Server Express Edition only). The .NET Data Provider never drops or creates the database. It creates or drops the database objects (tables, relations, constraints for example). The user must create the database first.
   - The IDbConnectionFactory.CreateConnection method treats the string parameter "nameOrConnectionString" as database name (initial catalog for Microsoft SQL Server) or a connection string. If the user does not provide the connection string for DbContext, SqlClient automatically connects to the SQL Express server on the local computer using the namespace of user-defined DbContext class as the initial catalog. For *SQL Anywhere*, that parameter can only contain a connection string. A database name is ignored and integrated login is used instead.
3. The Microsoft SQL Server SqlClient API maps a column with data annotation attribute TimeStamp to Microsoft SQL Server data type timestamp/rowversion. There are some misconceptions about Microsoft SQL Server timestamp/rowversion among developers. The Microsoft SQL Server timestamp/rowversion data type is different from *SQL Anywhere* and most other RDBMS:
   - The Microsoft SQL Server timestamp/rowversion is binary(8). It is does not support a combined date and time value. SQL Anywhere supports a data type called timestamp that is equivalent to the Microsoft SQL Server datetime data type.
   - Microsoft SQL Server timestamp/rowversion values are guaranteed to be unique. SQL Anywhere timestamp values are not unique.
   - A Microsoft SQL Server timestamp/rowversion value changes every time the row is updated.

   The TimeStamp data annotation attribute is not supported by the .NET Data Provider.
4. By default, Entity Framework 4.1 always sets the schema or owner name to dbo which is the default schema of Microsoft SQL Server. However, dbo is not appropriate for SQL Anywhere databases. For SQL Anywhere, you must specify the schema or owner name (GROUPO for example) with the table name either by using data annotations or the Fluent API. Here is an example:

   ```
   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" );
        }
    }
}
```

In this section:

Code First to a New Database [page 90]
If you plan to try Microsoft's Code First to a New Database tutorial, then there are some steps that you must do differently.

Code First to an Existing Database [page 92]
If you plan to try Microsoft's Code First to an Existing Database tutorial, then there are some steps that you must do differently.

Model First to a New Database [page 94]

If you plan to try Microsoft's Model First tutorial, then there are some steps that you must do differently.

Database First to an Existing Database [page 96]
If you plan to try Microsoft's Database First to an Existing Database tutorial, then there are some steps that you must do differently.

1.3.1.9.1 Code First to a New Database

If you plan to try Microsoft's Code First to a New Database tutorial, then there are some steps that you must do differently.

Prerequisites

You must have Microsoft Visual Studio and the .NET Framework installed on your computer.

You must have the CREATE ANY OBJECT system privilege.

Context

Microsoft's Code First to a New Database tutorial is designed for use with Microsoft's .NET data provider. The steps below provide guidance for using the SQL Anywhere .NET Data Provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Entity Framework 6, make sure that the SQL Anywhere .NET Data Provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

   ```
   cd %SQLANY17%\Assembly\v4.5
   SetupVSPackage.exe /i /v EF6
   ```

2. Before creating your Microsoft Visual Studio project, connect to the sample database using Interactive SQL and run the following SQL script to remove the Blogs, Posts, and Users tables, if you have already created them in another tutorial.

   ```
   DROP TABLE IF EXISTS [GROUPO].[Blogs];
   DROP TABLE IF EXISTS [GROUPO].[Posts];
   DROP TABLE IF EXISTS [GROUPO].[Users];
   ```

3. Start Microsoft Visual Studio. The steps that follow were performed successfully using Microsoft Visual Studio 2013 and Entity Framework 6.1.3.

4. One of the steps requires that you install the latest version of Entity Framework 6 into the project, using the NuGet Package Manager.
   This step creates an `App.config` file that is not suitable for use with SQL Anywhere.
5. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.data>
      <DbProviderFactories>
          <clear />
          <add name="SQL Anywhere 17 Data Provider"
invariant="Sap.Data.SQLAnywhere" description=".Net Framework Data
Provider for SQL Anywhere 17" type="Sap.Data.SQLAnywhere.SAFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400" />
      </DbProviderFactories>
  </system.data>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.SQLAnywhere.SAConnectionFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400">
        <parameters>
            <parameter value="DSN=SQL Anywhere 17 Demo;PWD=sql" />
        </parameters>
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.SQLAnywhere"
type="Sap.Data.SQLAnywhere.SAProviderServices, Sap.Data.SQLAnywhere.EF6,
Version=17.0.0.10094, Culture=neutral, PublicKeyToken=f222fc4333e0d400" />
    </providers>
  </entityFramework>
</configuration>
```

6. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
7. Once you have updated `App.config`, you must build your project.
8. Continue with the instructions in the remaining steps of the tutorial. When you add the code for the BloggingContext class, you must revise it as follows to ensure that the owner of the tables is GROUPO (otherwise, dbo is used).

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.HasDefaultSchema("GROUPO");
    }
}
```

Results

You have built an Entity Framework application that uses the Code First approach when the tables do not already exist in the database.

1.3.1.9.2 Code First to an Existing Database

If you plan to try Microsoft's Code First to an Existing Database tutorial, then there are some steps that you must do differently.

Prerequisites

You must have Microsoft Visual Studio and the .NET Framework installed on your computer.

You must have the CREATE ANY OBJECT system privilege.

Context

Microsoft's Code First to an Existing Database tutorial is designed for use with Microsoft's .NET data provider. The steps below provide guidance for using the SQL Anywhere .NET Data Provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Entity Framework 6, make sure that the SQL Anywhere .NET Data Provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

   ```
   cd %SQLANY17%\Assembly\v4.5
   SetupVSPackage.exe /i /v EF6
   ```

2. Before creating your Microsoft Visual Studio project, connect to the sample database using Interactive SQL and run the following SQL script to set up the Blogs, Posts, and Users tables. It is very similar to the one presented in Microsoft's tutorials but uses the GROUPO schema owner instead of *dbo*.

   ```
   CREATE TABLE [GROUPO].[Blogs] (
       [BlogId] INT DEFAULT AUTOINCREMENT NOT NULL,
       [Name] NVARCHAR (200) NULL,
       [Url]  NVARCHAR (200) NULL,
       CONSTRAINT [PK_GROUPO.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
   );

   CREATE TABLE [GROUPO].[Posts] (
   ```

```
    [PostId] INT DEFAULT AUTOINCREMENT NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_GROUPO.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_GROUPO.Posts_GROUPO.Blogs_BlogId] FOREIGN KEY ([BlogId])
        REFERENCES [GROUPO].[Blogs] ([BlogId]) ON DELETE CASCADE
);
INSERT INTO [GROUPO].[Blogs] ([Name],[Url])
VALUES ('SAP SQL Anywhere', 'http://scn.sap.com/community/sql-anywhere')

INSERT INTO [GROUPO].[Blogs] ([Name],[Url])
VALUES ('SAP Business Trends', 'http://scn.sap.com/community/business-trends')
CREATE TABLE [GROUPO].[Users]
(
    [Username] NVARCHAR(50) NOT NULL PRIMARY KEY,
    [DisplayName] LONG NVARCHAR NULL
)
```

3. Start Microsoft Visual Studio. The steps that follow were performed successfully using Microsoft Visual Studio 2013 and Entity Framework 6.1.3.
4. In the Microsoft Visual Studio *Server Explorer*, use the *.NET Framework Data Provider for SQL Anywhere 17* to create a data connection. For *ODBC data source*, use SQL Anywhere 17 Demo. Fill in the *UserID* and *Password* fields. The *Test Connection* button is used to ensure that a connection can be made to the database.
5. Immediately after creating your new project, install the latest version of Entity Framework 6 into the project, using the NuGet Package Manager.
   This step creates an `App.config` file that is not suitable for use with SQL Anywhere.
6. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.data>
      <DbProviderFactories>
          <clear />
          <add name="SQL Anywhere 17 Data Provider"
invariant="Sap.Data.SQLAnywhere" description=".Net Framework Data
Provider for SQL Anywhere 17" type="Sap.Data.SQLAnywhere.SAFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400" />
      </DbProviderFactories>
  </system.data>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.SQLAnywhere.SAConnectionFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400">
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.SQLAnywhere"
type="Sap.Data.SQLAnywhere.SAProviderServices, Sap.Data.SQLAnywhere.EF6,
Version=17.0.0.10094, Culture=neutral, PublicKeyToken=f222fc4333e0d400" />
    </providers>
```

```
  </entityFramework>
</configuration>
```

7. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
8. Once you have updated `App.config`, you must build your project. This must be done before the Reverse Engineer Model step.
9. In the *Entity Data Model Wizard*, select the option to include sensitive data in the connection string.
10. Rather than select all tables, expand GROUPO in *Tables* and select the Blogs and Posts tables.
11. Continue with the instructions in the remaining steps of the tutorial.

Results

You have built an Entity Framework application that uses the Code First approach for tables that already exist in the database.

1.3.1.9.3 Model First to a New Database

If you plan to try Microsoft's Model First tutorial, then there are some steps that you must do differently.

Prerequisites

You must have Microsoft Visual Studio and the .NET Framework installed on your computer.

You must have the CREATE ANY OBJECT system privilege.

Context

Microsoft's Model First tutorial is designed for use with Microsoft's .NET data provider. The steps below provide guidance for using the SQL Anywhere .NET Data Provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Entity Framework 6, make sure that the SQL Anywhere .NET Data Provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

```
cd %SQLANY17%\Assembly\v4.5
```

```
SetupVSPackage.exe /i /v EF6
```

2. Before creating your Microsoft Visual Studio project, connect to the sample database using Interactive SQL and run the following SQL script to remove the Blogs, Posts, and Users tables, if you have already created them in another tutorial.

```
DROP TABLE IF EXISTS [GROUPO].[Blogs];
DROP TABLE IF EXISTS [GROUPO].[Posts];
DROP TABLE IF EXISTS [GROUPO].[Users];
```

3. Start Microsoft Visual Studio.
4. In the Microsoft Visual Studio *Server Explorer*, use the *.NET Framework Data Provider for SQL Anywhere 17* to create a data connection. For *ODBC data source*, use SQL Anywhere 17 Demo. Fill in the *UserID* and *Password* fields. The *Test Connection* button is used to ensure that a connection can be made to the database.
5. Immediately after creating your new project, install the latest version of Entity Framework 6 into the project, using the NuGet Package Manager.
   This step creates an `App.config` file that is not suitable for use with SQL Anywhere.
6. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.data>
      <DbProviderFactories>
          <clear />
          <add name="SQL Anywhere 17 Data Provider"
invariant="Sap.Data.SQLAnywhere" description=".Net Framework Data
Provider for SQL Anywhere 17" type="Sap.Data.SQLAnywhere.SAFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400" />
      </DbProviderFactories>
  </system.data>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.SQLAnywhere.SAConnectionFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400">
        <parameters>
            <parameter value="DSN=SQL Anywhere 17 Demo;PWD=sql" />
        </parameters>
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.SQLAnywhere"
type="Sap.Data.SQLAnywhere.SAProviderServices, Sap.Data.SQLAnywhere.EF6,
Version=17.0.0.10094, Culture=neutral, PublicKeyToken=f222fc4333e0d400" />
    </providers>
  </entityFramework>
</configuration>
```

7. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.

8. Once you have updated `App.config`, you must build your project. This must be done before using the Entity Framework Designer.
9. Before the Generating the Database step, open the *Properties* of `BloggingModel.edmx [Diagram1]` (this is your design form) and set *Database Schema Name* to GROUPO and set *DDL Generation Template* to `SSDLToSA17.tt (VS)`.
10. In the *Generate Database Wizard*, select the option to include sensitive data in the connection string.
11. Continue with the instructions in the remaining steps of the tutorial.

Results

You have built an Entity Framework application that uses the Model First approach when the tables do not already exist in the database.

1.3.1.9.4 Database First to an Existing Database

If you plan to try Microsoft's Database First to an Existing Database tutorial, then there are some steps that you must do differently.

Prerequisites

You must have Microsoft Visual Studio and the .NET Framework installed on your computer.

You must have the CREATE ANY OBJECT system privilege.

Context

Microsoft's Database First to an Existing Database tutorial is designed for use with Microsoft's .NET data provider. The steps below provide guidance for using the SQL Anywhere .NET Data Provider instead. Review these steps before attempting the tutorial.

Procedure

1. For Entity Framework 6, make sure that the SQL Anywhere .NET Data Provider that supports this version is installed. Make sure there are no running instances of Microsoft Visual Studio. At a command prompt with administrator privileges, do the following:

   ```
   cd %SQLANY17%\Assembly\v4.5
   SetupVSPackage.exe /i /v EF6
   ```

2. Before creating your Microsoft Visual Studio project, connect to the sample database using Interactive SQL and run the following SQL script to set up the Blogs, Posts, and Users tables. It is very similar to the one presented in Microsoft's tutorials but uses the GROUPO schema owner instead of *dbo*.

```
CREATE TABLE [GROUPO].[Blogs] (
    [BlogId] INT DEFAULT AUTOINCREMENT NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url]  NVARCHAR (200) NULL,
    CONSTRAINT [PK_GROUPO.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [GROUPO].[Posts] (
    [PostId] INT DEFAULT AUTOINCREMENT NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_GROUPO.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_GROUPO.Posts_GROUPO.Blogs_BlogId] FOREIGN KEY ([BlogId])
        REFERENCES [GROUPO].[Blogs] ([BlogId]) ON DELETE CASCADE
);
INSERT INTO [GROUPO].[Blogs] ([Name],[Url])
VALUES ('SAP SQL Anywhere', 'http://scn.sap.com/community/sql-anywhere')

INSERT INTO [GROUPO].[Blogs] ([Name],[Url])
VALUES ('SAP Business Trends', 'http://scn.sap.com/community/business-trends')
CREATE TABLE [GROUPO].[Users]
(
    [Username] NVARCHAR(50) NOT NULL PRIMARY KEY,
    [DisplayName] LONG NVARCHAR NULL
)
```

3. Start Microsoft Visual Studio. The steps that follow were performed successfully using Microsoft Visual Studio 2013 and Entity Framework 6.1.3.
4. In the Microsoft Visual Studio *Server Explorer*, use the *.NET Framework Data Provider for SQL Anywhere 17* to create a data connection. For *ODBC data source*, use SQL Anywhere 17 Demo. Fill in the *UserID* and *Password* fields. The *Test Connection* button is used to ensure that a connection can be made to the database.
5. Immediately after creating your new project, install the latest version of Entity Framework 6 into the project, using the NuGet Package Manager.
   This step creates an `App.config` file that is not suitable for use with SQL Anywhere.
6. Replace the contents of `App.config` with the following.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://
go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.data>
      <DbProviderFactories>
          <clear />
          <add name="SQL Anywhere 17 Data Provider"
invariant="Sap.Data.SQLAnywhere" description=".Net Framework Data
Provider for SQL Anywhere 17" type="Sap.Data.SQLAnywhere.SAFactory,
```

```
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400" />
      </DbProviderFactories>
  </system.data>
  <entityFramework>
    <defaultConnectionFactory type="Sap.Data.SQLAnywhere.SAConnectionFactory,
Sap.Data.SQLAnywhere.EF6, Version=17.0.0.10094, Culture=neutral,
PublicKeyToken=f222fc4333e0d400">
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="Sap.Data.SQLAnywhere"
type="Sap.Data.SQLAnywhere.SAProviderServices, Sap.Data.SQLAnywhere.EF6,
Version=17.0.0.10094, Culture=neutral, PublicKeyToken=f222fc4333e0d400" />
    </providers>
  </entityFramework>
</configuration>
```

7. Update all occurrences of the data provider version number in `App.config`. The version number should match the version of the data provider that you have currently installed.
8. Once you have updated `App.config`, you must build your project. This must be done before the Reverse Engineer Model step.
9. In the *Entity Data Model Wizard*, select the option to include sensitive data in the connection string.
10. Rather than select all tables, expand GROUPO in *Tables* and select the Blogs and Posts tables.
11. Continue with the instructions in the remaining steps of the tutorial.

Results

You have built an Entity Framework application that uses the Database First approach for tables that already exist in the database.

1.3.1.10 .NET Tracing Support

The .NET Data Provider supports tracing using the .NET tracing feature.

By default, tracing is disabled. To enable tracing, specify the trace source in your application's configuration file. The following is an example of a configuration file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="Sap.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>
```

There are four types of trace listeners referenced in the configuration file shown above.

ConsoleTraceListener

Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the *Output* window.

DefaultTraceListener

This listener is automatically added to the Debug.Listeners and Trace.Listeners collections using the name "Default". Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the *Output* window. To avoid duplication of output produced by the ConsoleTraceListener, this listener is removed.

EventLogTraceListener

Tracing or debugging output is directed to an EventLog identified in the *initializeData* option. In the example, the event log is named MyEventLog. Writing to the system event log requires administrator privileges and is not a recommended method for debugging applications.

TextWriterTraceListener

Tracing or debugging output is directed to a TextWriter which writes the stream to the file identified in the *initializeData* option.

To disable tracing to any of the trace listeners described above, remove the corresponding *add* entry under *<listeners>*.

The trace configuration information is placed in the application's project folder in the `App.config` file. If the file does not exist, it can be created and added to the project using Microsoft Visual Studio by choosing *Add* ▸ *New Item* and selecting *Application Configuration File*.

The traceOutputOptions can be specified for any listener and include the following:

Callstack

Write the call stack, which is represented by the return value of the Environment.StackTrace property.

DateTime

Write the date and time.

LogicalOperationStack

Write the logical operation stack, which is represented by the return value of the CorrelationManager.LogicalOperationStack property.

None

Do not write any elements.

ProcessId

Write the process identity, which is represented by the return value of the Process.Id property.

ThreadId

Write the thread identity, which is represented by the return value of the Thread.ManagedThreadId property for the current thread.

Timestamp

Write the timestamp, which is represented by the return value of the System.Diagnostics.Stopwatch.GetTimeStamp method.

The example configuration file, shown earlier, specifies trace output options for the TextWriterTraceListener only.

You can limit what is traced by setting specific trace options. By default the numeric-valued trace option settings are all 0. The trace options that can be set include the following:

SASourceSwitch

SASourceSwitch can take any of the following values. If it is Off then there is no tracing.

Off

Does not allow any events through.

Critical

Allows only Critical events through.

Error

Allows Critical and Error events through.

Warning

Allows Critical, Error, and Warning events through.

Information

Allows Critical, Error, Warning, and Information events through.

Verbose

Allows Critical, Error, Warning, Information, and Verbose events through.

ActivityTracing

Allows the Stop, Start, Suspend, Transfer, and Resume events through.

All

Allows all events through.

Here is an example setting.

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

SATraceAllSwitch

All the trace options are enabled. You do not need to set any other options since they are all selected. You cannot disable individual options if you choose this option. For example, the following will not disable exception tracing.

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

SATraceExceptionSwitch

All exceptions are logged. Trace messages have the following form.

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

The nativeError=error_number text will only be displayed if there is an SAException object.

SATraceFunctionSwitch

All method scope entry/exits are logged. Trace messages have any of the following forms.

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

The nnn is an integer representing the scope nesting level 1, 2, 3, ... The optional parameter_names is a list of parameter names separated by spaces.

SATracePoolingSwitch

All connection pooling is logged. Trace messages have any of the following forms.

```
<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

All property setting and retrieval is logged. Trace messages have any of the following forms.

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

In this section:

Configuring a .NET Application for Tracing [page 102]

Enable tracing on the TableViewer sample application by creating a configuration file that references the ConsoleTraceListener and TextWriterTraceListener listeners, removes the default listener, and enables all switches that would otherwise be set to 0.

Related Information

Tracing Data Access

1.3.1.10.1 Configuring a .NET Application for Tracing

Enable tracing on the TableViewer sample application by creating a configuration file that references the ConsoleTraceListener and TextWriterTraceListener listeners, removes the default listener, and enables all switches that would otherwise be set to 0.

Prerequisites

You must have Microsoft Visual Studio installed.

Procedure

1. Open the TableViewer sample in Microsoft Visual Studio.

   Start Microsoft Visual Studio and open the *%SQLANYSAMP17%* `\SQLAnywhere\ADO.NET\TableViewer\TableViewer.sln`.
2. Create an application file named `App.config`.

   Choose *Add* > *New Item* and selecting *Application Configuration File*. Place the following configuration information in this file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="Sap.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. Rebuild the application.
4. Click *Debug* > *Start Debugging*.

Results

When the application finishes execution, the trace output is recorded in the `bin\Debug\myTrace.log` file.

Next Steps

View the trace log in the *Output* window of Microsoft Visual Studio.

Related Information

Tracing Data Access

1.3.1.11 The SQL Anywhere .NET Data Provider Unmanaged Code

The unmanaged code portion of the SQL Anywhere .NET Data Provider is contained in a shared object or DLL that is unloaded by the provider into a directory and subsequently loaded from there into memory.

On Windows, the steps in the load procedure for the unmanaged code DLL (`dbdata17.dll`) are as follows:

1. The provider looks for the dbdata DLL in the ADO.NET application's directory. For example, `c:\Project\bin\x64\Debug`
   If the DLL is found, then it is loaded and a version check is done. If the DLL version matches the provider version, then the application is launched. Otherwise, an attempt to delete the DLL occurs and the next step is performed.
2. The provider looks for the dbdata DLL in a subdirectory of the ADO.NET application's directory.
   The subdirectory name is formed from a GUID, version string, and executable type. For example, `c:\Project\bin\x64\Debug\{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_17011` (32-bit) or `c:\Project\bin\x64\Debug\{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_17011.x64)` (64-bit)
   If the DLL is found, then it is loaded and a version check is done. If the DLL version matches the provider version, then the application is launched. Otherwise, an attempt to delete the DLL occurs and the next step is performed.
3. The provider looks for the dbdata DLL in the directory where the provider was loaded from (this directory could be different than the application directory, for example, the Global Assembly Cache). If the DLL is found, then it is loaded and a version check is done. If the DLL version matches the provider version, then the application is launched. Otherwise, an attempt to delete the DLL occurs and the next step is performed.
4. The provider looks for the first writeable directory chosen in the following order:
   - The path identified by the Path.GetTempPath method. This is the first of the following:
     - The path specified by the TMP environment variable.

- The path specified by the TEMP environment variable.
- The path specified by the USERPROFILE environment variable.
- The Windows directory.

- The path identified by the Directory.GetCurrentDirectory method. This is the application's current working directory.
  The path to the provider DLL (for example, the Global Assembly Cache). This is the same path as in step 3.

5. The provider looks for the dbdata DLL in a subfolder of the folder identified in step 4. It starts with the directory at index 1. For example, `{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_17011.x64_1` If the DLL is found, then it is loaded and a version check is done. If the DLL version matches the provider version, then the application is launched. Otherwise, if the DLL was found but the version was wrong, an attempt is made to delete it. If the delete succeeds, then a new DLL is unpacked into the directory. Otherwise, the next directory (index 2, 3, and so on) is searched repeating step 5.

For step 3, no dbdata DLL will likely be found if the provider DLL is in the Global Assembly Cache (GAC). Typically, the provider DLL is located with the application executable. Ultimately, your application decides how the provider is loaded if it is not through the GAC. Placement of the dbdata DLL as described in step 2 is preferable to that of step 1, if you expect that the application will run as 32-bit in some installations and 64-bit in others. The setup in step 2 permits the installation of both 32-bit and 64-bit versions of the dbdata DLL.

It is the responsibility of the ADO.NET application developer to make copies of the dbdata DLL during the development/test phase from the %TEMP% directory and embed the DLL or DLLs in one of the directories described in step 1 or 2. The developer must ensure correct bitness (32/64 bit) and version match (for example, 17.0.11.7400) between the provider and the dbdata DLL for step 1 or 2 to work.

1.3.2 SQL Anywhere .NET Core Data Provider

The SQL Anywhere Provider for .NET Core is an ADO.NET driver that provides data access from .NET Core applications to SAP SQL Anywhere databases.

The driver is Sap.Data.SQLAnywhere.Core.v2.1.dll and is available on Microsoft Windows. It is included in the SAP SQL Anywhere Database Client.

ADO.NET Support

Most ADO.NET features are supported by the provider. The following ADO.NET functionality is not available for .NET Core:

- Distributed transaction enlistment with a transaction coordinator
- SACredential class
- SAConnection(string connectionString, SACredential credential) method
- SAPermission class
- SAFactory.CreatePermission method
- Entity Framework

1.3.3 .NET Data Provider Tutorials

The Simple and Table Viewer sample projects introduce you to .NET application programming using the .NET provider. A tutorial is included that takes you though the steps of building the Simple Viewer .NET database application using Microsoft Visual Studio.

These sample projects can be used with Microsoft Visual Studio 2005 or later versions. If you use Microsoft Visual Studio 2008 or later versions, you may have to run the Microsoft Visual Studio *Upgrade Wizard*.

In this section:

Tutorial: Using the Simple Code Sample in SimpleWin32 [page 105]
Use the Simple project as an example of how to obtain a result set from the database server using the .NET Data Provider.

Tutorial: Using the Table Viewer Code Sample [page 110]
Use the TableViewer project as an example of how to connect to a database, execute SQL statements, and display the results using a DataGrid object using the .NET Data Provider.

Tutorial: Developing a Simple .NET Database Application with Microsoft Visual Studio [page 114]
This tutorial takes you though the steps of building the Simple Viewer .NET database application using Visual Studio.

1.3.3.1 Tutorial: Using the Simple Code Sample in SimpleWin32

Use the Simple project as an example of how to obtain a result set from the database server using the .NET Data Provider.

Prerequisites

You must have the SELECT ANY TABLE system privilege.

Context

The Simple project is included with the samples. It demonstrates a simple listbox that is filled with the names from the Employees table. You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on your computer.

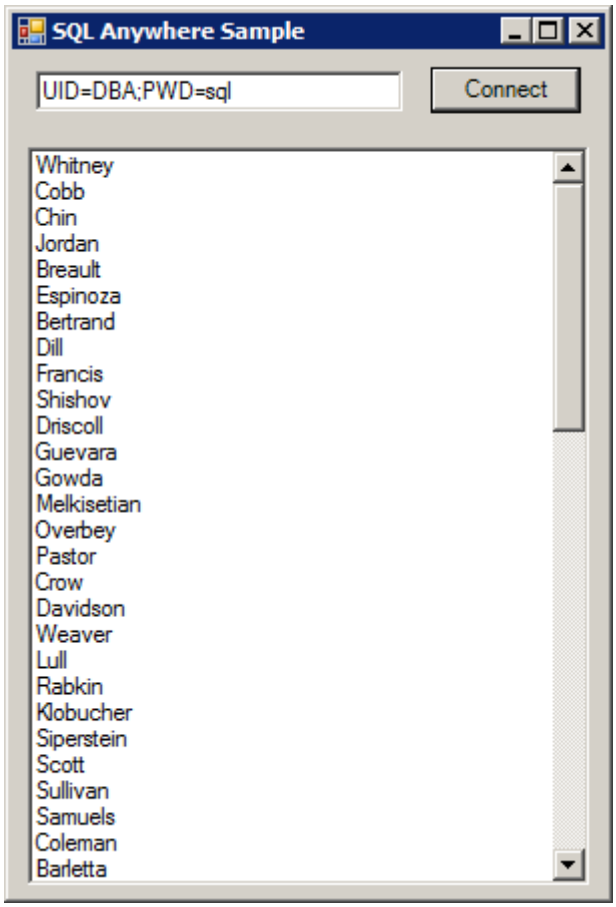
Procedure

1. Start Microsoft Visual Studio.
2. Click *File* > *Open* > *Project*.
3. Browse to *%SQLANYSAMP17%*`\SQLAnywhere\ADO.NET\SimpleWin32` and open the `Simple.sln` project.
4. You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on yourWhen you use the .NET Data Provider in a project, you must add a reference to the Data Provider. This has already been done in the Simple code sample. To view the reference to the Data Provider (`Sap.Data.SQLAnywhere`), open the *References* folder in the *Solution Explorer* window.
5. You must also add a `using` directive to your source code to reference the Data Provider classes. This has already been done in the Simple code sample. To view the `using` directive:
   - Open the source code for the project. In the *Solution Explorer* window, right-click `Form1.cs` and click *View Code*.
     In the `using` directives in the top section, you see the following line:

     ```
     using Sap.Data.SQLAnywhere;
     ```

     This line is required forMicrosoft C# projects. If you are using Microsoft Visual Basic .NET, add the equivalent `Imports` line to your source code.
6. Click *Debug* > *Start Without Debugging* or press Ctrl+F5 to run the Simple sample.
7. In the *SQL Anywhere Sample* window, modify the user ID and password credentials for the sample database and then click *Connect*.

The application connects to the sample database and puts the surname of each employee in the window, as follows:



8. Close the *SQL Anywhere Sample* window to shut down the application and disconnect from the sample database. This also shuts down the database server.

Results

You have built and executed a simple Microsoft .NET application that uses the .NET Data Provider to obtain a result set from a database.

Example

The complete application can be found in the samples directory at *%SQLANYSAMP17%* `\SQLAnywhere\ADO.NET\SimpleWin32`.

In this section:

The Simple Sample Project Explained [page 108]

By examining the code from the Simple project, some key features of the SQL Anywhere .NET Data Provider are illustrated.

Related Information

Using the SQL Anywhere .NET Data Provider in a Microsoft Visual Studio Project [page 57]

1.3.3.1.1 The Simple Sample Project Explained

By examining the code from the Simple project, some key features of the SQL Anywhere .NET Data Provider are illustrated.

The Simple project uses the sample database, *demo.db*, which is located in your samples directory.

The Simple project is described a few lines at a time. Not all code from the sample is included here. To see all the code, open the project file *%SQLANYSAMP17%*`\SQLAnywhere\ADO.NET\SimpleWin32\Simple.sln`.

Declaring Controls

The following code declares a textbox named txtConnectString, a button named btnConnect, and a listbox named listEmployees.

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

Connecting to the Database

The btnConnect_Click method declares and initializes an SAConnection connection object.

```
SAConnection conn = new SAConnection("Data Source=SQL Anywhere 17 Demo;"
    + txtConnectString.Text);
```

The SAConnection object uses the connection string to connect to the sample database when the Open method is called.

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

Defining a Query

A SQL statement is executed using an SACommand object. The following code declares and creates a command object using the SACommand constructor. This constructor accepts a string representing the query to be executed, along with the SAConnection object that represents the connection that the query is executed on.

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

Displaying the Results

The results of the query are obtained using an SADataReader object. The following code declares and creates an SADataReader object using the ExecuteReader constructor. This constructor is a member of the SACommand object, cmd, that was declared previously. ExecuteReader sends the command text to the connection for execution and builds an SADataReader.

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

The following code loops through the rows held in the SADataReader object and adds them to the listbox control. Each time the Read method is called, the data reader gets another row back from the result set. A new item is added to the listbox for each row that is read. The data reader uses the GetString method with an argument of 0 to get the first column from the result set row.

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

Finishing Off

The following code at the end of the method closes the data reader and connection objects.

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

Error Handling

Any errors that occur during execution and that originate with .NET Data Provider objects are handled by displaying them in a window. The following code catches the error and displays its message:

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

```
}
```

Related Information

Samples Directory
The SQL Anywhere Sample Database (demo.db)

1.3.3.2 Tutorial: Using the Table Viewer Code Sample

Use the TableViewer project as an example of how to connect to a database, execute SQL statements, and display the results using a DataGrid object using the .NET Data Provider.

Prerequisites

You must have Microsoft Visual Studio and the Microsoft .NET Framework installed on your computer.

You must have the SELECT ANY TABLE system privilege.

Context

The TableViewer project is included with the samples. The Table Viewer project is more complex than the Simple project. You can use it to connect to a database, select a table, and execute SQL statements on the database.

Procedure

1. Start Microsoft Visual Studio.
2. Click *File* > *Open* > *Project*.
3. Browse to *%SQLANYSAMP17%*`\SQLAnywhere\ADO.NET\TableViewer` and open the `TableViewer.sln` project.
4. To use the .NET Data Provider in a project, you must add a reference to the Data Provider DLL. This has already been done in the Table Viewer code sample. To view the reference to the Data Provider (`Sap.Data.SQLAnywhere`), open the *References* folder in the *Solution Explorer* window.
5. You must also add a `using` directive to your source code to reference the Data Provider classes. This has already been done in the Table Viewer code sample. To view the `using` directive:
   - Open the source code for the project. In the *Solution Explorer* window, right-click `TableViewer.cs` and click *View Code*.

- In the `using` directives in the top section, you see the following line:

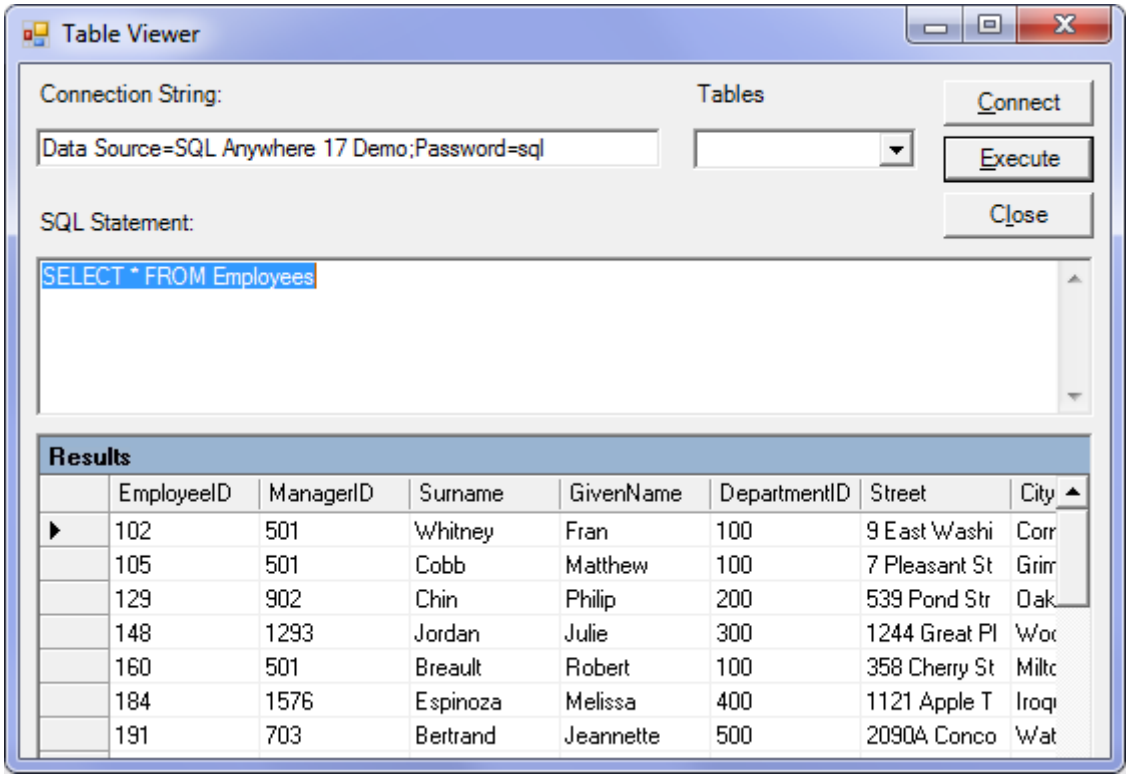
  ```
  using Sap.Data.SQLAnywhere;
  ```

  This line is required for Microsoft C# projects. If you are using Microsoft Visual Basic, add the equivalent `Imports` line to your source code.

6. Click *Debug* > *Start Without Debugging* or press Ctrl+F5 to run the Table Viewer sample.

   The application connects to the sample database.
7. In the *Table Viewer* window, modify the *Connection String* text box and enter a valid database password for the DBA user. Click *Connect*.
8. In the *Table Viewer* window, click *Execute*.

   The application retrieves the data from the Employees table in the sample database and puts the query results in the *Results* datagrid, as follows:



   You can also execute other SQL statements from this application: type a SQL statement in the *SQL Statement* pane, and then click *Execute*.
9. Close the *Table Viewer* window to shut down the application and disconnect from the sample database. This also shuts down the database server.

Results

You have built and executed a Microsoft .NET application that uses the .NET Data Provider to connect to a database, execute SQL statements, and display the results using a DataGrid object.

Example

The complete application can be found in the samples directory at *%SQLANYSAMP17%* `\SQLAnywhere\ADO.NET\TableViewer`.

In this section:

The Table Viewer Sample Project Explained [page 112]
By examining the code from the Table Viewer project, some key features of the SQL Anywhere .NET Data Provider are illustrated.

Related Information

Using the SQL Anywhere .NET Data Provider in a Microsoft Visual Studio Project [page 57]

1.3.3.2.1 The Table Viewer Sample Project Explained

By examining the code from the Table Viewer project, some key features of the SQL Anywhere .NET Data Provider are illustrated.

The Table Viewer project uses the sample database, *demo.db*, which is located in the samples directory.

The Table Viewer project is described a few lines at a time. Not all code from the sample is included here. To see all the code, open the project file *%SQLANYSAMP17%* `\SQLAnywhere\ADO.NET\TableViewer\TableViewer.sln`.

Declaring Controls

The following code declares a couple of Labels named label1 and label2, a TextBox named txtConnectString, a button named btnConnect, a TextBox named txtSQLStatement, a button named btnExecute, and a DataGrid named dgResults.

```
private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox txtConnectString;
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;
```

Declaring a Connection Object

The SAConnection type is used to declare an uninitialized connection object. The SAConnection object is used to represent a unique connection to a data source.

```
private SAConnection _conn;
```

Connecting to the Database

The Text property of the txtConnectString object has a default value of "Data Source=SQL Anywhere 17 Demo;Password=sql". This value can be overridden by the application user by typing a new value into the txtConnectString text box. You can see how this default value is set by opening up the region in `TableViewer.cs` labeled Windows Form Designer Generated Code. In this region, you find the following line of code.

```
this.txtConnectString.Text = "Data Source=SQL Anywhere 17 Demo;Password=passwd";
```

Later, the SAConnection object uses the connection string to connect to a database. The following code creates a new connection object with the connection string using the SAConnection constructor. It then establishes the connection by using the Open method.

```
_conn = new SAConnection( txtConnectString.Text );
_conn.Open();
```

Defining a Query

The Text property of the txtSQLStatement object has a default value of "SELECT * FROM Employees". This value can be overridden by the application user by typing a new value into the txtSQLStatement text box.

The SQL statement is executed using an SACommand object. The following code declares and creates a command object using the SACommand constructor. This constructor accepts a string representing the query to be executed, along with the SAConnection object that represents the connection that the query is executed on.

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

Displaying the Results

The results of the query are obtained using an SADataReader object. The following code declares and creates an SADataReader object using the ExecuteReader constructor. This constructor is a member of the SACommand object, cmd, that was declared previously. ExecuteReader sends the command text to the connection for execution and builds an SADataReader.

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

The following code connects the SADataReader object to the DataGrid object, which causes the result columns to appear on the screen. The SADataReader object is then closed.

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

Error Handling

If there is an error when the application attempts to connect to the database or when it populates the tables combo box, the following code catches the error and displays its message:

```
try
{
  _conn = new SAConnection( 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" );
}
```

Related Information

Samples Directory
The SQL Anywhere Sample Database (demo.db)

1.3.3.3 Tutorial: Developing a Simple .NET Database Application with Microsoft Visual Studio

This tutorial takes you though the steps of building the Simple Viewer .NET database application using Visual Studio.

Prerequisites

You must have the SELECT ANY TABLE system privilege. To insert new rows, you must be the owner of the table, or have the INSERT ANY TABLE privilege, or have INSERT privilege on the table. To change the content

of existing rows, you must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege. To delete rows, you must be the owner of the table, or have SELECT and DELETE privileges on the table.

1. Lesson 1: Creating a Table Viewer [page 115]
   Use Microsoft Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider to create an application that accesses one of the tables in the sample database, allowing you to examine rows and perform updates.
2. Lesson 2: Adding a Synchronizing Data Control [page 119]
   Add a datagrid control to the form developed in the previous lesson.

1.3.3.3.1 Lesson 1: Creating a Table Viewer

Use Microsoft Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider to create an application that accesses one of the tables in the sample database, allowing you to examine rows and perform updates.

Prerequisites

You must have Microsoft Visual Studio and the .NET Framework installed on your computer.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This tutorial is based on Microsoft Visual Studio and the .NET Framework. The complete application can be examined by opening the project *%SQLANYSAMP17%*.

Procedure

1. Start Microsoft Visual Studio.
2. Click *File* > *New* > *Project*.

   The *New Project* window appears.
   a. In the left pane of the *New Project* window, click either *Visual Basic* or *Visual C#* for the programming language.
   b. From the *Windows* subcategory, click *Windows Application* (VS 2005) or *Windows Forms Application* (VS 2008 or later).
   c. Select the .NET Framework version corresponding to the SQL Anywhere .NET Data Provider installed. For example, select *.NET Framework 4.5* from the dropdown list if the version you have installed is 4.5.

d. In the project *Name* field, type **`MySimpleViewer`**.
e. Click *OK* to create the new project.

3. Click *View* > *Server Explorer*.
4. In the *Server Explorer* window, right-click *Data Connections* and click *Add Connection*.
5. In the *Add Connection* window:
   a. If you have used *Add Connection* before, then click *Change* to change the data source.
   b. You see a list of data sources. Select *.NET Framework Data Provider for SQL Anywhere 17* from the list of data sources presented and click *OK*.
   c. Under *Data source*, click *ODBC data source* and select or type ***SQL Anywhere 17 Demo***.

> **Note**
>
> When using the Microsoft Visual Studio Add Connection wizard on 64-bit Windows, only the 64-bit System Data Source Names (DSN) are included with the User Data Source Names. Any 32-bit System Data Source Names are not displayed. In the Microsoft Visual Studio 32-bit design environment, the Test Connection button will attempt to establish a connection using the 32-bit equivalent of the 64-bit System DSN. If the 32-bit System DSN does not exist, the test will fail.

   d. Under *Login information*, in the *Password* field, enter the sample database password (*sql*).
   e. Click *Test Connection* to verify that you can connect to the sample database.
   f. Click *OK*.

   A new connection named *SQL Anywhere.demo17* appears in the *Server Explorer* window.
6. Expand the *SQL Anywhere.demo17* connection in the *Server Explorer* window until you see the table names.

   (Microsoft Visual Studio 2005 only) Try the following:

   a. Right-click the Products table and click *Show Table Data*.

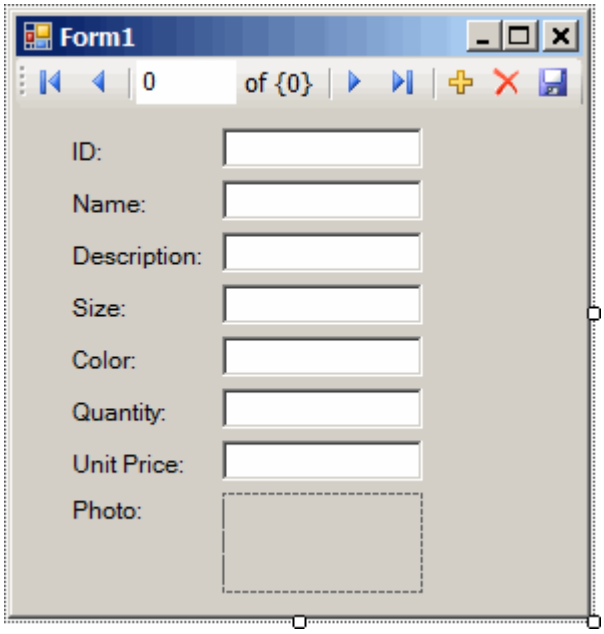
      This shows the rows and columns of the Products table in a window.
   b. Close the table data window.
7. Click *Data* > *Add New Data Source*. For later versions of Microsoft Visual Studio, this is *Project* > *Add New Data Source*.
8. In the *Data Source Configuration Wizard*, do the following:
   a. On the *Data Source Type* page, click *Database*, then click *Next*.
   b. (Microsoft Visual Studio 2010 or later) On the *Database Model* page, click *Dataset*, then click *Next*.
   c. On the *Data Connection* page, click *SQL Anywhere.demo 17*, click *Yes, include sensitive data in the connection string*, then click *Next*.
   d. On the *Save the Connection String* page, make sure that *Yes, save the connection as* is chosen and click *Next*.
   e. On the *Choose Your Database Objects* page, click *Tables*, then click *Finish*.
9. Click *Data* > *Show Data Sources*. For later versions of Microsoft Visual Studio, this might be *View* > *Other Windows* > *Data Sources*.

   The *Data Sources* window appears.

   Expand the Products table in the *Data Sources* window.

   a. Click *Products*, then click *Details* from the dropdown list.
   b. Click *Photo*, then click *Picture Box* from the dropdown list.

c. Click *Products* and drag it to your form (Form1).



A dataset control and several labeled text fields appear on the form.

10. On the form, click the picture box next to Photo.
    a. Change the shape of the box to a square.
    b. Click the right-arrow in the upper-right corner of the picture box.

       The *Picture Box Tasks* window opens.
    c. From the *Size Mode* dropdown list, click *Zoom*.
    d. To close the *Picture Box Tasks* window, click anywhere outside the window.
11. Build and run the project.
    a. Click *Build* > *Build Solution*.
    b. Click *Debug* > *Start Debugging*.

       The application connects to the sample database and displays the first row of the Products table in the text boxes and picture box.

c. You can use the buttons on the control to scroll through the rows of the result set.
d. You can go directly to a row in the result set by entering the row number in the scroll control.
e. You can update values in the result set using the text boxes and save them by clicking the *Save Data* button.

12. Shut down the application and then save your project.

Results

You have now created a simple, yet powerful, Microsoft .NET application using Microsoft Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider.

Next Steps

Proceed to the next lesson, where you add a datagrid control to the form developed in this lesson.

Task overview: Tutorial: Developing a Simple .NET Database Application with Microsoft Visual Studio [page 114]

Next task: Lesson 2: Adding a Synchronizing Data Control [page 119]

1.3.3.3.2 Lesson 2: Adding a Synchronizing Data Control

Add a datagrid control to the form developed in the previous lesson.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This control updates automatically as you navigate through the result set.

The complete application can be examined by opening the project *%SQLANYSAMP17%*`\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln`.

Procedure

1. Start Microsoft Visual Studio and load your MySimpleViewer project.
2. Right-click *DataSet1* in the *Data Sources* window and click *Edit DataSet with Designer*.
3. Right-click an empty area in the *DataSet Designer* window and click *Add* ▶ *TableAdapter*.
4. In the *TableAdapter Configuration Wizard*:
   a. On the *Choose Your Data Connection* page, click *Next*.
   b. On the *Choose A Command Type* page, click *Use SQL statements*, then click *Next*.
   c. On the *Enter a SQL Statement* page, click *Query Builder*.
   d. On the *Add Table* window, click the *Views* tab, then click *ViewSalesOrders*, and then click *Add*.
   e. Click *Close* to close the *Add Table* window.
5. Expand the *Query Builder* window so that all sections of the window are visible.
   a. Expand the *ViewSalesOrders* window so that all the checkboxes are visible.
   b. Click *Region*.
   c. Click *Quantity*.
   d. Click *ProductID*.
   e. In the grid below the *ViewSalesOrders* window, clear the checkbox under *Output* for the ProductID column.
   f. For the ProductID column, type a question mark (?) in the *Filter* cell and then click anywhere else on the form. This generates a WHERE clause for ProductID.

   A SQL query has been built that looks like the following:

```
SELECT   Region, Quantity
```

```
FROM     GROUPO.ViewSalesOrders
WHERE    (ProductID = :Param1)
```

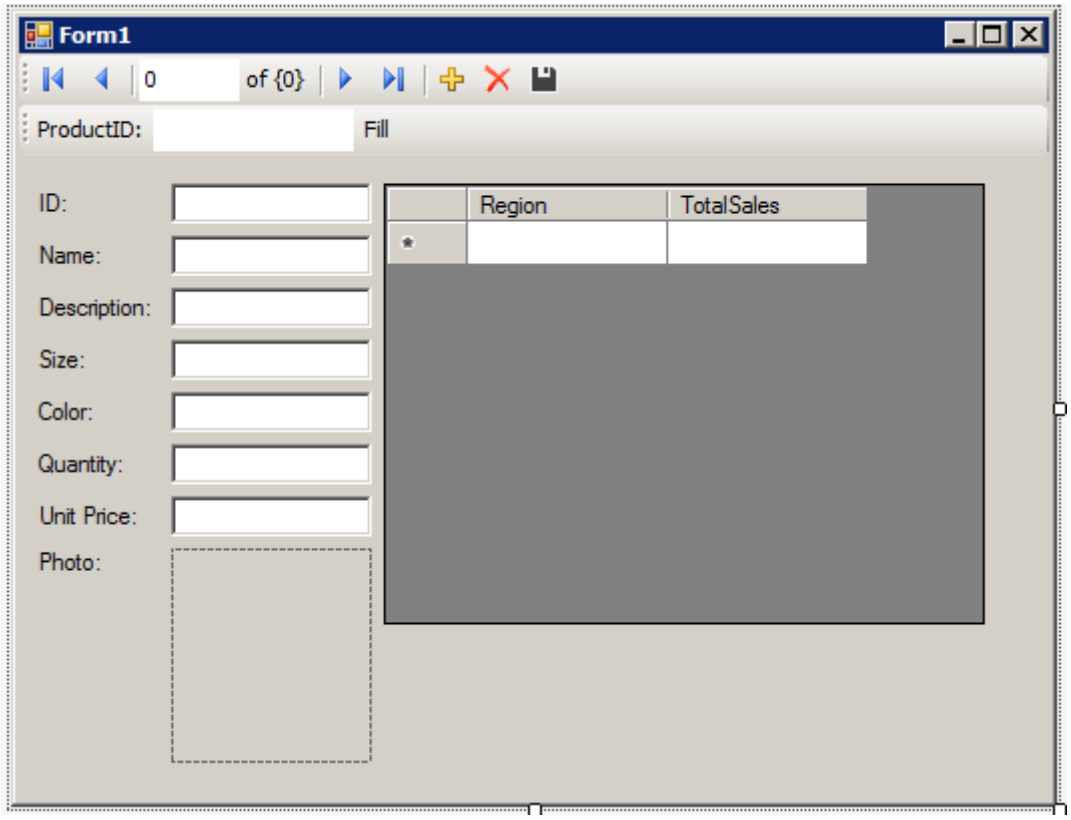
6. Modify the SQL query as follows:
   a. Change `Quantity` to `SUM(Quantity) AS TotalSales`.
   b. Add `GROUP BY Region` to the end of the query following the WHERE clause.

   The modified SQL query now looks like this:

```
SELECT   Region, SUM(Quantity) as TotalSales
FROM     GROUPO.ViewSalesOrders
WHERE    (ProductID = :Param1)
GROUP BY Region
```

7. Click *OK*.
8. Click *Finish*.

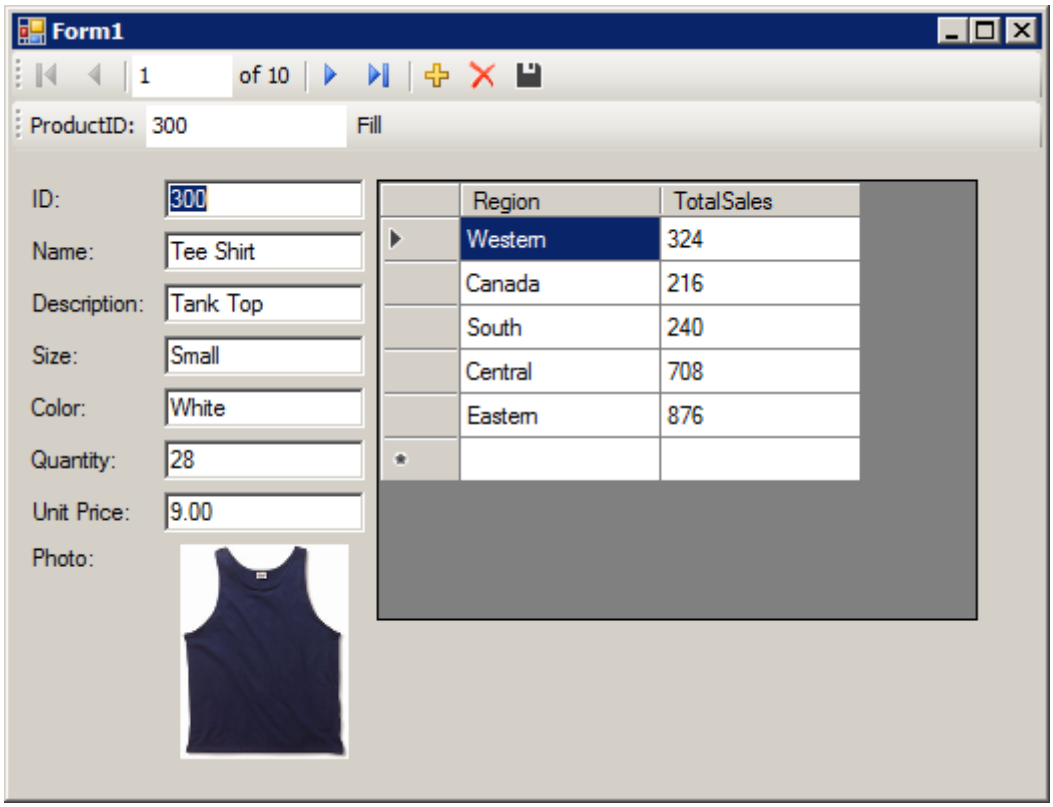
   A new *TableAdapter* called *ViewSalesOrders* has been added to the *DataSet Designer* window.
9. Click the form design tab (Form1.cs [Design]).
   - Stretch the form to the right to make room for a new control.
10. Expand ViewSalesOrders in the *Data Sources* window.
    a. Click *ViewSalesOrders* and click *DataGridView* from the dropdown list.
    b. Click *ViewSalesOrders* and drag it to your form (Form1).



   A datagrid view control appears on the form.
11. Build and run the project.
    - Click *Build* > *Build Solution*.

- Click *Debug* > *Start Debugging*.
- In the *Param1* or *ProductID* (Microsoft Visual Studio 2010 or later) text box, enter a product ID number such as 300 and click *Fill*.

The datagrid view displays a summary of sales by region for the product ID entered.



You can also use the other control on the form to move through the rows of the result set.

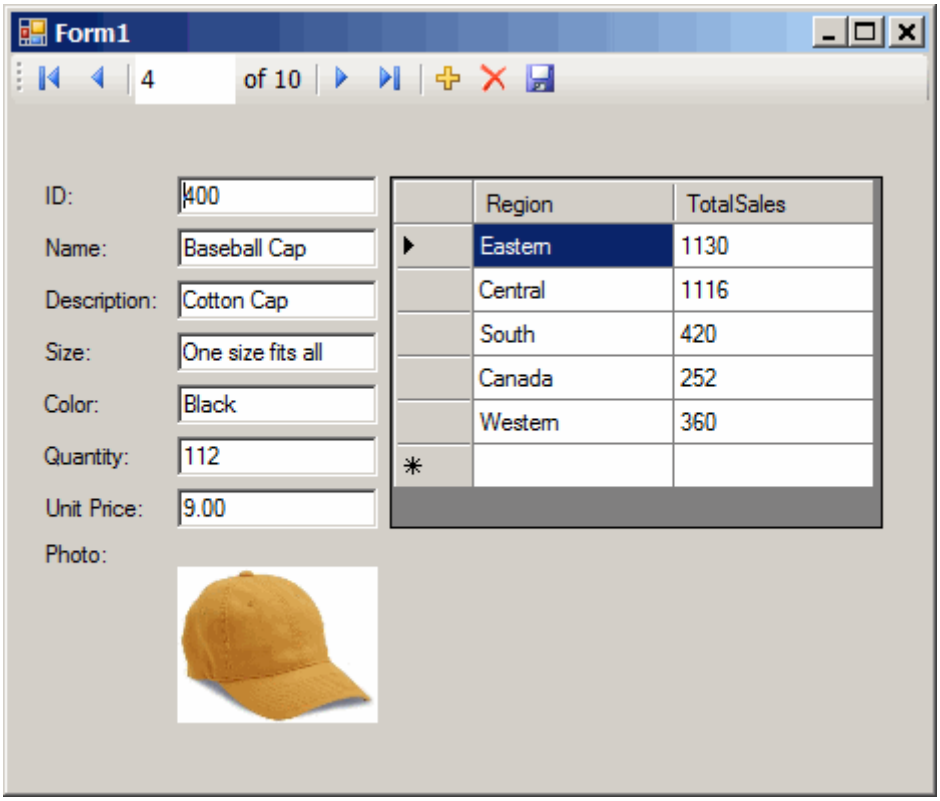
It would be ideal, however, if both controls could stay synchronized with each other. The next few steps show how to do this.

12. Shut down the application and then save your project.
13. Delete the Fill strip on the form since you do not need it.
    - On the design form (Form1), right-click the Fill strip to the right of the word *Fill*, then click *Delete*.

    The Fill strip is removed from the form.
14. Synchronize the two controls as follows.
    a. On the design form (Form1), right-click the ID text box, then click *Properties*.
    b. Click the *Events* button (it appears as a lightning bolt).
    c. Scroll down until you find the *TextChanged* event.
    d. Click *TextChanged*, then click *fillToolStripButton_Click* from the dropdown list. If you are using Microsoft Visual Basic, the event is called *FillToolStripButton_Click*.
    e. Double-click *fillToolStripButton_Click* and the form's code window opens on the `fillToolStripButton_Click` event handler.
    f. Find the reference to `param1ToolStripTextBox` or `productIDToolStripTextBox` (Microsoft Visual Studio 2010) and change this to `iDTextBox`. If you are using Microsoft Visual Basic, the text box is called `IDTextBox`.

g. Rebuild and run the project.

15. The application form now appears with a single navigation control.
    - The datagrid view displays an updated summary of sales by region corresponding to the current product as you move through the result set.



16. Shut down the application and then save your project.

Results

You have now added a control that updates automatically as you navigate through the result set.

In this tutorial, you saw how the powerful combination of Microsoft Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider can be used to create database applications.

Task overview: Tutorial: Developing a Simple .NET Database Application with Microsoft Visual Studio [page 114]

Previous task: Lesson 1: Creating a Table Viewer [page 115]

1.3.4 SQL Anywhere ASP.NET Providers

The SQL Anywhere ASP.NET providers replace the standard ASP.NET providers for Microsoft SQL Server, and allow you to run your website using SQL Anywhere.

There are five providers:

Membership Provider

The membership provider provides authentication and authorization services. Use the membership provider to create new users and passwords, and validate the identity of users.

Roles Provider

The roles provider provides methods for creating roles, adding users to roles, and deleting roles. Use the roles provider to assign users to groups and manage permissions.

Profiles Provider

The profiles provider provides methods for reading, storing, and retrieving user information. Use the profiles provider to save user preferences.

Web Parts Personalization Provider

The web parts personalization provider provides methods for loading and storing the personalized content and layout of web pages. Use the web parts personalization provider to allow users to create personalized views of your website.

Health Monitoring Provider

The health monitoring provider provides methods for monitoring the status of deployed web applications. Use the health monitoring provider to monitor application performance, identify failing applications or systems, and log and review significant events.

The database server schema used by the SQL Anywhere ASP.NET providers is identical to the schema used by the standard ASP.NET providers. The methodology used to manipulate and store data are identical.

When you have finished setting up the SQL Anywhere ASP.NET providers, you can use the Microsoft Visual Studio ASP.NET Web Site Administration Tool to create and manage users and roles. You can also use the Microsoft Visual Studio Login, LoginView, and PasswordRecovery tools to add security to your web site. Use the static wrapper classes to access more advanced provider functions, or to make your own login controls.

In this section:

ASP.NET Provider Database Configuration [page 124]
To implement the SQL Anywhere ASP.NET providers, you can add the required schema to a new or existing database.

Connection String Registration [page 126]
There are two methods for registering the connection string: using an ODBC data source or using a full connection string.

Registration of SQL Anywhere ASP.NET Providers [page 126]
Your web application must be configured to use the SQL Anywhere ASP.NET providers and not the default providers.

Membership Provider XML Attributes [page 128]
There are several XML attributes used to describe a membership provider.

Roles Provider Table Schema [page 129]

SARoleProvider stores role information in the aspnet_Roles table of the provider database. The namespace associated with SARoleProvider is Sap.Web.Security. Each record in the Roles table corresponds to one role.

Profile Provider Table Schema [page 130]
SAProfileProvider stores profile data in the aspnet_Profile table of the provider database. The namespace associated with SAProfileProvider is Sap.Web.Security. Each record in the Profile table corresponds to one user's persisted profile properties.

Web Part Personalization Provider Table Schema [page 130]
SAPersonalizationProvider preserves personalized user content in the aspnet_Paths table of the provider database. The namespace associated with SAPersonalizationProvider is Sap.Web.Security.

Health Monitoring Provider Table Schema [page 131]
SAWebEventProvider logs web events in the aspnet_WebEvent_Events table of the provider database. The namespace associated with SAWebEventProvider is Sap.Web.Security. Each record in the WebEvents_Events table corresponds to one web event.

1.3.4.1 ASP.NET Provider Database Configuration

To implement the SQL Anywhere ASP.NET providers, you can add the required schema to a new or existing database.

To add the schema to a database, run `SASetupAspNet.exe`. When executed, `SASetupAspNet.exe` connects to a database and creates tables and stored procedures required by the SQL Anywhere ASP.NET providers. All SQL Anywhere ASP.NET provider resources are prefixed with `aspnet_`. To minimize naming conflicts with existing database resources you can install provider database resources under any database user.

You can use a wizard or the command line to run `SASetupAspNet.exe`. To access the wizard, run the application, or execute a command line statement without arguments. When using the command line to access the `SASetupAspNet.exe`, use the question mark (-?) argument to display detailed help for configuring the database.

Setting Up the Database Connection

Connect with a user having the appropriate privileges for the database objects they are creating. The following minimum set of privileges are required.

- 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

Specifying a Resource Owner

The wizard and command line (*-U <user>*) allow you to specify the owner of the new resources (tables and procedures). By default, the owner is DBA. You can specify a different owner using the wizard. When you specify connection strings for the SQL Anywhere ASP.NET providers, the UserID (UID) must match the user you specify here. You do not need to grant the specified user any extra privileges; the user owns the resources and has full privileges on the tables and stored procedures that are created for this user by the wizard.

Selecting Features and Preserving Data

You can add or remove specific features. Common components are installed automatically. Selecting *Remove* for an uninstalled feature has no effect; selecting *Add* for a feature already installed reinstalls the feature. By default, the data in tables associated with the selected feature is preserved. If a user significantly changes the schema of a table, it might not be possible to automatically preserve the data stored in it. If a clean reinstall is required, data preservation can be turned off.

Membership and roles providers should be installed together. The effectiveness of the Microsoft Visual Studio ASP.NET Web Site Administration Tool is reduced when the membership provider is not installed with the roles provider.

Further Reading

A white paper called *Tutorial: Creating an ASP.NET Web Page Using SQL Anywhere* is available to demonstrate how to use SQL Anywhere and Microsoft Visual Studio 2010 to build a database-driven ASP.NET web site.

Related Information

SQL Anywhere and Microsoft .NET

1.3.4.2 Connection String Registration

There are two methods for registering the connection string: using an ODBC data source or using a full connection string.

- You can register an ODBC data source using the ODBC Data Source Administrator, and reference it by name.
- You can specify a full connection string. For example:

```
connectionString="SERVER=MyServer;DBN=MyDatabase;UID=DBA;PWD=passwd"
```

When you add the `<connectionStrings>` element to the `web.config` file, the connection string and its provider can be referenced by the application. Updates can be implemented in a single location.

XML Code Sample for Connection String Registration

```
<connectionStrings>
   <add name="MyConnectionString"
      connectionString="DSN=MyDataSource"
      providerName="Sap.Data.SQLAnywhere"/>
</connectionStrings>
```

1.3.4.3 Registration of SQL Anywhere ASP.NET Providers

Your web application must be configured to use the SQL Anywhere ASP.NET providers and not the default providers.

To register the SQL Anywhere ASP.NET providers:

- Add a reference to the Sap.Web.Security assembly to your web site.
- Add an entry for each provider to the `<system.web>` element in `web.config` file.
- Add the name of the SQL Anywhere ASP.NET provider to the `defaultProvider` attribute in the application.

The provider database can store data for multiple applications. For each application, the `applicationName` attribute must be the same for each SQL Anywhere ASP.NET provider. If you do not specify an `applicationName` value, an identical name is assigned to each provider in the provider database.

To reference a previously registered connection string, replace the `connectionString` attribute with the `connectionStringName` attribute.

XML Code Sample for Membership Provider Registration

```
<membership defaultProvider="SAMembershipProvider">
   <providers>
```

```
      <add name="SAMembershipProvider"
         type="Sap.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>
```

XML Code Sample for Roles Provider Registration

```
<roleManager enabled="true" defaultProvider="SARoleProvider">
   <providers>
      <add name="SARoleProvider"
         type="Sap.Web.Security.SARoleProvider"
         connectionStringName="MyConnectionString"
         applicationName="MyApplication"
         commandTimeout="30" />
   </providers>
</roleManager>
```

XML Code Sample for Profiles Provider Registration

```
<profile defaultProvider="SAProfileProvider">
   <providers>
      <add name="SAProfileProvider"
         type="Sap.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>
```

XML Code Sample for Personalization Provider Registration

```
<webParts>
```

```
   <personalization defaultProvider="SAPersonalizationProvider">
      <providers>
         <add name="SAPersonalizationProvider"
            type="Sap.Web.Security.SAPersonalizationProvider"
            connectionStringName="MyConnectionString"
            applicationName="MyApplication"
            commandTimeout="30" />
      </providers>
   </personalization>
</webParts>
```

XML Code Sample for Health Monitoring Provider Registration

```
<healthMonitoring enabled="true">
                  ...
   <providers>
      <add name="SAWebEventProvider"
         type="Sap.Web.Security.SAWebEventProvider"
         connectionStringName="MyConnectionString"
         commandTimeout="30"
         bufferMode="Notification"
         maxEventDetailsLength="Infinite" /
   </providers>
               ...
</healthMonitoring>
```

Related Information

Membership Provider XML Attributes [page 128]
Roles Provider Table Schema [page 129]
Profile Provider Table Schema [page 130]
Web Part Personalization Provider Table Schema [page 130]
Health Monitoring Provider Table Schema [page 131]

How To: Use Health Monitoring in ASP.NET 2.0

1.3.4.4 Membership Provider XML Attributes

There are several XML attributes used to describe a membership provider.

Column Name	Description
name	The name of the provider.
type	`Sap.Web.Security.SAMembershipProvider`
connectionStringName	The name of a connection string specified in the `<connectionStrings>` element.

Column Name	Description
connectionString	The connection string. Optional. Required if `connectionStringName` is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.
enablePasswordReset	Valid entries are true or false.
enablePasswordRetrieval	Valid entries are true or false.
maxInvalidPasswordAttempts	Valid entries are true or false.
minRequiredNonalphanumericCharacters	The minimum number of special characters that must be present in a valid password.
minRequiredPasswordLength	The minimum length required for a password.
passwordAttemptWindow	The time window between which consecutive failed attempts to provide a valid password or password answer are tracked.
passwordFormat	Valid entries are Clear, Hashed, or Encrypted.
requiresQuestionAndAnswer	Valid entries are true or false.
requiresUniqueEmail	Valid entries are true or false.
passwordStrengthRegularExpression	The regular expression used to evaluate a password.

1.3.4.5 Roles Provider Table Schema

SARoleProvider stores role information in the aspnet_Roles table of the provider database. The namespace associated with SARoleProvider is Sap.Web.Security. Each record in the Roles table corresponds to one role.

SARoleProvider uses the aspnet_UsersInRoles table to map roles to users.

Column Name	Description
name	The name of the provider.
type	`Sap.Web.Security.SARoleProvider`
connectionStringName	The name of a connection string specified in the `<connectionStrings>` element.
connectionString	The connection string. Optional. Required if `connectionStringName` is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.

1.3.4.6 Profile Provider Table Schema

SAProfileProvider stores profile data in the aspnet_Profile table of the provider database. The namespace associated with SAProfileProvider is Sap.Web.Security. Each record in the Profile table corresponds to one user's persisted profile properties.

Column Name	Description
name	The name of the provider.
type	`Sap.Web.Security.SAProfileProvider`
connectionStringName	The name of a connection string specified in the `<connectionStrings>` element.
connectionString	The connection string. Optional. Required if `connectionStringName` is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.

1.3.4.7 Web Part Personalization Provider Table Schema

SAPersonalizationProvider preserves personalized user content in the aspnet_Paths table of the provider database. The namespace associated with SAPersonalizationProvider is Sap.Web.Security.

SAPersonalizationProvider uses the aspnet_PersonalizationPerUser and aspnet_PersonalizationAllUsers tables to define the path for which the Web Parts personalization state has been saved. The PathID columns point to the column of the same name in the aspnet_Paths table.

Column name	Description
name	The name of the provider.
type	`Sap.Web.Security.SAPersonalizationProvider`
connectionStringName	The name of a connection string specified in the `<connectionStrings>` element.
connectionString	The connection string. Optional. Required if `connectionStringName` is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.

1.3.4.8 Health Monitoring Provider Table Schema

SAWebEventProvider logs web events in the aspnet_WebEvent_Events table of the provider database. The namespace associated with SAWebEventProvider is Sap.Web.Security. Each record in the WebEvents_Events table corresponds to one web event.

Column Name	Description
name	The name of the provider.
type	`Sap.Web.Security.SAWebEventProvider`
connectionStringName	The name of a connection string specified in the `<connectionStrings>` element.
connectionString	The connection string. Optional. Required if `connectionStringName` is not specified.
commandTimeout	The timeout value, in seconds, for server calls.
maxEventDetailsLength	The maximum length of the details string for each event or Infinite

Related Information

How To: Use Health Monitoring in ASP.NET 2.0

1.3.5 SQL Anywhere .NET API Reference

Use SQL Anywhere with .NET, including the API for the SQL Anywhere .NET Data Provider.

The SQL Anywhere .NET API reference is available in the *SQL Anywhere- .NET API Reference* at https://help.sap.com/viewer/e2cc7ae777e347529ca8d8db9258fe3c/LATEST/en-US.

1.4 OLE DB and ADO Development

An OLE DB provider for OLE DB and ADO applications is included with the software.

OLE DB is a set of Component Object Model (COM) interfaces developed by Microsoft, which provide applications with uniform access to data stored in diverse information sources and that are used to implement additional database services. These interfaces support the amount of DBMS functionality appropriate to the data store, enabling it to share its data.

ADO is an object model for programmatically accessing, editing, and updating a wide variety of data sources through OLE DB system interfaces. ADO is also developed by Microsoft. Most developers using the OLE DB programming interface do so by writing to the ADO API rather than directly to the OLE DB API.

Do not confuse the ADO interface with ADO.NET. ADO.NET is a separate interface.

Refer to the Microsoft Developer Network for documentation on OLE DB and ADO programming. For product-specific information about OLE DB and ADO development, use this document.

In this section:

OLE DB [page 132]
OLE DB is a data access model from Microsoft. It uses the Component Object Model (COM) interfaces and, unlike ODBC, OLE DB does not assume that the data source uses a SQL query processor.

ADO Programming with the OLE DB Provider [page 133]
ADO (Microsoft ActiveX Data Objects) is a data access object model exposed through an Automation interface, which allows client applications to discover the methods and properties of objects at runtime without any prior knowledge of the object.

OLE DB Connection Parameters [page 141]
OLE DB connection parameters are defined by Microsoft. The OLE DB provider supports a subset of these connection parameters.

OLE DB Connection Pooling [page 144]
The .NET Framework Data Provider for OLE DB automatically pools connections using OLE DB session pooling.

OLE DB Isolation Levels [page 145]
The OLE DB provider supports additional isolation levels. In the list below, the first 4 are standard OLE DB isolation levels. The last three are supported by the database server.

Microsoft Linked Servers [page 145]
A Microsoft Linked Server can be created that uses the OLE DB provider to obtain access to a database. SQL queries can be issued using either the Microsoft four-part table referencing syntax or the Microsoft OPENQUERY SQL function.

Supported OLE DB Interfaces [page 149]
The OLE DB API consists of a set of interfaces.

OLE DB Provider Registration [page 154]
When the SAOLEDB provider is installed using the installer provided with the software, the provider registers itself.

Related Information

SQL Anywhere .NET Data Provider [page 54]

1.4.1 OLE DB

OLE DB is a data access model from Microsoft. It uses the Component Object Model (COM) interfaces and, unlike ODBC, OLE DB does not assume that the data source uses a SQL query processor.

An **OLE DB provider**, named SAOLEDB, is included with the software. This provider is available for supported Windows platforms.

You can also access a database using the Microsoft OLE DB Provider for ODBC (MSDASQL), together with the ODBC driver included with SQL Anywhere.

Using SAOLEDB instead of MSDASQL has several benefits:

- Some features, such as updating through a cursor, are not available using the OLE DB/ODBC bridge.
- If you use the OLE DB provider, ODBC is not required in your deployment.
- MSDASQL allows OLE DB clients to work with any ODBC driver, but does not guarantee that you can use the full range of functionality of each ODBC driver. Using SAOLEDB, you can get full access to database software features from OLE DB programming environments.

In this section:

OLE DB Platform Support [page 133]
The OLE DB provider is designed to work with Microsoft Data Access Components (MDAC) 2.8 and later versions.

Distributed Transactions in OLE DB [page 133]
The OLE DB driver can be used as a resource manager in a distributed transaction environment.

1.4.1.1 OLE DB Platform Support

The OLE DB provider is designed to work with Microsoft Data Access Components (MDAC) 2.8 and later versions.

1.4.1.2 Distributed Transactions in OLE DB

The OLE DB driver can be used as a resource manager in a distributed transaction environment.

Related Information

Three-tier Computing and Distributed Transactions [page 716]

1.4.2 ADO Programming with the OLE DB Provider

ADO (Microsoft ActiveX Data Objects) is a data access object model exposed through an Automation interface, which allows client applications to discover the methods and properties of objects at runtime without any prior knowledge of the object.

Automation allows scripting languages like Microsoft Visual Basic to use a standard data access object model. ADO uses OLE DB to provide data access.

Using the OLE DB provider, you get full access to database software features from an ADO programming environment.

Using Microsoft Visual Basic and ADO, basic tasks such as connecting to a database, executing SQL queries, and retrieving result sets are demonstrated. This is not a complete guide to programming using ADO.

Code samples can be found in the *%SQLANYSAMP17%*`\SQLAnywhere\VBSampler\vbsampler.sln` project file.

For information about programming in ADO, see your development tool documentation.

In this section:

How to Connect to a Database Using the Connection Object [page 134]
The following Microsoft Visual Basic routine connects to a database using the Connection object.

How to Execute Statements Using the Command Object [page 136]
The following routine sends a simple SQL statement to the database using the Command object.

How to Obtain Result Sets Using the Recordset Object [page 137]
The ADO Recordset object represents the result set of a query. You can use it to view data from a database.

The Recordset Object [page 138]
The ADO Recordset represents a cursor. You can choose the type of cursor by declaring a CursorType property of the Recordset object before you open the Recordset. The choice of cursor type controls the actions you can take on the Recordset and has performance implications.

Row Updates Through a Cursor Using the Recordset Object [page 139]
The OLE DB provider lets you update a result set through a cursor. This capability is not available through the MSDASQL provider.

ADO Transactions [page 140]
By default, any change you make to the database using ADO is committed when it is executed. This includes explicit updates, and the UpdateBatch method on a Recordset.

1.4.2.1 How to Connect to a Database Using the Connection Object

The following Microsoft Visual Basic routine connects to a database using the Connection object.

Sample Code

You can try this routine by placing a command button named cmdTestConnection on a form, and pasting the routine into its Click event. Run the program and click the button to connect and then disconnect.

```
Private Sub cmdTestConnection_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdTestConnection.Click
```

```
    ' Declare variables
    Dim myPwd as String
    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 17 Demo;" + _
    "Password=" + myPwd
    myConn.Open()
    MsgBox("Connection succeeded")
    myConn.Close()
    Exit Sub
HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

Notes

The sample carries out the following tasks:

- It declares the variables used in the routine.
- It establishes a connection, using the OLE DB provider, to the sample database.
- It uses a Command object to execute a simple statement, which displays a message in the database server messages window.
- It closes the connection.

Related Information

OLE DB Connection Parameters [page 141]

1.4.2.2 How to Execute Statements Using the Command Object

The following routine sends a simple SQL statement to the database using the Command object.

Sample Code

You can try this routine by placing a command button named cmdUpdate on a form, and pasting the routine into its Click event. Run the program and click the button to connect, display a message in the database server messages window, and then disconnect.

```
Private Sub cmdUpdate_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdUpdate.Click

    ' Declare variables
    Dim myPwd as String
    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 17 Demo;" + _
    "Password=" + myPwd
    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
```

Notes

After establishing a connection, the example code creates a Command object, sets its CommandText property to an update statement, and sets its ActiveConnection property to the current connection. It then executes the update statement and displays the number of rows affected by the update in a window.

In this example, the update is sent to the database and committed when it is executed.

You can also perform updates through a cursor.

Related Information

ADO Transactions [page 140]
Row Updates Through a Cursor Using the Recordset Object [page 139]

1.4.2.3 How to Obtain Result Sets Using the Recordset Object

The ADO Recordset object represents the result set of a query. You can use it to view data from a database.

Sample Code

You can try this routine by placing a command button named cmdQuery on a form and pasting the routine into its Click event. Run the program and click the button to connect, display a message in the database server messages window, execute a query and display the first few rows in windows, and then disconnect.

```
Private Sub cmdQuery_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdQuery.Click
    ' Declare variables
    Dim myPwd as String
    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 17 Demo;" + _
    "Password=" + myPwd
    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
```

Notes

The Recordset object in this example holds the results from a query on the Customers table. The For loop scrolls through the first several rows and displays the CompanyName value for each row.

This is a simple example of using a cursor from ADO.

Related Information

The Recordset Object [page 138]

1.4.2.4 The Recordset Object

The ADO Recordset represents a cursor. You can choose the type of cursor by declaring a CursorType property of the Recordset object before you open the Recordset. The choice of cursor type controls the actions you can take on the Recordset and has performance implications.

Cursor Types

ADO has its own naming convention for cursor types.

The available cursor types, the corresponding cursor type constants, and the SQL Anywhere types they are equivalent to, are as follows:

ADO Cursor Type	ADO Constant	Database Software Type
Dynamic cursor	adOpenDynamic	Dynamic scroll cursor
Keyset cursor	adOpenKeyset	Scroll cursor
Static cursor	adOpenStatic	Insensitive cursor

ADO Cursor Type	ADO Constant	Database Software Type
Forward only	adOpenForwardOnly	No-scroll cursor

Sample Code

The following code sets the cursor type for an ADO Recordset object:

```
Dim myRS As New ADODB.Recordset
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic
```

Related Information

Cursor Types [page 25]
Cursor Properties [page 27]

1.4.2.5 Row Updates Through a Cursor Using the Recordset Object

The OLE DB provider lets you update a result set through a cursor. This capability is not available through the MSDASQL provider.

Updating Record Sets

You can update the database through a Recordset.

```
Private Sub cmdUpdateThroughCursor_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdUpdateThroughCursor.Click
    ' Declare variables
    Dim myPwd As String
    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 17 Demo;" + _
        "Password=" + myPwd
    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
```

Notes

If you use the adLockBatchOptimistic setting on the Recordset, the myRS.Update method does not make any changes to the database itself. Instead, it updates a local copy of the Recordset.

The myRS.UpdateBatch method makes the update to the database server, but does not commit it, because it is inside a transaction. If an UpdateBatch method was invoked outside a transaction, the change would be committed.

The myConn.CommitTrans method commits the changes. The Recordset object has been closed by this time, so there is no issue of whether the local copy of the data is changed or not.

1.4.2.6 ADO Transactions

By default, any change you make to the database using ADO is committed when it is executed. This includes explicit updates, and the UpdateBatch method on a Recordset.

However, the previous section illustrated that you can use the BeginTrans and RollbackTrans or CommitTrans methods on the Connection object to use transactions.

The transaction isolation level is set as a property of the Connection object. The IsolationLevel property can take on one of the following values:

ADO Isolation Level	Constant	Database Server Isolation Level
Unspecified	adXactUnspecified	Not applicable. Set to 0

ADO Isolation Level	Constant	Database Server Isolation Level
Chaos	adXactChaos	Unsupported. Set to 0
Browse	adXactBrowse	0
Read uncommitted	adXactReadUncommitted	0
Cursor stability	adXactCursorStability	1
Read committed	adXactReadCommitted	1
Repeatable read	adXactRepeatableRead	2
Isolated	adXactIsolated	3
Serializable	adXactSerializable	3
Snapshot	2097152	SNAPSHOT
Statement snapshot	4194304	STATEMENT SNAPSHOT
Readonly statement snapshot	8388608	READONLY STATEMENT SNAPSHOT

Related Information

Isolation Levels and Consistency

1.4.3 OLE DB Connection Parameters

OLE DB connection parameters are defined by Microsoft. The OLE DB provider supports a subset of these connection parameters.

A typical connection string looks like this:

```
"Provider=SAOLEDB;Data Source=myDsn;Initial Catalog=myDbn;User
ID=myUid;Password=myPwd"
```

Below are the OLE DB connection parameters that are supported by the provider. In some cases, OLE DB connection parameters are identical to (for example, Password) or resemble (for example, User ID) database server connection parameters. Note the use of spaces in many of these connection parameters.

Provider

This parameter is used to identify the OLE DB provider (SAOLEDB).

User ID

This connection parameter maps directly to the UserID (UID) connection parameter. For example: `User ID=DBA`.

Password

This connection parameter maps directly to the Password (PWD) connection parameter. For example: `Password=sql`.

Data Source

This connection parameter maps directly to the DataSourceName (DSN) connection parameter. For example: `Data Source=SQL Anywhere 17 Demo`.

Initial Catalog

This connection parameter maps directly to the DatabaseName (DBN) connection parameter. For example: `Initial Catalog=demo`.

Location

This connection parameter maps directly to the Host connection parameter. The parameter value has the same form as the Host parameter value. For example: `Location=localhost:4444`.

Downlevel

This connection parameter maps directly to the Delphi connection parameter. When set TRUE, this connection parameter forces the provider to map the SQL TIME data type to the less precise DBTYPE_DBTIME OLE DB data type (`Downlevel=true`). By default or when set FALSE, the provider maps the SQL TIME data type to DBTYPE_DBTIME2 (ordinal 145) which includes fractional seconds (`Downlevel=false`). Use the FALSE setting when migrating tables between relational database management systems.

When using the ODBC Data Source Administrator, check the *Delphi applications* box. Use this option for SAP PowerBuilder applications.

Extended Properties (or Location)

This connection parameter is used by OLE DB to pass in all the database server-specific connection parameters. For example: `Extended Properties="UserID=DBA;DBKEY=V3moj3952B;DBF=demo.db"`.

ADO uses this connection parameter to collect and pass in all the connection parameters that it does not recognize.

Some Microsoft connection windows have a field called *Prov String* or *Provider String*. The contents of this field are passed as the value to Extended Properties.

OLE DB Services

This connection parameter is not directly handled by the OLE DB provider. It controls connection pooling in ADO.

Prompt

This connection parameter governs how a connection attempt handles errors. The possible prompt values are 1, 2, 3, or 4. The meanings are DBPROMPT_PROMPT (1), DBPROMPT_COMPLETE (2), DBPROMPT_COMPLETEREQUIRED (3), and DBPROMPT_NOPROMPT (4).

The default prompt value is 4 which means the provider does not present a connect window. Setting the prompt value to 1 causes a connect window to always appear. Setting the prompt value to 2 causes a connect window to appear if the initial connection attempt fails. Setting the prompt value to 3 causes a connect window to appear if the initial connection attempt fails but the provider disables the controls for any information not required to connect to the data source.

Window Handle

The application can pass the handle of the parent window, if applicable, or a null pointer if either the window handle is not applicable or the provider does present any windows. The window handle value is typically 0 (NULL).

Other OLE DB connection parameters can be specified but they are ignored by the OLE DB provider.

When the OLE DB provider is invoked, it gets the property values for the OLE DB connection parameters. Here is a typical set of property values obtained from Microsoft's RowsetViewer application.

```
User ID 'DBA'
Password 'sql'
Location 'localhost:4444'
Initial Catalog 'demo'
Data Source 'testds'
Extended Properties 'appinfo=api=oledb'
Prompt 2
Window Handle 0
```

The connection string that the provider constructs from this set of parameter values is:

```
'DSN=testds;HOST=localhost:4444;DBN=demo;UID=DBA;PWD=sql;appinfo=api=oledb'
```

The OLE DB provider uses the connection string, Window Handle, and Prompt values as parameters to the database server connection call that it makes.

This is a simple ADO connection string example.

```
connection.Open "Provider=SAOLEDB;Location=localhost:4444;UserID=DBA;Pwd=sql"
```

ADO parses the connection string and passes all of the unrecognized connection parameters in Extended Properties. When the OLE DB provider is invoked, it gets the property values for the OLE DB connection parameters. Here is the set of property values obtained from the ADO application that used the connection string shown above.

```
User ID ''
Password ''
Location 'localhost:4444'
Initial Catalog ''
Data Source ''
Extended Properties 'UserID=DBA;Pwd=sql'
Prompt 4
Window Handle 0
```

The connection string that the provider constructs from this set of parameter values is:

```
'HOST=localhost:4444;UserID=DBA;Pwd=sql'
```

The provider uses the connection string, Window Handle, and Prompt values as parameters to the database server connection call that it makes.

The following two connection strings are equivalent.

```
Provider=SAOLEDB,Location=192.168.2.2:2638,Downlevel=TRUE;ProviderString='Databas
eName=demo;ServerName=SQLA'
Provider=SAOLEDB,Downlevel=TRUE;ProviderString='DatabaseName=demo;ServerName=SQLA
;Host=192.168.2.2:2638'
```

But the following is not (and is in error).

```
Provider=SAOLEDB,Downlevel=TRUE;ProviderString='DatabaseName=demo;ServerName=SQLA
;Location=192.168.2.2:2638'
```

OLE DB uses the value of the Location property to set the value of the Host connection parameter, but there is no such thing as a Location connection parameter.

Related Information

OLE DB Connection Pooling [page 144]
Userid (UID) Connection Parameter
Password (PWD) Connection Parameter
DataSourceName (DSN) Connection Parameter
DatabaseName (DBN) Connection Parameter
Host Connection Parameter

1.4.4 OLE DB Connection Pooling

The .NET Framework Data Provider for OLE DB automatically pools connections using OLE DB session pooling.

When the application closes the connection, it is not actually closed. Instead, the connection is held for a period of time. When your application re-opens a connection, ADO/OLE DB recognizes that the application is using an identical connection string and reuses the open connection. For example, if the application does an Open/Execute/Close 100 times, there is only 1 actual open and 1 actual close. The final close occurs after about 1 minute of idle time.

If a connection is terminated by external means (such as a forced disconnect using an administrative tool such as *SQL Central*), ADO/OLE DB does not know that this has occurred until the next interaction with the server. Caution should be exercised before resorting to forcible disconnects.

The flag that controls connection pooling is DBPROPVAL_OS_RESOURCEPOOLING (1). This flag can be turned off using a connection parameter in the connection string.

If you specify **`OLE DB Services=-2`** in your connection string, then connection pooling is disabled. Here is a sample connection string:

```
Provider=SAOLEDB;OLE DB Services=-2;...
```

If you specify **`OLE DB Services=-4`** in your connection string, then connection pooling and transaction enlistment are disabled. Here is a sample connection string:

```
Provider=SAOLEDB;OLE DB Services=-4;...
```

If you disable connection pooling, there is a performance penalty if your application frequently opens and closes connections using the same connection string.

Related Information

SQL Server Connection Pooling (ADO.NET)
Overriding Provider Service Defaults
OLE DB, ODBC, and Oracle Connection Pooling (ADO.NET)

1.4.5 OLE DB Isolation Levels

The OLE DB provider supports additional isolation levels. In the list below, the first 4 are standard OLE DB isolation levels. The last three are supported by the database server.

```
#define ISOLATIONLEVEL_READUNCOMMITTED              0x000100
#define ISOLATIONLEVEL_READCOMMITTED                0x001000
#define ISOLATIONLEVEL_REPEATABLEREAD               0x010000
#define ISOLATIONLEVEL_SERIALIZABLE                 0x100000
#define ISOLATIONLEVEL_SNAPSHOT                     0x200000
#define ISOLATIONLEVEL_STATEMENT_SNAPSHOT           0x400000
#define ISOLATIONLEVEL_READONLY_STATEMENT_SNAPSHOT  0x800000
```

The following is an example for specifying SNAPSHOT isolation.

```
hr = pITransactionLocal->StartTransaction( ISOLATIONLEVEL_SNAPSHOT, 0, NULL,
&ulTransactionLevel );
```

Related Information

ADO Transactions [page 140]

1.4.6 Microsoft Linked Servers

A Microsoft Linked Server can be created that uses the OLE DB provider to obtain access to a database. SQL queries can be issued using either the Microsoft four-part table referencing syntax or the Microsoft OPENQUERY SQL function.

An example of the four-part syntax follows.

```
SELECT * FROM SADATABASE.demo.GROUPO.Customers
```

In this example, SADATABASE is the name of the Linked Server, demo is the catalog or database name, GROUPO is the table owner in the database, and Customers is the table name in the database.

The other form uses the Microsoft OPENQUERY function.

```
SELECT * FROM OPENQUERY( SADATABASE, 'SELECT * FROM Customers' )
```

In the OPENQUERY syntax, the second SELECT statement ('SELECT * FROM Customers') is passed to the database server for execution.

For complex queries, OPENQUERY may be the better choice since the entire query is evaluated on the database server. With the four-part syntax, SQL Server may retrieve the contents of all tables referenced by the query before it can evaluate it (for example, queries with WHERE, JOIN, nested queries, and so on). For queries involving very large tables, processing time may be very poor when using four-part syntax. In the following four-part query example, SQL Server passes a simple SELECT on the entire table (no WHERE clause) to the database server via the OLE DB provider and then evaluates the WHERE condition itself.

```
SELECT ID, Surname, GivenName FROM [SADATABASE].[demo].[GROUPO].[Customers]
```

```
WHERE Surname = 'Elkins'
```

Instead of returning one row in the result set to SQL Server, all rows are returned and then this result set is reduced to one row by SQL Server. The following example produces an identical result but only one row is returned to SQL Server.

```
SELECT * FROM OPENQUERY( SADATABASE,
    'SELECT ID, Surname, GivenName FROM [GROUPO].[Customers]
    WHERE Surname = ''Elkins''' )
```

You can set up a Linked Server that uses the OLE DB provider using a Microsoft SQL Server interactive application or a SQL Server script.

In this section:

Setting Up a Linked Server Using an Interactive Application [page 146]
Use a Microsoft SQL Server interactive application to create a Microsoft Linked Server that uses the OLE DB provider to obtain access to a database.

Setting Up a Linked Server Using a Script [page 148]
Set up a Microsoft Linked Server definition using a Microsoft SQL Server script.

1.4.6.1 Setting Up a Linked Server Using an Interactive Application

Use a Microsoft SQL Server interactive application to create a Microsoft Linked Server that uses the OLE DB provider to obtain access to a database.

Prerequisites

Microsoft SQL Server 2000 or later.

Before setting up a Linked Server, there are a few things to consider when using Windows 7 or later versions of Windows. Microsoft SQL Server runs as a service on your system. Depending on how the service is set up on Windows 7 or later versions, a service may not be able to use shared memory connections, it may not be able to start a server, and it may not be able to access User Data Source definitions. For example, a service logged in as a *Network Service* cannot start servers, connect via shared memory, or access User Data Sources. For these situations, the database server must be started ahead of time and the TCPIP communication protocol must be used. Also, if a data source is to be used, it must be a System Data Source.

Procedure

1. For Microsoft SQL Server 2005/2008, start Microsoft SQL Server Management Studio. For other versions of Microsoft SQL Server, the name of this application and the steps to setting up a Linked Server may vary.

In the *Object Explorer* pane, expand *Server Objects* > *Linked Servers*. Right-click *Linked Servers* and then click *New Linked Server*.

2. Fill in the *General* page.

   The *Linked Server* field on the *General* page should contain a *Linked Server* name (like SADATABASE used in an earlier example).

   The *Other Data Source* option should be chosen, and *SQL Anywhere OLE DB Provider 17* should be chosen from the *Provider* list.

   The *Product Name* field can be anything you like (for example, your application name).

   The *Data Source* field can contain an ODBC data source name (DSN). This is a convenience option and a data source name is not required. If you use a System DSN, it must be a 32-bit DSN for 32-bit versions of Microsoft SQL Server or a 64-bit DSN for 64-bit versions of Microsoft SQL Server.

   ```
   Data Source: SQL Anywhere 17 Demo
   ```

   The *Provider String* field can contain additional connection parameters such as UserID (UID), ServerName (Server), and DatabaseFile (DBF).

   ```
   Provider string: Server=myserver;DBF=sample.db
   ```

   The *Location* field can contain the equivalent of the Host connection parameter (for example, localhost:4444 or 10.25.99.253:2638).

   ```
   Location: AppServer-pc:2639
   ```

   The *Initial Catalog* field can contain the name of the database to connect to (for example, demo). The database must have been previously started.

   ```
   Initial Catalog: demo
   ```

   The combination of these last four fields and the user ID and password from the *Security* page must contain enough information to successfully connect to a database server.

3. Instead of specifying the database user ID and password as a connection parameter in the *Provider String* field where it would be exposed in plain text, you can fill in the *Security* page.

   In Microsoft SQL Server 2005/2008, click the *Be made using this security context* option and fill in the *Remote login* and *With password* fields (the password is displayed as asterisks).

4. Go to the *Server Options* page.

   Enable the *RPC* and *RPC Out* options.

   The technique for doing this varies with different versions of Microsoft SQL Server. In Microsoft SQL Server 2000, there are two checkboxes that must be checked for these two options. In Microsoft SQL Server 2005/2008, the options are True/False settings. Make sure that they are set True. The *Remote Procedure Call* (*RPC*) options must be set to execute stored procedure/function calls in a database and pass parameters in and out successfully.

5. Choose the *Allow Inprocess* provider option.

   The technique for doing this varies with different versions of Microsoft SQL Server. In Microsoft SQL Server 2000, there is a *Provider Options* button that takes you to the page where you can choose this option.

   For Microsoft SQL Server 2005/2008, right-click the SAOLEDB.17 provider name under *Linked Servers*

Providers and click *Properties*. Make sure the *Allow Inprocess* checkbox is checked. If the *Inprocess* option is not chosen, queries fail.

6. Other provider options can be ignored. Several of these options pertain to Microsoft SQL Server backwards compatibility and have no effect on the way Microsoft SQL Server interacts with the OLE DB provider. Examples are *Nested queries* and *Supports LIKE operator*. Other options, when selected, may result in syntax errors or degraded performance.

Results

The Microsoft Linked Server is configured.

1.4.6.2 Setting Up a Linked Server Using a Script

Set up a Microsoft Linked Server definition using a Microsoft SQL Server script.

Prerequisites

Microsoft SQL Server.

Before setting up a Linked Server, there are a few things to consider when using Microsoft Windows 7 or later versions of Microsoft Windows. Microsoft SQL Server runs as a service on your system. Depending on how the service is set up on Microsoft Windows 7 or later versions, a service may not be able to use shared memory connections, it may not be able to start a server, and it may not be able to access User Data Source definitions. For example, a service logged in as a *Network Service* cannot start servers, connect via shared memory, or access User Data Sources. For these situations, the database server must be started ahead of time and the TCPIP communication protocol must be used. Also, if a data source is to be used, it must be a System Data Source.

Context

Make the appropriate changes to the following script using the steps below before running it on Microsoft SQL Server.

```
USE [master]
GO
EXEC master.dbo.sp_addlinkedserver @server=N'SADATABASE',
     @srvproduct=N'SAP DBMS', @provider=N'SAOLEDB.17',
     @datasrc=N'SQL Anywhere 17 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'password'
GO
-- Set global provider "allow in process" flag
EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.17', N'AllowInProcess', 1
```

Procedure

1. Choose a new Linked Server name (SADATABASE is used in the example).
2. Choose an optional data source name (SQL Anywhere 17 Demo is used in the example).
3. Choose an optional provider string (`N'host=localhost:4444;server=myserver;dbn=demo'` is used in the example).
4. Choose a remote user ID and password (N'DBA' and N'`password`' are used in the example).

Results

Your modified script can be run under Microsoft SQL Server to create a new Linked Server.

1.4.7 Supported OLE DB Interfaces

The OLE DB API consists of a set of interfaces.

The following table describes the support for each interface by the OLE DB driver.

Interface	Purpose	Limitations
IAccessor	Define bindings between client memory and data store values.	DBACCESSOR_PASSBYREF not supported.DBACCESSOR_OPTIMIZED not supported.
IAlterIndex IAlterTable	Alter tables, indexes, and columns.	Not supported.
IChapteredRowset	A chaptered rowset allows rows of a rowset to be accessed in separate chapters.	Not supported. The database server does not support chaptered rowsets.
IColumnsInfo	Get simple information about the columns in a rowset.	Supported.

Interface	Purpose	Limitations
IColumnsRowset	Get information about optional metadata columns in a rowset, and get a rowset of column metadata.	Supported.
ICommand	Execute SQL statements.	Does not support calling. ICommandProperties: GetProperties with DBPROPSET_PROPERTIESINERROR to find properties that could not have been set.
ICommandPersist	Persist the state of a command object (but not any active rowsets). These persistent command objects can subsequently be enumerated using the PROCEDURES or VIEWS rowset.	Supported.
ICommandPrepare	Prepare commands.	Supported.
ICommandProperties	Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.	Supported.
ICommandText	Set the SQL statement text for ICommand.	Only the DBGUID_DEFAULT SQL dialect is supported.
ICommandWithParameters	Set or get parameter information for a command.	No support for parameters stored as vectors of scalar values. No support for BLOB parameters.
IConvertType		Supported.
IDBAsynchNotify IDBAsynchStatus	Asynchronous processing. Notify client of events in the asynchronous processing of data source initialization, populating rowsets, and so on.	Not supported.
IDBCreateCommand	Create commands from a session.	Supported.
IDBCreateSession	Create a session from a data source object.	Supported.
IDBDataSourceAdmin	Create/destroy/modify data source objects, which are COM objects used by clients. This interface is not used to manage data stores (databases).	Not supported.
IDBInfo	Find information about keywords unique to this provider (that is, to find non-standard SQL keywords). Also, find information about literals, special characters used in text matching queries, and other literal information.	Supported.

Interface	Purpose	Limitations
IDBInitialize	Initialize data source objects and enumerators.	Supported.
IDBProperties	Manage properties on a data source object or enumerator.	Supported.
IDBSchemaRowset	Get information about system tables, in a standard form (a rowset).	Supported.
IErrorInfo IErrorLookup IErrorRecords	Microsoft ActiveX error object support.	Supported.
IGetDataSource	Returns an interface pointer to the session's data source object.	Supported.
IIndexDefinition	Create or drop indexes in the data store.	Not supported.
IMultipleResults	Retrieve multiple results (rowsets or row counts) from a command.	Supported.
IOpenRowset	Non-SQL way to access a database table by its name.	Supported. Opening a table by its name is supported, not by a GUID.
IParentRowset	Access chaptered/hierarchical rowsets.	Not supported.
IRowset	Access rowsets.	Supported.
IRowsetChange	Allow changes to rowset data, reflected back to the data store. InsertRow/SetData for BLOBs are not implemented.	Supported.
IRowsetChapterMember	Access chaptered/hierarchical rowsets.	Not supported.
IRowsetCurrentIndex	Dynamically change the index for a rowset.	Not supported.
IRowsetFind	Find a row within a rowset matching a specified value.	Not supported.
IRowsetIdentity	Compare row handles.	Not supported.
IRowsetIndex	Access database indexes.	Not supported.
IRowsetInfo	Find information about rowset properties or to find the object that created the rowset.	Supported.
IRowsetLocate	Position on rows of a rowset, using bookmarks.	Supported.
IRowsetNotify	Provides a COM callback interface for rowset events.	Supported.

Interface	Purpose	Limitations
IRowsetRefresh	Get the latest value of data that is visible to a transaction.	Not supported.
IRowsetResynch	Old OLE DB 1.x interface, superseded by IRowsetRefresh.	Not supported.
IRowsetScroll	Scroll through rowset to fetch row data.	Not supported.
IRowsetUpdate	Delay changes to rowset data until Update is called.	Supported.
IRowsetView	Use views on an existing rowset.	Not supported.
ISequentialStream	Retrieve a BLOB column.	Supported for reading only. No support for SetData with this interface.
ISessionProperties	Get session property information.	Supported.
ISourcesRowset	Get a rowset of data source objects and enumerators.	Supported.
ISQLErrorInfo ISupportErrorInfo	Microsoft ActiveX error object support.	Supported.
ITableDefinition ITableDefinitionWithConstraints	Create, drop, and alter tables, with constraints.	Supported.
ITransaction	Commit or abort transactions.	Not all the flags are supported.
ITransactionJoin	Support distributed transactions.	Not all the flags are supported.
ITransactionLocal	Handle transactions on a session. Not all the flags are supported.	Supported.
ITransactionOptions	Get or set options on a transaction.	Not supported.
IViewChapter	Work with views on an existing rowset, specifically to apply post-processing filters/sorting on rows.	Not supported.
IViewFilter	Restrict contents of a rowset to rows matching a set of conditions.	Not supported.
IViewRowset	Restrict contents of a rowset to rows matching a set of conditions, when opening a rowset.	Not supported.
IViewSort	Apply sort order to a view.	Not supported.

Support for TIME and TIMESTAMP WITH TIME ZONE Data Types

The data provider supports two different OLE DB data types for handling of the SQL TIME data type. These are DBTYPE_DBTIME and DBTYPE_DBTIME2.

The DBTYPE_DBTIME data structure does not contain fractional seconds. There is a loss of precision when SQL TIME data is returned to the client application using the DBTYPE_DBTIME data type. The ordinal value for DBTYPE_DBTIME is 134.

The DBTYPE_DBTIME2 data type, introduced by Microsoft with the release of SQL Server 2008, supports fractional seconds. There is no loss of precision when SQL TIME data is returned to the client application using the DBTYPE_DBTIME2 data type. The ordinal value for DBTYPE_DBTIME2 is 145.

By default, the data provider maps the TIME data type to DBTYPE_DBTIME2.

In this mode, the data provider can be used with Microsoft SQL Server Integration Services (SSIS) where table data can be exchanged (imported or exported) between the database server and Microsoft SQL Server without the loss of fractional seconds on TIME data types when there are 6 or less fractional seconds.

You can also use the data provider as a Microsoft Linked Server to fetch TIME data without the loss of precision.

If you require that the TIME data type be mapped to DBTYPE_DBTIME, then use the OLE DB connection parameter `Downlevel=TRUE` for backwards compatibility.

Also, if you use Borland Delphi, then use the connection parameter `Delphi=Yes` for backwards compatibility. Old versions of Borland Delphi do not handle DBTYPE_DBTIME2. When the `Delphi=Yes` option is specified in a connection string, the provider maps the TIME data type to DBTYPE_DBTIME. This option is labeled *Delphi applications* on the *ODBC* tab of the *ODBC Configuration for SQL Anywhere* dialog that appears when using the Microsoft ODBC Data Source Administrator.

The data provider maps the SQL TIMESTAMP WITH TIME ZONE data type to DBTYPE_DBTIMESTAMPOFFSET. There is no backwards compatibility option since there is no other OLE DB data type that corresponds to this SQL data type. The DBTYPE_DBTIMESTAMPOFFSET data type, introduced by Microsoft with the release of SQL Server 2008, supports fractional seconds and a time zone offset. The ordinal value for DBTYPE_DBTIMESTAMPOFFSET is 146.

Old versions of Borland Delphi do not support the DBTYPE_DBTIMESTAMPOFFSET data type. If you require the ability to fetch a TIMESTAMP WITH TIME ZONE column in a Borland Delphi application, the column must be cast as CHAR or VARCHAR data.

The underlying C/C++ data structures are described in the Microsoft SQL Server SDK header file called `sqlncli.h`.

Note that OLE DB schema rowset information is not affected by connection parameters. For example, schema rowsets such as `DBSCHEMA_PROVIDER_TYPES` and `DBSCHEMA_COLUMNS` always return ordinal 145 (DBTYPE_DBTIME2) for the TIME data type. Schema rowset information is implemented by the stored procedures listed in the table below and these can be modified if required.

Table 1: OLE DB Schema Rowset to Stored Procedure Mapping

Schema Rowset Name	Stored Procedure Owner and Name
DBSCHEMA_COLUMNS	dbo.sa_oledb_columns
DBSCHEMA_PROCEDURE_COLUMNS	dbo.sa_oledb_procedure_columns

Schema Rowset Name	Stored Procedure Owner and Name
DBSCHEMA_PROCEDURE_PARAMETERS	dbo.sa_oledb_procedure_parameters
DBSCHEMA_PROVIDER_TYPES	dbo.sa_oledb_provider_types

1.4.8 OLE DB Provider Registration

When the SAOLEDB provider is installed using the installer provided with the software, the provider registers itself.

This registration process includes making registry entries in the COM section of the registry, so that ADO can locate the DLL when the SAOLEDB provider is called. If you change the location of your DLL, you must re-register it.

Example

The following commands register the OLE DB provider when run from the directory where the provider is installed:

```
regsvr32 dboledb17.dll
regsvr32 dboledba17.dll
```

If you are using 64-bit Windows, the commands shown above register the 64-bit provider. The following commands can be used to register the 32-bit OLE DB provider:

```
c:\Windows\SysWOW64\regsvr32 dboledb17.dll
c:\Windows\SysWOW64\regsvr32 dboledba17.dll
```

1.5 ODBC Support

ODBC (Open Database Connectivity) is a standard call level interface (CLI) developed by Microsoft Corporation. It is based on the SQL Access Group CLI specification.

ODBC applications can run against any data source that provides an ODBC driver. ODBC is a good choice for a programming interface if you want your application to be portable to other data sources that have ODBC drivers.

ODBC is a low-level interface. Almost all database server functionality is available with this interface. ODBC is available as a DLL under Microsoft Windows operating systems. It is provided as a shared object library for UNIX and Linux.

In this section:

Requirements for Developing ODBC Applications [page 156]
You can develop applications using a variety of development tools and programming languages, as shown in the figure below, and access the database server using the ODBC API.

ODBC Application Development [page 158]
Every C/C++ source file that calls ODBC functions must include a platform-specific ODBC header file. Each platform-specific header file includes the main ODBC header file `odbc.h`, which defines all the functions, data types, and constant definitions required to write an ODBC program.

Sample ODBC Programs [page 163]
Sample ODBC programs are included with the software.

ODBC Handles [page 166]
ODBC applications use a small set of **handles** to track the ODBC context, database connections, and SQL statements. A handle is a pointer type and is a 64-bit value in 64-bit applications and a 32-bit value in 32-bit applications.

ODBC Connection Functions [page 168]
ODBC supplies three different connection functions.

Server Options Changed by ODBC [page 172]
The ODBC driver sets some temporary server options when connecting to the database server.

SQLSetConnectAttr Extended Connection Attributes [page 173]
The ODBC driver supports some extended connection attributes.

SQL_ATTR_CURRENT_CATALOG ODBC API Connection Attribute [page 175]
The SQL_ATTR_CURRENT_CATALOG attribute lets you define the default database in an ODBC application.

Considerations for the Windows DllMain Function [page 176]
ODBC functions should not be called directly or indirectly from the DllMain function in a Windows Dynamic Link Library. The DllMain entry point function is intended to perform only simple initialization and termination tasks. Calling ODBC functions like SQLFreeHandle, SQLFreeConnect, and SQLFreeEnv can create deadlocks and circular dependencies.

Ways to Execute SQL Statements [page 177]
ODBC includes several functions for executing SQL statements.

64-bit ODBC Considerations [page 182]
When you use an ODBC function like SQLBindCol, SQLBindParameter, or SQLGetData, some of the parameters are typed as SQLLEN or SQLULEN in the function prototype.

Data Alignment Requirements [page 185]
When you use SQLBindCol, SQLBindParameter, or SQLGetData, a C data type is specified for the column or parameter.

Result Sets in ODBC Applications [page 186]
ODBC applications use cursors to manipulate and update result sets. The software provides extensive support for different kinds of cursors and cursor operations.

Stored Procedure Considerations [page 192]
You can create and call stored procedures and process the results from an ODBC application.

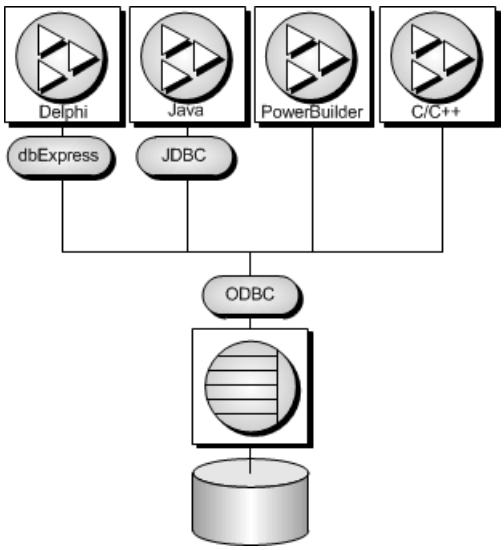
ODBC Escape Syntax [page 194]
You can use ODBC escape syntax from any ODBC application. This escape syntax allows you to call a set of common functions regardless of the database management system you are using.

Error Handling in ODBC [page 196]

Errors in ODBC are reported using the return value from each of the ODBC function calls and either the SQLError function or the SQLGetDiagRec function.

1.5.1 Requirements for Developing ODBC Applications

You can develop applications using a variety of development tools and programming languages, as shown in the figure below, and access the database server using the ODBC API.



To write ODBC applications for the database server, you need:

- A C/C++ compiler capable of creating programs for your environment.
- The Microsoft ODBC Software Development Kit. This is available on the Microsoft Developer Network, and provides documentation and additional tools for testing ODBC applications.

> **Note**
>
> Some application development tools, which already have ODBC support, provide their own programming interface that hides the ODBC interface. This documentation does not describe how to use those tools.

The ODBC driver manager

Microsoft Windows includes an ODBC driver manager. For UNIX, Linux, and macOS, a driver manager is included with the database server software.

In this section:

ODBC Conformance [page 157]
The ODBC driver supports ODBC 3.5, which is supplied as part of the Microsoft Data Access Kit 2.7.

1.5.1.1 ODBC Conformance

The ODBC driver supports ODBC 3.5, which is supplied as part of the Microsoft Data Access Kit 2.7.

Levels of ODBC Support

ODBC features are arranged according to level of conformance. Features are either **Core**, **Level 1**, or **Level 2**, with Level 2 being the most complete level of ODBC support.

Features Supported by the ODBC Driver

The ODBC 3.5 specification is supported as follows:

Core conformance

All Core level features are supported.

Level 1 conformance

All Level 1 features are supported, except for asynchronous execution of ODBC functions.

Multiple threads sharing a single connection are supported. The requests from the different threads are serialized by the database server.

Level 2 conformance

All Level 2 features are supported, except for the following ones:

- Three part names of tables and views. This is not applicable to the database server.
- Asynchronous execution of ODBC functions for specified individual statements.
- Ability to time out login requests and SQL queries.

Related Information

ODBC Programmer's Reference

1.5.2 ODBC Application Development

Every C/C++ source file that calls ODBC functions must include a platform-specific ODBC header file. Each platform-specific header file includes the main ODBC header file `odbc.h`, which defines all the functions, data types, and constant definitions required to write an ODBC program.

Perform the following tasks to include the ODBC header file in a C/C++ source file:

1. Add an include line referencing the appropriate platform-specific header file to your source file. The lines to use are as follows:

Operating System	Include Line
Windows	#include "ntodbc.h"
UNIX/Linux	#include "unixodbc.h"

2. Add the directory containing the header file to the include path for your compiler.
   Both the platform-specific header files and `odbc.h` are installed in the `SDK\Include` subdirectory of the database server software directory.
3. When building ODBC applications for UNIX or Linux, you might have to define the macro "UNIX" for 32-bit applications or "UNIX64" for 64-bit applications to obtain the correct data alignment and sizes. This step is not required if you are using one of the following supported compilers:
   - GNU C/C++ compiler on any supported platform
   - Intel C/C++ compiler for Linux (icc)
   - SunPro C/C++ compiler for Linux or Solaris
   - VisualAge C/C++ compiler for AIX
   - C/C++ compiler (cc/aCC) for HP-UX

Once your source code has been written, you are ready to compile and link the application.

In this section:

ODBC Applications on Windows [page 159]
When linking your application, you must link against the appropriate import library file to have access to the ODBC functions.

ODBC Applications on UNIX/Linux [page 159]
An ODBC driver manager for UNIX and Linux is included with the database server software and there are third party driver managers available. The following information describes how to build ODBC applications that do not use an ODBC driver manager.

The SQL Anywhere ODBC Driver Manager for UNIX/Linux [page 161]
An ODBC driver manager for UNIX and Linux is included with the database server software.

The unixODBC Driver Manager [page 162]
Versions of the unixODBC release before version 2.2.14 have incorrectly implemented some aspects of the 64-bit ODBC specification as defined by Microsoft. These differences cause problems when using the unixODBC driver manager with the 64-bit ODBC driver.

UTF-32 ODBC Driver Managers for UNIX/Linux [page 162]
Versions of ODBC driver managers that define SQLWCHAR as 32-bit (UTF-32) quantities cannot be used with the ODBC driver that supports wide calls since this driver is built for 16-bit SQLWCHAR.

1.5.2.1 ODBC Applications on Windows

When linking your application, you must link against the appropriate import library file to have access to the ODBC functions.

The import library defines entry points for the ODBC driver manager `odbc32.dll`. The driver manager in turn loads the ODBC driver `dbodbc17.dll`.

Typically, the import library is stored under the `Lib` directory structure of the Microsoft platform SDK:

Operating System	Import Library
Windows (32-bit)	`Lib\odbc32.lib`
Windows (64-bit)	`Lib\x64\odbc32.lib`

Example

The following command illustrates how to add the directory containing the platform-specific import library to the list of library directories in your LIB environment variable:

```
set LIB=%LIB%;c:\mssdk\v7.0\lib
```

The following command illustrates how to compile and link the application stored in `odbc.c` using the Microsoft compile and link tool:

```
cl odbc.c /Ic:\sa17\SDK\Include odbc32.lib
```

1.5.2.2 ODBC Applications on UNIX/Linux

An ODBC driver manager for UNIX and Linux is included with the database server software and there are third party driver managers available. The following information describes how to build ODBC applications that do not use an ODBC driver manager.

ODBC Driver

The ODBC driver is a shared object or shared library. Separate versions of the ODBC driver are supplied for single-threaded and multithreaded applications. A generic ODBC driver is supplied that will detect the threading model in use and direct calls to the appropriate single-threaded or multithreaded library.

The ODBC drivers are the following files:

Operating System	Threading Model	ODBC Driver
(all UNIX/Linux except macOS and HP-UX)	Generic	`libdbodbc17.so` (`libdbodbc17.so.1`)
(all UNIX/Linux except macOS and HP-UX)	Single threaded	`libdbodbc17_n.so` (`libdbodbc17_n.so.1`)
(all UNIX/Linux except macOS and HP-UX)	Multithreaded	`libdbodbc17_r.so` (`libdbodbc17_r.so.1`)
HP-UX	Generic	`libdbodbc17.sl` (`libdbodbc17.sl.1`)
HP-UX	Single threaded	`libdbodbc17_n.sl` (`libdbodbc17_n.sl.1`)
HP-UX	Multithreaded	`libdbodbc17_r.sl` (`libdbodbc17_r.sl.1`)
macOS	Generic	`libdbodbc17.dylib`
macOS	Single threaded	`libdbodbc17_n.dylib`
macOS	Multithreaded	`libdbodbc17_r.dylib`

The libraries are installed as symbolic links to the shared library with a version number (shown in parentheses).

In addition, the following bundles are also provided for macOS:

Operating System	Threading Model	ODBC Driver
macOS	Single threaded	`dbodbc17.bundle`
macOS	Multithreaded	`dbodbc17_r.bundle`

When linking an ODBC application on UNIX or Linux, link your application against the generic ODBC driver `libdbodbc17`. When deploying your application, ensure that the appropriate (or all) ODBC driver versions (non-threaded or threaded) are available in the user's library path.

Data Source Information

If the presence of an ODBC driver manager is not detected, the ODBC driver uses the system information file for data source information.

Related Information

ODBC Data Sources on UNIX/Linux

1.5.2.3 The SQL Anywhere ODBC Driver Manager for UNIX/Linux

An ODBC driver manager for UNIX and Linux is included with the database server software.

The `libdbodm`*17* shared object can be used on all supported UNIX and Linux platforms as an ODBC driver manager. The driver manager can be used to load any version 3.0 or later ODBC driver. The driver manager will not perform mappings between ODBC 1.0/2.0 calls and ODBC 3.x calls; therefore, applications using the driver manager must restrict their use of the ODBC feature set to version 3.0 and later. Also, the driver manager can be used by both threaded and non-threaded applications.

The driver manager can perform tracing of ODBC calls for any given connection. To turn on tracing, use the TraceLevel and TraceLog directives. These directives can be part of a connection string (in the case where SQLDriverConnect is being used) or within a DSN entry. The TraceLog directive identifies the tracing log file to contain the trace output for the connection. The TraceLevel directive governs the amount of tracing information wanted. The trace levels are:

NONE

No tracing information is printed.

MINIMAL

Routine name and parameters are included in the output.

LOW

In addition to the above, return values are included in the output.

MEDIUM

In addition to the above, the date and time of execution are included in the output.

HIGH

In addition to the above, parameter types are included in the output.

ALL

In addition to the above, process ID and thread ID are included in the trace output.

Third-party ODBC driver managers for UNIX and Linux are available also. Consult the documentation that accompanies these driver managers for information about their use.

Related Information

The unixODBC Driver Manager [page 162]
UTF-32 ODBC Driver Managers for UNIX/Linux [page 162]

1.5.2.4 The unixODBC Driver Manager

Versions of the unixODBC release before version 2.2.14 have incorrectly implemented some aspects of the 64-bit ODBC specification as defined by Microsoft. These differences cause problems when using the unixODBC driver manager with the 64-bit ODBC driver.

To avoid these problems, be aware of the differences. One of them is the definition of SQLLEN and SQLULEN. These are 64-bit types in the Microsoft 64-bit ODBC specification, and are expected to be 64-bit quantities by the 64-bit ODBC driver. Some implementations of unixODBC define these two types as 32-bit quantities and this will result in problems when interfacing to the 64-bit ODBC driver.

There are three things that you must do to avoid problems on 64-bit platforms.

1. Instead of including the unixODBC headers like `sql.h` and `sqlext.h`, include the `unixodbc.h` header file. This guarantees that you've the correct definitions for SQLLEN and SQLULEN. The header files in unixODBC 2.2.14 or later versions correct this problem.
2. You must ensure that you've used the correct types for all parameters. Use of the correct header file and the strong type checking of your C/C++ compiler should help in this area. You must also ensure that you've used the correct types for all variables that are set by the ODBC driver indirectly through pointers.
3. Don't use versions of the unixODBC driver manager before release 2.2.14. Link directly to the ODBC driver instead. For example, ensure that the libodbc shared object is linked to the ODBC driver shared object.

   ```
   libodbc.so.1 -> libdbodbc17_r.so.1
   ```

   To use the driver manager included with the database server software, put the location of the libdbodbcinst17_r.so and libdbodbc17.so in the LD_LIBRARY_PATH, ahead of the unixODBC drivers.
   To use the unixODBC drivers, put the location of unixODBC's libodbc.so and libodbcinst.so ahead of the ones in the product. Typically their location is `/usr/lib`.

Related Information

The SQL Anywhere ODBC Driver Manager for UNIX/Linux [page 161]
ODBC Applications on UNIX/Linux [page 159]
64-bit ODBC Considerations [page 182]

1.5.2.5 UTF-32 ODBC Driver Managers for UNIX/Linux

Versions of ODBC driver managers that define SQLWCHAR as 32-bit (UTF-32) quantities cannot be used with the ODBC driver that supports wide calls since this driver is built for 16-bit SQLWCHAR.

For these cases, an ANSI-only version of the ODBC driver is provided. This version of the ODBC driver does not support the wide call interface (for example, SQLConnectW).

The shared object name of the driver is `libdbodbcansi`*17*`_r`. Only a threaded variant of the driver is provided. On macOS, in addition to the dylib, the driver is also available in bundle form (`dbodbcansi`*17*`_r.bundle`). Certain frameworks, such as Real Basic, do not work with the dylib and require the bundle.

The regular ODBC driver treats SQLWCHAR strings as UTF-16 strings. This driver cannot be used with some ODBC driver managers, such as iODBC, which treat SQLWCHAR strings as UTF-32 strings. When dealing with Unicode-enabled drivers, these driver managers translate narrow calls from the application to wide calls into the driver. An ANSI-only driver gets around this behavior, allowing the driver to be used with such driver managers, as long as the application does not make any wide calls. Wide calls through iODBC, or any other driver manager with similar semantics, remain unsupported.

1.5.3 Sample ODBC Programs

Sample ODBC programs are included with the software.

You can find the samples in the *%SQLANYSAMP17%*`\SQLAnywhere` subdirectories (Microsoft Windows) and *$SQLANYSAMP17*`/sqlanywhere` subdirectories (UNIX and Linux).

The sample programs in directories starting with the 4 letters ODBC illustrate separate and simple ODBC tasks, such as connecting to a database and executing statements. A complete sample ODBC program is supplied in the file *%SQLANYSAMP17%*`\SQLAnywhere\C\odbc.c` (Microsoft Windows) and *$SQLANYSAMP17*`/sqlanywhere/c/odbc.c` (UNIX and Linux). This program performs the same actions as the Embedded SQL dynamic cursor example program that is in the same directory.

In this section:

Building the Sample ODBC Program for Windows [page 164]
Build and run a sample ODBC program to see how it performs ODBC tasks, such as connecting to a database and executing statements.

Building the Sample ODBC Program for UNIX/Linux [page 165]
Build and run a sample ODBC program to see how it performs ODBC tasks, such as connecting to a database and executing statements.

Run the ODBC Sample Programs [page 165]
You can load the sample ODBC program by running the file on the appropriate platform.

Related Information

Sample Embedded SQL Programs [page 272]

1.5.3.1 Building the Sample ODBC Program for Windows

Build and run a sample ODBC program to see how it performs ODBC tasks, such as connecting to a database and executing statements.

Prerequisites

A recent version of Microsoft Visual Studio is required.

For x86/x64 platform builds with Microsoft Visual Studio, you must set up the correct environment for compiling and linking. This is typically done using the Microsoft Visual Studio `vcvars32.bat` or `vcvars64.bat` (called `vcvarsamd64.bat` in older versions of Microsoft Visual Studio).

Context

A batch file located in the *%SQLANYSAMP17%*`\SQLAnywhere\C` directory can be used to compile and link all the sample applications.

Procedure

1. Open a command prompt and change the directory to the *%SQLANYSAMP17%*`\SQLAnywhere\C` directory.
2. Run the `build.bat` batch file.

   If you are getting build errors, try specifying the target platform (`x86` or `x64`) as an argument to `build.bat`. Here is an example.

   ```
   build x64
   ```

Results

The sample ODBC program is built.

1.5.3.2 Building the Sample ODBC Program for UNIX/Linux

Build and run a sample ODBC program to see how it performs ODBC tasks, such as connecting to a database and executing statements.

Context

A shell script located in the *$SQLANYSAMP17*`/sqlanywhere/c` directory can be used to compile and link all the sample applications.

Procedure

1. Open a command shell and change the directory to the *$SQLANYSAMP17*`/sqlanywhere/c` directory.

   Open a command shell and change the directory to the *$SQLANYSAMP17*`/samples/sqlanywhere/c` directory.
2. Run the `build.sh` shell script.

Results

The sample ODBC program is built.

1.5.3.3 Run the ODBC Sample Programs

You can load the sample ODBC program by running the file on the appropriate platform.

- For 32-bit Windows, run *%SQLANYSAMP17%*`\SQLAnywhere\C\odbcwin.exe`.
- For 64-bit Windows, run *%SQLANYSAMP17%*`\SQLAnywhere\C\odbcx64.exe`.
- For UNIX and Linux, run *$SQLANYSAMP17*`/sqlanywhere/c/odbc`.

After running the file, choose one of the tables in the sample database. For example, you can enter **`Customers`** or **`Employees`**.

1.5.4 ODBC Handles

ODBC applications use a small set of **handles** to track the ODBC context, database connections, and SQL statements. A handle is a pointer type and is a 64-bit value in 64-bit applications and a 32-bit value in 32-bit applications.

The following handles are used in ODBC applications:

Environment

The environment handle provides a global context in which to access data. Every ODBC application must allocate exactly one environment handle upon starting, and must free it at the end.

The following code illustrates how to allocate an environment handle:

```
SQLRETURN rc;
SQLHENV env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

Connection

A connection is the link between an application and a data source. An application can have several connections associated with its environment. Allocating a connection handle does not establish a connection; a connection handle must be allocated first and then used to establish a connection.

The following code illustrates how to allocate a connection handle:

```
SQLRETURN rc;
SQLHDBC  dbc;
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

Statement

A statement handle is used to execute SQL statements and set up or process any information associated with it, such as parameters and result sets. Each connection can have several statement handles associated with it. Statement handles are used both for query execution (cursor operations such as fetching) and for non-query execution (for example, INSERT, UPDATE, and DELETE statements).

The following code illustrates how to allocate a statement handle:

```
SQLRETURN rc;
SQLHSTMT stmt;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

In this section:

How to Allocate ODBC Handles [page 167]
ODBC defines four types of objects (environment, connection, statement, and descriptor) that are referenced in applications by using handles.

ODBC Example [page 168]
A simple ODBC program that connects to the sample database and immediately disconnects can be found in *%SQLANYSAMP17%*`\SQLAnywhere\ODBCConnect\odbcconnect.cpp`.

1.5.4.1 How to Allocate ODBC Handles

ODBC defines four types of objects (environment, connection, statement, and descriptor) that are referenced in applications by using handles.

The handle types available for ODBC programs are as follows:

Item	Handle Type
Environment	SQLHENV
Connection	SQLHDBC
Statement	SQLHSTMT
Descriptor	SQLHDESC

To use an ODBC handle, you perform the following tasks:

1. Call the SQLAllocHandle function.
2. Use the handle in subsequent function calls.
3. Free the object using SQLFreeHandle.

SQLAllocHandle takes the following parameters:

- an identifier for the type of item being allocated
- the handle of the parent item
- a pointer to the location of the handle to be allocated

SQLFreeHandle takes the following parameters:

- an identifier for the type of item being freed
- the handle of the item being freed

Example

The following code fragment allocates and frees an environment handle:

```
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 );
```

Related Information

Error Handling in ODBC [page 196]

SQLAllocHandle Function
SQLFreeHandle Function

1.5.4.2 ODBC Example

A simple ODBC program that connects to the sample database and immediately disconnects can be found in *%SQLANYSAMP17%*`\SQLAnywhere\ODBCConnect\odbcconnect.cpp`.

This example shows the steps required in setting up the environment to make a connection to a database server, as well as the steps required in disconnecting from the server and freeing up resources.

1.5.5 ODBC Connection Functions

ODBC supplies three different connection functions.

Which one you use depends on how you expect your application to be deployed and used:

SQLConnect

The simplest connection function.

SQLConnect takes a data source name and optional user ID and password. Use SQLConnect if you hard-code a data source name into your application.

SQLDriverConnect

Connects to a data source using a connection string.

SQLDriverConnect allows the application to use connection information that is external to the data source definition. Also, you can use SQLDriverConnect to request that the ODBC driver prompt for connection information.

SQLDriverConnect can also be used to connect without specifying a data source. The ODBC driver name is specified instead. The following example connects to a server and database that is already running.

```
SQLSMALLINT cso;
SQLCHAR     scso[2048];
SQLDriverConnect( hdbc, NULL,
   "Driver=SQL Anywhere 17;UID=DBA;PWD=passwd", SQL_NTS,
   scso, sizeof(scso)-1,
   &cso, SQL_DRIVER_NOPROMPT );
```

SQLBrowseConnect

Connects to a data source using a connection string, like SQLDriverConnect.

SQLBrowseConnect allows your application to build its own windows to prompt for connection information and to browse for data sources used by a particular driver.

In this section:

Establishing an ODBC Connection [page 169]
Establish an ODBC connection in your application to perform any database operations.

Related Information

Alphabetical List of Connection Parameters
SQLConnect Function
SQLDriverConnect Function
SQLBrowseConnect Function

1.5.5.1 Establishing an ODBC Connection

Establish an ODBC connection in your application to perform any database operations.

Context

You can find a complete sample in *%SQLANYSAMP17%*`\SQLAnywhere\ODBCConnect\odbcconnect.cpp`.

Procedure

1. Allocate an ODBC environment. For example:

```
SQLRETURN rc;
SQLHENV   env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

2. Declare the ODBC version. For example:

```
rc = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0 );
```

   By declaring that the application follows ODBC version 3, SQLSTATE values and some other version-dependent features are set to the proper behavior.
3. Allocate an ODBC connection handle. For example:

```
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

4. Set any connection attributes that must be set before connecting.

   Some connection attributes must be set before establishing a connection or after establishing a connection, while others can be set either before or after. The SQL_AUTOCOMMIT attribute is one that can be set before or after:

```
rc = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
0 );
```

   By default, ODBC operates in autocommit mode. This mode is turned off by setting SQL_AUTOCOMMIT to false.

5. If necessary, assemble the data source or connection string.

   Depending on your application, you may have a hard-coded data source or connection string, or you may store it externally for greater flexibility.
6. Establish an ODBC connection. For example:

```
if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
{
   printf( "dbc allocated\n" );
   rc = SQLConnect( dbc,
      (SQLCHAR *) "SQL Anywhere 17 Demo", SQL_NTS,
      (SQLCHAR *) my_userid, SQL_NTS,
      (SQLCHAR *) my_password, SQL_NTS );
   if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
   {
       // Successfully connected.
```

   Every string passed to ODBC has a corresponding length. If the length is unknown, you can pass SQL_NTS indicating that it is a **Null Terminated String** whose end is marked by the null character (\0) for ASCII strings or the null wide character (\0\0) for wide-character strings.

Results

The application, when built and run, establishes an ODBC connection.

In this section:

How to Set Connection Attributes [page 170]
: You use the SQLSetConnectAttr function to control details of the connection. For example, the following statement disables ODBC autocommit.

How to Get Connection Attributes [page 171]
: You use the SQLGetConnectAttr function to get details of the connection. For example, the following statement returns the connection state.

Threads and Connections in ODBC Applications [page 171]
: You can develop multithreaded ODBC applications. Use a separate connection for each thread.

ODBC Connection Failures [page 171]
: If you do not deploy all the files required for the ODBC driver, you may encounter the following error when attempting to connect to a database.

1.5.5.1.1 How to Set Connection Attributes

You use the SQLSetConnectAttr function to control details of the connection. For example, the following statement disables ODBC autocommit.

```
rc = SQLSetConnectAttr( dbc, SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
```

Many aspects of the connection can be controlled through the connection parameters.

Related Information

Alphabetical List of Connection Parameters
SQLSetConnectAttr Function

1.5.5.1.2 How to Get Connection Attributes

You use the SQLGetConnectAttr function to get details of the connection. For example, the following statement returns the connection state.

```
rc = SQLGetConnectAttr( dbc, SQL_ATTR_CONNECTION_DEAD,
   (SQLPOINTER)&closed, SQL_IS_INTEGER, 0 );
```

When using the SQLGetConnectAttr function to get the SQL_ATTR_CONNECTION_DEAD attribute, the value SQL_CD_TRUE is returned if the connection has been dropped even if no request has been sent to the server since the connection was dropped. Determining if the connection has been dropped is done without making a request to the server, and the dropped connection is detected within a few seconds. The connection can be dropped for several reasons, such as an idle timeout.

Related Information

SQLGetConnectAttr Function

1.5.5.1.3 Threads and Connections in ODBC Applications

You can develop multithreaded ODBC applications. Use a separate connection for each thread.

You can use a single connection for multiple threads. However, the database server does not allow more than one active request for any one connection at a time. If one thread executes a statement that takes a long time, all other threads must wait until the request is complete.

1.5.5.1.4 ODBC Connection Failures

If you do not deploy all the files required for the ODBC driver, you may encounter the following error when attempting to connect to a database.

```
[IM004][Microsoft][ODBC Driver Manager] Driver's SQLAllocHandle on
SQL_HANDLE_ENV failed
```

Check that you have installed all the files required for the correct operation of the ODBC driver.

Related Information

ODBC Client Deployment [page 836]

1.5.6 Server Options Changed by ODBC

The ODBC driver sets some temporary server options when connecting to the database server.

The following options are set as indicated.

date_format

yyyy-mm-dd

date_order

ymd

isolation_level

based on the SQL_ATTR_TXN_ISOLATION/SA_SQL_ATTR_TXN_ISOLATION attribute setting of SQLSetConnectAttr. The following options are available.

```
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

yyyy-mm-dd hh:nn:ss.ssssss

timestamp_with_time_zone_format

yyyy-mm-dd hh:nn:ss.ssssss +hh:nn

To guarantee consistent behavior of the ODBC driver, do not change the setting of these options.

Related Information

ODBC Transaction Isolation Levels [page 187]

1.5.7 SQLSetConnectAttr Extended Connection Attributes

The ODBC driver supports some extended connection attributes.

SA_REGISTER_MESSAGE_CALLBACK

Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

A message handler routine can be created to intercept these messages. The message handler callback prototype is as follows:

```
void SQL_CALLBACK message_handler(
SQLHDBC sqlany_dbc,
unsigned char  msg_type,
long code,
unsigned short  length,
char * message
);
```

The following possible values for `msg_type` are defined in `sqldef.h`.

MESSAGE_TYPE_INFO

The message type was INFO.

MESSAGE_TYPE_WARNING

The message type was WARNING.

MESSAGE_TYPE_ACTION

The message type was ACTION.

MESSAGE_TYPE_STATUS

The message type was STATUS.

MESSAGE_TYPE_PROGRESS

The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

A SQLCODE associated with the message may be provided in`code`. When not available, the`code` parameter value is 0.

The length of the message is contained in `length`.

A pointer to the message is contained in `message`. The message string is not null-terminated. Your application must be designed to handle this. The following is an example.

```
memcpy( mybuff, msg, len );
mybuff[ len ] = '\0';
```

To register the message handler in ODBC, call the SQLSetConnectAttr function as follows:

```
rc = SQLSetConnectAttr(
   hdbc,
   SA_REGISTER_MESSAGE_CALLBACK,
   (SQLPOINTER) &message_handler, SQL_IS_POINTER );
```

To unregister the message handler in ODBC, call the SQLSetConnectAttr function as follows:

```
rc = SQLSetConnectAttr(
   hdbc,
   SA_REGISTER_MESSAGE_CALLBACK,
   NULL, SQL_IS_POINTER );
```

SA_GET_MESSAGE_CALLBACK_PARM

To retrieve the value of the SQLHDBC connection handle that is passed to message handler callback routine, use SQLGetConnectAttr with the SA_GET_MESSAGE_CALLBACK_PARM parameter.

```
SQLHDBC callback_hdbc = NULL;
rc = SQLGetConnectAttr(
   hdbc,
   SA_GET_MESSAGE_CALLBACK_PARM,
   (SQLPOINTER) &callback_hdbc, 0, 0 );
```

The returned value is the same as the parameter value that is passed to the message handler callback routine.

SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK

This is used to register a file transfer validation callback function. Before allowing any transfer to take place, the ODBC driver will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the ODBC driver will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below.

The callback prototype is as follows:

```
int SQL_CALLBACK file_transfer_callback(
void * sqlca,
char * file_name,
int is_write
);
```

The `file_name` parameter is the name of the file to be read or written. The `is_write` parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the ODBC driver allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. If the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

SA_SQL_ATTR_TXN_ISOLATION

This is used to set an extended transaction isolation level. The following example sets a Snapshot isolation level:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SA_SQL_ATTR_TXN_ISOLATION,
      SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

Related Information

ODBC Transaction Isolation Levels [page 187]
MESSAGE Statement
progress_messages Option

1.5.8 SQL_ATTR_CURRENT_CATALOG ODBC API Connection Attribute

The SQL_ATTR_CURRENT_CATALOG attribute lets you define the default database in an ODBC application.

If you start a server with two or more databases, for example, first.db and second.db (specified on the command line in that order), then when you connect to the server, you can do one of three things:

1. Specify the database you want to connect to in your connection string (DatabaseName=b or DBN=b).
2. Omit the DBN parameter and connect to the default database, which would be the first one in the list.
3. Set the SQL_ATTR_CURRENT_CATALOG connection attribute to specify which database you want to connect to by default when you don't specify the DBN parameter.

SQL_ATTR_CURRENT_CATALOG can be set before or after connecting to a database server.

If you connect to the server and do not specify which database you want to connect to, then the server defaults to the first database specified on the command line (first.db). But if you set SQL_ATTR_CURRENT_CATALOG to second before making a connection, then the server defaults to second.db.

```
rc = SQLSetConnectAttr( dbc, SQL_ATTR_CURRENT_CATALOG, (SQLCHAR*)"second",
SQL_NTS );
```

If the connection string contains a DatabaseName (DBN) connection parameter, then that parameter overrides the default specified by SQL_ATTR_CURRENT_CATALOG. If the database has not been started by the server, then the connection fails.

If you set SQL_ATTR_CURRENT_CATALOG after making a connection, then a new default database is established and that database becomes the default when the application disconnects from and reconnects to the database server.

The ODBC call SQLGetConnectAttr(SQL_ATTR_CURRENT_CATALOG) returns the name of the default database when the application is not connected to a database (that is, before connecting or after disconnecting). While connected to a database, SQLGetConnectAttr() returns the name of the currently

connected database. This name may or may not be the same as the default database name, for example, if the DatabaseName (DBN) connection parameter had been used to override the default.

```
rc = SQLGetConnectAttr( dbc, SQL_ATTR_CURRENT_CATALOG, (SQLCHAR*)database_name,
   sizeof(database_name), &cbdatabase_name );
```

1.5.9 Considerations for the Windows DllMain Function

ODBC functions should not be called directly or indirectly from the DllMain function in a Windows Dynamic Link Library. The DllMain entry point function is intended to perform only simple initialization and termination tasks. Calling ODBC functions like SQLFreeHandle, SQLFreeConnect, and SQLFreeEnv can create deadlocks and circular dependencies.

The following code example illustrates a bad programming practice. When the Microsoft ODBC Driver Manager detects that the last access to the ODBC driver has completed, it will do a driver unload. When the ODBC driver shuts down, it stops any active threads. Thread termination results in a recursive thread detach call into DllMain. Since the call into DllMain is serialized, and a call is underway, the new thread detach call will never get started. The ODBC driver will wait forever for its threads to terminate and your application will hang.

```
BOOL APIENTRY 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 */
}
```

Related Information

Dynamic-Link Library Best Practices

1.5.10 Ways to Execute SQL Statements

ODBC includes several functions for executing SQL statements.

Direct execution

The database server parses the SQL statement, prepares an access plan, and executes the statement. Parsing and access plan preparation are called **preparing** the statement.

Prepared execution

The statement preparation is carried out separately from the execution. For statements that are to be executed repeatedly, this avoids repeated preparation and so improves performance.

In this section:

Direct Statement Execution [page 177]
To execute a SQL statement in an ODBC application, allocate a handle for the statement using SQLAllocHandle and then call the SQLExecDirect function to execute the statement.

Executing Statements with Bound Parameters [page 178]
Construct and execute SQL statements using bound parameters to set values for statement parameters at runtime.

Executing Prepared Statements [page 180]
Execute prepared statements to provide performance advantages for statements that are used repeatedly.

1.5.10.1 Direct Statement Execution

To execute a SQL statement in an ODBC application, allocate a handle for the statement using SQLAllocHandle and then call the SQLExecDirect function to execute the statement.

Any parameters must be included as part of the statement (for example, a WHERE clause must specify its arguments). Alternatively, you can also construct statements using bound parameters.

The SQLExecDirect function takes a statement handle, a SQL string, and a length or termination indicator, which in this case is a null-terminated string indicator. The statement may include parameters.

For a complete sample with error checking, see *%SQLANYSAMP17%* `\SQLAnywhere\ODBCExecute\odbcexecute.cpp`.

Example

The following example illustrates how to allocate a handle of type SQL_HANDLE_STMT named stmt on a connection with handle dbc:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

The following example illustrates how to declare a statement and execute it:

```
SQLCHAR deletestmt[ STMT_LEN ] =
  "DELETE FROM Departments WHERE DepartmentID = 201";
SQLExecDirect( stmt, deletestmt, SQL_NTS) ;
```

The deletestmt declaration should usually occur at the beginning of the function.

Related Information

Executing Statements with Bound Parameters [page 178]
SQLExecDirect Function

1.5.10.2 Executing Statements with Bound Parameters

Construct and execute SQL statements using bound parameters to set values for statement parameters at runtime.

Prerequisites

To run the example successfully, you need the following system privileges.

- INSERT on the Departments table

Context

Bound parameters are used with prepared statements to provide performance benefits for statements that are executed more than once.

Procedure

1. Allocate a handle for the statement using SQLAllocHandle.

   For example, the following statement allocates a handle of type `SQL_HANDLE_STMT` with name `stmt`, on a connection with handle `dbc`:

   ```
   SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
   ```

2. Bind parameters for the statement using SQLBindParameter.

   For example, the following lines declare variables to hold the values for the department ID, department name, and manager ID, and for the statement string. They then bind parameters to the first, second, and third parameters of a statement executed using the stmt statement handle.

   ```
   #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. Assign values to the parameters.

   For example, the following lines assign values to the parameters for the fragment of step 2.

   ```
   deptID = 201;
   strcpy( (char * ) deptName, "Sales East" );
   managerID = 902;
   ```

   Commonly, these variables would be set in response to user action.

4. Execute the statement using SQLExecDirect.

   For example, the following line executes the statement string held in `insertstmt` on the statement handle `stmt`.

   ```
   SQLExecDirect( stmt, insertstmt, SQL_NTS );
   ```

Results

When built and run, the application executes the SQL statement.

Next Steps

The above code fragments do not include error checking. For a complete sample, including error checking, see *%SQLANYSAMP17%*`\SQLAnywhere\ODBCExecute\odbcexecute.cpp`.

Related Information

Executing Prepared Statements [page 180]

1.5.10.3 Executing Prepared Statements

Execute prepared statements to provide performance advantages for statements that are used repeatedly.

Prerequisites

To run the example successfully, you need the following system privileges.

- INSERT on the Departments table

Procedure

1. Prepare the statement using SQLPrepare.

   For example, the following code fragment illustrates how to prepare an INSERT statement:

   ```
   rc = SQLPrepare( stmt,
       "INSERT INTO Departments( DepartmentID, DepartmentName,
   DepartmentHeadID ) "
       "VALUES (?, ?, ?)",
       SQL_NTS );
   ```

   In this example:

   rc

   Receives a return code that should be tested for success or failure of the operation.

   stmt

   Provides a handle to the statement so that it can be referenced later.

   ?

   The question marks are placeholders for statement parameters. A placeholder is put in the statement to indicate where host variables are to be accessed. A placeholder is either a question mark (?) or a host variable reference (a host variable name preceded by a colon). In the latter case, the host

variable name used in the actual text of the statement serves only as a placeholder indicating that a corresponding parameter is to be bound to it. It need not match the actual parameter name.

2. Bind statement parameter values using SQLBindParameter.

   For example, the following function call binds the value of the DepartmentID variable:

   ```
   rc = SQLBindParameter( stmt,
                   1,
                   SQL_PARAM_INPUT,
                   SQL_C_SSHORT,
                   SQL_INTEGER,
                   0,
                   0,
                   &sDeptID,
                   0,
                   &cbDeptID );
   ```

   In this example:

   rc

   Holds a return code that should be tested for success or failure of the operation.

   stmt

   is the statement handle.

   1

   indicates that this call sets the value of the first placeholder.

   SQL_PARAM_INPUT

   indicates that the parameter is an input statement.

   SQL_C_SHORT

   indicates the C data type being used in the application.

   SQL_INTEGER

   indicates SQL data type being used in the database.

   The next two parameters indicate the column precision and the number of decimal digits: both zero for integers.

   &sDeptID

   is a pointer to a buffer for the parameter value.

   0

   indicates the length of the buffer, in bytes.

   &cbDeptID

   is a pointer to a buffer for the length of the parameter value.

3. Bind the other two parameters and assign values to sDeptId.
4. Execute the statement:

   ```
   rc = SQLExecute( stmt );
   ```

   Steps 2 to 4 can be carried out multiple times.

5. Drop the statement.

Dropping the statement frees resources associated with the statement itself. You drop statements using SQLFreeHandle.

Results

When built and run, the application executes the prepared statements.

Next Steps

The above code fragments do not include error checking. For a complete sample, including error checking, see *%SQLANYSAMP17%*`\SQLAnywhere\ODBCPrepare\odbcprepare.cpp`.

Related Information

Prepared Statements [page 13]

1.5.11 64-bit ODBC Considerations

When you use an ODBC function like SQLBindCol, SQLBindParameter, or SQLGetData, some of the parameters are typed as SQLLEN or SQLULEN in the function prototype.

Depending on the date of the Microsoft ODBC API Reference documentation that you are looking at, you might see the same parameters described as SQLINTEGER or SQLUINTEGER.

SQLLEN and SQLULEN data items are 64 bits in a 64-bit ODBC application and 32 bits in a 32-bit ODBC application. SQLINTEGER and SQLUINTEGER data items are 32 bits on all platforms.

To illustrate the problem, the following ODBC function prototype was excerpted from an older copy of the Microsoft ODBC API Reference.

```
SQLRETURN SQLGetData(
     SQLHSTMT     StatementHandle,
     SQLUSMALLINT ColumnNumber,
     SQLSMALLINT  TargetType,
     SQLPOINTER   TargetValuePtr,
     SQLINTEGER   BufferLength,
     SQLINTEGER  *StrLen_or_IndPtr);
```

Compare this with the actual function prototype found in `sql.h` in Microsoft Visual Studio version 8.

```
SQLRETURN  SQL_API SQLGetData(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT  ColumnNumber,
    SQLSMALLINT   TargetType,
    SQLPOINTER    TargetValue,
```

```
    SQLLEN        BufferLength,
    SQLLEN       *StrLen_or_Ind);
```

As you can see, the BufferLength and StrLen_or_Ind parameters are now typed as SQLLEN, not SQLINTEGER. For the 64-bit platform, these are 64-bit quantities, not 32-bit quantities as indicated in the Microsoft documentation.

To avoid issues with cross-platform compilation, the database server software provides its own ODBC header files. For Windows platforms, include the `ntodbc.h` header file. For UNIX and Linux platforms, include the `unixodbc.h` header file. Use of these header files ensures compatibility with the corresponding ODBC driver for the target platform.

The following table lists some common ODBC types that have the same or different storage sizes on 64-bit and 32-bit platforms.

ODBC API	64-bit Platform	32-bit Platform
SQLINTEGER	32 bits	32 bits
SQLUINTEGER	32 bits	32 bits
SQLLEN	64 bits	32 bits
SQLULEN	64 bits	32 bits
SQLSETPOSIROW	64 bits	16 bits
SQL_C_BOOKMARK	64 bits	32 bits
BOOKMARK	64 bits	32 bits

If you declare data variables and parameters incorrectly, then you may encounter incorrect software behavior.

The following table summarizes the ODBC API function prototypes that have changed with the introduction of 64-bit support. The parameters that are affected are noted. The parameter name as documented by Microsoft is shown in parentheses when it differs from the actual parameter name used in the function prototype. The parameter names are those used in the Microsoft Visual Studio version 8 header files.

ODBC API	Parameter (Documented Parameter Name)
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)

ODBC API	Parameter (Documented Parameter Name)
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
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

Some values passed into and returned from ODBC API calls through pointers have changed to accommodate 64-bit applications. For example, the following values for the SQLSetStmtAttr and SQLSetDescField functions are no longer SQLINTEGER/SQLUINTEGER. The same rule applies to the corresponding parameters for the SQLGetStmtAttr and SQLGetDescField functions.

ODBC API	Type for 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

ODBC API	Type for Value/ValuePtr Variable
SQLSetStmtAttr(SQL_ATTR_ROW_NUMBER)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROWS_FETCHED_PTR)	SQLULEN * value
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

Related Information

INFO: ODBC 64-Bit API Changes in MDAC 2.7

1.5.12 Data Alignment Requirements

When you use SQLBindCol, SQLBindParameter, or SQLGetData, a C data type is specified for the column or parameter.

On certain platforms, the storage (memory) provided for each column must be properly aligned to fetch or store a value of the specified type. The ODBC driver checks for proper data alignment. When an object is not properly aligned, the ODBC driver will issue an *"Invalid string or buffer length"* message (*SQLSTATE* HY090 or S1090).

The following table lists memory alignment requirements for processors such as Sun Sparc, Itanium-IA64, and ARM-based devices. The memory address of the data value must be a multiple of the indicated value.

C Data Type	Alignment Required
SQL_C_CHAR	none
SQL_C_BINARY	none
SQL_C_GUID	none
SQL_C_BIT	none
SQL_C_STINYINT	none
SQL_C_UTINYINT	none
SQL_C_TINYINT	none
SQL_C_NUMERIC	none

C Data Type	Alignment Required
SQL_C_DEFAULT	none
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 (buffer size must be a multiple of 2 on all platforms)
SQL_C_SLONG	4
SQL_C_ULONG	4
SQL_C_LONG	4
SQL_C_FLOAT	4
SQL_C_DOUBLE	8 (4 for ARM)
SQL_C_SBIGINT	8
SQL_C_UBIGINT	8

The x86, x64, and PowerPC platforms do not require memory alignment. The x64 platform includes Advanced Micro Devices (AMD) AMD64 processors and Intel Extended Memory 64 Technology (EM64T) processors.

1.5.13 Result Sets in ODBC Applications

ODBC applications use cursors to manipulate and update result sets. The software provides extensive support for different kinds of cursors and cursor operations.

In this section:

ODBC Transaction Isolation Levels [page 187]
You can use SQLSetConnectAttr to set the transaction isolation level for a connection.

ODBC Cursor Characteristics [page 188]
ODBC functions that execute statements and manipulate result sets, use cursors to perform their tasks. Applications open a cursor implicitly whenever they execute a SQLExecute or SQLExecDirect function.

Data Retrieval [page 189]
To retrieve rows from a database, you execute a SELECT statement using SQLExecute or SQLExecDirect. This opens a cursor on the statement.

Row Updates and Deletes Through a Cursor [page 191]

You can update and delete rows through a cursor.

Bookmarks [page 192]
ODBC provides **bookmarks**, which are values used to identify rows in a cursor. The ODBC driver supports bookmarks for value-sensitive and insensitive cursors. For example, the ODBC cursor types SQL_CURSOR_STATIC and SQL_CURSOR_KEYSET_DRIVEN support bookmarks while cursor types SQL_CURSOR_DYNAMIC and SQL_CURSOR_FORWARD_ONLY do not.

Related Information

Cursor Principles [page 19]

1.5.13.1 ODBC Transaction Isolation Levels

You can use SQLSetConnectAttr to set the transaction isolation level for a connection.

The characteristics that determine the transaction isolation level that the software provides include the following:

SQL_TXN_READ_UNCOMMITTED

Set isolation level to 0. When this attribute value is set, it isolates any data read from changes by others and changes made by others cannot be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read. This is the default value for isolation level.

SQL_TXN_READ_COMMITTED

Set isolation level to 1. When this attribute value is set, it does not isolate data read from changes by others, and changes made by others can be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read.

SQL_TXN_REPEATABLE_READ

Set isolation level to 2. When this attribute value is set, it isolates any data read from changes by others, and changes made by others cannot be seen. The re-execution of the read statement is affected by others. This supports a repeatable read.

SQL_TXN_SERIALIZABLE

Set isolation level to 3. When this attribute value is set, it isolates any data read from changes by others, and changes made by others cannot be seen. The re-execution of the read statement is not affected by others. This supports a repeatable read.

SA_SQL_TXN_SNAPSHOT

Set isolation level to Snapshot. When this attribute value is set, it provides a single view of the database for the entire transaction.

SA_SQL_TXN_STATEMENT_SNAPSHOT

Set isolation level to Statement-snapshot. When this attribute value is set, it provides less consistency than Snapshot isolation, but may be useful when long running transactions result in too much space being used in the temporary file by the version store.

SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT

Set isolation level to Readonly-statement-snapshot. When this attribute value is set, it provides less consistency than Statement-snapshot isolation, but avoids the possibility of update conflicts. Therefore, it is most appropriate for porting applications originally intended to run under different isolation levels.

The allow_snapshot_isolation database option must be set to On to use the Snapshot, Statement-snapshot, or Readonly-statement-snapshot settings.

Example

The following fragment sets the isolation level to Snapshot:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION,
      SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

Related Information

SQLSetConnectAttr Function
Microsoft Open Database Connectivity (ODBC)

1.5.13.2 ODBC Cursor Characteristics

ODBC functions that execute statements and manipulate result sets, use cursors to perform their tasks. Applications open a cursor implicitly whenever they execute a SQLExecute or SQLExecDirect function.

For applications that move through a result set only in a forward direction and do not update the result set, cursor behavior is relatively straightforward. By default, ODBC applications request this behavior. ODBC defines a read-only, forward-only cursor, and the database server provides a cursor optimized for performance in this case.

For applications that scroll both forward and backward through a result set, such as many graphical user interface applications, cursor behavior is more complex. ODBC defines a variety of **scrollable cursors** to allow you to build in the behavior that suits your application. The database server provides a full set of cursors to match the ODBC scrollable cursor types.

You set the required ODBC cursor characteristics by calling the SQLSetStmtAttr function that defines statement attributes. You must call SQLSetStmtAttr before executing a statement that creates a result set.

You can use SQLSetStmtAttr to set many cursor characteristics. The characteristics that determine the cursor type that the database server supplies include the following:

SQL_ATTR_CURSOR_SCROLLABLE

Set to SQL_SCROLLABLE for a scrollable cursor and SQL_NONSCROLLABLE for a forward-only cursor. SQL_NONSCROLLABLE is the default.

SQL_ATTR_CONCURRENCY

Set to one of the following values:

SQL_CONCUR_READ_ONLY

Disallow updates. SQL_CONCUR_READ_ONLY is the default.

SQL_CONCUR_LOCK

Use the lowest level of locking sufficient to ensure that the row can be updated.

SQL_CONCUR_ROWVER

Use optimistic concurrency control, employing a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

SQL_CONCUR_VALUES

Use optimistic concurrency control, employing a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

Example

The following fragment requests a read-only, scrollable cursor:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE,
      SQL_SCROLLABLE, SQL_IS_UINTEGER );
```

Related Information

Data Retrieval [page 189]
Value-sensitive Cursors [page 37]
SQLSetStmtAttr Function

1.5.13.3 Data Retrieval

To retrieve rows from a database, you execute a SELECT statement using SQLExecute or SQLExecDirect. This opens a cursor on the statement.

You then use SQLFetch or SQLFetchScroll to fetch rows through the cursor. These functions fetch the next rowset of data from the result set and return data for all bound columns. Using SQLFetchScroll, rowsets can be specified at an absolute or relative position or by bookmark. SQLFetchScroll replaces the older SQLExtendedFetch from the ODBC 2.0 specification.

When an application frees the statement using SQLFreeHandle, it closes the cursor.

To fetch values from a cursor, your application can use either SQLBindCol or SQLGetData. If you use SQLBindCol, values are automatically retrieved on each fetch. If you use SQLGetData, you must call it for each column after each fetch.

SQLGetData is used to fetch values in pieces for columns such as LONG VARCHAR or LONG BINARY. As an alternative, you can set the SQL_ATTR_MAX_LENGTH statement attribute to a value large enough to hold the entire value for the column. The default value for SQL_ATTR_MAX_LENGTH is 256 KB.

The ODBC driver implements SQL_ATTR_MAX_LENGTH in a different way than intended by the ODBC specification. The intended meaning for SQL_ATTR_MAX_LENGTH is that it be used as a mechanism to truncate large fetches. This might be done for a "preview" mode where only the first part of the data is displayed. For example, instead of transmitting a 4 MB blob from the server to the client application, only the first 500 bytes of it might be transmitted (by setting SQL_ATTR_MAX_LENGTH to 500). The ODBC driver does not support this implementation.

When you use SQLBindCol to bind a NUMERIC or DECIMAL column to a SQL_C_NUMERIC target type, the data value is stored in a 128-bit field (*val*) of a SQL_NUMERIC_STRUCT. This field can only accommodate a maximum precision of 38. The database server supports a maximum NUMERIC precision of 127. When the precision of a NUMERIC or DECIMAL column is greater than 38, the column should be bound as SQL_C_CHAR to avoid loss of precision.

The following code fragment opens a cursor on a query and retrieves data through the cursor. Error checking has been omitted to make the example easier to read. The fragment is taken from a complete sample, which can be found in *%SQLANYSAMP17%*`\SQLAnywhere\ODBCSelect\odbcselect.cpp`.

```
int main( int argc, char* argv[] )
{
    #define DEPT_NAME_LEN 40
    SQLLEN          cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;
    SQLCHAR         deptName[ DEPT_NAME_LEN + 1];
    SQLSMALLINT     deptID, managerID;
    SQLHENV         env;
    SQLHDBC         dbc;
    SQLHSTMT        stmt;
    SQLRETURN       retcode;
    argc = ProcessOptions( argv );
    if( argc < 0 ) return( 1 );
    retcode = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
        /* Set the ODBC version environment attribute */
        retcode = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION,
(void*)SQL_OV_ODBC3, 0);
        retcode = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
            retcode = SQLConnect( dbc,
                    (SQLCHAR*) DataSourceName, SQL_NTS,
                    (SQLCHAR*) UserName, SQL_NTS,
                    (SQLCHAR*) Password, SQL_NTS );
            if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
                retcode = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
                if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
                    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);
                    retcode = SQLExecDirect( stmt, (SQLCHAR * )
                            "SELECT DepartmentID, DepartmentName,
DepartmentHeadID "
                            "FROM Departments "
                            "ORDER BY DepartmentID", SQL_NTS );
                    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO) {
                        while( ( 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 );
    return( (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) ? 0 :
1 );
}
```

The number of row positions you can fetch in a cursor is governed by the size of an integer. You can fetch rows numbered up to number 2147483646, which is one less than the value that can be held in a 32-bit integer. When using negative numbers (rows from the end) you can fetch down to one more than the largest negative value that can be held in an integer.

1.5.13.4 Row Updates and Deletes Through a Cursor

You can update and delete rows through a cursor.

When you use positioned update statements, you do not need to execute a SELECT ... FOR UPDATE statement.

Cursors are automatically updatable as long as the following conditions are met:

- The underlying query supports updates.
  That is to say, as long as a data manipulation statement on the columns in the result is meaningful, then positioned data manipulation statements can be carried out on the cursor.
  The ansi_update_constraints database option limits the type of queries that are updatable.
- The cursor type supports updates.
  If you are using a read-only cursor, you cannot update the result set.

ODBC provides two alternatives for carrying out positioned updates and deletes:

- Use the SQLSetPos function.
  Depending on the parameters supplied (SQL_POSITION, SQL_REFRESH, SQL_UPDATE, SQL_DELETE) SQLSetPos sets the cursor position and allows an application to refresh data, or update, or delete data in the result set.
  This is the method to use with the database server.
- Send positioned UPDATE and DELETE statements using SQLExecute. This method should not be used with the database server.

Related Information

ansi_update_constraints Option

1.5.13.5 Bookmarks

ODBC provides **bookmarks**, which are values used to identify rows in a cursor. The ODBC driver supports bookmarks for value-sensitive and insensitive cursors. For example, the ODBC cursor types SQL_CURSOR_STATIC and SQL_CURSOR_KEYSET_DRIVEN support bookmarks while cursor types SQL_CURSOR_DYNAMIC and SQL_CURSOR_FORWARD_ONLY do not.

Before ODBC 3.0, a database could specify only whether it supported bookmarks or not: there was no interface to provide this information for each cursor type. There was no way for a database server to indicate for what kind of cursor bookmarks were supported. For ODBC 2 applications, the ODBC driver returns that it does support bookmarks. There is therefore nothing to prevent you from trying to use bookmarks with dynamic cursors; however, do not use this combination.

1.5.14 Stored Procedure Considerations

You can create and call stored procedures and process the results from an ODBC application.

Procedures and Result Sets

There are two types of procedures: those that return result sets and those that do not. You can use SQLNumResultCols to tell the difference: the number of result columns is zero if the procedure does not return a result set. If there is a result set, you can fetch the values using SQLFetch or SQLExtendedFetch just like any other cursor.

Parameters to procedures should be passed using parameter markers (question marks). Use SQLBindParameter to assign a storage area for each parameter marker, whether it is an INPUT, OUTPUT, or INOUT parameter.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure-defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the stored procedure definition. To avoid this problem, you can use column aliases in your procedure result set cursor.

Example

Example 1

This example creates and calls a procedure that does not return a result set. The procedure takes one INOUT parameter, and increments its value. In the example, the variable `num_columns` has the value zero, since the procedure does not return a result set. Error checking has been omitted to make the example easier to read.

```
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 );
```

Example 2

This example calls a procedure that returns a result set. In the example, the variable `num_columns` will have the value 2 since the procedure returns a result set with two columns. Again, error checking has been omitted to make the example easier to read.

```
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 )
   {
      rc = SQLMoreResults( stmt );
      if( rc == SQL_NO_DATA ) break;
   }
   else
   {
      do_something( ID, Surname );
   }
}
SQLCloseCursor( stmt );
SQLFreeHandle( SQL_HANDLE_STMT, stmt );
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions

1.5.15 ODBC Escape Syntax

You can use ODBC escape syntax from any ODBC application. This escape syntax allows you to call a set of common functions regardless of the database management system you are using.

> **Note**
>
> If you do not use escape syntax, then turn off escape syntax parsing in your ODBC application by setting the NOSCAN statement attribute. This improves performance by stopping the ODBC driver from scanning all SQL statements before sending them to the database server for execution. The following statement sets the NOSCAN statement attribute:
>
> ```
> SQLSetStmtAttr ( hstmt, SQL_ATTR_NOSCAN, (SQLPOINTER) SQL_NOSCAN_ON,
> SQL_IS_UINTEGER );
> ```

The general form for the escape syntax is:

```
{ keyword parameters }
```

The set of keywords includes the following:

{d date-string}

The date string is any date value accepted by the database server.

{t time-string}

The time string is any time value accepted by the database server.

{ts date-string time-string}

The date/time string is any timestamp value accepted by the database server.

{guid uuid-string}

The uuid-string is any valid GUID string, for example, 41dfe9ef-db91-11d2-8c43-006008d26a6f.

{oj outer-join-expr}

The outer-join-expr is a valid OUTER JOIN expression accepted by the database server.

{? = call func(p1, ...)}

The function is any valid function call accepted by the database server.

{call proc(p1, ...)}

The procedure is any valid stored procedure call accepted by the database server.

{fn func(p1, ...)}

The function is any one of the library of functions listed below.

You can use the escape syntax to access a library of functions implemented by the ODBC driver that includes number, string, time, date, and system functions.

For example, to obtain the current date in a database management system-neutral way, you would execute the following:

```
SELECT { FN CURDATE() }
```

The following tables list the functions that are supported by the ODBC driver.

ODBC Driver Supported Functions

Numeric Functions	String Functions	System Functions	Time/Date Functions
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH	CONVERT	CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		TIMESTAMPADD
ROUND	RTRIM		TIMESTAMPDIFF
SIGN	SOUNDEX		WEEK
SIN	SPACE		YEAR
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

ODBC TIMESTAMPADD, TIMESTAMPDIFF

The ODBC driver maps the TIMESTAMPADD and TIMESTAMPDIFF functions to the corresponding database server DATEADD and DATEDIFF functions. The syntax for the TIMESTAMPADD and TIMESTAMPDIFF functions is as follows.

```
{ fn TIMESTAMPADD( interval, integer-expr, timestamp-expr ) }
```

Returns the timestamp calculated by adding `integer-expr` intervals of type `interval` to `timestamp-expr`. Valid values of `interval` are shown below.

```
{ fn TIMESTAMPDIFF( interval, timestamp-expr1, timestamp-expr2 ) }
```

Returns the integer number of intervals of type `interval` by which `timestamp-expr2` is greater than `timestamp-expr1`. Valid values of `interval` are shown below.

interval	DATEADD/DATEDIFF Date-Part Mapping
SQL_TSI_YEAR	YEAR
SQL_TSI_QUARTER	QUARTER
SQL_TSI_MONTH	MONTH
SQL_TSI_WEEK	WEEK
SQL_TSI_DAY	DAY
SQL_TSI_HOUR	HOUR
SQL_TSI_MINUTE	MINUTE
SQL_TSI_SECOND	SECOND
SQL_TSI_FRAC_SECOND	MICROSECOND - The DATEADD and DATEDIFF functions do not support a resolution of nanoseconds.

Interactive SQL

The ODBC escape syntax is identical to the JDBC escape syntax. In Interactive SQL, which uses JDBC, the braces *must* be doubled. There must not be a space between successive braces: "{{" is acceptable, but "{ {" is not. As well, you cannot use newline characters in the statement. The escape syntax cannot be used in stored procedures because they are not parsed by Interactive SQL.

For example, to obtain the number of weeks in February 2013, execute the following in Interactive SQL:

```
SELECT {{ fn TIMESTAMPDIFF(SQL_TSI_WEEK, '2013-02-01T00:00:00',
'2013-03-01T00:00:00'  ) }}
```

1.5.16 Error Handling in ODBC

Errors in ODBC are reported using the return value from each of the ODBC function calls and either the SQLError function or the SQLGetDiagRec function.

The SQLError function was used in ODBC versions up to, but not including, version 3. As of version 3 the SQLError function has been deprecated and replaced by the SQLGetDiagRec function.

Every ODBC function returns a SQLRETURN, which is one of the following status codes:

Status Code	Description
SQL_SUCCESS	No error.
SQL_SUCCESS_WITH_INFO	The function completed, but a call to SQLError will indicate a warning. The most common case for this status is that a value being returned is too long for the buffer provided by the application.
SQL_ERROR	The function did not complete because of an error. Call SQLError to get more information about the problem.
SQL_INVALID_HANDLE	An invalid environment, connection, or statement handle was passed as a parameter. This often happens if a handle is used after it has been freed, or if the handle is the null pointer.
SQL_NO_DATA_FOUND	There is no information available. The most common use for this status is when fetching from a cursor; it indicates that there are no more rows in the cursor.
SQL_NEED_DATA	Data is needed for a parameter. This is an advanced feature described in the ODBC SDK documentation under SQLParamData and SQLPutData.

Every environment, connection, and statement handle can have one or more errors or warnings associated with it. Each call to SQLError or SQLGetDiagRec returns the information for one error and removes the information for that error. If you do not call SQLError or SQLGetDiagRec to remove all errors, the errors are removed on the next function call that passes the same handle as a parameter.

Each call to SQLError passes three handles for an environment, connection, and statement. The first call uses SQL_NULL_HSTMT to get the error associated with a connection. Similarly, a call with both SQL_NULL_DBC and SQL_NULL_HSTMT get any error associated with the environment handle.

Each call to SQLGetDiagRec can pass either an environment, connection or statement handle. The first call passes in a handle of type SQL_HANDLE_DBC to get the error associated with a connection. The second call passes in a handle of type SQL_HANDLE_STMT to get the error associated with the statement that was just executed.

SQLError and SQLGetDiagRec return SQL_SUCCESS if there is an error to report (*not* SQL_ERROR), and SQL_NO_DATA_FOUND if there are no more errors to report.

Example

Example 1

The following code fragment uses SQLError and return codes:

```
// ODBC 2.0
RETCODE rc;
HENV env;
HDBC dbc;
HSTMT stmt;
SDWORD err_native;
UCHAR err_state[6];
UCHAR err_msg[ 512 ];
SWORD err_ind;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
   printf( "Allocation failed\n" );
   for(;;)
   {
      rc = SQLError( env, dbc, SQL_NULL_HSTMT, err_state, &err_native,
               err_msg, sizeof(err_msg), &err_ind );
      if( rc  < SQL_SUCCESS ) break;
      if( rc == SQL_NO_DATA_FOUND ) break;
      printf( "[%s:%d] %s\n", err_state, err_native, err_msg );
   }
   return;
}
rc = SQLExecDirect( stmt,
       "DELETE FROM SalesOrderItems WHERE ID=2015",
       SQL_NTS );
if( rc == SQL_ERROR )
{
   printf( "Failed to delete items\n" );
   for(;;)
   {
      rc = SQLError( env, dbc, stmt, err_state, &err_native,
               err_msg, sizeof(err_msg), &err_ind );
      if( rc  < SQL_SUCCESS ) break;
      if( rc == SQL_NO_DATA_FOUND ) break;
      printf( "[%s:%d] %s\n", err_state, err_native, err_msg );
   }
   return;
}
```

Example 2

The following code fragment uses SQLGetDiagRec and return codes:

```
// ODBC 3.0
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT rec;
SQLINTEGER err_native;
SQLCHAR err_state[6];
SQLCHAR err_msg[ 512 ];
SQLSMALLINT err_ind;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
   printf( "Failed to allocate handle\n" );
   for( rec = 1; ; rec++ )
   {
      rc = SQLGetDiagRec( SQL_HANDLE_DBC, dbc, rec, err_state, &err_native,
               err_msg, sizeof(err_msg), &err_ind );
      if( rc  < SQL_SUCCESS ) break;
      if( rc == SQL_NO_DATA_FOUND ) break;
      printf( "[%s:%d] %s\n", err_state, err_native, err_msg );
   }
```

```
   return;
}
rc = SQLExecDirect( stmt,
       "DELETE FROM SalesOrderItems WHERE ID=2015",
       SQL_NTS );
if( rc == SQL_ERROR )
{
   printf( "Failed to delete items\n" );
   for( rec = 1; ; rec++ )
   {
      rc = SQLGetDiagRec( SQL_HANDLE_STMT, stmt, rec, err_state, &err_native,
                err_msg, sizeof(err_msg), &err_ind );
      if( rc  < SQL_SUCCESS ) break;
      if( rc == SQL_NO_DATA_FOUND ) break;
      printf( "[%s:%d] %s\n", err_state, err_native, err_msg );
   }
   return;
}
```

1.6 Java in the Database

The database server supports a mechanism for executing Java classes from within the database environment. Using Java methods from the database server provides powerful ways of adding programming logic to a database.

Java support in the database offers the following:

- Reuse Java components in the different layers of your application (client, middle-tier, or server) and use them wherever it makes the most sense to you.
- Java provides a more powerful language than the SQL stored procedure language for building logic into the database.
- Java can be used in the database server without jeopardizing the integrity, security, or robustness of the database and the server.

The SQLJ Standard

Java in the database is based on the SQLJ Part 1 proposed standard (ANSI/INCITS 331.1-1999). SQLJ Part 1 provides specifications for calling Java static methods as SQL stored procedures and functions.

In this section:

Introduction to Java in the Database [page 200]
SQL stored procedure syntax is extended to permit the calling of Java methods from SQL.

Java Error Handling [page 201]
Errors in Java applications generate an exception object representing the error (called **throwing an exception**). A thrown exception terminates a Java program unless it is caught and handled properly at some level of the application.

Tutorial: Using Java in the Database [page 202]
This tutorial describes the steps involved in creating Java methods and calling them from SQL.

How to Install Java Classes into a Database [page 209]
You can install Java classes into a database as a single class or a JAR.

Special Features of Java Classes in the Database [page 213]
The following material describes features of Java classes when used in the database.

How to Start and Stop the Java VM [page 218]
The Java VM loads automatically whenever the first Java operation is carried out. You can use the START JAVA and STOP JAVA statements to manually start and stop the Java VM.

Shutdown Hooks in the Java VM [page 218]
The built-in Java VM ClassLoader, which is used in providing JAVA in the database support allows applications to install shutdown hooks.

1.6.1 Introduction to Java in the Database

SQL stored procedure syntax is extended to permit the calling of Java methods from SQL.

The following support is provided:

You can run Java from the database

An external Java VM runs your Java code on behalf of the database server.

You can access data from Java

Your Java code can access data from the database.

SQL is preserved

The use of Java does not alter the behavior of existing SQL statements or other aspects of non-Java relational database behavior.

Java provides several features that make it ideal for use in database applications:

- Compile-time error checking.
- Built-in error handing with a well-defined error handling methodology.
- Built-in garbage collection (memory recovery).
- Elimination of many bug-prone programming techniques.
- Strong security features.
- Platform-independent execution of Java code.

The Java language is more powerful than SQL. Java is an object-oriented language, so its instructions (source code) come in the form of classes. To execute Java in a database, you write the Java instructions outside the database and compile them outside the database into compiled classes (byte code), which are binary files holding Java instructions.

Compiled classes can be called from client applications as easily and in the same way as stored procedures. Java classes can contain both information about the subject and some computational logic. For example, you could design, write, and compile Java code to create an Employees class complete with various methods that perform operations on an Employees table. You install your Java classes as objects into a database and write SQL cover functions or procedures to invoke the methods in the Java classes.

Once installed, you can execute these classes from the database server using stored procedures. For example, the following statement creates the interface to a Java procedure:

```
CREATE PROCEDURE MyMethod( )
EXTERNAL NAME 'JDBCExample.MyMethod()V'
LANGUAGE JAVA;
```

The database server facilitates a runtime environment for Java classes, not a Java development environment. You need a Java development environment, such as the Java Development Kit (JDK), to write and compile Java. You also need a Java Runtime Environment to execute Java classes.

You can use many of the classes that are part of the Java API as included in the Java Development Kit. You can also use classes created and compiled by Java developers.

The database server launches a Java VM. The Java VM interprets compiled Java instructions and runs them on behalf of the database server. The database server starts the Java VM automatically when needed: you do not have to take any explicit action to start or stop the Java VM.

The SQL request processor in the database server has been extended so it can call into the Java VM to execute Java instructions. It can also process requests from the Java VM to enable data access from Java.

Related Information

How to Install Java Classes into a Database [page 209]
CREATE PROCEDURE Statement [External Call]

1.6.2 Java Error Handling

Errors in Java applications generate an exception object representing the error (called **throwing an exception**). A thrown exception terminates a Java program unless it is caught and handled properly at some level of the application.

Both Java API classes and custom-created classes can throw exceptions. In fact, users can create their own exception classes that throw their own custom-created classes of errors.

If there is no exception handler in the body of the method where the exception occurred, then the search for an exception handler continues up the call stack. If the top of the call stack is reached and no exception handler has been found, the default exception handler of the Java interpreter running the application is called and the program terminates.

If a SQL statement calls a Java method, and an unhandled exception is thrown, a SQL error is generated. The full text of the Java exception plus the Java stack trace is displayed in the server messages window.

1.6.3 Tutorial: Using Java in the Database

This tutorial describes the steps involved in creating Java methods and calling them from SQL.

Prerequisites

You must have the following system privileges.

- MANAGE ANY EXTERNAL ENVIRONMENT
- SET ANY SYSTEM OPTION
- SERVER OPERATOR
- MANAGE ANY EXTERNAL OBJECT
- CREATE PROCEDURE

Context

It shows you how to compile and install a Java class into the database to make it available for use. It also shows you how to access the class and its members and methods from SQL statements.

It is assumed that you have a Java Development Kit (JDK) installed, including the Java compiler (javac) and Java VM.

Source code and batch files for the sample are provided in *%SQLANYSAMP17%* `\SQLAnywhere\JavaInvoice`.

1. Lesson 1: Compiling a Java Program [page 203]
   Write Java code and compile it as the first step to using Java in the database.
2. Lesson 2: Selecting a Java VM [page 204]
   Set up the database server to locate a Java Virtual Machine (VM). Since you can specify a different Java VM for each database, the ALTER EXTERNAL ENVIRONMENT statement can be used to indicate the location (path) of the Java VM.
3. Lesson 3: Installing a Java Class [page 206]
   Install Java classes into a database so that they can be used from SQL.
4. Lesson 4: Calling Methods in a Java Class [page 207]
   Create stored procedures or functions that act as wrappers that call the Java methods in the class.

1.6.3.1 Lesson 1: Compiling a Java Program

Write Java code and compile it as the first step to using Java in the database.

Prerequisites

Install a Java Development Kit (JDK), including the Java compiler (javac) and a Java Runtime Environment (JRE).

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The database server uses the CLASSPATH environment variable to locate a file during the installation of classes.

Procedure

1. Open a command prompt and go to the *%SQLANYSAMP17%*`\SQLAnywhere\JavaInvoice` folder.

   ```
   cd %SQLANYSAMP17%\SQLAnywhere\JavaInvoice
   ```

2. Compile the Java source code example using the following command:

   ```
   javac Invoice.java
   ```

3. This step is optional. Before starting the database server, make sure that the location of your compiled class file is included in the CLASSPATH environment variable. It is the CLASSPATH of the database server that is used, not the CLASSPATH of the client running Interactive SQL. Here is an example:

   ```
   SET CLASSPATH=%SQLANYSAMP17%\SQLAnywhere\JavaInvoice
   ```

Results

The `javac` command creates a class file that can be installed into the database.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Java in the Database [page 202]

Next task: Lesson 2: Selecting a Java VM [page 204]

1.6.3.2 Lesson 2: Selecting a Java VM

Set up the database server to locate a Java Virtual Machine (VM). Since you can specify a different Java VM for each database, the ALTER EXTERNAL ENVIRONMENT statement can be used to indicate the location (path) of the Java VM.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

If you do not have a Java Runtime Environment (JRE) installed, you can install and use any Java JRE as long as it is version 1.6 or later (JRE 6 or later). Most Java installers set up one of the JAVA_HOME or JAVAHOME environment variables. If neither of these environment variables exist, you can create one manually, and point it to the root directory of your Java VM. However, this configuration is not required if you use the ALTER EXTERNAL ENVIRONMENT statement.

Procedure

1. Use Interactive SQL to start the personal database server and connect to the sample database.

   ```
   dbisql -c "DSN=SQL Anywhere 17 Demo;PWD=sql"
   ```

2. This step is optional. Execute a statement like the following.

   ```
   ALTER EXTERNAL ENVIRONMENT JAVA
   LOCATION 'c:\\jdk1.8.0\\jre\\bin\\java.exe';
   ```

   If the location of the Java VM is specified using the LOCATION clause of the ALTER EXTERNAL ENVIRONMENT JAVA statement and the location specified is incorrect, then the database server will not load the Java VM.

If the location of the Java VM is not specified using the LOCATION clause, then the database server searches for the location of the Java VM as follows:

- Check the JAVA_HOME environment variable.
- Check the JAVAHOME environment variable.
- Check the system PATH.
- If the VM cannot be located, return an error.

3. This step is optional. Use the java_vm_options database option to specify any additional command-line options that are required to start the Java VM. Replace `java-options` with a string of valid Java VM options.

```
SET OPTION PUBLIC.java_vm_options=java-options;
```

4. Use the START JAVA statement to start the Java VM.

```
START JAVA;
```

This statement attempts to preload the Java VM. If the database server is not able to locate and start the Java VM, then an error message is issued. This statement is optional since the database server automatically loads the Java VM when it is required.

Results

The LOCATION clause of the ALTER EXTERNAL ENVIRONMENT JAVA statement indicates the location of the Java VM. The START JAVA statement loads the Java VM.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Java in the Database [page 202]

Previous task: Lesson 1: Compiling a Java Program [page 203]

Next task: Lesson 3: Installing a Java Class [page 206]

Related Information

ALTER EXTERNAL ENVIRONMENT Statement
START JAVA Statement
java_vm_options Option

1.6.3.3 Lesson 3: Installing a Java Class

Install Java classes into a database so that they can be used from SQL.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The database server uses the class path defined by the -cp database server option and the java_class_path database option to locate a file during the installation of classes. If the file listed in the INSTALL JAVA statement is located in a directory or ZIP file specified by the database server's class path, the server successfully locates the file and installs the class.

Procedure

Use Interactive SQL to execute a statement like the following. The `path` to the location of your compiled class file is not required if it can be located using the database server's class path. The `path`, if specified, must be accessible to the database server.

```
INSTALL JAVA NEW
FROM FILE 'path\\Invoice.class';
```

If an error occurs as a result of this step, check that your JAVA VM path is set correctly. The following statement returns the path to the JAVA VM executable that the database server will use.

```
SELECT db_property('JavaVM');
```

If this result is NULL, then you do not have your path set correctly. If you set the path using the ALTER EXTERNAL ENVIRONMENT JAVA LOCATION statement, then you can determine the current setting as follows:

```
SELECT location FROM SYS.SYSEXTERNENV WHERE name = 'java';
```

If the path is not set or it is not set correctly then return to step 2 of the previous lesson.

Results

The class is now installed into the sample database.

Subsequent changes made to the class file are *not* reflected automatically in the copy of the class file in the database. Whenever the class file is recompiled, use the INSTALL JAVA UPDATE statement to reload the class file into the database.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Java in the Database [page 202]

Previous task: Lesson 2: Selecting a Java VM [page 204]

Next task: Lesson 4: Calling Methods in a Java Class [page 207]

Related Information

How to Install Java Classes into a Database [page 209]
INSTALL JAVA Statement

1.6.3.4 Lesson 4: Calling Methods in a Java Class

Create stored procedures or functions that act as wrappers that call the Java methods in the class.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The Java class file containing the compiled methods from the Invoice example has been loaded into the database.

Procedure

1. Create the following SQL stored procedure to call the Invoice.main method in the sample class:

   ```
   CREATE PROCEDURE InvoiceMain( IN arg1 CHAR(50) )
   EXTERNAL NAME 'Invoice.main([Ljava/lang/String;)V'
   LANGUAGE JAVA;
   ```

   This stored procedure acts as a wrapper to the Java method.
2. Call the stored procedure to call the Java method:

   ```
   CALL InvoiceMain( 'to you' );
   ```

   If you examine the database server message log, you see the message "Hello to you" written there. The database server has redirected the output there from System.out.
3. The following stored procedures illustrate how to pass arguments to and retrieve return values from the Java methods in the Invoice class. If you examine the Java source code, you see that the `init` method of the Invoice class takes both string and double arguments. String arguments are specified using `Ljava/lang/String;`. Double arguments are specified using `D`. The method returns void and this is specified using `V` after the right parenthesis.

   ```
   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. The following functions call Java methods that take no arguments and return a double (`D`) or a string (`Ljava/lang/String;`)

   ```
   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. The following illustrates a sample call to the stored procedure that acts as a wrapper for the `init` method of the Invoice class:

```
CALL init( 'Shirt', 10.00, 'Jacket', 25.00 );
```

6. The following SELECT statement calls several of the other methods in the Invoice class:

```
SELECT getLineItem1Description() as Item1,
       getLineItem1Cost() as Item1Cost,
       getLineItem2Description() as Item2,
       getLineItem2Cost() as Item2Cost,
       rateOfTaxation() as TaxRate,
       totalSum() as Cost;
```

The SELECT statement returns six columns.

Item1	Item1Cost	Item2	Item2Cost	TaxRate	Cost
Shirt	10	Jacket	25	0.15	40.25

Results

You have created stored procedures or functions that act as wrappers for the methods in the Java class. These lessons have taken you through the steps involved in writing Java methods and calling them from SQL.

Task overview: Tutorial: Using Java in the Database [page 202]

Previous task: Lesson 3: Installing a Java Class [page 206]

Related Information

CREATE PROCEDURE Statement [External Call]
CREATE FUNCTION Statement [External Call]

1.6.4 How to Install Java Classes into a Database

You can install Java classes into a database as a single class or a JAR.

A single class

You can install a single class into a database from a compiled class file. Class files typically have extension `.class`.

A JAR

You can install a set of classes all at once if they are in either a compressed or uncompressed JAR file. JAR files typically have the extension `.jar` or `.zip`. The database server supports all compressed JAR files created with the JAR utility, and some other JAR compression schemes.

In this section:

Class File Creation [page 210]
The first step to using Java in the database is to create a Java application or class file.

Installing a Class File [page 210]
Make your Java class available within the database by installing the class into the database.

Installing a JAR File [page 211]
Install the JAR file into the database to make it available within the database.

Updating Classes and JAR Files [page 212]
Replace classes and JAR files with updated copies by using *SQL Central*.

1.6.4.1 Class File Creation

The first step to using Java in the database is to create a Java application or class file.

Although the details of each step may differ depending on whether you are using a Java development tool, the steps involved in creating your own class generally include the following steps:

1. Define your class.
   Write the Java code that defines your class.
2. Name and save your class.
   Save your class declaration (Java code) in a file with the extension `.java`. Make certain the name of the file is the same as the name of the class and that the case of both names is identical.
   For example, a class called Utility should be saved in a file called `Utility.java`.
3. Compile your class.
   This step turns your class declaration containing Java code into a new, separate file containing byte code. The name of the new file is the same as the Java code file, but has an extension of `.class`. You can run a compiled Java class in a Java Runtime Environment, regardless of the platform you compiled it on or the operating system of the runtime environment.

1.6.4.2 Installing a Class File

Make your Java class available within the database by installing the class into the database.

Prerequisites

To install a class, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

You must know the path and file name of the class file you want to install.

Procedure

1. Use the *SQL Central SQL Anywhere 17* plug-in to connect to the database.
2. Open the *External Environments* folder.
3. Under this folder, open the *Java* folder.
4. Right-click the right pane and click *New* > *Java Class*.
5. Follow the instructions in the wizard.

Results

The class is installed into the database and is ready for use.

Related Information

INSTALL JAVA Statement

1.6.4.3 Installing a JAR File

Install the JAR file into the database to make it available within the database.

Prerequisites

To install a JAR, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

You must know the path and file name of the JAR file you want to install. A JAR file can have the extension JAR or ZIP. Each JAR file must have a name in the database. Usually, you use the same name as the JAR file, without the extension. For example, if you install a JAR file named `myjar.zip`, you would generally give it a JAR name of `myjar`.

Context

It is useful and common practice to collect sets of related classes together in packages, and to store one or more packages in a **JAR file**.

Procedure

1. In *SQL Central*, connect to the database.
2. Open the *External Environments* folder.
3. Under this folder, open the *Java* folder.
4. Right-click the right pane and click *New* > *JAR File*.
5. Follow the instructions in the wizard.

Results

A JAR file has been installed into the database and is ready for use.

Related Information

INSTALL JAVA Statement

1.6.4.4 Updating Classes and JAR Files

Replace classes and JAR files with updated copies by using *SQL Central*.

Prerequisites

To update a class or JAR, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

You must have a newer version of the compiled class file or JAR file available.

Context

Only new connections established after installing the class, or that use the class for the first time after installing the class, use the new definition. Once the Java VM loads a class definition, it stays in memory until the connection closes. If you have been using a Java class or objects based on a class in the current connection, disconnect and reconnect to use the new class definition.

Procedure

1. Use the *SQL Central SQL Anywhere 17* plug-in to connect to the database.
2. Open the *External Environments* folder.
3. Under this folder, open the *Java* folder.
4. Locate the subfolder containing the class or JAR file you want to update.
5. Click the class or JAR file and then click *File* > *Update*.
6. In the *Update* window, specify the location and name of the class or JAR file to be updated. You can click *Browse* to search for it.

Results

Only new connections established after installing the class, or that use the class for the first time after installing the class, use the new definition. Once the Java VM loads a class definition, it stays in memory until the connection closes. If you have been using a Java class or objects based on a class in the current connection, disconnect and reconnect to use the new class definition.

Next Steps

You can also update a Java class or JAR file by right-clicking the class or JAR file name and choosing *Update*.

As well, you can update a Java class or JAR file by clicking *Update Now* on the *General* tab of its *Properties* window.

Related Information

INSTALL JAVA Statement

1.6.5 Special Features of Java Classes in the Database

The following material describes features of Java classes when used in the database.

In this section:

How to Call the Main Method [page 214]
You typically start Java applications (outside the database) by running the Java VM on a class that has a main method.

Threads in Java Applications [page 215]
With features of the java.lang.Thread package, you can use multiple threads in a Java application.

No Such Method Exception [page 215]
: If you supply an incorrect number of arguments when calling a Java method, or if you use an incorrect data type, the Java VM responds with a `java.lang.NoSuchMethodException` error. Check the number and type of arguments.

How to Return Result Sets from Java Methods [page 215]
: Write a Java method that returns a result set to the calling environment, and wrap this method in a SQL stored procedure declared with EXTERNAL NAME and LANGUAGE JAVA clauses.

Values Returned from Java via Stored Procedures [page 217]
: You can use stored procedures created using the EXTERNAL NAME and LANGUAGE JAVA clauses as wrappers around Java methods. There is a special technique for returning parameter data from your Java method to SQL OUT or INOUT parameters in the stored procedure.

Security Management for Java [page 218]
: Java provides security managers that you can use to control user access to security-sensitive features of your applications, such as file access and network access.

1.6.5.1 How to Call the Main Method

You typically start Java applications (outside the database) by running the Java VM on a class that has a main method.

For example, the Invoice class in the file *%SQLANYSAMP17%*`\SQLAnywhere\JavaInvoice\Invoice.java` has a main method. When you execute the class from the command line using a command such as the following, it is the main method that executes.

```
java Invoice
```

Calling the Main Method of a Class from SQL

Perform the following steps to call the main method of a class written in Java:

1. Declare the Java main method with an array of strings as an argument:

   ```
   import java.io.*;
   public class JavaClass
   {
       public static void main( String[] args )
       {
           for ( int i = 0; i  < args.length; i++ )
               System.out.println( args[i] );
       }
   }
   ```

2. Compile the Java class and install it in the database.

   ```
   INSTALL JAVA
   NEW
   FROM FILE 'C:\\temp\\JavaClass.class';
   ```

3. Create a stored procedure that wraps the method `main`.

```
CREATE PROCEDURE JavaMain( IN arg CHAR(50) )
EXTERNAL NAME 'JavaClass.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

4. Invoke the main method using the SQL CALL statement.

```
CALL JavaMain( 'Hello, world' );
```

   Due to the limitations of the SQL language, only a single string can be passed.
   Check the database server messages window for a "hello" message from the Java application.

Related Information

CREATE PROCEDURE Statement [External Call]

1.6.5.2 Threads in Java Applications

With features of the java.lang.Thread package, you can use multiple threads in a Java application.

You can synchronize, suspend, resume, interrupt, or stop threads in Java applications.

1.6.5.3 No Such Method Exception

If you supply an incorrect number of arguments when calling a Java method, or if you use an incorrect data type, the Java VM responds with a `java.lang.NoSuchMethodException` error. Check the number and type of arguments.

Related Information

Lesson 4: Calling Methods in a Java Class [page 207]

1.6.5.4 How to Return Result Sets from Java Methods

Write a Java method that returns a result set to the calling environment, and wrap this method in a SQL stored procedure declared with EXTERNAL NAME and LANGUAGE JAVA clauses.

Perform the following tasks to return result sets from a Java method:

1. Ensure that the Java method is declared as public and static in a public class.

2. For each result set you expect the method to return, ensure that the method has a parameter of type java.sql.ResultSet[]. These result set parameters must all occur at the end of the parameter list.
3. In the method, first create an instance of java.sql.ResultSet and then assign it to one of the ResultSet[] parameters.
4. Create a SQL stored procedure using EXTERNAL NAME and LANGUAGE JAVA clauses. This type of procedure is a wrapper around a Java method. You can use a cursor on the SQL procedure result set in the same way as any other procedure that returns result sets.

Example

The following simple class has a single method that executes a query and passes the result set back to the calling environment.

```
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;
  }
}
```

You can expose the result set using a CREATE PROCEDURE statement that indicates the number of result sets returned from the procedure and the signature of the Java method.

A CREATE PROCEDURE statement indicating a result set could be defined as follows:

```
CREATE PROCEDURE result_set( )
  RESULT ( SurName person_name_t )
  DYNAMIC RESULT SETS 1
  EXTERNAL NAME 'MyResultSet.return_rset([Ljava/sql/ResultSet;)V'
  LANGUAGE JAVA;
```

You can open a cursor on this procedure, just as you can with any SQL procedure returning result sets.

The string `([Ljava/sql/ResultSet;)V` is a Java method signature that is a compact character representation of the number and type of the parameters and return value.

Related Information

How to Return Result Sets from Java [page 251]
CREATE PROCEDURE Statement [External Call]

1.6.5.5 Values Returned from Java via Stored Procedures

You can use stored procedures created using the EXTERNAL NAME and LANGUAGE JAVA clauses as wrappers around Java methods. There is a special technique for returning parameter data from your Java method to SQL OUT or INOUT parameters in the stored procedure.

Java does not have explicit support for INOUT or OUT parameters. Instead, you can use an array of the parameter. For example, to use an integer OUT parameter, create an array of exactly one integer:

```
public class Invoice
{
  public static boolean testOut( int[] param )
  {
    param[0] = 123;
    return true;
  }
}
```

The following procedure uses the testOut method:

```
CREATE PROCEDURE testOut( OUT p INTEGER )
EXTERNAL NAME 'Invoice.testOut([I)Z'
LANGUAGE JAVA;
```

The string `([I)Z` is a Java method signature, indicating that the method has a single parameter, which is an array of integers, and returns a Boolean value. Define the method so that the method parameter you want to use as an OUT or INOUT parameter is an array of a Java data type that corresponds to the SQL data type of the OUT or INOUT parameter.

To test this, call the stored procedure with an uninitialized variable.

```
CREATE VARIABLE zap INTEGER;
CALL testOut( zap );
SELECT zap;
```

The result set is `123`.

Related Information

CREATE PROCEDURE Statement [External Call]

1.6.5.6 Security Management for Java

Java provides security managers that you can use to control user access to security-sensitive features of your applications, such as file access and network access.

1.6.6 How to Start and Stop the Java VM

The Java VM loads automatically whenever the first Java operation is carried out. You can use the START JAVA and STOP JAVA statements to manually start and stop the Java VM.

To load the Java VM explicitly in readiness for carrying out Java operations, you can do so by executing the following statement:

```
START JAVA;
```

You can unload the Java VM when Java is not in use using the STOP JAVA statement. The syntax is:

```
STOP JAVA;
```

1.6.7 Shutdown Hooks in the Java VM

The built-in Java VM ClassLoader, which is used in providing JAVA in the database support allows applications to install shutdown hooks.

These shutdown hooks are similar to the shutdown hooks that applications install with the JVM Runtime. When a connection that is using Java in the database support executes a STOP JAVA statement or disconnects, the ClassLoader for that connection runs all shutdown hooks that have been installed for that particular connection prior to unloading. For regular Java in the database applications that install all Java classes within the database, the installation of shutdown hooks should not be necessary. The ClassLoader shutdown hooks should be used with extreme caution and should only be used to clean up any system-wide resources that were allocated for the particular connection that is stopping Java. Also, jdbc:default JDBC requests are not allowed within shutdown hooks since the jdbc:default connection is already closed prior to the ClassLoader shutdown hook being called.

To install a shutdown hook with the Java VM ClassLoader, an application must include `sajvm.jar` in the Java classpath and it needs to execute code similar to the following:

```
SDHookThread hook = new SDHookThread( ... );
ClassLoader classLoader = Thread.currentThread( ).getContextClassLoader( );
((ianywhere.sa.jvm.SAClassLoader)classLoader).addShutdownHook( hook );
```

The SDHookThread class extends the standard Thread class and that the above code must be executed by a class that was loaded by the ClassLoader for the current connection. Any class that is installed within the database and that is later called via an external environment call is automatically executed by the correct Java VM ClassLoader.

To remove a shutdown hook from the Java VM ClassLoader list, an application must execute code similar to the following:

```
ClassLoader classLoader = Thread.currentThread( ).getContextClassLoader( );
((ianywhere.sa.jvm.SAClassLoader)classLoader).removeShutdownHook( hook );
```

The above code must be executed by a class that was loaded by the ClassLoader for the current connection.

1.7 XS JavaScript Application Programming (Deprecated)

The SQL Anywhere XS JavaScript driver can be used to connect to SQL Anywhere databases, issue SQL queries, and obtain result sets.

The SQL Anywhere XS JavaScript driver allows users to interact with the database from the JavaScript environment. The SQL Anywhere XS API is based on the SAP HANA XS JavaScript Database API provided by the SAP HANA XS engine. Drivers are available for various versions of Node.js.

The XS JavaScript driver also supports server-side applications written in JavaScript using SQL Anywhere external environment support.

The SQL Anywhere .XS JavaScript API reference is available in the *SQL Anywhere- XS JavaScript API Reference* at https://help.sap.com/viewer/78bac38a62a2496996d5a27467561290/LATEST/en-US.

> **Note**
>
> In addition to the XS JavaScript driver, a lightweight, minimally-featured Node.js driver is available that can handle small result sets quite well. Node.js application programming using this driver is described elsewhere in the documentation. The lightweight driver is perfect for simple web applications that need a quick and easy connection to the database server to retrieve small result sets. However, if your JavaScript application needs to handle large results, have greater control, or have access to a fuller application programming interface, then you should use the XS JavaScript driver.

Node.js must be installed on your computer and the folder containing the Node.js executable should be included in your PATH. Node.js software is available at nodejs.org .

For Node.js to locate the driver, make sure that the NODE_PATH environment variable includes the location of the XS JavaScript driver. The following is an example for Microsoft Windows:

```
SET NODE_PATH=%SQLANY17%\Node
```

> **Note**
>
> On macOS 10.11 or a later version, set the SQLANY_API_DLL environment variable to the full path for `libdbcapi_r.dylib`.

The following illustrates a simple Node.js application that uses the XS JavaScript driver.

```
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server     : 'demo',
             UserID     : 'DBA',
             Password   : 'sql'
           };
```

```
var conn = sqla.createConnection( cstr );
conn.connect();
console.log( 'Connected' );
stmt = conn.prepareStatement( "SELECT * FROM Customers" );
stmt.execute();
var result = stmt.getResultSet();
while ( result.next() )
{
    console.log( result.getString(1) +
            " " + result.getString(3) +
            " " + result.getString(2) );
}
conn.disconnect();
console.log( 'Disconnected' );
```

This program connects to the sample database, executes a SQL SELECT statement, and displays only the first three columns of the result set, namely the ID, GivenName, and Surname of each customer. It then disconnects from the database.

Suppose this JavaScript code was stored in the file `xs-sample.js`. To run this program, open a command prompt and execute the following statement. Make sure that the NODE_PATH environment variable is set appropriately.

```
node xs-sample.js
```

The following JavaScript example illustrates the use of prepared statements and batches. It creates a database table and populates it with the first 100 positive integers.

```
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server     : 'demo',
             UserID     : 'DBA',
             Password   : 'sql'
           };
var conn = sqla.createConnection( cstr );
conn.connect();
conn.prepareStatement( "CREATE OR REPLACE TABLE mytable( col1 INT)" ).execute();
var stmt = conn.prepareStatement( "INSERT INTO mytable VALUES(?)" );
var BATCHSIZE = 100;
stmt.setBatchSize(BATCHSIZE);
for ( var i = 1; i <= BATCHSIZE; i++ )
{
    stmt.setInteger( 1, i );
    stmt.addBatch();
}
stmt.executeBatch();
stmt.close();
conn.commit();
conn.disconnect();
```

The following JavaScript example illustrates the use of prepared statements and exception handling. The `getcustomer` function returns a hash corresponding to the table row for the specified customer, or an error message.

```
function getcustomer( conn, customer_id )
{
    try
    {
        var query = 'SELECT * FROM Customers WHERE ID=?';

        var pstmt = conn.prepareStatement(query);
        pstmt.setInteger( 1, customer_id );
        var rs = pstmt.executeQuery();
        if ( rs.next() )
```

```
        {
            return(
            {
                id          : rs.getInteger(1),
                surname     : rs.getString(2),
                givenname   : rs.getString(3),
                street      : rs.getString(4),
                city        : rs.getString(5),
                state       : rs.getString(6),
                country     : rs.getString(7),
                postalcode  : rs.getString(8),
                phone       : rs.getString(9),
                companyname : rs.getString(10)
            } );
        }
        else
        {
            return 'No customer with ID=' + customer_id;
        }
    }
    catch (ex)
    {
        return ex.message;
    }
    finally
    {
        if( pstmt )
        {
            pstmt.close();
        }
    }
}
var sqla = require( 'sqlanywhere-xs' );
var cstr = { Server     : 'demo',
             UserID     : 'DBA',
             Password   : 'sql'
            };
var conn = sqla.createConnection( cstr );
conn.connect();
for ( var i = 200; i < 225; i++ )
{
    console.log( getcustomer( conn, i ) );
}
conn.close();
conn.disconnect();
```

Related Information

nodejs.org

1.8 JDBC Support

JDBC is a call-level interface for Java applications. JDBC provides you with a uniform interface to a wide range of relational databases, and provides a common base on which higher level tools and interfaces can be built.

JDBC is now a standard part of Java and is included in the JDK.

The software includes the SQL Anywhere JDBC driver, which is a Type 2 driver.

The software also supports the use of jConnect, a pure Java JDBC driver available from SAP.

In addition to using JDBC as a client-side application programming interface, you can also use JDBC inside the database server to access data by using the Java in the database feature.

In this section:

JDBC Applications [page 223]
You can develop Java applications that use the JDBC API to connect to a database. Several of the applications included with the database software use JDBC.

JDBC Drivers [page 224]
The SQL Anywhere JDBC driver and the jConnect driver are supported.

SQL Anywhere JDBC Driver [page 225]
The SQL Anywhere JDBC driver is recommended for its performance and feature benefits when compared to the pure Java jConnect JDBC driver. However, the SQL Anywhere JDBC driver does not provide a pure-Java solution.

The jConnect JDBC Driver [page 226]
To use JDBC from an applet, you must use the jConnect JDBC driver to connect to a database.

JDBC Program Structure [page 232]
JDBC applications typically connect to a database, execute one or more SQL statements, process result sets, and then disconnect from the database.

Differences Between Client- and Server-side JDBC Applications [page 233]
There are some minor differences between client-side and server-side JDBC applications.

A Sample Client-side JDBC Application [page 234]
A typical JDBC application connects to a database server, issues SQL queries, processes multiple results sets, and then terminates.

A Sample Server-side JDBC Application [page 238]
A typical JDBC application connects to a database server, issues SQL queries, processes multiple results sets, and then terminates.

Notes on JDBC Connections [page 241]
Be aware that there are differences between JDBC on the client side and on the server side. Aspects such as autocommit behavior and isolation levels are described here.

Server-side Data Access Using JDBC [page 242]
Database transaction logic implemented as Java methods that can be called from SQL can offer significant advantages over traditional SQL stored procedures.

JDBC Callbacks [page 253]
The SQL Anywhere JDBC driver supports two asynchronous callbacks, one for handling the SQL MESSAGE statement and the other for validating requests for file transfers. These are similar to the callbacks supported by the ODBC driver.

JDBC Escape Syntax [page 256]
You can use JDBC escape syntax from any JDBC application, including Interactive SQL. This escape syntax allows you to call stored procedures regardless of the database management system you are using.

SQL Anywhere JDBC API Support [page 259]

Some optional methods of the java.sql.Blob interface are not supported by the SQL Anywhere JDBC driver.

1.8.1 JDBC Applications

You can develop Java applications that use the JDBC API to connect to a database. Several of the applications included with the database software use JDBC.

JDBC can be used both from client applications and inside the database. Java classes using JDBC provide a more powerful alternative to SQL stored procedures for incorporating programming logic into the database.

JDBC provides a SQL interface for Java applications: to access relational data from Java, you do so using JDBC calls.

The phrase **client application** applies both to applications running on a user's computer and to logic running on a middle-tier application server.

The examples illustrate the distinctive features of JDBC applications. For more information about JDBC programming, see any JDBC programming book.

You can use JDBC in the following ways:

JDBC on the client

Java client applications can make JDBC calls to a database server. The connection takes place through a JDBC driver.

The SQL Anywhere JDBC driver, which is a Type 2 JDBC driver, is included with the software. The jConnect driver for pure Java applications, which is a Type 4 JDBC driver, is also supported.

JDBC in the database

Java classes installed into a database can make JDBC calls to access and modify data in the database using an internal JDBC driver.

JDBC resources

Example source code

You can find source code for the examples in the directory *%SQLANYSAMP17%*`\SQLAnywhere\JDBC`.

Related Information

The jConnect JDBC Driver [page 226]
Downloading jConnect

1.8.2 JDBC Drivers

The SQL Anywhere JDBC driver and the jConnect driver are supported.

These JDBC drivers have the following characteristics:

SQL Anywhere JDBC driver

This driver communicates with the database server using the Command Sequence client/server protocol. Its behavior is consistent with ODBC, Embedded SQL, and OLE DB applications. The SQL Anywhere JDBC driver is the recommended JDBC driver for connecting to databases. The driver can be used only with JRE 1.6 or later.

The driver performs automatic JDBC driver registration with the JAVA VM. It is therefore sufficient to have the sajdbc4.jar file in the class file path and simply call DriverManager.getConnection() with a URL that begins with jdbc:sqlanywhere.

The JDBC driver contains manifest information to allow it to be loaded as an OSGi (Open Services Gateway initiative) bundle.

With the SQL Anywhere JDBC driver, metadata for NCHAR data now returns the column type as java.sql.Types.NCHAR, NVARCHAR, or LONGNVARCHAR. In addition, applications can now fetch NCHAR data using the Get/SetNString or Get/SetNClob methods instead of the Get/SetString and Get/SetClob methods.

jConnect

This driver is a 100% pure Java driver. It communicates with the database server using the TDS client/server protocol.

When choosing which driver to use, consider the following factors:

Features

The SQL Anywhere JDBC driver provides fully scrollable cursors when connected to a database. The jConnect JDBC driver provides scrollable cursors when connected to a database, but the result set is cached on the client side. The jConnect JDBC driver provides fully scrollable cursors when connected to an Adaptive Server Enterprise database.

Pure Java

The jConnect driver is a Type 4 driver and hence a pure Java solution. The SQL Anywhere JDBC driver is a Type 2 driver and hence is not a pure Java solution.

> ⓘ Note
>
> The SQL Anywhere JDBC driver does not load the SQL Anywhere ODBC driver and hence is not a Type 1 driver.

Performance

The SQL Anywhere JDBC driver provides better performance for most purposes than the jConnect driver.

Compatibility

The TDS protocol used by the jConnect driver is shared with Adaptive Server Enterprise. Some aspects of the driver's behavior are governed by this protocol, and are configured to be compatible with Adaptive Server Enterprise.

Related Information

SAP SQL Anywhere Components by Platforms
Downloading jConnect

1.8.3 SQL Anywhere JDBC Driver

The SQL Anywhere JDBC driver is recommended for its performance and feature benefits when compared to the pure Java jConnect JDBC driver. However, the SQL Anywhere JDBC driver does not provide a pure-Java solution.

In this section:

How to Load the SQL Anywhere JDBC Driver [page 225]
Ensure that the JDBC driver is in your class file path.

SQL Anywhere JDBC Driver Connection Strings [page 226]
To connect to a database via the SQL Anywhere JDBC driver, supply a URL for the database.

Related Information

JDBC Drivers [page 224]

1.8.3.1 How to Load the SQL Anywhere JDBC Driver

Ensure that the JDBC driver is in your class file path.

```
set classpath=%SQLANY17%\java\sajdbc4.jar;%classpath%
```

The JDBC driver takes advantage of the new automatic driver registration. The driver is automatically loaded at execution startup when it is in the class file path.

Required Files

The Java component of the JDBC driver is included in the `sajdbc4.jar` file installed into the `Java` subdirectory of your software installation. For Windows, the native component is `dbjdbc`*`17`*`.dll` in the `bin32` or `bin64` subdirectory of your software installation; for UNIX and Linux, the native component is `libdbjdbc`*`17`*`.so`. This component must be in the system path. For macOS 10.11 or a later version, the `java.library.path` must include the full path to `libdbjdbc17.dylib`. For example: java -Djava.library.path=$SQLANY17/lib64.

Related Information

JDBC Client Deployment [page 847]

1.8.3.2 SQL Anywhere JDBC Driver Connection Strings

To connect to a database via the SQL Anywhere JDBC driver, supply a URL for the database.

The following is an example of how to connect to a database.

```
Connection con = DriverManager.getConnection( "jdbc:sqlanywhere:DSN=SQL Anywhere
17 Demo;Password="+pwd );
```

The URL contains jdbc:sqlanywhere: followed by a connection string. If the `sajdbc4.jar` file is in your class file path, then the SQL Anywhere JDBC driver is loaded automatically and handles the URL. As shown in the example, an ODBC data source (DSN) may be specified for convenience, but you can also use explicit connection parameters, separated by semicolons, in addition to or instead of the data source connection parameter.

If you do not use a data source, you must specify all required connection parameters in the connection string:

```
Connection con =
DriverManager.getConnection( "jdbc:sqlanywhere:UserID=DBA;Password=passwd;Start=.
.." );
```

The Driver connection parameter is not required since neither the ODBC driver nor ODBC driver manager is used. If present, it is ignored.

The following is another example in which a connection is made to the database SalesDB on the server Acme running on the host computer Elora using TCP/IP port 2638.

```
Connection con = DriverManager.getConnection(

"jdbc:sqlanywhere:UserID=DBA;Password=passwd;Host=Elora:2638;ServerName=Acme;Data
baseName=SalesDB" );
```

Related Information

Alphabetical List of Connection Parameters

1.8.4 The jConnect JDBC Driver

To use JDBC from an applet, you must use the jConnect JDBC driver to connect to a database.

The jConnect driver is available as a separate download from the SAP Software Download Center on the SAP Service Marketplace. Search for the SDK FOR SAP ASE. Documentation for jConnect is included in

the install. This link may help you locate the driver: http://sqlanywhere-forum.sap.com/questions/23450/jconnect-software-developer-kit-download.

jConnect is supplied as a JAR file named `jconn42.jar`. This file is located in your jConnect install location.

Setting the Class File Path for jConnect

For your application to use jConnect, the jConnect classes must be in your class file path at compile time and run time, so that the Java compiler and Java runtime can locate the necessary files.

The following command adds the jConnect driver to an existing CLASSPATH environment variable (where `jconnect-path` is your jConnect installation directory).

```
set classpath=jconnect-path\classes\jconn42.jar;%classpath%
```

Importing the jConnect Classes

The classes in jConnect are all in com.sybase.jdbc42.jdbc. You can import these classes at the beginning of each source file if required:

```
import com.sybase.jdbc42.jdbc.*
```

Encrypting Passwords

Passwords can be encrypted for jConnect connections. The following example illustrates this.

```
Connection  connection = null;
Properties props = new Properties();
props.put( "ENCRYPT_PASSWORD", "true" );
props.put( "User", "DBA" );
props.put( "Password", "passwd" );
try {
    Driver drvr = (Driver)
(Class.forName( "com.sybase.jdbc42.jdbc.SybDriver" ).newInstance());
    connection = drvr.connect( "jdbc:sybase:Tds:localhost:2638", props );
}
catch( Exception e ) {
    System.err.println( "Error! Could not connect" );
    System.err.println( e.getMessage() );
    printExceptions( (SQLException)e );
    connection = null;
}
```

In this section:

Installing jConnect System Objects into a Database [page 228]
Add the jConnect system objects to your database so that you can use jConnect to access system table information (database metadata).

How to Load the jConnect Driver [page 229]
Ensure that the jConnect driver is in your class file path. The driver file `jconn42.jar` is located in the `classes` subdirectory of your jConnect installation.

jConnect Driver Connection Strings [page 229]
To connect to a database via jConnect, supply a URL for the database.

Related Information

Downloading jConnect

1.8.4.1 Installing jConnect System Objects into a Database

Add the jConnect system objects to your database so that you can use jConnect to access system table information (database metadata).

Prerequisites

You must have the ALTER DATABASE, BACKUP DATABASE, and SERVER OPERATOR system privileges, and must be the only connection to the database.

Back up your database files before upgrading. If you attempt to upgrade a database and it fails, then the database becomes unusable.

Context

jConnect system objects are installed into a database by default when you use the dbinit utility. You can add the jConnect system objects to the database when creating the database or at a later time by upgrading the database. You can upgrade a database from *SQL Central* as follows.

Procedure

1. Use *SQL Central* to connect to the database.
2. Click *Tools* > *SQL Anywhere 17* and then click *Upgrade Database*.
3. Follow the instructions in the *Upgrade Database Wizard*.

Results

The jConnect system objects are added to the database.

Alternatively, use the Upgrade utility (dbupgrad) to connect to your database and install the jConnect system objects to the database. The following is an example.

```
dbupgrad -c "UID=DBA;PWD=passwd;SERVER=myServer;DBN=myDatabase
```

Related Information

Database Backup and Recovery
Upgrade Utility (dbupgrad)

1.8.4.2 How to Load the jConnect Driver

Ensure that the jConnect driver is in your class file path. The driver file `jconn42.jar` is located in the `classes` subdirectory of your jConnect installation.

```
set classpath=.;c:\jConnect-nn\classes\jconn42.jar;%classpath%
```

The driver is automatically loaded at execution startup when it is in the class file path.

1.8.4.3 jConnect Driver Connection Strings

To connect to a database via jConnect, supply a URL for the database.

The following is an example of how to connect to a database.

```
Connection con = DriverManager.getConnection( "jdbc:sybase:Tds:localhost:2638",
"DBA", "sql" );
```

The URL is composed in the following way:

```
jdbc:sybase:Tds:host:port
```

The individual components are:

jdbc:sybase:Tds

The jConnect JDBC driver, using the TDS application protocol.

host

The IP address or name of the computer on which the server is running. If you are establishing a same-host connection, you can use localhost, which means the computer system you are logged into.

port

The port number on which the database server listens. The default port number used by the database server is 2638.

The connection string must be less than 253 characters in length.

If you are using the personal server, make sure to include the TCP/IP support option when starting the server.

In this section:

How to Specify a Database with a jConnect Connection String [page 230]
Each database server can have one or more databases loaded at a time. If the URL you supply when connecting via jConnect specifies a server, but does not specify a database, then the connection attempt is made to the default database on the server.

Database Options Set for jConnect Connections [page 231]
When an application connects to the database using the jConnect driver, the sp_tsql_environment stored procedure is called. The sp_tsql_environment procedure sets some database options for compatibility with Adaptive Server Enterprise behavior.

1.8.4.3.1 How to Specify a Database with a jConnect Connection String

Each database server can have one or more databases loaded at a time. If the URL you supply when connecting via jConnect specifies a server, but does not specify a database, then the connection attempt is made to the default database on the server.

You can specify a particular database by providing an extended form of the URL in one of the following ways.

Using the ServiceName Parameter

```
jdbc:sybase:Tds:host:port?ServiceName=database
```

The question mark followed by a series of assignments is a standard way of providing arguments to a URL. The case of ServiceName is not significant, and there must be no spaces around the = sign. The `database` parameter is the database name, not the server name. The database name must not include the path or file suffix. For example:

```
Connection con = DriverManager.getConnection(
      "jdbc:sybase:Tds:localhost:2638?ServiceName=demo", "DBA", "sql");
```

Using the RemotePWD Parameter

A workaround exists for passing additional connection parameters to the server.

This technique allows you to provide additional connection parameters such as the database name, or a database file, using the RemotePWD field. You set RemotePWD as a Properties field using the put method.

The following code illustrates how to use the field.

```
import java.util.Properties;
.
.
.
Properties props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "passwd" );
props.put( "RemotePWD", ",DatabaseFile=mydb.db" );

Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", props );
```

As shown in the example, a comma must precede the DatabaseFile connection parameter. Using the DatabaseFile parameter, you can start a database on a server using jConnect. By default, the database is started with AutoStop=YES. If you specify utility_db with a DatabaseFile (DBF) or DatabaseName (DBN) connection parameter (for example, DBN=utility_db), then the utility database is started automatically.

This example also shows additional connection parameters:

```
props.put( "RemotePWD", ",Server=demo17;DBF=demo.db;DBN=demo" );
```

Related Information

The Utility Database (utility_db)

1.8.4.3.2 Database Options Set for jConnect Connections

When an application connects to the database using the jConnect driver, the sp_tsql_environment stored procedure is called. The sp_tsql_environment procedure sets some database options for compatibility with Adaptive Server Enterprise behavior.

Related Information

Open Client/jConnect TDS Compatibility Options
sp_tsql_environment System Procedure

1.8.5 JDBC Program Structure

JDBC applications typically connect to a database, execute one or more SQL statements, process result sets, and then disconnect from the database.

The following are elements of a typical JDBC application:

Create a Connection object

The getConnection method of the DriverManager class creates a Connection object, and establishes a connection with a database.

Create a Statement object

The createStatement method of the Connection object creates a Statement object.

Execute a SQL statement

The execute method of the Statement object executes a SQL statement within the database environment. If one or more result sets are available, the boolean result `true` is returned.

Process one or more result sets

The getResultSet and getMoreResults methods of the Statement object are used to obtain result sets.

The ResultSet object is used to obtain the data returned from the SQL statement, one row at a time.

Loop over the rows of each result set

The next method of the ResultSet object is used to position to the next available row in the result set. When no more rows are available, the boolean result `false` is returned.

For each row, retrieve the values

Values are retrieved for each column in the ResultSet object by identifying either the name or position of the column. You can use the getString method to get the value from a column on the current row.

Release JDBC objects at the appropriate time

The close method of each JDBC object releases resources to the system.

The following example demonstrates the principles outlined above.

```
import java.io.*;
import java.sql.*;
public class results
{
    public static void main(String[] args) throws Exception
    {
        Connection conn = null;
        try
        {
            conn = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
            String SQL = "BEGIN\n"
                + " SELECT * FROM Departments "
                + "     ORDER BY DepartmentID;\n"
                + " SELECT EmployeeID, Surname, GivenName "
                + "     FROM Employees e "
                + "     JOIN Departments d "
                + "     ON DepartmentHeadID = EmployeeID "
                + "     ORDER BY d.DepartmentID\n"
                + "END";
            Statement stmt = conn.createStatement();
            if( stmt.execute(SQL) )
            {
                ResultSet rs = stmt.getResultSet();
```

```
                while( rs != null )
                {
                    System.out.println( "\n---Result set---\n" );
                    while (rs.next())
                    {
                        System.out.println(rs.getString(1)
                                + ", " + rs.getString(2)
                                + ", " + rs.getString(3) );
                    }
                    if( stmt.getMoreResults() )
                    {
                        rs = stmt.getResultSet();
                    }
                    else
                    {
                        rs.close();
                        rs = null;
                    }
                }
            }
            stmt.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        if( conn != null) conn.close();
    }
}
```

Java objects can use JDBC objects to interact with a database and get data for their own use.

1.8.6 Differences Between Client- and Server-side JDBC Applications

There are some minor differences between client-side and server-side JDBC applications.

Client-side

When using JDBC from a client computer, a connection is establish using either the SQL Anywhere JDBC driver or the jConnect JDBC driver. Connection details such as user ID and password are passed as arguments to DriverManager.getConnection which establishes the connection. The database server runs on the same or some other computer system. The JDBC application is contained in one or more class files that are accessible from the client computer.

Server-side

When using JDBC from a database server, a connection already exists. The string "jdbc:default:connection" is passed to DriverManager.getConnection, which allows the JDBC application to work within the context of the current user connection. This is a quick, efficient, and safe operation because the client user has already passed database security to establish a connection. Credentials such as user ID and password, having been provided once, do not need to be provided again. A connection from the JDBC application can only be made back to the database that the launched the application. The JDBC application is contained in one or more class files that are stored in the database.

You can write JDBC classes so that they can run both at the client and at the server by employing a single conditional statement for constructing the URL. An external connection requires connection information such as user ID, password, host name, and port number, while the internal connection requires only "jdbc:default:connection".

1.8.7 A Sample Client-side JDBC Application

A typical JDBC application connects to a database server, issues SQL queries, processes multiple results sets, and then terminates.

The following complete Java application connects to a running database, issues a SQL query, processes and displays multiple results sets, and then terminates. It uses the SQL Anywhere JDBC driver by default to connect to the database. To use the jConnect driver, you pass in a database user ID and password, the driver name (jConnect), and an optional SQL query (enclosed in quotation marks) on the command line.

Database metadata is always available when using the SQL Anywhere JDBC driver.

To access database system tables (database metadata) from a JDBC application that uses jConnect, you must add a set of jConnect system objects to your database. These procedures are installed to all databases by default. The dbinit -i option prevents this installation.

This example assumes that a database server has already been started using the sample database. The source code for this example can be found in *%SQLANYSAMP17%*`\SQLAnywhere\JDBC\JDBCConnect.java`.

```
import java.io.*;
import java.sql.*;
public class JDBCConnect
{
    public static void main( String args[] )
    {
        try
        {
            String userID = "";
            String password = "";
            String driver = "jdbc4";
            String SQL =
                  "BEGIN"
                + " SELECT * FROM Departments"
                + "     ORDER BY DepartmentID;"
                + " SELECT d.DepartmentID, GivenName, Surname, EmployeeID"
                + "     FROM Employees e"
                + "     JOIN Departments d"
                + "     ON DepartmentHeadID = EmployeeID"
                + "     ORDER BY d.DepartmentID;"
                + "END";
            if( args.length > 0 ) userID = args[0];
            if( args.length > 1 ) password = args[1];
            if( args.length > 2 ) driver = args[2];
            if( args.length > 3 ) SQL = args[3];
            Connection con;
            if( driver.compareToIgnoreCase( "jconnect" ) == 0 )
            {
                con = DriverManager.getConnection(
                    "jdbc:sybase:Tds:localhost:2638", userID, password);
            }
            else
            {
                con = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=" + userID + ";pwd=" + password);
            }
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(SQL);
            while( rs != null )
            {
                while (rs.next())
                {
                    for( int i = 1;
                         i <= rs.getMetaData().getColumnCount();
                         i++ )
```

```
                    {
                        if( i > 1 ) System.out.print(", ");
                        System.out.print(rs.getString(i));
                    }
                    System.out.println();
                }
                if( stmt.getMoreResults() )
                {
                    System.out.println();
                    rs = stmt.getResultSet();
                }
                else
                {
                    rs.close();
                    rs = null;
                }
            }
            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);
    }
}
```

In this section:

How the Client-side JDBC Application Works [page 236]
This JDBC application connects to a database server, issues SQL queries, processes multiple result sets, and then terminates.

Running the Sample Client-side JDBC Application [page 236]
Compile and execute a sample JDBC application to learn the steps required to create a working JDBC application.

Related Information

The jConnect JDBC Driver [page 226]
A Sample Server-side JDBC Application [page 238]

1.8.7.1 How the Client-side JDBC Application Works

This JDBC application connects to a database server, issues SQL queries, processes multiple result sets, and then terminates.

Importing Packages

The application requires a couple of packages, which are imported in the first lines of `JDBCConnect.java`:

- The java.io package contains the Java input/output classes, which are required for printing to the command prompt window.
- The java.sql package contains the JDBC classes, which are required for all JDBC applications.

Application Structure

Each Java application requires a class with a method named `main`, which is the method invoked when the program starts. In this simple example, JDBCConnect.main is the only public method in the application.

The JDBCConnect.main method carries out the following tasks:

1. Obtains a database user ID, password, JDBC driver name, and SQL query from the optional command-line arguments. Depending on which driver is selected, the SQL Anywhere JDBC driver or the jConnect 7.0 driver is loaded if they are in the class file path.
2. Connects to a running database using the selected JDBC driver URL. The getConnection method establishes a connection using the specified URL.
3. Creates a statement object, which is the container for the SQL statement.
4. Creates a result set object by executing a SQL query.
5. Iterates through the result set, printing the column information.
6. Checks for additional result sets and repeats the previous step if another result set is available.
7. Closes each of the result set, statement, and connection objects.

1.8.7.2 Running the Sample Client-side JDBC Application

Compile and execute a sample JDBC application to learn the steps required to create a working JDBC application.

Prerequisites

A Java Development Kit (JDK) must be installed.

Context

Two different types of connections using JDBC can be made. One is the client-side connection and the other is the server-side connection. The following example uses a client-side connection.

Procedure

1. At a command prompt, change to the *%SQLANYSAMP17%*`\SQLAnywhere\JDBC` directory.
2. Start a database server with the sample database on your local computer using the following command:

   ```
   dbsrv17 "%SQLANYSAMP17%\demo.db"
   ```
3. Set the CLASSPATH environment variable. The SQL Anywhere JDBC driver is contained in `sajdbc4.jar`.

   ```
   set classpath=.;%SQLANY17%\java\sajdbc4.jar
   ```

   If you are using the jConnect driver instead, then set the CLASSPATH as follows (where `jconnect-path` is your jConnect installation directory).

   ```
   set classpath=.;jconnect-path\classes\jconn4.jar
   ```
4. Run the following command to compile the example:

   ```
   javac JDBCConnect.java
   ```
5. Run the following command to execute the example:

   ```
   java JDBCConnect Browser browse
   ```

   A default SQL query is executed.

   If the attempt to connect fails, an error message appears instead. Confirm that you have executed all the steps as required. Check that your CLASSPATH is correct. An incorrect setting may result in a failure to locate a class.
6. Optional. Include command-line arguments to select a different user ID, password, and JDBC driver:

   ```
   java JDBCConnect Updater update jconnect
   ```
7. Optional. Include SQL queries on the command line:

   ```
   java JDBCConnect Browser browse jdbc4 "SELECT * FROM Customers"
   ```

Results

Result sets are displayed.

1.8.8 A Sample Server-side JDBC Application

A typical JDBC application connects to a database server, issues SQL queries, processes multiple results sets, and then terminates.

The following complete Java application connects to a running database using a server-side connection, issues a SQL query, processes and displays multiple results sets, and then terminates.

Establishing a connection from a server-side JDBC application is more straightforward than establishing an external connection. Because the user is already connected to the database, the application simply uses the current connection.

The source code for this example is a modified version of the `JDBCConnect.java` example and is located in *%SQLANYSAMP17%*`\SQLAnywhere\JDBC\JDBCConnect2.java`.

```
import java.io.*;
import java.sql.*;
public class JDBCConnect2
{
    public static void main( String args[] )
    {
        try
        {
            String SQL =
                  "BEGIN"
                + " SELECT * FROM Departments"
                + "     ORDER BY DepartmentID;"
                + " SELECT d.DepartmentID, GivenName, Surname, EmployeeID"
                + "     FROM Employees e"
                + "     JOIN Departments d"
                + "     ON DepartmentHeadID = EmployeeID"
                + "     ORDER BY d.DepartmentID;"
                + "END";
            if( args.length > 0 && args[0] != null && args[0].length() > 0 )
                SQL = args[0];
            Connection con = DriverManager.getConnection(
                "jdbc:default:connection" );
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(SQL);
            while( rs != null )
            {
                while (rs.next())
                {
                    for( int i = 1;
                         i <= rs.getMetaData().getColumnCount();
                         i++ )
                    {
                        if( i > 1 ) System.out.print(", ");
                        System.out.print(rs.getString(i));
                    }
                    System.out.println();
                }
                if( stmt.getMoreResults() )
                {
                    System.out.println();
                    rs = stmt.getResultSet();
                }
                else
                {
                    rs.close();
                    rs = null;
                }
            }
            stmt.close();
```

```
            con.close();
        }
        catch (SQLException sqe)
        {
            System.out.println("Unexpected exception : " +
                               sqe.toString() + ", sqlstate = " +
                               sqe.getSQLState());
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

In this section:

How the Server-side JDBC Application Works [page 239]
The server-side JDBC application is almost identical to the sample client-side JDBC application, with the following exceptions:

Running the Sample Server-side JDBC Application [page 239]
Compile and execute a sample JDBC application to learn the steps required to create a working server-side JDBC application.

1.8.8.1 How the Server-side JDBC Application Works

The server-side JDBC application is almost identical to the sample client-side JDBC application, with the following exceptions:

1. The user ID, password, and JDBC driver arguments are not required.
2. It connects to the default running database using the current connection. The URL in the getConnection call has been changed as follows:

   ```
   Connection con = DriverManager.getConnection(
       "jdbc:default:connection" );
   ```
3. Output is redirected to the server messages window.
4. The `System.exit()` statements have been removed.

1.8.8.2 Running the Sample Server-side JDBC Application

Compile and execute a sample JDBC application to learn the steps required to create a working server-side JDBC application.

Prerequisites

A Java Development Kit (JDK) must be installed.

You must have the following system privileges.

- MANAGE ANY EXTERNAL OBJECT to install Java classes
- CREATE PROCEDURE and CREATE EXTERNAL REFERENCE to create external procedures

Context

Two different types of connections using JDBC can be made. One is the client-side connection and the other is the server-side connection. The following example uses a server-side connection.

Procedure

1. At a command prompt, change to the *%SQLANYSAMP17%*`\SQLAnywhere\JDBC` directory.

   ```
   cd %SQLANYSAMP17%\SQLAnywhere\JDBC
   ```

2. For server-side JDBC, it is not necessary to set the CLASSPATH environment variable unless the server is started from a different current working directory.

   ```
   set classpath=.;%SQLANYSAMP17%\SQLAnywhere\JDBC
   ```

3. Start a database server with the sample database on your local computer using the following command:

   ```
   dbsrv17 "%SQLANYSAMP17%\demo.db"
   ```

4. Enter the following command to compile the example:

   ```
   javac JDBCConnect2.java
   ```

5. Install the class into the sample database using Interactive SQL. Execute the following statement (a path to the class file may be required):

   ```
   INSTALL JAVA NEW
   FROM FILE 'JDBCConnect2.class';
   ```

   You can also install the class using *SQL Central*. While connected to the sample database, open the *Java* subfolder under *External Environments* and click *File* > *New* > *Java Class*. Then follow the instructions in the wizard.

6. Define a stored procedure named JDBCConnect that acts as a wrapper for the JDBCConnect2.main method in the class:

   ```
   CREATE OR REPLACE PROCEDURE JDBCConnect(IN arg LONG VARCHAR)
     EXTERNAL NAME 'JDBCConnect2.main([Ljava/lang/String;)V'
     LANGUAGE JAVA;
   ```

7. Call the JDBCConnect2.main method as follows:

   ```
   CALL JDBCConnect('');
   ```

   The first time a Java class is called in a session, the Java VM must be loaded. This might take a few seconds.

8. Confirm that some result sets appear in the database server messages window.

   If the attempt to connect fails, an error message appears instead. Confirm that you have executed all the steps as required.
9. Try executing a different SQL query as follows:

   ```
   CALL JDBCConnect('SELECT * FROM Products;SELECT * FROM Departments');
   ```

   Some different result sets appear in the database server messages window.

Results

Result sets are displayed in the database server messages window.

1.8.9 Notes on JDBC Connections

Be aware that there are differences between JDBC on the client side and on the server side. Aspects such as autocommit behavior and isolation levels are described here.

Autocommit behavior

The JDBC specification requires that, by default, a COMMIT is performed after each data manipulation statement. Currently, the client-side JDBC behavior is to commit (autocommit is true) and the server-side behavior is to not commit (autocommit is false). To obtain the same behavior in both client-side and server-side applications, you can use a statement such as the following:

```
con.setAutoCommit( false );
```

In this statement, con is the current connection object. You could also set autocommit to true.

Setting transaction isolation level

To set the transaction isolation level, the application must call the Connection.setTransactionIsolation method with one of the following values.

For the SQL Anywhere JDBC driver use:

- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE
- sap.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
- sap.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_SNAPSHOT
- sap.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_READONLY_SNAPSHOT

The following example sets the transaction isolation level to SNAPSHOT using the SQL Anywhere JDBC driver.

```
try
```

```
{
    con.setTransactionIsolation(
        sap.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 );
}
```

Connection defaults

From server-side JDBC, only the first call to `getConnection( "jdbc:default:connection" )` creates a new connection with the default values. Subsequent calls return a wrapper of the current connection with all connection properties unchanged. If you set autocommit to false in your initial connection, any subsequent getConnection calls within the same Java code return a connection with autocommit set to false.

Ensure that closing a connection restores the connection properties to their default values, so that subsequent connections are obtained with standard JDBC values. The following code achieves this:

```
Connection con =
    DriverManager.getConnection("jdbc:default:connection");
boolean oldAutoCommit = con.getAutoCommit();
try
{
     // main body of code here
}
finally
{
    con.setAutoCommit( oldAutoCommit );
}
```

This discussion applies not only to autocommit, but also to other connection properties such as transaction isolation level and read-only mode.

Connection failure using the SQL Anywhere JDBC driver

If you do not deploy all the files required for the JDBC driver, you may encounter the error `Invalid ODBC handle` when attempting to connect to a database. Check that you have installed all the files required for the correct operation of the JDBC driver.

Related Information

JDBC Client Deployment [page 847]

1.8.10 Server-side Data Access Using JDBC

Database transaction logic implemented as Java methods that can be called from SQL can offer significant advantages over traditional SQL stored procedures.

The interface to a Java method is implemented using specialized SQL stored procedure definition syntax. Calls to Java methods, including those that use JDBC, closely parallel calls to SQL stored procedures that are

comprised entirely of SQL statements. Although the following topics demonstrate how to use JDBC from the database (server-side JDBC), the examples also demonstrate how to write JDBC for a client-side application.

As with other programming interfaces, SQL statements in JDBC can be either **static** or **dynamic**. Static SQL statements are constructed in the Java application and sent to the database. The database server parses the statement, selects an execution plan, and executes the statement.

If the same or similar SQL statement is executed many times (many inserts into one table, for example), there can be significant overhead when using static SQL because the statement preparation step has to be executed each time.

In contrast, a dynamic SQL statement contains place holders. The statement, prepared once using these place holders, can be executed many times without the additional expense of preparing it each time.

The following topics provide examples of both static and dynamic SQL statement execution as well as execution of batches (wide inserts, deletes, updates, and merges).

In this section:

Setting up the Server-side JDBC Example [page 244]
Compile and install a sample Java application into the database.

Inserts, Updates, and Deletes Using JDBC [page 245]
Static SQL statements such as INSERT, UPDATE, and DELETE, which do not return result sets, are executed using the executeUpdate method of the Statement class. Statements, such as CREATE TABLE and other data definition statements, can also be executed using executeUpdate.

Using Static INSERT and DELETE Statements from JDBC [page 246]
A sample JDBC application is called from the database server to insert and delete rows in the Departments table using static SQL statements.

How to Use Prepared Statements for More Efficient Access [page 247]
If you use the Statement interface, you parse each statement that you send to the database, generate an access plan, and execute the statement. The steps before execution are called **preparing** the statement.

Using Prepared INSERT and DELETE Statements from JDBC [page 248]
Call a sample JDBC application from the database server to insert and delete rows in the Departments table using prepared statements.

JDBC Batch Methods [page 250]
The addBatch method of the PreparedStatement class is used for performing batched (or wide) inserts, deletes, updates, and merges. The following are some guidelines to using this method.

How to Return Result Sets from Java [page 251]
You must write a Java method that returns one or more result sets to the calling environment, and wrap this method in a SQL stored procedure.

Returning Result Sets from JDBC [page 251]
Call a sample JDBC application from the database server to return several result sets.

Privilege Requirements When Using JDBC in the Database [page 252]
With the correct set of privileges, users can execute methods in Java classes.

1.8.10.1 Setting up the Server-side JDBC Example

Compile and install a sample Java application into the database.

Prerequisites

A Java Development Kit (JDK) must be installed.

To install a class, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Context

The source code for the JDBC example is located in *%SQLANYSAMP17%* `\SQLAnywhere\JDBC\JDBCExample.java`.

Procedure

1. At a command prompt, compile the `JDBCExample.java` source code.

   ```
   javac JDBCExample.java
   ```

2. Connect to the database from Interactive SQL.
3. Install the `JDBCExample.class` file into the sample database by executing the following statement in Interactive SQL:

   ```
   INSTALL JAVA NEW
   FROM FILE 'JDBCExample.class';
   ```

   If the database server was not started from the same directory as the class file and the path to the class file is not listed in the database server's class path, then you will have to include the path to the class file in the INSTALL statement. The database server's class path is defined by the -cp database server option and the java_class_path database option.

   You can also install the class using *SQL Central*. While connected to the sample database, open the *Java* subfolder under *External Environments* and click *File* > *New* > *Java Class*. Follow the instructions in the wizard.

Results

The JDBCExample class file is installed in the database and ready for demonstration. This class file is used in subsequent topics.

1.8.10.2 Inserts, Updates, and Deletes Using JDBC

Static SQL statements such as INSERT, UPDATE, and DELETE, which do not return result sets, are executed using the executeUpdate method of the Statement class. Statements, such as CREATE TABLE and other data definition statements, can also be executed using executeUpdate.

The addBatch, clearBatch, and executeBatch methods of the Statement class may also be used. Because the JDBC specification is unclear on the behavior of the executeBatch method of the Statement class, the following notes should be considered when using this method with the SQL Anywhere JDBC driver:

- Processing of the batch stops immediately upon encountering a SQL exception or result set. If processing of the batch stops, then a BatchUpdateException is thrown by the executeBatch method. Calling the getUpdateCounts method on the BatchUpdateException will return an integer array of row counts where the set of counts prior to the batch failure will contain a valid non-negative update count; while all counts at the point of the batch failure and beyond will contain a -1 value. Casting the BatchUpdateException to a SQLException will provide additional details as to why batch processing was stopped.
- The batch is only cleared when the clearBatch method is explicitly called. As a result, calling the executeBatch method repeatedly will re-execute the batch over and over again. In addition, calling execute(sql_query) or executeQuery(sql_query) will correctly execute the specified SQL query, but will not clear the underlying batch. Hence, calling the executeBatch method followed by execute(sql_query) followed by the executeBatch method again will execute the set of batched statements, then execute the specified SQL query, and then execute the set of batched statements again.

The following code fragment illustrates how to execute an INSERT statement. It uses a Statement object that has been passed to the InsertStatic method as an argument.

```
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();
  }
}
```

Notes

- This code fragment is part of the `JDBCExample.java` file included in the *%SQLANYSAMP17%* `\SQLAnywhere\JDBC` directory.
- The executeUpdate method returns an integer that reflects the number of rows affected by the operation. In this case, a successful INSERT would return a value of one (1).

- When run as a server-side class, the output from `System.out.println` goes to the database server messages window.

1.8.10.3 Using Static INSERT and DELETE Statements from JDBC

A sample JDBC application is called from the database server to insert and delete rows in the Departments table using static SQL statements.

Prerequisites

To create an external procedure, you must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges. You must also have SELECT, DELETE, and INSERT privileges on the database object you are modifying.

Procedure

1. Connect to the sample database from Interactive SQL.
2. Ensure the JDBCExample class has been installed.
3. Define a stored procedure named JDBCExample that acts as a wrapper for the JDBCExample.main method in the class:

   ```
   CREATE PROCEDURE JDBCExample( IN arg LONG VARCHAR )
     EXTERNAL NAME 'JDBCExample.main([Ljava/lang/String;)V'
     LANGUAGE JAVA;
   ```
4. Call the JDBCExample.main method as follows:

   ```
   CALL JDBCExample( 'insert' );
   ```

   The argument string `'insert'` causes the InsertStatic method to be invoked.
5. Confirm that a row has been added to the Departments table.

   ```
   SELECT * FROM Departments;
   ```

   The example program displays the updated contents of the Departments table in the database server messages window.
6. There is a similar method in the example class called DeleteStatic that shows how to delete the row that has just been added. Call the JDBCExample.main method as follows:

   ```
   CALL JDBCExample( 'delete' );
   ```

   The argument string `'delete'` causes the DeleteStatic method to be invoked.

7. Confirm that the row has been deleted from the Departments table.

```
SELECT * FROM Departments;
```

The updated contents of the Departments table are displayed.

Results

Rows are inserted and deleted from a table using static SQL statements in a server-side JDBC application. The updated contents of the Departments table are displayed in the database server messages window.

Related Information

Setting up the Server-side JDBC Example [page 244]

1.8.10.4 How to Use Prepared Statements for More Efficient Access

If you use the Statement interface, you parse each statement that you send to the database, generate an access plan, and execute the statement. The steps before execution are called **preparing** the statement.

You can achieve performance benefits if you use the PreparedStatement interface. This allows you to prepare a statement using placeholders, and then assign values to the placeholders when executing the statement.

Using prepared statements is particularly useful when carrying out many similar actions, such as inserting many rows.

Example

The following example illustrates how to use the PreparedStatement interface, although inserting a single row is not a good use of prepared statements.

The following InsertDynamic method of the JDBCExample class carries out a prepared statement:

```
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();
  }
}
```

Notes

- This code fragment is part of the `JDBCExample.java` file included in the *%SQLANYSAMP17%* `\SQLAnywhere\JDBC` directory.
- The executeUpdate method returns an integer that reflects the number of rows affected by the operation. In this case, a successful INSERT would return a value of one (1).
- When run as a server-side class, the output from `System.out.println` goes to the database server messages window.

1.8.10.5 Using Prepared INSERT and DELETE Statements from JDBC

Call a sample JDBC application from the database server to insert and delete rows in the Departments table using prepared statements.

Prerequisites

To create an external procedure, you must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges. You must also have SELECT, DELETE, and INSERT privileges on the database object you are modifying.

Procedure

1. Connect to the sample database from Interactive SQL.
2. Ensure the JDBCExample class has been installed.
3. Define a stored procedure named JDBCInsert that acts as a wrapper for the JDBCExample.Insert method in the class:

   ```
   CREATE PROCEDURE JDBCInsert( IN arg1 INTEGER, IN arg2 LONG VARCHAR )
     EXTERNAL NAME 'JDBCExample.Insert(ILjava/lang/String;)V'
     LANGUAGE JAVA;
   ```

4. Call the JDBCExample.Insert method as follows:

   ```
   CALL JDBCInsert( 202, 'Southeastern Sales' );
   ```

   The Insert method causes the InsertDynamic method to be invoked.
5. Confirm that a row has been added to the Departments table.

   ```
   SELECT * FROM Departments;
   ```

   The example program displays the updated contents of the Departments table in the database server messages window.
6. There is a similar method in the example class called DeleteDynamic that shows how to delete the row that has just been added.

   Define a stored procedure named JDBCDelete that acts as a wrapper for the JDBCExample.Delete method in the class:

   ```
   CREATE PROCEDURE JDBCDelete( IN arg1 INTEGER )
     EXTERNAL NAME 'JDBCExample.Delete(I)V'
     LANGUAGE JAVA;
   ```

7. Call the JDBCExample.Delete method as follows:

   ```
   CALL JDBCDelete( 202 );
   ```

   The Delete method causes the DeleteDynamic method to be invoked.
8. Confirm that the row has been deleted from the Departments table.

   ```
   SELECT * FROM Departments;
   ```

   The updated contents of the Departments table are displayed.

Results

Rows are inserted and deleted from a table using prepared SQL statements in a server-side JDBC application. The updated contents of the Departments table are displayed in the database server messages window.

Related Information

Prepared Statements [page 13]
Setting up the Server-side JDBC Example [page 244]

1.8.10.6 JDBC Batch Methods

The addBatch method of the PreparedStatement class is used for performing batched (or wide) inserts, deletes, updates, and merges. The following are some guidelines to using this method.

1. A SQL statement should be prepared using one of the prepareStatement methods of the Connection class.

   ```
   String sqlStr = "INSERT INTO Departments( DepartmentID, DepartmentName )
   VALUES ( ? , ? )";
   PreparedStatement pstmt = con.prepareStatement( sqlStr );
   ```

2. The parameters for the prepared statement should be set and then added as a batch. The following outline creates `n` batches with `m` parameters in each batch:

   ```
   for( i=0; i < n; i++ )
   {
       pstmt.set...( 1, param_1 );
       pstmt.set...( 2, param_2 );
       .
       .
       .
       pstmt.set...( m , param_m );
       pstmt.addBatch();
   }
   ```

   The following example creates 5 batches with 2 parameters in each batch. The first parameter in an integer and the second parameter is a string.:

   ```
   for( i=0; i < 5; i++ )
   {
       pstmt.setInt( 1, idValue[i] );
       pstmt.setString( 2, name[i] );
       pstmt.addBatch();
   }
   ```

3. The batch must be executed using the executeBatch method of the PreparedStatement class.

   ```
   int[] affected = pstmt.executeBatch();
   ```

BLOB parameters are not supported in batches.

When using the SQL Anywhere JDBC driver to perform batched inserts, use a small column size. Using batched inserts to insert large binary or character data into long binary or long varchar columns is not recommended and may degrade performance. The performance can decrease because the JDBC driver must allocate large amounts of memory to hold each of the batched insert rows.

In all other cases, batched inserts, deletes, updates, and merges should provide better performance than using individual operations.

1.8.10.7 How to Return Result Sets from Java

You must write a Java method that returns one or more result sets to the calling environment, and wrap this method in a SQL stored procedure.

The following code fragment illustrates how multiple result sets can be returned to the caller of this Java procedure. It uses three executeQuery statements to obtain three different result sets.

```
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();
}
```

Notes

- This server-side JDBC example is part of the `JDBCExample.java` file included in the *%SQLANYSAMP17%* `\SQLAnywhere\JDBC` directory.
- It obtains a connection to the default running database by using getConnection.
- The executeQuery methods return result sets.

1.8.10.8 Returning Result Sets from JDBC

Call a sample JDBC application from the database server to return several result sets.

Prerequisites

To create an external procedure, you must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges. You must also have SELECT, DELETE, and INSERT privileges on the database object you are modifying.

Procedure

1. Connect to the sample database from Interactive SQL.
2. Ensure the JDBCExample class has been installed.
3. Define a stored procedure named JDBCResults that acts as a wrapper for the JDBCExample.Results method in the class.

   For example:

   ```
   CREATE PROCEDURE JDBCResults(OUT args LONG VARCHAR)
     DYNAMIC RESULT SETS 3
     EXTERNAL NAME 'JDBCExample.Results([Ljava/sql/ResultSet;)V'
     LANGUAGE JAVA;
   ```

   The example returns 3 result sets.
4. Set the following Interactive SQL options so you can see all the results of the query:
   a. Click *Tools* > *Options*.
   b. Click .
   c. Click the *Results* tab.
   d. Set the value for *Maximum Number Of Rows To Display* to **5000**.
   e. Click *OK*.
5. Call the JDBCExample.Results method.

   ```
   CALL JDBCResults();
   ```
6. Check each of the three results tabs, *Result Set 1*, *Result Set 2*, and *Result Set 3*.

Results

Three different result sets are returned from a server-side JDBC application.

Related Information

Setting up the Server-side JDBC Example [page 244]

1.8.10.9 Privilege Requirements When Using JDBC in the Database

With the correct set of privileges, users can execute methods in Java classes.

Access privileges

Like all Java classes in the database, classes containing JDBC statements can be accessed by any user if the GRANT EXECUTE statement has granted them privilege to execute the stored procedure that is acting as a wrapper for the Java method.

Execution privileges

Java classes execute with the privileges of the stored procedure that is acting as a wrapper for the Java method (by default, this is SQL SECURITY DEFINER).

1.8.11 JDBC Callbacks

The SQL Anywhere JDBC driver supports two asynchronous callbacks, one for handling the SQL MESSAGE statement and the other for validating requests for file transfers. These are similar to the callbacks supported by the ODBC driver.

Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

A message handler routine can be created to intercept these messages. The following is an example of a message handler callback routine.

```
class T_message_handler implements sap.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;
    }
}
```

A client file transfer request can be validated. Before allowing any transfer to take place, the JDBC driver will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the JDBC driver will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below. The following is an example of a file transfer validation callback routine.

```
class T_filetrans_callback implements
sap.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
    }
}
```

The filename argument is the name of the file to be read or written. The is_write parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the JDBC driver allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. If the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

The following sample Java application demonstrates the use of the callbacks supported by the SQL Anywhere JDBC driver. Place the file *%SQLANY17%*`\java\sajdbc4.jar` in your classpath.

```
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();

((sap.jdbc4.sqlanywhere.IConnection)con).setASAMessageHandler( message_worker );

            // create and register validate file transfer callback
            T_filetrans_callback filetran_worker = new T_filetrans_callback();

((sap.jdbc4.sqlanywhere.IConnection)con).setSAValidateFileTransferCallback( filet
ran_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=password" );
        }
        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();
        }
    }
}
```

Related Information

MESSAGE Statement

1.8.12 JDBC Escape Syntax

You can use JDBC escape syntax from any JDBC application, including Interactive SQL. This escape syntax allows you to call stored procedures regardless of the database management system you are using.

The general form for the escape syntax is:

```
{ keyword parameters }
```

The set of keywords includes the following:

{d date-string}

The date string is any date value accepted by the database server.

{t time-string}

The time string is any time value accepted by the database server.

{ts date-string time-string}

The date/time string is any timestamp value accepted by the database server.

{guid uuid-string}

The uuid-string is any valid GUID string, for example, 41dfe9ef-db91-11d2-8c43-006008d26a6f.

{oj outer-join-expr}

The outer-join-expr is a valid OUTER JOIN expression accepted by the database server.

{? = call func(p1, ...)}

The function is any valid function call accepted by the database server.

{call proc(p1, ...)}

The procedure is any valid stored procedure call accepted by the database server.

{fn func(p1, ...)}

The function is any one of the library of functions listed below.

You can use the escape syntax to access a library of functions implemented by the JDBC driver that includes number, string, time, date, and system functions.

For example, to obtain the current date in a database management system-neutral way, you would execute the following:

```
SELECT { FN CURDATE() }
```

The functions that are available depend on the JDBC driver that you are using. The following tables list the functions that are supported by the SQL Anywhere JDBC driver and by the jConnect driver.

SQL Anywhere JDBC Driver Supported Functions

Numeric Functions	String Functions	System Functions	Time/Date Functions
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH		CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
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		

Numeric Functions	String Functions	System Functions	Time/Date Functions
TRUNCATE			

jConnect Supported Functions

Numeric Functions	String Functions	System Functions	Time/Date Functions
ABS	ASCII	DATABASE	CURDATE
ACOS	CHAR	IFNULL	CURTIME
ASIN	CONCAT	USER	DAYNAME
ATAN	DIFFERENCE	CONVERT	DAYOFMONTH
ATAN2	LCASE		DAYOFWEEK
CEILING	LENGTH		HOUR
COS	REPEAT		MINUTE
COT	RIGHT		MONTH
DEGREES	SOUNDEX		MONTHNAME
EXP	SPACE		NOW
FLOOR	SUBSTRING		QUARTER
LOG	UCASE		SECOND
LOG10			TIMESTAMPADD
PI			TIMESTAMPDIFF
POWER			YEAR
RADIANS			
RAND			
ROUND			
SIGN			
SIN			
SQRT			
TAN			

JDBC TIMESTAMPADD, TIMESTAMPDIFF

The JDBC driver maps the TIMESTAMPADD and TIMESTAMPDIFF functions to the corresponding DATEADD and DATEDIFF functions. The syntax for the TIMESTAMPADD and TIMESTAMPDIFF functions is as follows.

```
{ fn TIMESTAMPADD( interval, integer-expr, timestamp-expr ) }
```

Returns the timestamp calculated by adding `integer-expr` intervals of type `interval` to `timestamp-expr`. Valid values of `interval` are shown below.

```
{ fn TIMESTAMPDIFF( interval, timestamp-expr1, timestamp-expr2 ) }
```

Returns the integer number of intervals of type `interval` by which `timestamp-expr2` is greater than `timestamp-expr1`. Valid values of `interval` are shown below.

Interval	DATEADD/DATEDIFF Date-Part Mapping
SQL_TSI_YEAR	YEAR
SQL_TSI_QUARTER	QUARTER
SQL_TSI_MONTH	MONTH
SQL_TSI_WEEK	WEEK
SQL_TSI_DAY	DAY
SQL_TSI_HOUR	HOUR
SQL_TSI_MINUTE	MINUTE
SQL_TSI_SECOND	SECOND
SQL_TSI_FRAC_SECOND	MICROSECOND - The DATEADD and DATEDIFF functions do not support a resolution of nanoseconds.

Interactive SQL

In Interactive SQL, the braces *must* be doubled. There must not be a space between successive braces: "{{" is acceptable, but "{ {" is not. As well, you cannot use newline characters in the statement. The escape syntax cannot be used in stored procedures because they are not parsed by Interactive SQL.

For example, to obtain the number of weeks in February 2013, execute the following in Interactive SQL:

```
SELECT {{ fn TIMESTAMPDIFF(SQL_TSI_WEEK, '2013-02-01T00:00:00',
'2013-03-01T00:00:00'  ) }}
```

1.8.13 SQL Anywhere JDBC API Support

Some optional methods of the java.sql.Blob interface are not supported by the SQL Anywhere JDBC driver.

These optional methods are:

- 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);

1.9 Node.js Application Programming

The Node.js API can be used to connect to SQL Anywhere databases, issue SQL queries, and obtain result sets.

The Node.js driver allows users to connect and perform queries on the database using JavaScript on Joyent's Node.js software platform. Drivers are available for various versions of Node.js.

The API interface is very similar to the SAP HANA Node.js Client, and allows users to connect, disconnect, execute, and prepare statements.

The driver is available for install through the NPM (Node Packaged Modules) web site: https://npmjs.org/ .

It can also be downloaded from https://github.com/sqlanywhere/node-sqlanywhere .

The SQL Anywhere Node.js API reference is available in the *SQL Anywhere- Node.js API Reference* at https://help.sap.com/viewer/09fbca22f0344633b8951c3e9d624d28/LATEST/en-US.

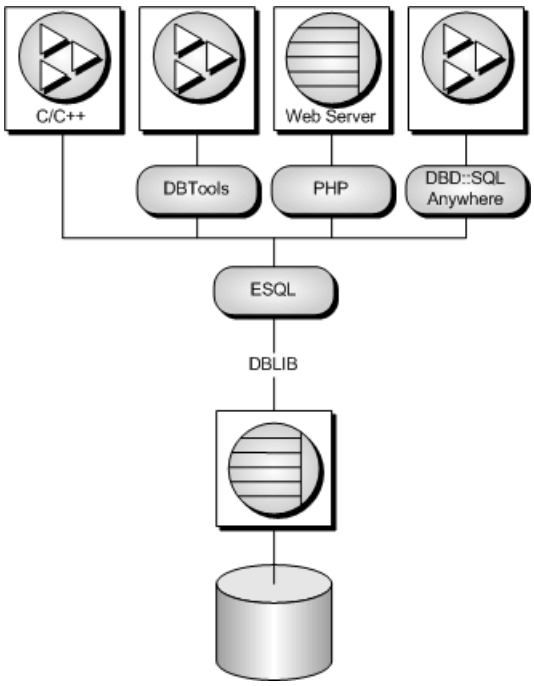
1.10 Embedded SQL

SQL statements embedded in a C or C++ source file are referred to as Embedded SQL. A preprocessor translates these statements into calls to a runtime library. Embedded SQL is an ISO/ANSI and IBM standard.

Embedded SQL is portable to other databases and other environments, and is functionally equivalent in all operating environments. It is a comprehensive, low-level interface that provides all the functionality available in the product. Embedded SQL requires knowledge of C or C++ programming languages.

Embedded SQL Applications

You can develop C or C++ applications that access the database server using the Embedded SQL interface. The command line database tools are examples of applications developed in this manner.



Embedded SQL is a database programming interface for the C and C++ programming languages. It consists of SQL statements intermixed with (embedded in) C or C++ source code. These SQL statements are translated by a **Embedded SQL preprocessor** into C or C++ source code, which you then compile.

At runtime, Embedded SQL applications use an **interface library** called DBLIB to communicate with a database server. DBLIB is a dynamic link library (**DLL**) or shared object on most platforms.

- On Windows operating systems, the interface library is `dblib17.dll`.
- On UNIX and Linux operating systems, the interface library is `libdblib17.so`, `libdblib17.sl`, or `libdblib17.a`, depending on the operating system.
- On macOS, the interface library is `libdblib17.dylib.1`.

Two flavors of Embedded SQL are provided. Static Embedded SQL is simpler to use, but is less flexible than dynamic Embedded SQL.

In this section:

Development Process Overview [page 263]
Once the program has been successfully preprocessed and compiled, it can be linked with an **import library** for DBLIB to form an executable file. When the database server is running, this executable file uses DBLIB to interact with the database server.

The Embedded SQL Preprocessor [page 264]
The Embedded SQL preprocessor is an executable named `sqlpp`.

Supported Compilers [page 268]
Both Windows and UNIX/Linux compilers have been used with the Embedded SQL preprocessor.

Embedded SQL Header Files [page 269]
All header files are installed in the `SDK\Include` subdirectory of your software installation directory.

Import Libraries [page 269]

Import libraries are installed in subdirectories of your software installation directory.

Embedded SQL Program Example [page 270]
A very simple example of an Embedded SQL program is presented here.

Structure of Embedded SQL Programs [page 271]
SQL statements are placed (embedded) within regular C or C++ code. All Embedded SQL statements start with the words EXEC SQL and end with a semicolon (;).

Loading DBLIB Dynamically under Windows [page 271]
Load DBLIB dynamically from your Embedded SQL application using the `esqldll.c` module in the `SDK\C` subdirectory of your software installation directory so that you do not need to link against the import library.

Sample Embedded SQL Programs [page 272]
Sample Embedded SQL programs are included with the software.

Embedded SQL Data Types [page 277]
To transfer information between a program and the database server, every piece of data must have a data type.

Host Variables in Embedded SQL [page 281]
Host variables are C variables that are identified to the Embedded SQL preprocessor. Host variables can be used to send values to the database server or receive values from the database server.

The SQL Communication Area (SQLCA) [page 290]
The **SQL Communication Area** (**SQLCA**) is an area of memory that is used on every database request for communicating statistics and errors from the application to the database server and back to the application.

Static and Dynamic SQL [page 296]
There are two ways to embed SQL statements into a C program: statically or dynamically.

The SQL Descriptor Area (SQLDA) [page 300]
The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file `sqlda.h`.

How to Fetch Data Using Embedded SQL [page 311]
Fetching data in Embedded SQL is done using the SELECT statement.

Wide Inserts Using Embedded SQL [page 318]
The INSERT statement can be used to insert more than one row at a time, which may improve performance. This is called a **wide insert** or an **array insert**.

Wide Deletes Using Embedded SQL [page 322]
The DELETE statement can be used to delete an arbitrary set of rows, which may improve performance. This is called a **wide delete** or an **array delete**.

Wide Merges Using Embedded SQL [page 325]
The MERGE statement can be used to merge multiple sets of rows into a table, which may improve performance. This is called a **wide merge** or an **array merge**.

How to Send and Retrieve Long Values Using Embedded SQL [page 329]
The method for sending and retrieving LONG VARCHAR, LONG NVARCHAR, and LONG BINARY values in Embedded SQL applications is different from that for other data types.

Simple Stored Procedures in Embedded SQL [page 336]
You can create and call stored procedures using Embedded SQL.

Request Management with Embedded SQL [page 339]
Since a typical Embedded SQL application must wait for the completion of each database request before carrying out the next step, an application that uses multiple execution threads can carry on with other tasks.

Database Backup with Embedded SQL [page 339]
The recommended way to backup a database is to use the BACKUP statement.

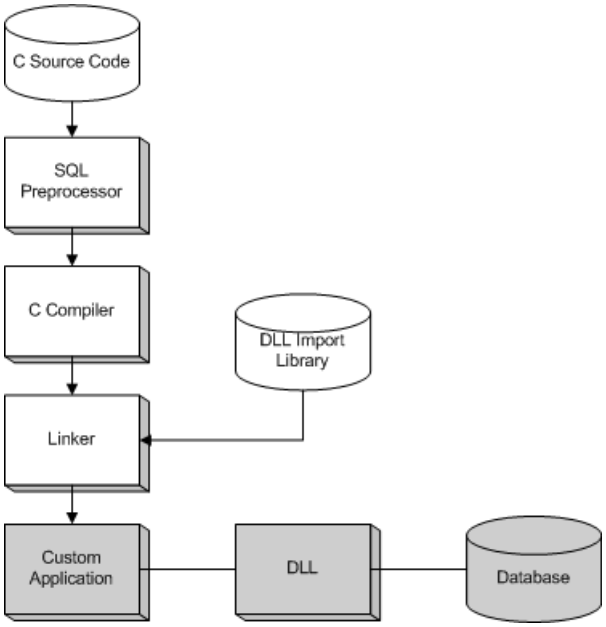
Library Function Reference [page 340]
The Embedded SQL preprocessor generates calls to functions in the interface library or DLL. In addition to the calls generated by the Embedded SQL preprocessor, a set of library functions is provided to make database operations easier to perform.

Embedded SQL Statement Summary [page 386]
All Embedded SQL statements must be preceded with EXEC SQL and end with a semicolon (;).

1.10.1 Development Process Overview

Once the program has been successfully preprocessed and compiled, it can be linked with an **import library** for DBLIB to form an executable file. When the database server is running, this executable file uses DBLIB to interact with the database server.

The database server does not have to be running when the program is preprocessed.



For Windows, there are 32-bit and 64-bit import libraries for Microsoft Visual C++. The use of import libraries is one method for developing applications that call functions in DLLs. However, it is better to avoid the use of import libraries by dynamically loading the library.

Related Information

Loading DBLIB Dynamically under Windows [page 271]

1.10.2 The Embedded SQL Preprocessor

The Embedded SQL preprocessor is an executable named `sqlpp`.

The preprocessor command line is as follows:

```
sqlpp [ options ] sql-filename [ output-filename ]
```

The preprocessor translates the Embedded SQL statements in a C or C++ source file into C code and places the result in an output file. A C or C++ compiler is then used to process the output file. The normal extension for source programs with Embedded SQL is `.sqc`. The default output file name is the `sql-filename` with an extension of `.c`. If the `sql-filename` has a `.c` extension, then the default output file name extension will change to `.cc`.

> **Note**
>
> When an application is rebuilt to use a new major version of the database interface library, the Embedded SQL files must be preprocessed with the same version's Embedded SQL preprocessor.

The following table describes the preprocessor options.

Option	Description
-d	Generate code that reduces data space size. Data structures are reused and initialized at execution time before use. This increases code size.

Option	Description
-e `level`	Flag as an error any static Embedded SQL that is not part of a specified standard. The `level` value indicates the standard to use. For example, `sqlpp -e c03 ...` flags any syntax that is not part of the core ANSI/ISO SQL Standard. The supported `level` values are: **c08** Flag syntax that is not core SQL/2008 syntax **p08** Flag syntax that is not full SQL/2008 syntax **c03** Flag syntax that is not core SQL/2003 syntax **p03** Flag syntax that is not full SQL/2003 syntax **c99** Flag syntax that is not core SQL/1999 syntax **p99** Flag syntax that is not full SQL/1999 syntax **e92** Flag syntax that is not entry-level SQL/1992 syntax **i92** Flag syntax that is not intermediate-level SQL/1992 syntax **f92** Flag syntax that is not full-SQL/1992 syntax **t** Flag non-standard host variable types **u** Flag syntax that is not supported by UltraLite For compatibility with previous software versions, you can also specify e, i, and f, which correspond to e92, i92, and f92, respectively.
-h `width`	Limit the maximum length of lines output by sqlpp to `width`. The continuation character is a backslash (\) and the minimum value of `width` is 10.
-k	Notify the preprocessor that the program to be compiled includes a user declaration of SQLCODE. The definition must be of type long, but does not need to be in a declaration section.

Option	Description
-m `mode`	Specify the cursor updatability mode if it is not specified explicitly in the Embedded SQL application. The `mode` can be one of: **HISTORICAL** In previous versions, Embedded SQL cursors defaulted to either FOR UPDATE or READ ONLY (depending on the query and the ansi_update_constraints option value). Explicit cursor updatability was specified on DECLARE CURSOR. Use this option to preserve this behavior. **READONLY** Embedded SQL cursors default to READ ONLY. Explicit cursor updatability is specified on PREPARE. This is the default behavior. READ ONLY cursors can result in improved performance.
-n	Generate line number information in the C file. This consists of `#line` directives in the appropriate places in the generated C code. If the compiler that you are using supports the `#line` directive, this option makes the compiler report errors on line numbers in the SQC file (the one with the Embedded SQL) as opposed to reporting errors on line numbers in the C file generated by the Embedded SQL preprocessor. Also, the `#line` directives are used indirectly by the source level debugger so that you can debug while viewing the SQC source file.
-o `operating-system`	Specify the target operating system. The supported operating systems are: **WINDOWS** Microsoft Windows. **UNIX** Use this option if you are creating a 32-bit Linux application. **UNIX64** Use this option if you are creating a 64-bit Linux application.
-q	Quiet mode. Do not print messages.
-r-	Generate non-reentrant code.
-s `len`	Set the maximum size string that the preprocessor puts into the C file. Strings longer than this value are initialized using a list of characters ('a', 'b', 'c', and so on). Most C compilers have a limit on the size of string literal they can handle. This option is used to set that upper limit. The default value is 500.

Option	Description
-u	Generate code for UltraLite.
-w `level`	Flag as a warning any static Embedded SQL that is not part of a specified standard. The `level` value indicates the standard to use. For example, `sqlpp -w c08 ...` flags any SQL syntax that is not part of the core SQL/2008 syntax. The supported `level` values are: **c08** Flag syntax that is not core SQL/2008 syntax **p08** Flag syntax that is not full SQL/2008 syntax **c03** Flag syntax that is not core SQL/2003 syntax **p03** Flag syntax that is not full SQL/2003 syntax **c99** Flag syntax that is not core SQL/1999 syntax **p99** Flag syntax that is not full SQL/1999 syntax **e92** Flag syntax that is not entry-level SQL/1992 syntax **i92** Flag syntax that is not intermediate-level SQL/1992 syntax **f92** Flag syntax that is not full-SQL/1992 syntax **t** Flag non-standard host variable types **u** Flag syntax that is not supported by UltraLite For compatibility with previous software versions, you can also specify e, i, and f, which correspond to e92, i92, and f92, respectively.
-x	Change multibyte strings to escape sequences so that they can pass through compilers.

Option	Description
`-z cs`	Specify the collation sequence. For a list of recommended collation sequences, run `dbinit -l` at a command prompt. The collation sequence is used to help the preprocessor understand the characters used in the source code of the program, for example, in identifying alphabetic characters suitable for use in identifiers. If -z is not specified, the pre-processor attempts to determine a reasonable collation to use based on the operating system and the SALANG and SACHARSET environment variables.
`sql-filename`	A C or C++ program containing Embedded SQL to be processed.
`output-filename`	The C language source file created by the Embedded SQL preprocessor.

Related Information

SQL Preprocessor Error Messages
SQLCA Management for Multithreaded or Reentrant Code [page 293]
sql_flagger_error_level Option
sql_flagger_warning_level Option
SQLFLAGGER Function [Miscellaneous]
sa_ansi_standard_packages System Procedure
UltraLite Embedded SQL API Reference
SACHARSET Environment Variable
SALANG Environment Variable

1.10.3 Supported Compilers

Both Windows and UNIX/Linux compilers have been used with the Embedded SQL preprocessor.

The following compilers have been used:

Operating System	Compiler	Version
Windows	Microsoft Visual C++	6.0 or later
UNIX/Linux	GNU or native compiler	

1.10.4 Embedded SQL Header Files

All header files are installed in the `SDK\Include` subdirectory of your software installation directory.

File Name	Description
`sqlca.h`	Main header file included in all Embedded SQL programs. This file includes the structure definition for the SQL Communication Area (SQLCA) and prototypes for all Embedded SQL database interface functions.
`sqlda.h`	SQL Descriptor Area structure definition included in Embedded SQL programs that use dynamic SQL.
`sqldef.h`	Definition of Embedded SQL interface data types. This file also contains structure definitions and return codes needed for starting the database server from a C program.
`sqlerr.h`	Definitions for error codes returned in the sqlcode field of the SQLCA.
`sqlstate.h`	Definitions for ANSI/ISO SQL standard error states returned in the sqlstate field of the SQLCA.
`pshpk1.h, pshpk4.h, poppk.h`	These headers ensure that structure packing is handled correctly.

1.10.5 Import Libraries

Import libraries are installed in subdirectories of your software installation directory.

On Windows platforms, all import libraries are installed in the `SDK\Lib` subdirectory of your software installation directory. Windows import libraries are stored in the `SDK\Lib\x86` and `SDK\Lib\x64` subdirectories. An export definition list is stored in `SDK\Lib\Def\dblib.def`.

On UNIX and Linux platforms, all import libraries are installed in the `lib32` and `lib64` subdirectories, under the software installation directory.

On macOS platforms, all import libraries are installed in the `System/lib32` and `System/lib64` subdirectories, under the software installation directory.

Operating System	Compiler	Import Library
Windows	Microsoft Visual C++	`dblibtm.lib`
UNIX/Linux (unthreaded applications)	All compilers	`libdblib`*17*`.so`, `libdbtasks`*17*`.so`, `libdblib`*17*`.sl`, `libdbtasks`*17*`.sl`
UNIX/Linux (threaded applications)	All compilers	`libdblib`*17*`_r.so`, `libdbtasks`*17*`_r.so`,`libdblib`*17*`_r.sl`, `libdbtasks`*17*`_r.sl`

Operating System	Compiler	Import Library
macOS (threaded applications)	All compilers	`libdblib`*17*`.dylib`, `libdbtasks`*17*`.dylib`
macOS (threaded applications)	All compilers	`libdblib`*17*`_r.dylib`, `libdbtasks`*17*`_r.dylib`

The `libdbtasks`*17* libraries are called by the `libdblib`*17* libraries. Some compilers locate `libdbtasks`*17* automatically. For others, you must specify it explicitly.

1.10.6 Embedded SQL Program Example

A very simple example of an Embedded SQL program is presented here.

```
#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 );
}
```

This example connects to the database, updates the last name of employee number 195, commits the change, and exits. There is virtually no interaction between the Embedded SQL code and the C code. The only thing the C code is used for in this example is control flow. The WHENEVER statement is used for error checking. The error action (GOTO in this example) is executed after any SQL statement that causes an error.

Related Information

How to Fetch Data Using Embedded SQL [page 311]

1.10.7 Structure of Embedded SQL Programs

SQL statements are placed (embedded) within regular C or C++ code. All Embedded SQL statements start with the words EXEC SQL and end with a semicolon (;).

Normal C language comments are allowed in the middle of Embedded SQL statements.

Every C program using Embedded SQL must contain the following statement before any other Embedded SQL statements in the source file.

```
EXEC SQL INCLUDE SQLCA;
```

Every C program using Embedded SQL must initialize a SQLCA first:

```
db_init( &sqlca );
```

One of the first Embedded SQL statements executed by the C program must be a CONNECT statement. The CONNECT statement is used to establish a connection with the database server and to specify the user ID that is used for authorizing all statements executed during the connection.

Some Embedded SQL statements do not generate any C code, or do not involve communication with the database. These statements are allowed before the CONNECT statement. Most notable are the INCLUDE statement and the WHENEVER statement for specifying error processing.

Every C program using Embedded SQL must finalize any SQLCA that has been initialized.

```
db_fini( &sqlca );
```

1.10.8 Loading DBLIB Dynamically under Windows

Load DBLIB dynamically from your Embedded SQL application using the `esqldll.c` module in the `SDK\C` subdirectory of your software installation directory so that you do not need to link against the import library.

Context

This task is an alternative to the usual technique of linking an application against a static import library for a Dynamic Link Library (DLL) that contains the required function definitions.

A similar task can be used to dynamically load DBLIB on UNIX and Linux platforms.

Procedure

1. Your application must call db_init_dll to load the DBLIB DLL, and must call db_fini_dll to free the DBLIB DLL. The db_init_dll call must be before any function in the database interface, and no function in the interface can be called after db_fini_dll.

You must still call the db_init and db_fini library functions.

2. You must include the `esqldll.h` header file before the EXEC SQL INCLUDE SQLCA statement or include `sqlca.h` in your Embedded SQL program. The `esqldll.h` header file includes `sqlca.h`.
3. A SQL OS macro must be defined. The header file `sqlos.h`, which is included by `sqlca.h`, attempts to determine the appropriate macro and define it. However, certain combinations of platforms and compilers may cause this to fail. In this case, you must add a #define to the top of this file, or make the definition using a compiler option. Define the _SQL_OS_WINDOWS for all Windows operating systems.
4. Compile `esqldll.c`.
5. Instead of linking against the import library, link the object module `esqldll.obj` with your Embedded SQL application objects.

Results

The DBLIB interface DLL loads dynamically when you run your Embedded SQL application.

Example

You can find a sample program illustrating how to load the interface library dynamically in the *%SQLANYSAMP17%*`\SQLAnywhere\ESQLDynamicLoad` directory. The source code is in `sample.sqc`.

The following example compiles and links `sample.sqc` with the code from `esqldll.c` on Windows.

```
sqlpp sample.sqc
cl sample.c %SQLANY17%\sdk\c\esqldll.c /I%SQLANY17%\sdk\include Advapi32.lib
```

1.10.9 Sample Embedded SQL Programs

Sample Embedded SQL programs are included with the software.

They are placed in the *%SQLANYSAMP17%*`\SQLAnywhere\C` directory.

- The static cursor Embedded SQL example, `cur.sqc`, demonstrates the use of static SQL statements.
- The dynamic cursor Embedded SQL example, `dcur.sqc`, demonstrates the use of dynamic SQL statements.

To reduce the amount of code that is duplicated by the sample programs, the mainlines and the data printing functions have been placed into a separate file. This is `mainch.c` for character mode systems and `mainwin.c` for windowing environments.

The sample programs each supply the following three routines, which are called from the mainlines:

WSQLEX_Init

Connects to the database and opens the cursor.

WSQLEX_Process_Command

Processes commands from the user, manipulating the cursor as necessary.

WSQLEX_Finish

Closes the cursor and disconnects from the database.

The function of the mainline is to:

1. Call the WSQLEX_Init routine.
2. Loop, getting commands from the user and calling WSQL_Process_Command until the user quits.
3. Call the WSQLEX_Finish routine.

Connecting to the database is done with the Embedded SQL CONNECT statement supplying the appropriate user ID and password.

In addition to these samples, you may find other programs and source files included with the software that demonstrate features available for particular platforms.

In this section:

Static Cursor Sample [page 273]
This example demonstrates the use of static cursors.

Running the Static Cursor Sample Program [page 274]
Run the static cursor sample program.

Dynamic Cursor Sample [page 275]
This sample demonstrates the use of cursors for a dynamic SQL SELECT statement.

Running the Dynamic Cursor Sample Program [page 276]
Run the dynamic cursor sample program.

1.10.9.1 Static Cursor Sample

This example demonstrates the use of static cursors.

The particular cursor used here retrieves certain information from the Employees table in the sample database. The cursor is declared statically, meaning that the actual SQL statement to retrieve the information is hard coded into the source program. This is a good starting point for learning how cursors work. The Dynamic Cursor sample takes this first example and converts it to use dynamic SQL statements.

The open_cursor routine both declares a cursor for the specific SQL query and also opens the cursor.

Printing a page of information is done by the print routine. It loops `pagesize` times, fetching a single row from the cursor and printing it out. The fetch routine checks for warning conditions, such as rows that cannot be found (SQLCODE 100), and prints appropriate messages when they arise. In addition, the cursor is repositioned by this program to the row before the one that appears at the top of the current page of data.

The move, top, and bottom routines use the appropriate form of the FETCH statement to position the cursor. This form of the FETCH statement doesn't actually get the data. It only positions the cursor. Also, a general relative positioning routine, move, has been implemented to move in either direction depending on the sign of the parameter.

When the user quits, the cursor is closed and the database connection is also released. The cursor is closed by a ROLLBACK WORK statement, and the connection is released by a DISCONNECT.

Related Information

Running the Static Cursor Sample Program [page 274]
Dynamic Cursor Sample [page 275]

1.10.9.2 Running the Static Cursor Sample Program

Run the static cursor sample program.

Prerequisites

For Windows, a recent version of Microsoft Visual Studio is required.

For x86/x64 platform builds with Microsoft Visual Studio, you must set up the correct environment for compiling and linking. This is typically done using the Microsoft Visual Studio `vcvars32.bat` or `vcvars64.bat` (called `vcvarsamd64.bat` in older versions of Microsoft Visual Studio).

Context

The executable files and corresponding source code are located in the *%SQLANYSAMP17%*`\SQLAnywhere\C` directory.

Procedure

1. Start the sample database, *demo.db*.
2. Files to build the sample programs are supplied with the sample code.

   For Windows, run the `build.bat` batch file.

   If you are getting build errors, try specifying the target platform (`x86` or `x64`) as an argument to `build.bat`. Here is an example.

   ```
   build x64
   ```

   For UNIX and Linux, use the shell script `build.sh` to build the example.
3. For the 32-bit Windows example, run the file `curwin.exe`.

   For the 64-bit Windows example, run the file `curx64.exe`.

   For the UNIX/Linux example, run the file `cur`.
4. Follow the on-screen instructions.

Results

The various commands manipulate a database cursor and print the query results on the screen. Enter the letter of the command that you want to perform. Some systems may require you to press Enter after the letter.

1.10.9.3 Dynamic Cursor Sample

This sample demonstrates the use of cursors for a dynamic SQL SELECT statement.

The dynamic cursor sample program (dcur) allows the user to select a table to look at with the N command. The program then presents as much information from that table as fits on the screen.

When this program is run, it prompts for a connection string. The following is an example.

```
UID=DBA;PWD=sql;DBF=demo.db
```

The C program with the Embedded SQL is located in the *%SQLANYSAMP17%*`\SQLAnywhere\C` directory.

The dcur program uses the Embedded SQL interface function db_string_connect to connect to the database. This function provides the extra functionality to support the connection string that is used to connect to the database.

The open_cursor routine first builds the SELECT statement

```
SELECT * FROM table-name
```

where `table-name` is a parameter passed to the routine. It then prepares a dynamic SQL statement using this string.

The Embedded SQL DESCRIBE statement is used to fill in the SQLDA structure with the results of the SELECT statement.

> **Note**
>
> An initial guess is taken for the size of the SQLDA (3). If this is not big enough, the actual size of the SELECT list returned by the database server is used to allocate a SQLDA of the correct size.
>
> The SQLDA structure is then filled with buffers to hold strings that represent the results of the query. The fill_s_sqlda routine converts all data types in the SQLDA to DT_STRING and allocates buffers of the appropriate size.

A cursor is then declared and opened for this statement. The rest of the routines for moving and closing the cursor remain the same.

The fetch routine is slightly different: it puts the results into the SQLDA structure instead of into a list of host variables. The print routine has changed significantly to print results from the SQLDA structure up to the width of the screen. The print routine also uses the name fields of the SQLDA to print headings for each column.

Related Information

Running the Dynamic Cursor Sample Program [page 276]
Static Cursor Sample [page 273]

1.10.9.4 Running the Dynamic Cursor Sample Program

Run the dynamic cursor sample program.

Prerequisites

For Windows, a recent version of Microsoft Visual Studio is required.

For x86/x64 platform builds with Microsoft Visual Studio, you must set up the correct environment for compiling and linking. This is typically done using the Microsoft Visual Studio `vcvars32.bat` or `vcvars64.bat` (called `vcvarsamd64.bat` in older versions of Microsoft Visual Studio).

Context

The executable files and corresponding source code are located in the *%SQLANYSAMP17%*`\SQLAnywhere\C` directory.

Procedure

1. Start the sample database, *demo.db*.
2. Files to build the sample programs are supplied with the sample code.

   For Windows, run the `build.bat` batch file.

   If you are getting build errors, try specifying the target platform (`x86` or `x64`) as an argument to `build.bat`. Here is an example.

   ```
   build x64
   ```

   For UNIX and Linux, use the shell script `build.sh` to build the example.
3. For the 32-bit Windows example, run the file `dcurwin.exe`.

   For the 64-bit Windows example, run the file `dcurx64.exe`.

   You can call a function in an external library from a stored procedure or function. You can call functions in a DLL under Windows operating systems and in a shared object on UNIX and Linux.

For the UNIX/Linux example, run the file `dcur`.

4. Each sample program presents a console-type user interface and prompts you for a command. Enter the following connection string to connect to the sample database:

   ```
   DSN=SQL Anywhere 17 Demo;PWD=sql
   ```

5. Each sample program prompts you for a table. Choose one of the tables in the sample database. For example, you can enter **`Customers`** or **`Employees`**.
6. Follow the on-screen instructions.

Results

The various commands manipulate a database cursor and print the query results on the screen. Enter the letter of the command that you want to perform. Some systems may require you to press Enter after the letter.

1.10.10 Embedded SQL Data Types

To transfer information between a program and the database server, every piece of data must have a data type.

The Embedded SQL data type constants are prefixed with DT_, and can be found in the `sqldef.h` header file. You can create a host variable of any one of the supported types. You can also use these types in a SQLDA structure for passing data to and from the database.

You can define variables of these data types using the DECL_ macros listed in `sqlca.h`. For example, a variable holding a BIGINT value could be declared with DECL_BIGINT.

The following data types are supported by the Embedded SQL programming interface:

DT_BIT

8-bit signed integer.

DT_SMALLINT

16-bit signed integer.

DT_UNSSMALLINT

16-bit unsigned integer.

DT_TINYINT

8-bit signed integer.

DT_BIGINT

64-bit signed integer.

DT_UNSBIGINT

64-bit unsigned integer.

DT_INT

32-bit signed integer.

DT_UNSINT

32-bit unsigned integer.

DT_FLOAT

4-byte floating-point number.

DT_DOUBLE

8-byte floating-point number.

DT_DECIMAL

Packed decimal number (proprietary format).

```
typedef struct TYPE_DECIMAL {
  char array[1];
} TYPE_DECIMAL;
```

DT_STRING

Null-terminated character string, in the CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.

DT_NSTRING

Null-terminated character string, in the NCHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.

DT_DATE

Null-terminated character string that is a valid date.

DT_TIME

Null-terminated character string that is a valid time.

DT_TIMESTAMP

Null-terminated character string that is a valid timestamp.

DT_FIXCHAR

Fixed-length blank-padded character string, in the CHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.

DT_NFIXCHAR

Fixed-length blank-padded character string, in the NCHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.

DT_VARCHAR

Varying length character string, in the CHAR character set, with a two-byte length field. When sending data, you must set the length field. The maximum length that can be sent is 32767 bytes. When fetching data, the database server sets the length field. The maximum length that can be fetched is 32767 bytes. The data is not null-terminated or blank-padded. The sqldata field points to this data area that is exactly sqllen + 2 bytes long.

```
typedef struct VARCHAR {
 a_sql_ulen len;
 char       array[1];
} VARCHAR;
```

DT_NVARCHAR

Varying length character string, in the NCHAR character set, with a two-byte length field. When sending data, you must set the length field. The maximum length that can be sent is 32767 bytes. When fetching

data, the database server sets the length field. The maximum length that can be fetched is 32767 bytes. The data is not null-terminated or blank-padded. The sqldata field points to this data area that is exactly sqllen + 2 bytes long.

```
typedef struct NVARCHAR {
 a_sql_ulen len;
 char       array[1];
} NVARCHAR;
```

DT_LONGVARCHAR

Long varying length character string, in the CHAR character set.

```
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;
```

The LONGVARCHAR structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null-terminated or blank-padded.

DT_LONGNVARCHAR

Long varying length character string, in the NCHAR character set. The macro defines a structure, as follows:

```
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;
```

The LONGNVARCHAR structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null-terminated or blank-padded.

DT_BINARY

Varying length binary data with a two-byte length field. When sending data, you must set the length field. The maximum length that can be sent is 32767 bytes. When fetching data, the database server sets the length field. The maximum length that can be fetched is 32767 bytes. The data is not null-terminated or blank-padded. The sqldata field points to this data area that is exactly sqllen + 2 bytes long.

```
typedef struct BINARY {
 a_sql_ulen len;
 char       array[1];
} BINARY;
```

DT_LONGBINARY

Long binary data. The macro defines a structure, as follows:

```
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;
```

The LONGBINARY structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement.

DT_TIMESTAMP_STRUCT

SQLDATETIME structure with fields for each part of a timestamp.

```
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;
```

The SQLDATETIME structure can be used to retrieve fields of DATE, TIME, and TIMESTAMP type (or anything that can be converted to one of these). Often, applications have their own formats and date manipulation code. Fetching data in this structure makes it easier for you to manipulate this data. DATE, TIME, and TIMESTAMP fields can also be fetched and updated with any character type.

If you use a SQLDATETIME structure to enter a date, time, or timestamp into the database, the day_of_year and day_of_week members are ignored.

DT_VARIABLE

Null-terminated character string. The character string must be the name of a SQL variable whose value is used by the database server. This data type is used only for supplying data to the database server. It cannot be used when fetching data from the database server.

The structures are defined in the `sqlca.h` file. The VARCHAR, NVARCHAR, BINARY, DECIMAL, and LONG data types are not useful for declaring host variables because they contain a one-character array. However, they are useful for allocating variables dynamically or typecasting other variables.

DATE and TIME Database Types

There are no corresponding Embedded SQL interface data types for the various DATE and TIME database types. These database types are all fetched and updated using either the SQLDATETIME structure or character strings.

Related Information

How to Send and Retrieve Long Values Using Embedded SQL [page 329]
GET DATA Statement [ESQL]
date_format Option
date_order Option
time_format Option
timestamp_format Option

1.10.11 Host Variables in Embedded SQL

Host variables are C variables that are identified to the Embedded SQL preprocessor. Host variables can be used to send values to the database server or receive values from the database server.

Host variables are quite easy to use, but they have some restrictions. Dynamic SQL is a more general way of passing information to and from the database server using a structure known as the SQL Descriptor Area (SQLDA). The Embedded SQL preprocessor automatically generates a SQLDA for each statement in which host variables are used.

Host variables cannot be used in batches. Host variables cannot be used within a subquery in a SET statement.

In this section:

Embedded SQL Host Variable Declaration [page 282]
Host variables are defined by putting them into a **declaration section**. According to the ANSI Embedded SQL standard, host variables are defined by surrounding the normal C variable declarations with the following:

Embedded SQL Host Variable Data Types [page 283]
Only a limited number of C data types are supported as host variables. Also, certain host variable types do not have a corresponding C type.

Embedded SQL Host Variable Usage [page 286]
Host variables can be used in a variety of circumstances.

Embedded SQL Indicator Variables [page 287]
Indicator variables are C variables that hold supplementary information when you are fetching or putting data.

Related Information

Static and Dynamic SQL [page 296]

1.10.11.1 Embedded SQL Host Variable Declaration

Host variables are defined by putting them into a **declaration section**. According to the ANSI Embedded SQL standard, host variables are defined by surrounding the normal C variable declarations with the following:

```
EXEC SQL BEGIN DECLARE SECTION;
/* C variable declarations */
EXEC SQL END DECLARE SECTION;
```

These host variables can then be used in place of value constants in any SQL statement. When the database server executes the statement, the value of the host variable is used. Host variables cannot be used in place of table or column names: dynamic SQL is required for this. The variable name is prefixed with a colon (:) in a SQL statement to distinguish it from other identifiers allowed in the statement.

Using the Embedded SQL preprocessor, C language code is only scanned inside a DECLARE SECTION. So, TYPEDEF types and structures are not allowed, but initializers on the variables are allowed inside a DECLARE SECTION.

Example

The following sample code illustrates the use of host variables on an INSERT statement. The variables are filled in by the program and then inserted into the database:

```
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 );
```

Related Information

Static Cursor Sample [page 273]

1.10.11.2 Embedded SQL Host Variable Data Types

Only a limited number of C data types are supported as host variables. Also, certain host variable types do not have a corresponding C type.

Macros defined in the `sqlca.h` header file can be used to declare host variables of the following types: NCHAR, VARCHAR, NVARCHAR, LONGVARCHAR, LONGNVARCHAR, BINARY, LONGBINARY, DECIMAL, DT_FIXCHAR, DT_NFIXCHAR, DATETIME (SQLDATETIME), BIT, BIGINT, or UNSIGNED BIGINT. They are used as follows:

```
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;
```

The preprocessor recognizes these macros within an Embedded SQL declaration section and treats the variable as the appropriate type. Do not use the DECIMAL (DT_DECIMAL, DECL_DECIMAL) type since the format of decimal numbers is proprietary.

The following table lists the C variable types that are allowed for host variables and their corresponding Embedded SQL interface data types.

C Data Type	Embedded SQL Interface Type	Description
`short si;` `short int si;`	DT_SMALLINT	16-bit signed integer.
`unsigned short int usi;`	DT_UNSSMALLINT	16-bit unsigned integer.
`long l;` `long int l;`	DT_INT	32-bit signed integer.
`unsigned long int ul;`	DT_UNSINT	32-bit unsigned integer.
`DECL_BIGINT ll;`	DT_BIGINT	64-bit signed integer.
`DECL_UNSIGNED_BIGINT ull;`	DT_UNSBIGINT	64-bit unsigned integer.
`float f;`	DT_FLOAT	4-byte single-precision floating-point value.

C Data Type	Embedded SQL Interface Type	Description
`double d;`	DT_DOUBLE	8-byte double-precision floating-point value.
`char a[n]; /*n>=1*/`	DT_STRING	Null-terminated string, in CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings. This variable holds n-1 bytes plus the null terminator.
`char *a;`	DT_STRING	Null-terminated string, in CHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.
`DECL_NCHAR a[n]; /*n>=1*/`	DT_NSTRING	Null-terminated string, in NCHAR character set. The string is blank-padded if the database is initialized with blank-padded strings. This variable holds n-1 bytes plus the null terminator.
`DECL_NCHAR *a;`	DT_NSTRING	Null-terminated string, in NCHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.
`DECL_VARCHAR(n) a;`	DT_VARCHAR	Varying length character string, in CHAR character set, with 2-byte length field. Not null-terminated or blank-padded. The maximum value for n is 32767 (bytes).
`DECL_NVARCHAR(n) a;`	DT_NVARCHAR	Varying length character string, in NCHAR character set, with 2-byte length field. Not null-terminated or blank-padded. The maximum value for n is 32767 (bytes).
`DECL_LONGVARCHAR(n) a;`	DT_LONGVARCHAR	Varying length long character string, in CHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.
`DECL_LONGNVARCHAR(n) a;`	DT_LONGNVARCHAR	Varying length long character string, in NCHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.
`DECL_BINARY(n) a;`	DT_BINARY	Varying length binary data with 2-byte length field. The maximum value for n is 32767 (bytes).
`DECL_LONGBINARY(n) a;`	DT_LONGBINARY	Varying length long binary data with three 4-byte length fields.
`char a; /*n=1*/ DECL_FIXCHAR(n) a;`	DT_FIXCHAR	Fixed length character string, in CHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).

C Data Type	Embedded SQL Interface Type	Description
`DECL_NCHAR a; /*n=1*/ DECL_NFIXCHAR(n) a;`	DT_NFIXCHAR	Fixed length character string, in NCHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).
`DECL_DATETIME a;`	DT_TIMESTAMP_STRUCT	SQLDATETIME structure

Character Sets

For DT_FIXCHAR, DT_STRING, DT_VARCHAR, and DT_LONGVARCHAR, character data is in the application's CHAR character set, which is usually the character set of the application's locale. An application can change the CHAR character set either by using the CHARSET connection parameter, or by calling the db_change_char_charset function.

For DT_NFIXCHAR, DT_NSTRING, DT_NVARCHAR, and DT_LONGNVARCHAR, data is in the application's NCHAR character set. By default, the application's NCHAR character set is the same as the CHAR character set. An application can change the NCHAR character set by calling the db_change_nchar_charset function.

Data Lengths

Regardless of the CHAR and NCHAR character sets in use, all data lengths are specified in bytes.

If character set conversion occurs between the server and the application, it is the application's responsibility to ensure that buffers are sufficiently large to handle the converted data, and to issue additional GET DATA statements if data is truncated.

Pointers to Char

The database interface considers a host variable declared as a **pointer to char** (`char * a`) to be 32767 bytes long. Any host variable of type pointer to char used to retrieve information from the database must point to a buffer large enough to hold any value that could possibly come back from the database.

This is potentially quite dangerous because someone could change the definition of the column in the database to be larger than it was when the program was written. This could cause random memory corruption problems. It is better to use a declared array, even as a parameter to a function, where it is passed as a pointer to char. This technique allows the Embedded SQL statements to know the size of the array.

Scope of Host Variables

A standard host-variable declaration section can appear anywhere that C variables can normally be declared. This includes the parameter declaration section of a C function. The C variables have their normal scope

(available within the block in which they are defined). However, since the Embedded SQL preprocessor does not scan C code, it does not respect C blocks.

As far as the Embedded SQL preprocessor is concerned, host variables are global to the source file; two host variables cannot have the same name.

Related Information

Locales
CharSet (CS) Connection Parameter
db_change_char_charset Function [page 349]
db_change_nchar_charset Function [page 350]
GET DATA Statement [ESQL]

1.10.11.3 Embedded SQL Host Variable Usage

Host variables can be used in a variety of circumstances.

Host variables can be used with the following:

- SELECT, INSERT, UPDATE, and DELETE statements in any place where a number or string constant is allowed.
- The INTO clause of SELECT and FETCH statements.
- Host variables can also be used in place of a statement name, a cursor name, or an option name in statements specific to Embedded SQL.
- For CONNECT, DISCONNECT, and SET CONNECT statements, a host variable can be used in place of a server name, database name, connection name, user ID, password, or connection string.
- For SET OPTION and GET OPTION, a host variable can be used in place of the option value.

Host variables cannot be used in the following circumstances:

- Host variables cannot be used in place of a table name or a column name in any statement.
- Host variables cannot be used in batches.
- Host variables cannot be used within a subquery in a SET statement.

SQLCODE and SQLSTATE host variables

The ISO/ANSI standard allows an Embedded SQL source file to declare the following special host variables within an Embedded SQL declaration section:

```
long SQLCODE;
char SQLSTATE[6];
```

If used, these variables are set after any Embedded SQL statement that makes a database request (EXEC SQL statements other than DECLARE SECTION, INCLUDE, WHENEVER SQLCODE, and so on). As a consequence,

the SQLCODE and SQLSTATE host variables must be visible in the scope of every Embedded SQL statement that generates database requests.

The following is valid 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 ...
}
```

The following is not valid Embedded SQL because SQLSTATE is not defined in the scope of the function sub2:

```
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...
}
```

The Embedded SQL preprocessor -k option permits the declaration of the SQLCODE variable outside the scope of an Embedded SQL declaration section.

Related Information

The Embedded SQL Preprocessor [page 264]

1.10.11.4 Embedded SQL Indicator Variables

Indicator variables are C variables that hold supplementary information when you are fetching or putting data.

There are several distinct uses for indicator variables:

NULL values

To enable applications to handle NULL values.

String truncation

To enable applications to handle cases when fetched values must be truncated to fit into host variables.

Conversion errors

To hold error information.

An indicator variable is a host variable of type a_sql_len that is placed immediately following a regular host variable in a SQL statement. For example, in the following INSERT statement, :ind_phone is an indicator variable:

```
EXEC SQL INSERT INTO Employees
 VALUES (:employee_number, :employee_name,
 :employee_initials, :employee_phone:ind_phone );
```

On a fetch or execute where no rows are received from the database server (such as when an error or end of result set occurs), then indicator values are unchanged.

> **Note**
>
> To allow for the future use of 32 and 64-bit lengths and indicators, the use of short int for Embedded SQL indicator variables is deprecated. Use a_sql_len instead.

In this section:

Indicator Variables: The SQL NULL Value [page 288]
Do not confuse the SQL concept of NULL with the C language constant of the same name.

Indicator Variables: Truncated Values [page 289]
Indicator variables indicate whether any fetched values were truncated to fit into a host variable. This enables applications to handle truncation appropriately.

Indicator Variables: Conversion Errors [page 289]
By default, the conversion_error database option is set to On, and any data type conversion failure leads to an error, with no row returned.

Summary of Indicator Variable Values [page 290]
Indicator values are used to convey information about retrieved column values.

1.10.11.4.1 Indicator Variables: The SQL NULL Value

Do not confuse the SQL concept of NULL with the C language constant of the same name.

In the SQL language, NULL represents either an unknown attribute or inapplicable information. The C language NULL is referred to as the null pointer constant and represents a pointer value that does not point to a memory location.

When NULL is used in this documentation, it refers to the SQL database meaning given above.

NULL is not the same as any value of the column's defined type. So, something extra is required beyond regular host variables to pass NULL values to the database or receive NULL results back. **Indicator variables** are used for this purpose.

Using Indicator Variables when Inserting NULL

An INSERT statement could include an indicator variable as follows:

```
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 );
```

If the indicator variable has a value of -1, a NULL is written. If it has a value of 0, the actual value of employee_phone is written.

Using Indicator Variables when Fetching NULL

Indicator variables are also used when receiving data from the database. They are used to indicate that a NULL value was fetched (indicator is negative). If a NULL value is fetched from the database and an indicator variable is not supplied, an error is generated (SQLE_NO_INDICATOR).

1.10.11.4.2 Indicator Variables: Truncated Values

Indicator variables indicate whether any fetched values were truncated to fit into a host variable. This enables applications to handle truncation appropriately.

If a value is truncated on fetching, the indicator variable is set to a positive value, containing the actual length of the database value before truncation. If the actual length of the database value is greater than 32767 bytes, then the indicator variable contains 32767.

1.10.11.4.3 Indicator Variables: Conversion Errors

By default, the conversion_error database option is set to On, and any data type conversion failure leads to an error, with no row returned.

You can use indicator variables to tell which column produced a data type conversion failure. If you set the database option conversion_error to Off, any data type conversion failure gives a CANNOT_CONVERT warning,

rather than an error. If the column that suffered the conversion error has an indicator variable, that variable is set to a value of -2.

If you set the conversion_error option to Off when inserting data into the database, a value of NULL is inserted when a conversion failure occurs.

1.10.11.4.4 Summary of Indicator Variable Values

Indicator values are used to convey information about retrieved column values.

The following table provides a summary of indicator variable usage.

Indicator Value	Supplying Value to Database	Receiving Value from Database
> 0	Host variable value	Retrieved value was truncated (actual length in indicator variable).
0	Host variable value	Fetch successful, or conversion_error set to On.
-1	NULL value	NULL result.
-2	NULL value	Conversion error (when conversion_error is set to Off only). SQLCODE indicates a CANNOT_CONVERT warning.
< -2	NULL value	NULL result.

Related Information

GET DATA Statement [ESQL]

1.10.12 The SQL Communication Area (SQLCA)

The **SQL Communication Area** (**SQLCA**) is an area of memory that is used on every database request for communicating statistics and errors from the application to the database server and back to the application.

The SQLCA is used as a handle for the application-to-database communication link. It is passed in to all database library functions that must communicate with the database server. It is implicitly passed on all Embedded SQL statements.

A global SQLCA variable is defined in the interface library. The Embedded SQL preprocessor generates an external reference for the global SQLCA variable and an external reference for a pointer to it. The external reference is named sqlca and is of type SQLCA. The pointer is named sqlcaptr. The actual global variable is declared in the import library.

The SQLCA is defined by the `sqlca.h` header file, included in the `SDK\Include` subdirectory of your software installation directory.

SQLCA Provides Error Codes

You reference the SQLCA to test for a particular error code. The sqlcode and sqlstate fields contain error codes when a database request has an error. Some C macros are defined for referencing the sqlcode field, the sqlstate field, and some other fields.

In this section:

SQLCA Fields [page 291]
The SQLCA is a structure with several fields.

SQLCA Management for Multithreaded or Reentrant Code [page 293]
You can use Embedded SQL statements in multithreaded or reentrant code.

Multiple SQLCAs [page 295]
You must not use the Embedded SQL preprocessor option (-r-) that generates non-reentrant code. Reentrant code is a little larger and a little slower because statically initialized global variables cannot be used. However, these effects are minimal.

1.10.12.1 SQLCA Fields

The SQLCA is a structure with several fields.

These fields have the following meanings:

sqlcaid

An 8-byte character field that contains the string SQLCA as an identification of the SQLCA structure. This field helps in debugging when you are looking at memory contents.

sqlcabc

A 32-bit integer that contains the length of the SQLCA structure (136 bytes).

sqlcode

A 32-bit integer that specifies the error code when the database detects an error on a request. Definitions for the error codes can be found in the header file `sqlerr.h`. The error code is 0 (zero) for a successful operation, positive for a warning, and negative for an error.

sqlerrml

The length of the information in the sqlerrmc field.

sqlerrmc

Zero or more character strings to be inserted into an error message. Some error messages contain one or more placeholder strings (`%1`, `%2`, ...) that are replaced with the strings in this field.

For example, if a `Table Not Found` error is generated, sqlerrmc contains the table name, which is inserted into the error message at the appropriate place.

sqlerrp

Reserved.

sqlerrd

A utility array of 32-bit integers.

sqlwarn

Reserved.

sqlstate

The SQLSTATE status value. The ANSI SQL standard defines this type of return value from a SQL statement in addition to the SQLCODE value. The SQLSTATE value is always a five-character null-terminated string, divided into a two-character class (the first two characters) and a three-character subclass. Each character can be a digit from 0 through 9 or an uppercase alphabetic character A through Z.

Any class or subclass that begins with 0 through 4 or A through H is defined by the SQL standard; other classes and subclasses are implementation defined. The SQLSTATE value '00000' means that there has been no error or warning.

sqlerror Array

The sqlerror field array has the following elements.

sqlerrd[1] (SQLIOCOUNT)

The actual number of input/output operations that were required to complete a statement.

The database server does not set this number to zero for each statement. Your program can set this variable to zero before executing a sequence of statements. After the last statement, this number is the total number of input/output operations for the entire statement sequence.

sqlerrd[2] (SQLCOUNT)

The value of this field depends on which statement is being executed.

INSERT, UPDATE, PUT, and DELETE statements

The number of rows that were affected by the statement.

OPEN and RESUME statements

On a cursor OPEN or RESUME, this field is filled in with either the actual number of rows in the cursor (a value greater than *or equal to* 0) or an estimate thereof (a negative number whose absolute value is the estimate). It is the actual number of rows if the database server can compute it without counting the rows. The database can also be configured to always return the actual number of rows using the row_counts option.

FETCH cursor statement

The SQLCOUNT field is filled if a SQLE_NOTFOUND warning is returned. It contains the number of rows by which a FETCH RELATIVE or FETCH ABSOLUTE statement goes outside the range of possible cursor positions (a cursor can be on a row, before the first row, or after the last row). For a wide fetch, SQLCOUNT is the number of rows actually fetched, and is less than or equal to the number of rows requested. During a wide fetch, SQLE_NOTFOUND is only set if no rows are returned.

The value is 0 if the row was not found, but the position is valid, for example, executing FETCH RELATIVE 1 when positioned on the last row of a cursor. The value is positive if the attempted fetch was beyond the end of the cursor, and negative if the attempted fetch was before the beginning of the cursor.

GET DATA statement

The SQLCOUNT field holds the actual length of the value.

DESCRIBE statement

If the WITH VARIABLE RESULT clause is used to describe procedures that may have more than one result set, SQLCOUNT is set to one of the following values:

0

The result set may change: the procedure call should be described again following each OPEN statement.

1

The result set is fixed. No re-describing is required.

For the SQLE_SYNTAX_ERROR syntax error, the field contains the approximate character position within the statement where the error was detected.

sqlerrd[3] (SQLIOESTIMATE)

The estimated number of input/output operations that are required to complete the statement. This field is given a value on an OPEN or EXPLAIN statement.

Related Information

SQL Anywhere Error Messages
Wide Fetches Using Embedded SQL [page 315]
SQL Anywhere Error Messages Sorted by SQLCODE
SQL Anywhere Error Messages Sorted by SQLSTATE

1.10.12.2 SQLCA Management for Multithreaded or Reentrant Code

You can use Embedded SQL statements in multithreaded or reentrant code.

However, if you use a single connection, you are restricted to one active request per connection. In a multithreaded application, do not use the same connection to the database on each thread unless you use a semaphore to control access.

There are no restrictions on using separate connections on each thread that wants to use the database. The SQLCA is used by the runtime library to distinguish between the different thread contexts. So, each thread wanting to use the database concurrently must have its own SQLCA. The exception is that a thread can use the db_cancel_request function to cancel a statement executing on a different thread using that thread's SQLCA.

The following is an example of reentrant multithreaded Embedded SQL code.

```
#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 "password";
     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 );
}
```

Related Information

Request Management with Embedded SQL [page 339]

1.10.12.3 Multiple SQLCAs

You must not use the Embedded SQL preprocessor option (-r-) that generates non-reentrant code. Reentrant code is a little larger and a little slower because statically initialized global variables cannot be used. However, these effects are minimal.

Each SQLCA used in your program must be initialized with a call to db_init and cleaned up at the end with a call to db_fini.

The Embedded SQL statement SET SQLCA is used to tell the Embedded SQL preprocessor to use a different SQLCA for database requests. Usually, a statement such as `EXEC SQL SET SQLCA 'task_data->sqlca';` is used at the top of your program or in a header file to set the SQLCA reference to point at task specific data. Performance is unaffected because this statement does not generate any code. It changes the state within the preprocessor so that any reference to the SQLCA uses the given string.

Each thread must have its own SQLCA. This requirement also applies to code in a shared library (in a DLL, for example) that uses Embedded SQL and is called by more than one thread in your application.

You can use the multiple SQLCA support in any of the supported Embedded SQL environments, but it is only required in reentrant code.

You do not need to use multiple SQLCAs to connect to more than one database or have more than one connection to a single database.

Each SQLCA can have one unnamed connection. Each SQLCA has an active or current connection.

All operations on a given database connection must use the same SQLCA that was used when the connection was established.

> **Note**
>
> Operations on different connections are subject to the normal record locking mechanisms and may cause each other to block and possibly to deadlock.

Related Information

SET SQLCA Statement [ESQL]

SET CONNECTION Statement [Interactive SQL] [ESQL]

1.10.13 Static and Dynamic SQL

There are two ways to embed SQL statements into a C program: statically or dynamically.

In this section:

Static SQL Statements [page 296]
All standard SQL data manipulation and data definition statements can be embedded in a C program by prefixing them with EXEC SQL and suffixing the statement with a semicolon (;). These statements are referred to as **static** statements.

Dynamic SQL Statements [page 297]
In the C language, strings are stored in arrays of characters. Dynamic SQL statements are constructed in C language strings. These statements can then be executed using the PREPARE and EXECUTE statements.

Dynamic SELECT Statement [page 298]
A SELECT statement that returns only a single row can be prepared dynamically, followed by an EXECUTE with an INTO clause to retrieve the one-row result. SELECT statements that return multiple rows, however, are managed using dynamic cursors.

1.10.13.1 Static SQL Statements

All standard SQL data manipulation and data definition statements can be embedded in a C program by prefixing them with EXEC SQL and suffixing the statement with a semicolon (;). These statements are referred to as **static** statements.

Static statements can contain references to host variables. Host variables can only be used in place of string or numeric constants. They cannot be used to substitute column names or table names; dynamic statements are required to perform those operations.

Related Information

Host Variables in Embedded SQL [page 281]

1.10.13.2 Dynamic SQL Statements

In the C language, strings are stored in arrays of characters. Dynamic SQL statements are constructed in C language strings. These statements can then be executed using the PREPARE and EXECUTE statements.

These SQL statements cannot reference host variables in the same manner as static statements since the C language variables are not accessible by name when the C program is executing.

To pass information between the statements and the C language variables, a data structure called the **SQL Descriptor Area** (**SQLDA**) is used. This structure is set up for you by the Embedded SQL preprocessor if you specify a list of host variables on the EXECUTE statement in the USING clause. These variables correspond by position to placeholders in the appropriate positions of the prepared statement.

A **placeholder** is put in the statement to indicate where host variables are to be accessed. A placeholder is either a question mark (?) or a host variable reference as in static statements (a host variable name preceded by a colon). In the latter case, the host variable name used in the actual text of the statement serves only as a placeholder indicating a reference to the SQL descriptor area.

A host variable used to pass information to the database is called a **bind variable**.

Example

```
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;
```

This method requires you to know how many host variables there are in the statement. Usually, this is not the case. So, you can set up your own SQLDA structure and specify this SQLDA in the USING clause on the EXECUTE statement.

The DESCRIBE BIND VARIABLES statement returns the host variable names of the bind variables that are found in a prepared statement. This makes it easier for a C program to manage the host variables. The general method is as follows:

```
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 Contents

The SQLDA consists of an array of variable descriptors. Each descriptor describes the attributes of the corresponding C program variable or the location that the database stores data into or retrieves data from:

- data type
- length if `type` is a string type
- memory address
- indicator variable

Indicator Variables and NULL

The indicator variable is used to pass a NULL value to the database or retrieve a NULL value from the database. The database server also uses the indicator variable to indicate truncation conditions encountered during a database operation. The indicator variable is set to a positive value when not enough space was provided to receive a database value.

Related Information

The SQL Descriptor Area (SQLDA) [page 300]
Embedded SQL Indicator Variables [page 287]
EXECUTE Statement [ESQL]

1.10.13.3 Dynamic SELECT Statement

A SELECT statement that returns only a single row can be prepared dynamically, followed by an EXECUTE with an INTO clause to retrieve the one-row result. SELECT statements that return multiple rows, however, are managed using dynamic cursors.

With dynamic cursors, results are put into a host variable list or a SQLDA that is specified on the FETCH statement (FETCH INTO and FETCH USING DESCRIPTOR). Since the number of SELECT list items is usually unknown, the SQLDA route is the most common. The DESCRIBE SELECT LIST statement sets up a SQLDA

with the types of the SELECT list items. Space is then allocated for the values using the fill_sqlda or fill_s_sqlda functions, and the information is retrieved by the FETCH USING DESCRIPTOR statement.

The typical scenario is as follows:

```
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;
```

> **Note**
>
> To avoid consuming unnecessary resources, ensure that statements are dropped after use.

The dynamic cursor example illustrates the use of cursors with a dynamic SELECT statement.

Related Information

PREPARE Statement [ESQL]
DESCRIBE Statement [ESQL]
Dynamic Cursor Sample [page 275]
Library Function Reference [page 340]

1.10.14 The SQL Descriptor Area (SQLDA)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file `sqlda.h`.

There are functions in the database interface shared library or DLL that you can use to manage SQLDAs.

When host variables are used with static SQL statements, the preprocessor constructs a SQLDA for those host variables. It is this SQLDA that is actually passed to and from the database server.

In this section:

The SQLDA Header File [page 300]
The SQLDA (SQL Descriptor Area) data structure is described by the `sqlda.h` header file.

SQLDA Fields [page 301]
The SQLDA (SQL Descriptor Area) is a data structure consisting of a number of fields.

SQLDA Host Variable Descriptions (struct sqlvar) [page 302]
Each sqlvar structure in the SQLDA describes a host variable.

SQLDA sqllen Field Values After a DESCRIBE [page 305]
The DESCRIBE statement gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

SQLDA sqllen Field Values When Sending Data [page 307]
The manner in which the length of the data being sent to the database is determined depends on the data type.

SQLDA sqllen Field Values When Retrieving Data [page 308]
The manner in which the length of the data being retrieved from the database is determined depends on the data type.

Related Information

Library Function Reference [page 340]

1.10.14.1 The SQLDA Header File

The SQLDA (SQL Descriptor Area) data structure is described by the `sqlda.h` header file.

The contents of `sqlda.h` are as follows:

```
#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
  #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   sqldaid[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
```

1.10.14.2 SQLDA Fields

The SQLDA (SQL Descriptor Area) is a data structure consisting of a number of fields.

The SQLDA fields have the following meanings:

Field	Description
sqldaid	An 8-byte character field that contains the string SQLDA as an identification of the SQLDA structure. This field helps in debugging when you are looking at memory contents.

Field	Description
sqldabc	A 32-bit integer containing the length of the SQLDA structure.
sqln	The number of variable descriptors allocated in the sqlvar array.
sqld	The number of variable descriptors that are valid (contain information describing a host variable). This field is set by the DESCRIBE statement. As well, you can set it when supplying data to the database server.
sqlvar	An array of descriptors of type struct sqlvar, each describing a host variable.

1.10.14.3 SQLDA Host Variable Descriptions (struct sqlvar)

Each sqlvar structure in the SQLDA describes a host variable.

The fields of the sqlvar structure are defined in the `sqlda.h` header file and have the following meanings:

sqltype

The type and flags of the variable that is described by this descriptor. The type is extracted using DT_TYPES and the flags are extracted using DT_FLAGS.

```
a_sql_type sql_type = sqlvar->sqltype & DT_TYPES;
a_sql_type sql_flag = sqlvar->sqltype & DT_FLAGS;
```

The DT_NULLS_ALLOWED bit of the flags indicates whether NULL values are allowed. Valid types and flags are defined in the `sqlhosttype.h` header file.

This field is filled by the DESCRIBE statement. You can change this field to any type when supplying data to the database server or retrieving data from the database server as long as the sqldata field is changed accordingly to match the new type. Any necessary type conversion is done automatically (for example, binary integer to ASCII string).

sqllen

The length of the variable. A sqllen value has type a_sql_len. What the length actually means depends on the type information and how the SQLDA is being used.

For LONG VARCHAR, LONG NVARCHAR, and LONG BINARY data types, the array_len field of the LONGVARCHAR, LONGNVARCHAR, or LONGBINARY data type structure is used instead of the sqllen field. In these cases, sqllen is usually set to 32767 (the maximum).

Otherwise, the sqllen value describes the length of the variable in bytes. For example, sqllen is 1280 for a VARCHAR(1280) variable.

sqldata

A pointer to the memory occupied by this variable. This memory must correspond to the type of the variable and its length.

The amount of memory required for specific types is returned by the following macros defined in the `sqlca.h` and `sqlda.h` header files.

DT_BINARY

_BINARYSIZE(n) returns the amount of storage required to store a BINARY of the specified length.

DT_VARCHAR

_VARCHARSIZE(n) returns the amount of storage required to store a VARCHAR of the specified length.

DT_DECIMAL

DECIMALSTORAGE(n) returns the amount of storage required to store a DECIMAL of the specified length.

DT_LONGBINARY

LONGBINARYSIZE(n) returns the amount of storage required to store a LONGBINARY of the specified length.

DT_LONGNVARCHAR

LONGNVARCHARSIZE(n) returns the amount of storage required to store a LONGNVARCHAR of the specified length.

DT_LONGVARCHAR

LONGVARCHARSIZE(n) returns the amount of storage required to store a LONGVARCHAR of the specified length.

All other types have fixed storage sizes depending on their type and size. Some examples follow.

If the type is DT_INT, then sqldata points to a 32-bit binary integer data value.

```
int ival = (int *)sqlvar->sqldata;
```

If the type is DT_VARCHAR, then sqldata points to a VARCHAR structure (which is defined in `sqlca.h`).

```
typedef struct VARCHAR {
    a_sql_ulen          len;
    char                array[1];
} VARCHAR;
sqlvar->sqldata = malloc( _VARCHARSIZE( sqlvar->sqllen ) );
VARCHAR *vc = (VARCHAR *)sqlvar->sqldata;
```

DT_NVARCHAR and DT_BINARY types have structures similar to DT_VARCHAR.

If the type is DT_LONGVARCHAR, then sqldata points to a LONGVARCHAR structure (which is defined in `sqlca.h`).

```
typedef struct LONGVARCHAR {
    a_sql_uint32        array_len;
    a_sql_uint32        stored_len;
    a_sql_uint32        untrunc_len;
    char                array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
sqlvar->sqldata = malloc( LONGVARCHARSIZE( 128*1024 ) );
LONGVARCHAR *lvc = (LONGVARCHAR *)sqlvar->sqldata;
lvc->array_len = 128*1024;
```

DT_LONGNVARCHAR and DT_LONGBINARY types have structures similar to DT_LONGVARCHAR.

For UPDATE and INSERT statements, this variable is not involved in the operation if the sqldata pointer is a null pointer. For a FETCH, no data is returned if the sqldata pointer is a null pointer. In other words, the column returned by the sqldata pointer is an **unbound column**.

If the DESCRIBE statement uses LONG NAMES, this field holds the long name of the result set column. If, in addition, the DESCRIBE statement is a DESCRIBE USER TYPES statement, then this field holds the long name of the user-defined data type, instead of the column. If the type is a base type, the field is empty.

sqlind

A pointer to the indicator value. An indicator value has type a_sql_len. A negative indicator value indicates a NULL value. A positive indicator value indicates that this variable has been truncated by a FETCH statement, and the indicator value contains the length of the data before truncation. A value of -2 indicates a conversion error if the conversion_error database option is set to Off.

If the sqlind pointer is the null pointer, no indicator variable pertains to this host variable.

The sqlind field is also used by the DESCRIBE statement to indicate parameter types. If the type is a user-defined data type, this field is set to DT_HAS_USERTYPE_INFO. In this case, perform a DESCRIBE USER TYPES to obtain information about the user-defined data types.

sqlname

A VARCHAR-like structure, as follows:

```
struct sqlname {
   short int  length;
   char       data[ SQL_MAX_NAME_LEN ];
};
```

It is filled by a DESCRIBE statement and is not otherwise used. This field has a different meaning for the two formats of the DESCRIBE statement:

SELECT LIST

The name data buffer is filled with the column heading of the corresponding item in the SELECT list.

BIND VARIABLES

The name data buffer is filled with the name of the host variable that was used as a bind variable, or "?" if an unnamed parameter marker is used.

On a DESCRIBE SELECT LIST statement, any indicator variables present are filled with a flag indicating whether the SELECT list item is updatable or not. The DT_UPDATABLE flag and others are defined in the `sqlhosttype.h` header file.

If the DESCRIBE statement is a DESCRIBE USER TYPES statement, then this field holds the long name of the user-defined data type instead of the column. If the type is a base type, the field is empty.

Related Information

Embedded SQL Indicator Variables [page 287]
Embedded SQL Data Types [page 277]
conversion_error Option

1.10.14.4 SQLDA sqllen Field Values After a DESCRIBE

The DESCRIBE statement gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

The following table indicates the values of the sqllen and sqltype structure members returned by the DESCRIBE statement for the various database types (both SELECT LIST and BIND VARIABLE DESCRIBE statements). For a user-defined database data type, the base type is described.

Your program can use the types and lengths returned from a DESCRIBE, or you may use another type. The database server performs type conversions between any two types. The memory pointed to by the sqldata field must correspond to the sqltype and sqllen fields. The Embedded SQL type is obtained by a bitwise AND of sqltype with DT_TYPES (sqltype & DT_TYPES).

Database Field Type	Embedded SQL Type Returned	Length (in Bytes) Returned on Describe
BIGINT	DT_BIGINT	8
BINARY(n)	DT_BINARY	n
BIT	DT_BIT	1
CHAR(n)	DT_FIXCHAR[1]	n times maximum data expansion when converting from database character set to the client's CHAR character set. If this length would be more than 32767 bytes, then the Embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
CHAR(n CHAR)	DT_FIXCHAR[1]	n times maximum character length in the client's CHAR character set. If this length would be more than 32767 bytes, then the Embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
DATE	DT_DATE	length of longest formatted string
DECIMAL(p,s)	DT_DECIMAL	low byte of length field in SQLDA set to p, and high byte set to s. See PRECISION and SCALE macros 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

Database Field Type	Embedded SQL Type Returned	Length (in Bytes) Returned on Describe
NCHAR(n)	DT_FIXCHAR / DT_NFIXCHAR[2]	n times maximum character length in the client's NCHAR character set. If this length would be more than 32767 bytes, then the Embedded SQL type returned is DT_LONGNVARCHAR with a length of 32767 bytes.
NVARCHAR(n)	DT_VARCHAR / DT_NVARCHAR[2]	n times maximum character length in the client's NCHAR character set. If this length would be more than 32767 bytes, then the Embedded SQL type returned is DT_LONGNVARCHAR with a length of 32767 bytes.
REAL	DT_FLOAT	4
SMALLINT	DT_SMALLINT	2
TIME	DT_TIME	length of longest formatted string
TIMESTAMP	DT_TIMESTAMP	length of longest formatted string
TINYINT	DT_TINYINT	1
UNSIGNED BIGINT	DT_UNSBIGINT	8
UNSIGNED INT	DT_UNSINT	4
UNSIGNED SMALLINT	DT_UNSSMALLINT	2
VARCHAR(n)	DT_VARCHAR[1]	n times maximum data expansion when converting from database character set to the client's CHAR character set. If this length would be more than 32767 bytes, then the Embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
VARCHAR(n CHAR)	DT_VARCHAR[1]	n times maximum character length in the client's CHAR character set. If this length would be more than 32767, then the Embedded SQL type returned is DT_LONGVARCHAR with length 32767.

[1] The type returned for CHAR and VARCHAR may be DT_LONGVARCHAR if the maximum byte length in the client's CHAR character set is greater than 32767 bytes.

[2] The type returned for NCHAR and NVARCHAR may be DT_LONGNVARCHAR if the maximum byte length in the client's NCHAR character set is greater than 32767 bytes. NCHAR, NVARCHAR, and LONG NVARCHAR are described by default as either DT_FIXCHAR, DT_VARCHAR, or DT_LONGVARCHAR, respectively. If the db_change_nchar_charset function has been called, the types are described as DT_NFIXCHAR, DT_NVARCHAR, and DT_LONGNVARCHAR, respectively.

Related Information

Embedded SQL Data Types [page 277]
CHAR Data Type
NCHAR Data Type
NVARCHAR Data Type
VARCHAR Data Type
db_change_nchar_charset Function [page 350]

1.10.14.5 SQLDA sqllen Field Values When Sending Data

The manner in which the length of the data being sent to the database is determined depends on the data type.

For non-long data types, the `sqllen` field of the `sqlvar` structure in the SQLDA represents the maximum size of the data buffer. For example, a column described as VARCHAR(300) will have a `sqllen` value of 300, representing the maximum length for that column. For blank-padded data types such as DT_FIXCHAR, the `sqllen` field represents the maximum size of the data buffer. For fixed-size data types such as integers and DT_TIMESTAMP_ STRUCT, the `sqllen` field is ignored and the length need not be specified. For long data types, the `array_len` field specifies the maximum length of the data buffer. The `sqllen` field is never modified when you send or retrieve data.

Only the data types displayed in the table below are allowed. The DT_DATE, DT_TIME, and DT_TIMESTAMP data types are treated the same as DT_STRING when you send or retrieve information. The value is formatted as a character string in the current date format.

Embedded SQL Data Type	Program Action to Set the Length
DT_BIGINT	Fixed size. No action required.
DT_BINARY(n)	Set the `len` field to the length in bytes of data in the `array` field of the BINARY structure. The maximum length is 32767.
DT_BIT	Fixed size. No action required.
DT_DATE	The length is determined by the terminating null character.
DT_DOUBLE	Fixed size. No action required.
DT_FIXCHAR(n)	The `sqllen` field determines the length in bytes of the `sqldata` area. The data must be blank-padded. The maximum length is 32767.
DT_FLOAT	Fixed size. No action required.
DT_INT	Fixed size. No action required.
DT_LONGBINARY	The `sqllen` field is ignored. The `array_len` field specifies the maximum length of the `array` field. Set the `stored_len` field to the actual length in bytes of data in the `array` field. More than 32767 bytes can be sent.

Embedded SQL Data Type	Program Action to Set the Length
DT_LONGNVARCHAR	The `sqllen` field is ignored. The `array_len` field specifies the maximum length of the `array` field. Set the `stored_len` field to the actual length in bytes of data in the `array` field. More than 32767 bytes can be sent.
DT_LONGVARCHAR	The `sqllen` field is ignored. The `array_len` field specifies the maximum length of the `array` field. Set the `stored_len` field to the actual length in bytes of data in the `array` field. More than 32767 bytes can be sent.
DT_NFIXCHAR(n)	The `sqllen` field determines the length in bytes of the `sqldata` area. The data must be blank-padded. The maximum length is 32767.
DT_NSTRING	The length is determined by the terminating null character.
DT_NVARCHAR(n)	Set the `len` field to the length in bytes of data in the `array` field of the NVARCHAR structure.
DT_SMALLINT	Fixed size. No action required.
DT_STRING	The length is determined by the terminating null character.
DT_TIME	The length is determined by the terminating null character.
DT_TIMESTAMP	The length is determined by the terminating null character.
DT_TIMESTAMP_STRUCT	Fixed size. No action required.
DT_UNSBIGINT	Fixed size. No action required.
DT_UNSINT	Fixed size. No action required.
DT_UNSSMALLINT	Fixed size. No action required.
DT_VARCHAR(n)	Set the `len` field to the length in bytes of data in the `array` field of the VARCHAR structure. The maximum length is 32767.
DT_VARIABLE	The length is determined by the terminating null character.

Related Information

Sending LONG Data Using Dynamic SQL [page 335]

1.10.14.6 SQLDA sqllen Field Values When Retrieving Data

The manner in which the length of the data being retrieved from the database is determined depends on the data type.

For non-long data types, the `sqllen` field of the `sqlvar` structure in the SQLDA represents the maximum size of the data buffer. For example, a column described as VARCHAR(300) will have a `sqllen` value of 300, representing the maximum length for that column. For blank-padded data types such as DT_FIXCHAR,

the `sqllen` field represents the maximum size of the data buffer. For fixed-size data types such as integers and DT_TIMESTAMP_ STRUCT, the `sqllen` field is ignored and the length need not be specified. For long data types, the `array_len` field specifies the maximum length of the data buffer. The `sqllen` field is never modified when you send or retrieve data.

Only the data types displayed in the table below are allowed. The DT_DATE, DT_TIME, and DT_TIMESTAMP data types are treated the same as DT_STRING when you send or retrieve information. The value is formatted as a character string in the current date format.

Embedded SQL Data Type	How the Length of the Data Area Is Specified Before Fetching a Value	How the Length of the Value Is Determined After Fetching a Value
DT_BIGINT	Fixed size. No action required.	Fixed size. No action required.
DT_BINARY(n)	The `sqllen` field is set to the maximum length in bytes of the `array` field of the BINARY structure plus the size of the len field (n+2). The maximum sqllen value is 32767.	The `len` field of the BINARY structure is updated to the actual length in bytes of data in the `array` field of the BINARY structure. The length will not exceed 32765.
DT_BIT	Fixed size. No action required.	Fixed size. No action required.
DT_DATE	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	A null character is placed at the end of the string.
DT_DOUBLE	No action required.	No action required.
DT_FIXCHAR(n)	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	The value is blank-padded to the `sqllen` value.
DT_FLOAT	Fixed size. No action required.	Fixed size. No action required.
DT_INT	Fixed size. No action required.	Fixed size. No action required.
DT_LONGBINARY	The `sqllen` field is ignored. The `array_len` field specifies the maximum length of the `array` field. More than 32767 bytes can be fetched.	The `stored_len` and `untrunc_len` fields are updated. The `stored_len` field contains the amount of data that was fetched. The `untrunc_len` field contains the amount of data that could be fetched.
DT_LONGNVARCHAR	The `sqllen` field is ignored. The `array_len` field specifies the maximum length of the `array` field. More than 32767 bytes can be fetched.	The `stored_len` and `untrunc_len` fields are updated. The `stored_len` field contains the amount of data that was fetched. The `untrunc_len` field contains the amount of data that could be fetched.
DT_LONGVARCHAR	The `sqllen` field is ignored. The `array_len` field specifies the maximum length of the `array` field. More than 32767 bytes can be fetched.	The `stored_len` and `untrunc_len` fields are updated. The `stored_len` field contains the amount of data that was fetched. The `untrunc_len` field contains the amount of data that could be fetched.

Embedded SQL Data Type	How the Length of the Data Area Is Specified Before Fetching a Value	How the Length of the Value Is Determined After Fetching a Value
DT_NFIXCHAR	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	The value is blank-padded to the `sqllen` value.
DT_NSTRING	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	The string is at most 32766 bytes long. A null character is placed at the end of the string.
DT_NVARCHAR(n)	The `sqllen` field is set to the maximum length in bytes of the `array` field of the NVARCHAR structure plus the size of the len field (n+2). The maximum sqllen value is 32767.	The `len` field of the NVARCHAR structure is updated to the actual length in bytes of data in the `array` field of the NVARCHAR structure. The length will not exceed 32765.
DT_SMALLINT	Fixed size. No action required.	Fixed size. No action required.
DT_STRING	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	The string is at most 32766 bytes long. A null character is placed at the end of the string.
DT_TIME	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	A null character is placed at the end of the string.
DT_TIMESTAMP	The `sqllen` field is set to the length in bytes of the `sqldata` area. The maximum length is 32767.	A null character is placed at the end of the string.
DT_TIMESTAMP_ STRUCT	Fixed size. No action required.	Fixed size. No action required.
DT_UNSBIGINT	Fixed size. No action required.	Fixed size. No action required.
DT_UNSINT	Fixed size. No action required.	Fixed size. No action required.
DT_UNSSMALLINT	Fixed size. No action required.	Fixed size. No action required.
DT_VARCHAR(n)	The `sqllen` field is set to the maximum length in bytes of the `array` field of the VARCHAR structure plus the size of the len field (n+2). The maximum sqllen value is 32767.	The `len` field of the VARCHAR structure is updated to the actual length in bytes of data in the `array` field. The length will not exceed 32765.

Related Information

Retrieving LONG Data Using Dynamic SQL [page 332]

1.10.15 How to Fetch Data Using Embedded SQL

Fetching data in Embedded SQL is done using the SELECT statement.

There are two cases:

The SELECT statement returns at most one row

Use an INTO clause to assign the returned values directly to host variables.

The SELECT statement may return multiple rows

Use cursors to manage the rows of the result set.

In this section:

SELECT Statements That Return at Most One Row [page 311]
A single row query retrieves at most one row from the database.

Cursors in Embedded SQL [page 312]
A cursor is used to retrieve rows from a query that has multiple rows in its result set.

Wide Fetches Using Embedded SQL [page 315]
The FETCH statement can be modified to fetch more than one row at a time, which may improve performance. This is called a **wide fetch** or an **array fetch**.

Related Information

FETCH Statement [ESQL] [SP]

1.10.15.1 SELECT Statements That Return at Most One Row

A single row query retrieves at most one row from the database.

A single-row query SELECT statement has an INTO clause following the SELECT list and before the FROM clause. The INTO clause contains a list of host variables to receive the value for each SELECT list item. There must be the same number of host variables as there are SELECT list items. The host variables may be accompanied by indicator variables to indicate NULL results.

When the SELECT statement is executed, the database server retrieves the results and places them in the host variables. If the query results contain more than one row, the database server returns an error.

If the query results in no rows being selected, an error is returned indicating that no rows can be found (SQLCODE 100). Errors and warnings are returned in the SQLCA structure.

Example

The following code fragment returns 1 if a row from the Employees table is fetched successfully, 0 if the row doesn't exist, and -1 if an error occurs.

```
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 */
  }
}
```

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.15.2 Cursors in Embedded SQL

A cursor is used to retrieve rows from a query that has multiple rows in its result set.

A cursor is a handle or an identifier for the SQL query and a position within the result set. Cursor management in Embedded SQL involves the following steps:

1. Declare a cursor for a particular SELECT statement, using the DECLARE CURSOR statement.
2. Open the cursor using the OPEN statement.
3. Retrieve results one row at a time from the cursor using the FETCH statement.
4. Fetch rows until the `Row Not Found` warning is returned.
   Errors and warnings are returned in the SQLCA structure.

5. Close the cursor, using the CLOSE statement.

By default, cursors are automatically closed at the end of a transaction (on COMMIT or ROLLBACK). Cursors that are opened with a WITH HOLD clause are kept open for subsequent transactions until they are explicitly closed.

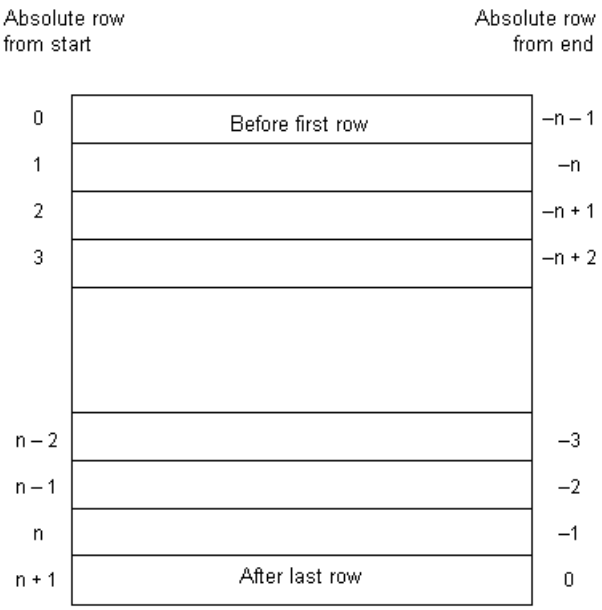
The following is a simple example of cursor usage:

```
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 Positioning

A cursor is positioned in one of three places:

- On a row
- Before the first row
- After the last row



When a cursor is opened, it is positioned before the first row. The cursor position can be moved using the FETCH statement. It can be positioned to an absolute position either from the start or from the end of the query results. It can also be moved relative to the current cursor position.

There are special **positioned** versions of the UPDATE and DELETE statements that can be used to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, an error is returned indicating that there is no corresponding row in the cursor.

The PUT statement can be used to insert a row into a cursor.

Cursor Positioning Problems

Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database server does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear until the cursor is closed and opened again.

This occurs if a temporary table had to be created to open the cursor.

The UPDATE statement can cause a row to move in the cursor. This happens if the cursor has an ORDER BY clause that uses an existing index (a temporary table is not created).

Related Information

Cursor Principles [page 19]
The SQL Communication Area (SQLCA) [page 290]
CLOSE statement [ESQL] [SP]

DECLARE CURSOR Statement [ESQL] [SP]
DELETE Statement (Positioned) [ESQL] [SP]
FETCH Statement [ESQL] [SP]
OPEN Statement [ESQL] [SP]
PUT Statement [ESQL]
UPDATE (Positioned) Statement [ESQL] [SP]
Static Cursor Sample [page 273]
Dynamic Cursor Sample [page 275]

1.10.15.3 Wide Fetches Using Embedded SQL

The FETCH statement can be modified to fetch more than one row at a time, which may improve performance. This is called a **wide fetch** or an **array fetch**.

Wide puts and inserts are also supported.

To use wide fetches in Embedded SQL, include the FETCH statement in your code as follows:

```
EXEC SQL FETCH ... ARRAY nnn
```

where ARRAY nnn is the last item of the FETCH statement. The fetch count nnn can be a host variable. The number of variables in the SQLDA must be the product of nnn and the number of columns per row. The first row is placed in SQLDA variables 0 to (columns per row) - 1, and so on.

Each column must be of the same type in each row of the SQLDA, or a SQLDA_INCONSISTENT error is returned.

The server returns in SQLCOUNT the number of records that were fetched, which is always greater than zero unless there is an error or warning. On a wide fetch, a SQLCOUNT of 1 with no error condition indicates that one valid row has been fetched.

Example

The following example code illustrates the use of wide fetches. The complete example is found in *%SQLANYSAMP17%*\SQLAnywhere\esqlwidefetch\widefetch.sqc.

```
static SQLDA * PrepareSQLDA(
        a_sql_statement_number  stat0,
        unsigned                width,
        unsigned                *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].sqllen = sqlda->sqlvar[col].sqllen;
            // 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 )
{
    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 )
{
    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_sqlda( sqlda );
err:
    return( SQLCODE );
}
int main( int argc, char *argv[] )
{
    int     result_code;

    argc = ProcessOptions( argv );
    if( argc < 0 ) {
        return( 1 );
    }
    db_init( &sqlca );
    db_string_connect( &sqlca, ConnectStr );
    if( SQLCODE != SQLE_NOERROR ) {
        PrintSQLError();
        goto err;
    }
    result_code = DoQuery( QueryStr, FetchWidth );
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( (result_code == 0) ? EXIT_OKAY : EXIT_FAIL );
err:
    db_fini( &sqlca );
    return( EXIT_FAIL );
}
```

Notes on Using Wide Fetches

- In the function PrepareSQLDA, the SQLDA memory is allocated using the alloc_sqlda function. This allows space for indicator variables, rather than using the alloc_sqlda_noind function.
- If the number of rows fetched is fewer than the number requested, but is not zero (at the end of the cursor for example), the SQLDA items corresponding to the rows that were not fetched are returned as NULL by setting the indicator value. If no indicator variables are present, an error is generated (SQLE_NO_INDICATOR: no indicator variable for NULL result).
- If a row being fetched has been updated, generating a SQLE_ROW_UPDATED_WARNING warning, the fetch stops on the row that caused the warning. The values for all rows processed to that point (including the row that caused the warning) are returned. SQLCOUNT contains the number of rows that were fetched, including the row that caused the warning. All remaining SQLDA items are marked as NULL.
- If a row being fetched has been deleted or is locked, generating a SQLE_NO_CURRENT_ROW or SQLE_LOCKED error, SQLCOUNT contains the number of rows that were read before the error. This does not include the row that caused the error. The SQLDA does not contain values for any of the rows since SQLDA values are not returned on errors. The SQLCOUNT value can be used to reposition the cursor, if necessary, to read the rows.

Related Information

EXECUTE Statement [ESQL]

FETCH Statement [ESQL] [SP]
PUT Statement [ESQL]
alloc_sqlda Function [page 342]
alloc_sqlda_noind Function [page 343]

1.10.16 Wide Inserts Using Embedded SQL

The INSERT statement can be used to insert more than one row at a time, which may improve performance. This is called a **wide insert** or an **array insert**.

To use wide inserts in Embedded SQL, prepare and then execute an INSERT statement in your code as follows:

```
EXEC SQL EXECUTE ... ARRAY nnn
```

where ARRAY nnn is the last part of the EXECUTE statement. The batch size nnn can be a host variable. The number of variables in the SQLDA must be the product of the batch size and the number of placeholders in the statement to be executed.

Each variable must be of the same type in each row of the SQLDA, or a SQLDA_INCONSISTENT error is returned.

Example

The following complete code example illustrates the use of wide inserts.

```
// [wideinsert.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 ) ) );
}
unsigned        RowsToInsert = 20;
unsigned short  BatchSize    = 5;
char *          ConnectStr   = "";
static void Usage()
{
    fprintf( stderr, "Usage: wideinsert [options] \n" );
    fprintf( stderr, "Options:\n" );
    fprintf( stderr, "   -n nnn          : number of rows to insert (default:
20)\n" );
    fprintf( stderr, "   -b nnn          : insert nnn rows at a time (default:
5)\n" );
    fprintf( stderr, "   -c conn_str     : database connection string (required)
\n" );
```

```
}
static int ArgumentIsASwitch( char * arg )
{
#if defined( UNIX )
    return ( arg[0] == '-' );
#else
    return ( arg[0] == '-' ) || ( arg[0] == '/' );
#endif
}
static int ProcessOptions( char * argv[] )
{
    int             argc;
    char *          arg;
    char            opt;
#define _get_arg_param()                                                \
            arg += 2;                                                   \
            if( !arg[0] ) arg = argv[++argc];                           \
            if( arg == NULL )                                           \
            {                                                           \
                fprintf( stderr, "Missing argument parameter\n" );      \
                return( -1 );                                           \
            }
    for( argc = 1; (arg = argv[argc]) != NULL; ++ argc )
    {
        if( !ArgumentIsASwitch( arg ) ) break;
        opt = arg[1];
        switch( opt )
        {
        case 'n':
            _get_arg_param();
            RowsToInsert = atol( arg );
            break;
        case 'b':
            _get_arg_param();
            BatchSize = (unsigned short) atol( arg );
            break;
        case 'c':
            _get_arg_param();
            ConnectStr = arg;
            break;
        default:
            fprintf( stderr, "**** Unknown option: -%c\n", opt );
            Usage();
            return( -1 );
        }
    }
    if( ConnectStr[0] == '\0' )
    {
        fprintf( stderr, "A database connection string is required.\n" );
        Usage();
        return( -1 );
    }
    return( argc );
}
static int DoInsert( unsigned rows_to_insert, unsigned short batch_size )
{
    SQLDA                   *sqlda;
    SQLVAR                  *var;
    int                     row_number = 1;

    EXEC SQL BEGIN DECLARE SECTION;
    char                    insert_stmt[ 100 ];
    unsigned short          array_size;
    unsigned                row_count;
    EXEC SQL END DECLARE SECTION;
    // Create the table for inserting the rows
    EXEC SQL DROP TABLE IF EXISTS WideInsertSample;
    EXEC SQL CREATE TABLE WideInsertSample( Col1 int, Col2 char(20), Col3 int );
```

```
    #define NUM_PARAMS 3
    sprintf( insert_stmt, "INSERT INTO WideInsertSample( Col1, Col2, Col3 )
VALUES( ?, ?, ? )" );

    sqlda = alloc_sqlda_noind( batch_size * NUM_PARAMS );
    if( sqlda == NULL )
    {
        printf( "Error allocating SQLDA\n" );
        return( SQLE_NO_MEMORY );
    }

    // Prepare the static parts of the SQLDA object for the insert
    sqlda->sqld = batch_size * NUM_PARAMS;
    for( unsigned short current_row = 0; current_row < batch_size; current_row+
+ )
    {
        var = &sqlda->sqlvar[ current_row * NUM_PARAMS + 0 ];
        var->sqltype = DT_INT;
        var->sqllen = sizeof( int );
        var = &sqlda->sqlvar[ current_row * NUM_PARAMS + 1 ];
        var->sqltype = DT_STRING;
        var->sqllen = 30;
        var = &sqlda->sqlvar[ current_row * NUM_PARAMS + 2 ];
        var->sqltype = DT_INT;
        var->sqllen = sizeof( int );
   }

    fill_sqlda( sqlda );

    printf( "Insert %u rows into table \"WideInsertSample\" with batch size
%u.\n",
        rows_to_insert,
        batch_size );

    EXEC SQL PREPARE wide_insert_stmt FROM :insert_stmt;
    while( rows_to_insert > 0 )
    {
        if( rows_to_insert > batch_size )
        {
            array_size = batch_size;
        }
        else
        {
            array_size = (unsigned short) rows_to_insert;
        }
        // Fill in data values
        for( unsigned short current_row = 0; current_row < array_size;
current_row++ )
        {
            *(int *)sqlda->sqlvar[ current_row * NUM_PARAMS + 0 ].sqldata =
row_number;
            sprintf( (char *)sqlda->sqlvar[ current_row * NUM_PARAMS +
1 ].sqldata,
                     "This is row #%u", row_number );
            *(int *)sqlda->sqlvar[ current_row * NUM_PARAMS + 2 ].sqldata =
row_number * 2;
            row_number++;
        }
        // Insert a batch of rows
        EXEC SQL EXECUTE wide_insert_stmt USING DESCRIPTOR sqlda
ARRAY :array_size;
        printf( "Inserted %u rows; ", SQLCOUNT );

        EXEC SQL SELECT COUNT(*) INTO :row_count FROM WideInsertSample;
        printf( "total %u rows in table.\n", row_count );

        rows_to_insert -= array_size;
    }
```

```
    printf( "Done.\n" );

    EXEC SQL COMMIT;
    EXEC SQL DROP STATEMENT wide_insert_stmt;
    free_filled_sqlda( sqlda );
err:
    return( SQLCODE );
}
int main( int argc, char *argv[] )
{
    int     result_code;

    argc = ProcessOptions( argv );
    if( argc < 0 )
    {
        return( 1 );
    }
    db_init( &sqlca );
    db_string_connect( &sqlca, ConnectStr );
    if( SQLCODE != SQLE_NOERROR )
    {
        PrintSQLError();
        goto err;
    }
    result_code = DoInsert( RowsToInsert, BatchSize );
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( (result_code == 0) ? EXIT_OKAY : EXIT_FAIL );
err:
    db_fini( &sqlca );
    return( EXIT_FAIL );
}
```

Notes on Using Wide Inserts

- The size of the SQLDA is based on the size of the batch (the maximum number of rows that you want to insert at a time) multiplied by the number of parameters or columns that you want to insert (NUM_PARAMS in the following examples). Memory for the SQLDA is allocated using the alloc_sqlda_noind function. This function does not allocate space for indicator variables.

  ```
  sqlda = alloc_sqlda_noind( batch_size * NUM_PARAMS );
  ```

- The entire SQLDA is initialized one row and column at a time. In general, the position in the SQLDA for each row and column is calculated using zero-based offsets as in the following example:

  ```
  var = &sqlda->sqlvar[ row_index * num_params + parameter_index ]
  var->sqltype = column_type;
  var->sqllen = column_size;
  ```

- Once this has been done, the fill_sqlda routine can be called to allocate buffers for the column values in all rows of the batch. Values are stuffed into the buffers using zero-based offsets prior to executing the prepared INSERT statement. The following is an example for storing an integer value.

  ```
  *(int *)sqlda->sqlvar[ current_row * NUM_PARAMS + current_column ].sqldata =
  row_number * 2;
  ```

- The prepared INSERT statement is executed using the EXEC SQL EXECUTE statement and the number of rows to insert is specified by the ARRAY clause. The following is an example.

  ```
  EXEC SQL EXECUTE wide_insert_stmt USING DESCRIPTOR sqlda ARRAY :array_size;
  ```

- The number of rows that were inserted is returned in SQLCOUNT, which is always greater than zero unless there is an error or warning.

  ```
  printf( "Inserted %u rows; ", SQLCOUNT );
  ```

1.10.17 Wide Deletes Using Embedded SQL

The DELETE statement can be used to delete an arbitrary set of rows, which may improve performance. This is called a **wide delete** or an **array delete**.

To use wide deletes in Embedded SQL, prepare and then execute a DELETE statement in your code as follows:

```
EXEC SQL EXECUTE ... ARRAY nnn
```

where ARRAY nnn is the last part of the EXECUTE statement. The batch size nnn can be a host variable. The number of variables in the SQLDA must be the product of the batch size and the number of placeholders in the statement to be executed.

Each variable must be of the same type in each row of the SQLDA, or a SQLDA_INCONSISTENT error is returned.

Example

The following complete code example illustrates the use of wide deletes.

```
// [widedelete.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 ) ) );
}
unsigned        RowsToDelete[100];
unsigned short  BatchSize    = 0;
char *          ConnectStr   = "";
static void Usage()
{
    fprintf( stderr, "Usage: widedelete [options] \n" );
    fprintf( stderr, "Options:\n" );
```

```
    fprintf( stderr, "   -n nnn          : row number to delete (specify up to
100 -n arguments)\n" );
    fprintf( stderr, "   -c conn_str     : database connection string (required)
\n" );
}
static int ArgumentIsASwitch( char * arg )
{
#if defined( UNIX )
    return ( arg[0] == '-' );
#else
    return ( arg[0] == '-' ) || ( arg[0] == '/' );
#endif
}
static int ProcessOptions( char * argv[] )
{
    int             argc;
    char *          arg;
    char            opt;
#define _get_arg_param()                                                \
            arg += 2;                                                   \
            if( !arg[0] ) arg = argv[++argc];                           \
            if( arg == NULL )                                           \
            {                                                           \
                fprintf( stderr, "Missing argument parameter\n" );      \
                return( -1 );                                           \
            }
    for( argc = 1; (arg = argv[argc]) != NULL; ++ argc )
    {
        if( !ArgumentIsASwitch( arg ) ) break;
        opt = arg[1];
        switch( opt )
        {
        case 'n':
            _get_arg_param();
            RowsToDelete[BatchSize++] = atol( arg );
            break;
        case 'c':
            _get_arg_param();
            ConnectStr = arg;
            break;
        default:
            fprintf( stderr, "**** Unknown option: -%c\n", opt );
            Usage();
            return( -1 );
        }
    }
    if( ConnectStr[0] == '\0' )
    {
        fprintf( stderr, "A database connection string is required.\n" );
        Usage();
        return( -1 );
    }
    return( argc );
}
static int DoDelete( unsigned *rows_to_delete, unsigned short batch_size )
{
    SQLDA                   *sqlda;
    SQLVAR                  *var;

    EXEC SQL BEGIN DECLARE SECTION;
    char                    delete_stmt[ 100 ];
    unsigned short          array_size;
    unsigned                row_count;
    EXEC SQL END DECLARE SECTION;
    sprintf( delete_stmt, "DELETE FROM WideInsertSample WHERE Col1 = ?" );

    sqlda = alloc_sqlda_noind( batch_size );
    if( sqlda == NULL )
```

```
    {
        printf( "Error allocating SQLDA\n" );
        return( SQLE_NO_MEMORY );
    }

    // Prepare the static parts of the SQLDA object for the delete
    sqlda->sqld = batch_size;
    for( unsigned short current_row = 0; current_row < batch_size; current_row+
+ )
    {
        var = &sqlda->sqlvar[ current_row ];
        var->sqltype = DT_INT;
        var->sqllen = sizeof( int );
    }

    fill_sqlda( sqlda );

    printf( "Delete %u rows from table \"WideInsertSample\".\n", batch_size );

    EXEC SQL PREPARE wide_delete_stmt FROM :delete_stmt;
    array_size = batch_size;
    // Fill in data values
    for( unsigned short current_row = 0; current_row < array_size; current_row+
+ )
    {
        *(int *)sqlda->sqlvar[ current_row ].sqldata =
rows_to_delete[ current_row ];
    }
    // Delete a batch of rows
    EXEC SQL EXECUTE wide_delete_stmt USING DESCRIPTOR sqlda ARRAY :array_size;
    printf( "Deleted %u rows; ", SQLCOUNT );

    EXEC SQL SELECT COUNT(*) INTO :row_count FROM WideInsertSample;
    printf( "total %u rows in table.\n", row_count );

    EXEC SQL COMMIT;
    EXEC SQL DROP STATEMENT wide_delete_stmt;
    free_filled_sqlda( sqlda );
err:
    return( SQLCODE );
}
int main( int argc, char *argv[] )
{
    int     result_code;

    argc = ProcessOptions( argv );
    if( argc < 0 )
    {
        return( 1 );
    }
    db_init( &sqlca );
    db_string_connect( &sqlca, ConnectStr );
    if( SQLCODE != SQLE_NOERROR )
    {
        PrintSQLError();
        goto err;
    }
    result_code = DoDelete( RowsToDelete, BatchSize );
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( (result_code == 0) ? EXIT_OKAY : EXIT_FAIL );
err:
    db_fini( &sqlca );
    return( EXIT_FAIL );
}
```

Notes on Using Wide Deletes

- To try this example, use the example in the wide inserts topic to populate the WideInsertSample table.
- The size of the SQLDA is based on the size of the batch (the maximum number of rows that you want to delete at a time) multiplied by the number of parameters in the DELETE statement. In this example, there is only 1 parameter but you could have more. Memory for the SQLDA is allocated using the alloc_sqlda_noind function. This function does not allocate space for indicator variables.

  ```
  sqlda = alloc_sqlda_noind( batch_size );
  ```

- The entire SQLDA is initialized one row and parameter at a time. In general, the position in the SQLDA for each row and parameter is calculated using zero-based offsets as in the following example (for this DELETE example, `num_params` is 1):

  ```
  var = &sqlda->sqlvar[ row_index * num_params + parameter_index ]
  var->sqltype = column_type;
  var->sqllen = column_size;
  ```

- Once this has been done, the fill_sqlda routine can be called to allocate buffers for the parameter values in all rows of the batch. Values are stuffed into the buffers using zero-based offsets prior to executing the prepared DELETE statement. The following is an example for storing the integer row number.

  ```
  *(int *)sqlda->sqlvar[ current_row ].sqldata = rows_to_delete[ current_row ];
  ```

- The prepared DELETE statement is executed using the EXEC SQL EXECUTE statement and the number of rows to delete is specified by the ARRAY clause. The following is an example.

  ```
  EXEC SQL EXECUTE wide_delete_stmt USING DESCRIPTOR sqlda ARRAY :array_size;
  ```

- The number of rows that were deleted is returned in SQLCOUNT, which is always greater than zero unless no row matched any of the specified rows (for example, the rows were already deleted) or there is an error or warning.

  ```
  printf( "Deleted %u rows; ", SQLCOUNT );
  ```

1.10.18 Wide Merges Using Embedded SQL

The MERGE statement can be used to merge multiple sets of rows into a table, which may improve performance. This is called a **wide merge** or an **array merge**.

To use wide merges in Embedded SQL, prepare and then execute a MERGE statement in your code as follows:

```
EXEC SQL EXECUTE ... ARRAY nnn
```

where ARRAY `nnn` is the last part of the EXECUTE statement. The batch size `nnn` can be a host variable. The number of variables in the SQLDA must be the product of the batch size and the number of placeholders in the statement to be executed.

Each variable must be of the same type in each row of the SQLDA, or a SQLDA_INCONSISTENT error is returned.

Example

The following complete code example illustrates a wide merge.

```
// [widemerge.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 ) ) );
}
char *          ConnectStr   = "";
static void Usage()
{
    fprintf( stderr, "Usage: widemerge [options] \n" );
    fprintf( stderr, "Options:\n" );
    fprintf( stderr, "   -c conn_str     : database connection string (required)
\n" );
}
static int ArgumentIsASwitch( char * arg )
{
#if defined( UNIX )
    return ( arg[0] == '-' );
#else
    return ( arg[0] == '-' ) || ( arg[0] == '/' );
#endif
}
static int ProcessOptions( char * argv[] )
{
    int             argc;
    char *          arg;
    char            opt;
#define _get_arg_param()                                                \
            arg += 2;                                                   \
            if( !arg[0] ) arg = argv[++argc];                           \
            if( arg == NULL )                                           \
            {                                                           \
                fprintf( stderr, "Missing argument parameter\n" );      \
                return( -1 );                                           \
            }
    for( argc = 1; (arg = argv[argc]) != NULL; ++ argc )
    {
        if( !ArgumentIsASwitch( arg ) ) break;
        opt = arg[1];
        switch( opt )
        {
        case 'c':
            _get_arg_param();
            ConnectStr = arg;
            break;
        default:
            fprintf( stderr, "**** Unknown option: -%c\n", opt );
            Usage();
            return( -1 );
        }
    }
```

```
    if( ConnectStr[0] == '\0' )
    {
        fprintf( stderr, "A database connection string is required.\n" );
        Usage();
        return( -1 );
    }
    return( argc );
}
static int DoMerge()
{
    SQLDA                   *sqlda;
    SQLVAR                  *var;
    char                    *region_id;
    unsigned                rep_id;

    EXEC SQL BEGIN DECLARE SECTION;
    char                    merge_stmt[ 200 ];
    unsigned short          batch_size;
    unsigned                row_count;
    EXEC SQL END DECLARE SECTION;
    // Create the table for inserting/merging the rows
    EXEC SQL CREATE OR REPLACE TABLE LocalSalesOrders
            (
                ID                  integer NOT NULL,
                CustomerID          integer NOT NULL,
                OrderDate           date NOT NULL,
                FinancialCode       char(2) NULL,
                Region              char(7) NULL,
                SalesRepresentative integer NOT NULL,
                CONSTRAINT SalesOrdersKey PRIMARY KEY (ID)
            );
    strcpy( merge_stmt,
            "MERGE INTO LocalSalesOrders "
            "USING WITH AUTO NAME"
            "("
            "    SELECT * FROM SalesOrders"
            "    WHERE Region = ? AND SalesRepresentative = ?"
            ") AS line_items "
            "ON PRIMARY KEY "
            "WHEN NOT MATCHED THEN INSERT" );
    #define NUM_PARAMS 2

    batch_size = 4;

    sqlda = alloc_sqlda_noind( batch_size * NUM_PARAMS );
    if( sqlda == NULL )
    {
        printf( "Error allocating SQLDA\n" );
        return( SQLE_NO_MEMORY );
    }
    // Prepare the static parts of the SQLDA object for the merge
    sqlda->sqld = batch_size * NUM_PARAMS;
    for( unsigned short current_row = 0; current_row < batch_size; current_row+
+ )
    {
        var = &sqlda->sqlvar[ current_row * NUM_PARAMS + 0 ];
        var->sqltype = DT_STRING;
        var->sqllen = 10;
        var = &sqlda->sqlvar[ current_row * NUM_PARAMS + 1 ];
        var->sqltype = DT_INT;
        var->sqllen = sizeof( int );
    }

    fill_sqlda( sqlda );

    printf( "Merge %u rowsets into table.\n", batch_size );

    EXEC SQL PREPARE wide_merge_stmt FROM :merge_stmt;
```

```
    // Fill in data values
    for( unsigned short current_row = 0; current_row < batch_size; current_row+
+ )
    {
        switch( current_row % 4 ) {
        case 0:
            region_id = "Eastern";
            rep_id = 299;
            break;
        case 1:
            region_id = "Western";
            rep_id = 299;
            break;
        case 2:
            region_id = "Eastern";
            rep_id = 856;
            break;
        case 3:
            region_id = "Western";
            rep_id = 856;
            break;
        }
        sprintf( (char *)sqlda->sqlvar[ current_row * NUM_PARAMS + 0 ].sqldata,
region_id );
                *(int *)sqlda->sqlvar[ current_row * NUM_PARAMS + 1 ].sqldata =
rep_id;
    }
    // Merge batches of rowsets
    EXEC SQL EXECUTE wide_merge_stmt USING DESCRIPTOR sqlda ARRAY :batch_size;
    printf( "Merged %u rows; ", SQLCOUNT );
    // Verify count
    EXEC SQL SELECT COUNT(*) INTO :row_count FROM LocalSalesOrders;
    printf( "total %u rows in table.\n", row_count );

    EXEC SQL COMMIT;
    EXEC SQL DROP STATEMENT wide_merge_stmt;
    free_filled_sqlda( sqlda );
err:
    return( SQLCODE );
}
int main( int argc, char *argv[] )
{
    int     result_code;

    argc = ProcessOptions( argv );
    if( argc < 0 )
    {
        return( 1 );
    }
    db_init( &sqlca );
    db_string_connect( &sqlca, ConnectStr );
    if( SQLCODE != SQLE_NOERROR )
    {
        PrintSQLError();
        goto err;
    }
    result_code = DoMerge();
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( (result_code == 0) ? EXIT_OKAY : EXIT_FAIL );
err:
    db_fini( &sqlca );
    return( EXIT_FAIL );
}
```

Notes on Using Wide Merges

- To try this example, use the sample database.
- The size of the SQLDA is based on the size of the batch (the maximum number of merges) multiplied by the number of placeholder parameters in the MERGE statement. In this example, there are 2 parameters. Memory for the SQLDA is allocated using the alloc_sqlda_noind function. This function does not allocate space for indicator variables.

```
sqlda = alloc_sqlda_noind( batch_size * NUM_PARAMS );
```

- The entire SQLDA is initialized one row and parameter at a time. In general, the position in the SQLDA for each row and parameter is calculated using zero-based offsets as in the following example:

```
var = &sqlda->sqlvar[ row_index * num_params + parameter_index ];
var->sqltype = column_type;
var->sqllen = column_size;
```

- Once this has been done, the fill_sqlda routine can be called to allocate buffers for the parameter values in all rows of the batch. Values are stuffed into the buffers using zero-based offsets prior to executing the prepared MERGE statement. The following is an example for storing the region string and the representative number for a given row.

```
sprintf( (char *)sqlda->sqlvar[ current_row * NUM_PARAMS + 0 ].sqldata,
region_id );
         *(int *)sqlda->sqlvar[ current_row * NUM_PARAMS + 1 ].sqldata =
rep_id;
```

- The prepared MERGE statement is executed using the EXEC SQL EXECUTE statement and the size of the batch is specified by the ARRAY clause. The following is an example.

```
EXEC SQL EXECUTE wide_merge_stmt USING DESCRIPTOR sqlda ARRAY :batch_size;
```

- The number of rows that were affected is returned in SQLCOUNT, which is always greater than zero unless no row matched any of the specified rows (for example, the rows were already present) or there is an error or warning.

```
printf( "Merged %u rows; ", SQLCOUNT );
```

1.10.19 How to Send and Retrieve Long Values Using Embedded SQL

The method for sending and retrieving LONG VARCHAR, LONG NVARCHAR, and LONG BINARY values in Embedded SQL applications is different from that for other data types.

The standard SQLDA fields are limited to 32767 bytes of data as the fields holding the length information (sqllen, *sqlind) are 16-bit values. Changing these values to 32-bit values would break existing applications.

The method of describing LONG VARCHAR, LONG NVARCHAR, and LONG BINARY values is the same as for other data types.

Static SQL Structures

Separate fields are used to hold the allocated, stored, and untruncated lengths of LONG BINARY, LONG VARCHAR, and LONG NVARCHAR data types. The static SQL data types are defined in `sqlca.h` as follows:

```
#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];  \
         }
```

The `size+1` allocation does not indicate that the LONGVARCHAR/LONGNVARCHAR `array` is null-terminated by the client library. The extra byte is included for those applications that wish to null-terminate the chunk that has been fetched from the database. Use the `stored_len` field to determine the amount of data fetched.

When any of these macros are used in an Embedded SQL DECLARE SECTION, the `array_len` field is initialized to `size`. Otherwise, the `array_len` field is not initialized.

Dynamic SQL Structures

For dynamic SQL, set the sqltype field to DT_LONGVARCHAR, DT_LONGNVARCHAR, or DT_LONGBINARY as appropriate. The associated LONGVARCHAR, LONGNVARCHAR, and LONGBINARY structure is as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32    array_len;
    a_sql_uint32    stored_len;
    a_sql_uint32    untrunc_len;
    char            array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

Structure Member Definitions

For both static and dynamic SQL structures, the structure members are defined as follows:

array_len

(Sending and retrieving.) The number of bytes allocated for the array part of the structure.

stored_len

(Sending and retrieving.) The number of bytes stored in the array. Always less than or equal to array_len and untrunc_len.

untrunc_len

(Retrieving only.) The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to stored_len. If truncation occurs, this value is larger than array_len.

In this section:

Retrieving LONG Data Using Static SQL [page 331]
Retrieve a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value using static SQL.

Retrieving LONG Data Using Dynamic SQL [page 332]
Retrieve a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value using dynamic SQL.

Sending LONG Data Using Static SQL [page 334]
Send LONG values to the database using static SQL from an Embedded SQL application.

Sending LONG Data Using Dynamic SQL [page 335]
Send LONG values to the database using dynamic SQL from an Embedded SQL application.

1.10.19.1 Retrieving LONG Data Using Static SQL

Retrieve a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value using static SQL.

Procedure

1. Declare a host variable of type DECL_LONGVARCHAR, DECL_LONGNVARCHAR, or DECL_LONGBINARY, as appropriate. The array_len field is filled in automatically.
2. Retrieve the data using FETCH, GET DATA, or EXECUTE INTO. The following information is set:

   sqlind

   The sqlind field points to an indicator. The indicator value is negative if the value is NULL, 0 if there is no truncation, and is the positive untruncated length in bytes up to a maximum of 32767. If the indicator value is positive, use the untrunc_len field instead.

   stored_len

   The number of bytes stored in the array. Always less than or equal to array_len and untrunc_len.

   untrunc_len

   The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to stored_len. If truncation occurs, this value is larger than array_len.

   array

   This area contains the data fetched. The data is not null-terminated.

Results

The LONG data is retrieved using static SQL.

Example

The following code fragment illustrates the mechanics of retrieving a LONG VARCHAR using static Embedded SQL. It is not intended to be a practical application.

```
EXEC SQL BEGIN DECLARE SECTION;
// SQLPP initializes longdata.array_len to 128K
DECL_LONGVARCHAR(131072) longdata;
EXEC SQL END DECLARE SECTION;
// Init longdata for fetching data, not required
// since these fields are updated by FETCH
longdata.stored_len = 0;
longdata.untrunc_len = 0;
memset( longdata.array, 0, 131072 );
EXEC SQL FETCH ABSOLUTE 1 c1 INTO :longdata;
printf( "Length fetched %d, Actual length %d, Data[0..19] \"%20.20s\"\n",
    longdata.stored_len, longdata.untrunc_len, longdata.array );
```

Related Information

Embedded SQL Indicator Variables [page 287]

1.10.19.2 Retrieving LONG Data Using Dynamic SQL

Retrieve a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value using dynamic SQL.

Procedure

1. Set the sqltype field to DT_LONGVARCHAR, DT_LONGNVARCHAR, or DT_LONGBINARY as appropriate.
2. Set the sqldata field to point to the LONGVARCHAR, LONGNVARCHAR, or LONGBINARY host variable structure.

   You can use the LONGVARCHARSIZE(n), LONGNVARCHARSIZE(n), or LONGBINARYSIZE(n) macro to determine the total number of bytes to allocate to hold n bytes of data in the array field.
3. Set the array_len field of the host variable structure to the number of bytes allocated for the array field.
4. Retrieve the data using FETCH, GET DATA, or EXECUTE INTO. The following information is set:

   sqlind

The sqlind field points to an indicator. The indicator value is negative if the value is NULL, 0 if there is no truncation, and is the positive untruncated length in bytes up to a maximum of 32767. If the indicator value is positive, use the untrunc_len field instead.

stored_len

The number of bytes stored in the array. Always less than or equal to array_len and untrunc_len.

untrunc_len

The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to stored_len. If truncation occurs, this value is larger than array_len.

array

This area contains the data fetched. The data is not null-terminated.

Results

The LONG data is retrieved using dynamic SQL.

Example

The following code fragment illustrates the mechanics of retrieving LONG VARCHAR data using dynamic Embedded SQL. It is not intended to be a practical application:

```
#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
  // (sqllen is unused with DT_LONG types)
  sqlda->sqld = 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 );
}
```

Related Information

Embedded SQL Indicator Variables [page 287]

1.10.19.3 Sending LONG Data Using Static SQL

Send LONG values to the database using static SQL from an Embedded SQL application.

Procedure

1. Declare a host variable of type DECL_LONGVARCHAR, DECL_LONGNVARCHAR, or DECL_LONGBINARY, as appropriate.
2. If you are sending NULL, set the indicator variable to a negative value.
3. Set the stored_len field of the host variable structure to the number of bytes of data in the array field.
4. Send the data by opening the cursor or executing the statement.

Results

The Embedded SQL application is ready to send LONG values to the database.

Example

The following code fragment illustrates the mechanics of sending a LONG VARCHAR using static Embedded SQL. It is not intended to be a practical application.

```
#define CURRENT_LEN (64*1024)
EXEC SQL BEGIN DECLARE SECTION;
// SQLPP initializes longdata.array_len to 128K
DECL_LONGVARCHAR(131072) longdata;
EXEC SQL END DECLARE SECTION;
void set_test_var()
{
  // init longdata for sending data
  longdata.stored_len = CURRENT_LEN;
  memset( longdata.array, 'a', CURRENT_LEN );
  printf( "Setting test_var to %d a's\n", CURRENT_LEN );
  EXEC SQL SET test_var = :longdata;
```

```
}
```

Related Information

Embedded SQL Indicator Variables [page 287]

1.10.19.4 Sending LONG Data Using Dynamic SQL

Send LONG values to the database using dynamic SQL from an Embedded SQL application.

Procedure

1. Set the sqltype field to DT_LONGVARCHAR, DT_LONGNVARCHAR, or DT_LONGBINARY, as appropriate.
2. If you are sending NULL, set `* sqlind` to a negative value.
3. If you are not sending NULL, set the sqldata field to point to the LONGVARCHAR, LONGNVARCHAR, or LONGBINARY host variable structure.

   You can use the LONGVARCHARSIZE(n), LONGNVARCHARSIZE(n), or LONGBINARYSIZE(n) macros to determine the total number of bytes to allocate to hold n bytes of data in the array field.
4. Set the array_len field of the host variable structure to the number of bytes allocated for the array field.
5. Set the stored_len field of the host variable structure to the number of bytes of data in the array field. This must not be more than array_len.
6. Send the data by opening the cursor or executing the statement.

Results

The Embedded SQL application is ready to send LONG values to the database.

Related Information

Embedded SQL Indicator Variables [page 287]

1.10.20 Simple Stored Procedures in Embedded SQL

You can create and call stored procedures using Embedded SQL.

You can embed a CREATE PROCEDURE just like any other data definition statement, such as CREATE TABLE. You can also embed a CALL statement to execute a stored procedure. The following code fragment illustrates both creating and executing a stored procedure 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 );
```

To pass host variable values to a stored procedure or to retrieve the output variables, you prepare and execute a CALL statement. The following code fragment illustrates the use of host variables. Both the USING and INTO clauses are used on the EXECUTE statement.

```
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=passwd';
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;
```

In this section:

Stored Procedures with Result Sets [page 337]
Database procedures can also contain SELECT statements. The procedure is declared using a RESULT clause to specify the number, name, and types of the columns in the result set.

Related Information

CREATE PROCEDURE Statement

EXECUTE Statement [ESQL]
PREPARE Statement [ESQL]

1.10.20.1 Stored Procedures with Result Sets

Database procedures can also contain SELECT statements. The procedure is declared using a RESULT clause to specify the number, name, and types of the columns in the result set.

Result set columns are different from output parameters. For procedures with result sets, the CALL statement can be used in place of a SELECT statement in the cursor declaration.

```
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 this example, the procedure has been invoked with an OPEN statement rather than an EXECUTE statement. The OPEN statement causes the procedure to execute until it reaches a SELECT statement. At this point, C1 is a cursor for the SELECT statement within the database procedure. You can use all forms of the FETCH statement (backward and forward scrolling) until you are finished with it. The CLOSE statement stops execution of the procedure.

If there had been another statement following the SELECT in the procedure, it would not have been executed. To execute statements following a SELECT, use the RESUME cursor-name statement. The RESUME statement either returns the warning SQLE_PROCEDURE_COMPLETE or it returns SQLE_NOERROR indicating that there is another cursor. The example illustrates a two-select procedure:

```
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;
```

Dynamic Cursors for CALL Statements

These examples have used static cursors. Full dynamic cursors can also be used for the CALL statement.

The DESCRIBE statement works fully for procedure calls. A DESCRIBE OUTPUT produces a SQLDA that has a description for each of the result set columns.

If the procedure does not have a result set, the SQLDA has a description for each INOUT or OUT parameter for the procedure. A DESCRIBE INPUT statement produces a SQLDA having a description for each IN or INOUT parameter for the procedure.

DESCRIBE ALL

DESCRIBE ALL describes IN, INOUT, OUT, and RESULT set parameters. DESCRIBE ALL uses the indicator variables in the SQLDA to provide additional information.

The DT_PROCEDURE_IN and DT_PROCEDURE_OUT bits are set in the indicator variable when a CALL statement is described. DT_PROCEDURE_IN indicates an IN or INOUT parameter and DT_PROCEDURE_OUT indicates an INOUT or OUT parameter. Procedure RESULT columns have both bits clear.

After a DESCRIBE OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE).

Multiple result sets

If you have a procedure that returns multiple result sets, you must re-describe after each RESUME statement if the result sets change shapes.

Describe the cursor, not the statement, to re-describe the current position of the cursor.

Related Information

Dynamic SELECT Statement [page 298]
CREATE PROCEDURE Statement
DESCRIBE Statement [ESQL]

1.10.21 Request Management with Embedded SQL

Since a typical Embedded SQL application must wait for the completion of each database request before carrying out the next step, an application that uses multiple execution threads can carry on with other tasks.

If you must use a single execution thread, then some degree of multitasking can be accomplished by registering a callback function using the db_register_a_callback function with the DB_CALLBACK_WAIT option. Your callback function is called repeatedly by the interface library while the database server or client library is busy processing your database request.

In your callback function, you cannot start another database request but you can cancel the current request using the db_cancel_request function. You can use the db_is_working function in your message handlers to determine if you have a database request in progress.

Related Information

db_register_a_callback Function [page 360]
db_cancel_request Function [page 348]
db_is_working Function [page 356]

1.10.22 Database Backup with Embedded SQL

The recommended way to backup a database is to use the BACKUP statement.

The db_backup function provides another way to perform an online backup in Embedded SQL applications. The Backup utility also makes use of this function.

You can also interface directly to the Backup utility using the Database Tools DBBackup function.

Undertake to write a program using the db_backup function only if your backup requirements are not satisfied by the any of the other backup methods.

Related Information

BACKUP DATABASE Statement
Backup Utility (dbbackup)
db_backup Function [page 344]

1.10.23 Library Function Reference

The Embedded SQL preprocessor generates calls to functions in the interface library or DLL. In addition to the calls generated by the Embedded SQL preprocessor, a set of library functions is provided to make database operations easier to perform.

Prototypes for these functions are included by the EXEC SQL INCLUDE SQLCA statement.

DLL Entry Points

The DLL entry points are the same except that the prototypes have a modifier appropriate for DLLs.

You can declare the entry points in a portable manner using _esqlentry_, which is defined in `sqlca.h`. On Windows platforms, it resolves to the value __stdcall.

In this section:

alloc_sqlda Function [page 342]
Allocate a SQLDA.

alloc_sqlda_noind Function [page 343]
Allocate a SQLDA with no space for indicator variables.

db_backup Function [page 344]
Back up a database.

db_cancel_request Function [page 348]
Cancel an active request.

db_change_char_charset Function [page 349]
Change the application's CHAR character set for this connection.

db_change_nchar_charset Function [page 350]
Change the application's NCHAR character set for this connection.

db_find_engine Function [page 351]
Return status information about a local database server.

db_fini Function [page 352]
Free resources used by the database interface or DLL.

db_get_property Function [page 353]
Obtain information about the database interface or the server to which you are connected.

db_init Function [page 355]
Initialize the database interface library.

db_is_working Function [page 356]
Indicates whether a database request is in progress.

db_locate_servers Function [page 356]
Obtain information about all database servers on the local network that are listening on TCP/IP.

db_locate_servers_ex Function [page 358]
Obtain information about all database servers on the local network that are listening on TCP/IP.

db_register_a_callback Function [page 360]
Register a callback function.

db_start_database Function [page 364]
Start a database on a server.

db_start_engine Function [page 365]
Start a database server.

db_stop_database Function [page 367]
Stop a database on a server.

db_stop_engine Function [page 368]
Stop a database server.

db_string_connect Function [page 369]
Connect to a database on a database server.

db_string_disconnect Function [page 370]
Disconnect the current or other connection from a database on a database server.

db_string_ping_server Function [page 371]
Determine if a server can be located, and optionally, if it a successful connection to a database can be made.

db_time_change Function [page 372]
Notify the server that the time has changed on the client.

DBAlloc Function [page 373]
Allocate memory for a SQLDA variable.

DBFree Function [page 374]
Free memory the was allocated using DBAlloc or DBRealloc.

DBRealloc Function [page 375]
Reallocate memory for a SQLDA variable.

fill_s_sqlda Function [page 376]
Allocate space for each variable described in each descriptor of a SQLDA, changing all data types to DT_STRING.

fill_sqlda Function [page 377]
Allocate space for each variable described in each descriptor of a SQLDA.

fill_sqlda_ex Function [page 378]
Allocate space for each variable described in each descriptor of a SQLDA, with special processing for LONG data types.

free_filled_sqlda Function [page 379]
Free memory allocated to each sqldata pointer and the space allocated for the SQLDA itself.

free_sqlda Function [page 380]
Free memory allocated to a SQLDA.

free_sqlda_noind Function [page 381]
Free memory allocated to a SQLDA, ignoring indicator variable pointers.

sql_needs_quotes Function [page 381]
Determine if quotes are needed for a SQL identifier.

sqlda_storage Function [page 382]

Determine the amount of storage required to store a value.

sqlda_string_length Function [page 384]
Determine the amount of storage required to store a value as a C string.

sqlerror_message Function [page 385]
Obtain the error message text.

1.10.23.1 alloc_sqlda Function

Allocate a SQLDA.

Syntax

```
struct sqlda * alloc_sqlda( unsigned numvar );
```

Parameters

numvar

The number of variable descriptors to allocate.

Returns

Pointer to a SQLDA if successful and returns the null pointer if there is not enough memory available.

Remarks

Allocates a SQLDA with descriptors for `numvar` variables. The sqln field of the SQLDA is initialized to `numvar`. Space is allocated for the indicator variables, the indicator pointers are set to point to this space, and the indicator value is initialized to zero. A null pointer is returned if memory cannot be allocated. Use this function instead of the alloc_sqlda_noind function.

Related Information

The SQL Descriptor Area (SQLDA) [page 300]
fill_s_sqlda Function [page 376]
fill_sqlda Function [page 377]

fill_sqlda_ex Function [page 378]
free_sqlda Function [page 380]

1.10.23.2 alloc_sqlda_noind Function

Allocate a SQLDA with no space for indicator variables.

Syntax

```
struct sqlda * alloc_sqlda_noind( unsigned numvar );
```

Parameters

numvar

The number of variable descriptors to allocate.

Returns

Pointer to a SQLDA if successful and returns the null pointer if there is not enough memory available.

Remarks

Allocates a SQLDA with descriptors for `numvar` variables. The sqln field of the SQLDA is initialized to `numvar`. Space is not allocated for indicator variables; the indicator pointers are set to the null pointer. A null pointer is returned if memory cannot be allocated.

Related Information

The SQL Descriptor Area (SQLDA) [page 300]
fill_s_sqlda Function [page 376]
fill_sqlda Function [page 377]
fill_sqlda_ex Function [page 378]
free_sqlda_noind Function [page 381]

1.10.23.3 db_backup Function

Back up a database.

Syntax

```
void db_backup(
SQLCA * sqlca,
int op,
int file_num,
unsigned long page_num,
struct sqlda * sqlda);
```

Parameters

sqlca

A pointer to a SQLCA structure.

op

The action or operation to be performed.

file_num

The file number of the database.

page_num

The page number of the database. A value in the range 0 to the maximum number of pages less 1.

sqlda

A pointer to a SQLDA structure.

Authorization

Must be connected as a user with BACKUP DATABASE system privilege, or have the SYS_RUN_REPLICATION_ROLE system role.

Remarks

Although this function provides one way to add backup features to an application, the recommended way to do this task is to use the BACKUP statement.

The action performed depends on the value of the op parameter:

DB_BACKUP_START

Must be called before a backup can start. Only one backup can be running per database at one time against any given database server. Database checkpoints are disabled until the backup is complete

(db_backup is called with an `op` value of DB_BACKUP_END). If the backup cannot start, the SQLCODE is SQLE_BACKUP_NOT_STARTED. Otherwise, the SQLCOUNT field of the `sqlca` is set to the database page size. Backups are processed one page at a time.

The `file_num`, `page_num`, and `sqlda` parameters are ignored.

DB_BACKUP_OPEN_FILE

Open the database file specified by `file_num`, which allows pages of the specified file to be backed up using DB_BACKUP_READ_PAGE. Valid file numbers are 0 through DB_BACKUP_MAX_FILE for the root database files, and 0 through DB_BACKUP_TRANS_LOG_FILE for the transaction log file. If the specified file does not exist, the SQLCODE is SQLE_NOTFOUND. Otherwise, SQLCOUNT contains the number of pages in the file, SQLIOESTIMATE contains a 32-bit value (POSIX time_t) that identifies the time that the database file was created, and the operating system file name is in the `sqlerrmc` field of the SQLCA.

The `page_num` and `sqlda` parameters are ignored.

DB_BACKUP_READ_PAGE

Read one page of the database file specified by `file_num`. The `page_num` should be a value from 0 to one less than the number of pages returned in SQLCOUNT by a successful call to db_backup with the DB_BACKUP_OPEN_FILE operation. Otherwise, SQLCODE is set to SQLE_NOTFOUND. The `sqlda` descriptor should be set up with one variable of type DT_BINARY or DT_LONG_BINARY pointing to a buffer. The buffer should be large enough to hold binary data of the size returned in the SQLCOUNT field on the call to db_backup with the DB_BACKUP_START operation.

DT_BINARY data contains a two-byte length followed by the actual binary data, so the buffer must be two bytes longer than the page size.

> **Note**
>
> This call makes a copy of the specified database page into the buffer, but it is up to the application to save the buffer on some backup media.

DB_BACKUP_READ_RENAME_LOG

This action is the same as DB_BACKUP_READ_PAGE, except that after the last page of the transaction log has been returned, the database server renames the transaction log and starts a new one.

If the database server is unable to rename the log at the current time (for example in version 7.0.x or earlier databases there may be incomplete transactions), the SQLE_BACKUP_CANNOT_RENAME_LOG_YET error is set. In this case, do not use the page returned, but instead reissue the request until you receive SQLE_NOERROR and then write the page. Continue reading the pages until you receive the SQLE_NOTFOUND condition.

The SQLE_BACKUP_CANNOT_RENAME_LOG_YET error may be returned multiple times and on multiple pages. In your retry loop, add a delay so as not to slow the server down with too many requests.

When you receive the SQLE_NOTFOUND condition, the transaction log has been backed up successfully and the file has been renamed. The name for the old transaction file is returned in the `sqlerrmc` field of the SQLCA.

Check the sqlda->sqlvar[0].sqlind value after a db_backup call. If this value is greater than zero, the last log page has been written and the transaction log file has been renamed. The new name is still in sqlca.sqlerrmc, but the SQLCODE value is SQLE_NOERROR.

Do not call db_backup again after this, except to close files and finish the backup. If you do, you get a second copy of your backed up log file and you receive SQLE_NOTFOUND.

DB_BACKUP_CLOSE_FILE

Must be called when processing of one file is complete to close the database file specified by `file_num`.

The `page_num` and `sqlda` parameters are ignored.

DB_BACKUP_END

Must be called at the end of the backup. No other backup can start until this backup has ended. Checkpoints are enabled again.

The `file_num`, `page_num` and `sqlda` parameters are ignored.

DB_BACKUP_PARALLEL_START

Starts a parallel backup. Like DB_BACKUP_START, only one backup can be running against a database at one time on any given database server. Database checkpoints are disabled until the backup is complete (until db_backup is called with an `op` value of DB_BACKUP_END). If the backup cannot start, you receive SQLE_BACKUP_NOT_STARTED. Otherwise, the SQLCOUNT field of the sqlca is set to the database page size.

The `file_num` parameter instructs the database server to rename the transaction log and start a new one after the last page of the transaction log has been returned. If the value is non-zero then the transaction log is renamed or restarted. Otherwise, it is not renamed and restarted. This parameter eliminates the need for the DB_BACKUP_READ_RENAME_LOG operation, which is not allowed during a parallel backup operation.

The `page_num` parameter informs the database server of the maximum size of the client's buffer, in database pages. On the server side, the parallel backup readers try to read sequential blocks of pages. This value lets the server know how large to allocate these blocks: passing a value of `nnn` lets the server know that the client is willing to accept at most `nnnn` database pages at a time from the server. The server may return blocks of pages of less than size `nnn` if it is unable to allocate enough memory for blocks of `nnn` pages. If the client does not know the size of database pages until after the call to DB_BACKUP_PARALLEL_START, this value can be provided to the server with the DB_BACKUP_INFO operation. This value must be provided before the first call to retrieve backup pages (DB_BACKUP_PARALLEL_READ).

> **Note**
>
> If you are using db_backup to start a parallel backup, db_backup does not create writer threads. The caller of db_backup must receive the data and act as the writer.

DB_BACKUP_INFO

This parameter provides additional information to the database server about the parallel backup. The `file_num` parameter indicates the type of information being provided, and the `page_num` parameter provides the value. You can specify the following additional information with DB_BACKUP_INFO:

DB_BACKUP_INFO_CHKPT_LOG

This is the client-side equivalent to the WITH CHECKPOINT LOG option of the BACKUP statement. A `page_num` value of DB_BACKUP_CHKPT_COPY indicates COPY, while the value DB_BACKUP_CHKPT_NOCOPY indicates NO COPY. If this value is not provided it defaults to COPY.

DB_BACKUP_INFO_PAGES_IN_BLOCK

The `page_num` argument contains the maximum number of pages that should be sent back in one block.

DB_BACKUP_INFO_WAIT_AFTER_END Waits until transactions are complete to rename or truncate the transaction log. This is the client-side equivalent to the WAIT AFTER END option of the BACKUP

statement. The `page_num` value is ignored. DB_BACKUP_INFO_WAIT_AFTER_END is ignored by version 16 and earlier SQL Anywhere database servers.

DB_BACKUP_PARALLEL_READ

This operation reads a block of pages from the database server. Before invoking this operation, use the DB_BACKUP_OPEN_FILE operation to open all the files that you want to back up. DB_BACKUP_PARALLEL_READ ignores the `file_num` and `page_num` arguments.

The sqlda descriptor should be set up with one variable of type DT_LONGBINARY pointing to a buffer. The buffer should be large enough to hold binary data of the size `nnn` pages (specified in the DB_BACKUP_START_PARALLEL operation, or in a DB_BACKUP_INFO operation).

The server returns a sequential block of database pages for a particular database file. The page number of the first page in the block is returned in the SQLCOUNT field. The file number that the pages belong to is returned in the SQLIOESTIMATE field, and this value matches one of the file numbers used in the DB_BACKUP_OPEN_FILE calls. The size of the data returned is available in the `stored_len` field of the DT_LONGBINARY variable, and is always a multiple of the database page size. While the data returned by this call contains a block of sequential pages for a given file, it is not safe to assume that separate blocks of data are returned in sequential order, or that all of one database file's pages are returned before another database file's pages. The caller should be prepared to receive portions of another individual file out of sequential order, or of any opened database file on any given call.

An application should make repeated calls to this operation until the size of the read data is 0, or the value of sqlda->sqlvar[0].sqlind is greater than 0. If the backup is started with transaction log renaming/restarting, SQLERROR could be set to SQLE_BACKUP_CANNOT_RENAME_LOG_YET. In this case, do not use the pages returned, but instead reissue the request until you receive SQLE_NOERROR, and then write the data. The SQLE_BACKUP_CANNOT_RENAME_LOG_YET error may be returned multiple times and on multiple pages. In your retry loop, add a delay so the database server is not slowed down by too many requests. Continue reading the pages until either of the first two conditions are met.

The dbbackup utility uses the following algorithm. This is *not* C code, and does not include error checking.

```
sqlda->sqld = 1;
sqlda->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 */
sqlda->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 == SQLE_NOERROR
    /* 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, &sqlda );
  if SQLCODE != SQLE_NOERROR
     break;
  if buffer->stored_len == 0 || sqlda->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
```

Related Information

The SQL Communication Area (SQLCA) [page 290]
The SQL Descriptor Area (SQLDA) [page 300]
BACKUP DATABASE Statement
Embedded SQL Data Types [page 277]

1.10.23.4 db_cancel_request Function

Cancel an active request.

Syntax

```
int db_cancel_request( SQLCA * sqlca );
```

Parameters

sqlca

A pointer to a SQLCA structure.

Returns

1 when the cancel request is sent; 0 if no request is sent.

Remarks

Cancels the currently active database server request. This function checks to make sure a database server request is active before sending the cancel request.

A non-zero return value does not mean that the request was canceled. There are a few critical timing cases where the cancel request and the response from the database or server cross. In these cases, the cancel simply has no effect, even though the function still returns TRUE.

The db_cancel_request function can be called asynchronously. This function and db_is_working are the only functions in the database interface library that can be called asynchronously using a SQLCA that might be in use by another request.

If you cancel a request that is carrying out a cursor operation, the position of the cursor is indeterminate. You must locate the cursor by its absolute position or close it, following the cancel.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.5 db_change_char_charset Function

Change the application's CHAR character set for this connection.

Syntax

```
unsigned int db_change_char_charset(
SQLCA * sqlca,
char * charset );
```

Parameters

sqlca

A pointer to a SQLCA structure.

charset

A string representing the character set.

Returns

1 if the change is successful; 0 otherwise.

Remarks

Data sent and fetched using DT_FIXCHAR, DT_VARCHAR, DT_LONGVARCHAR, and DT_STRING types are in the CHAR character set.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Recommended Character Sets and Collations

1.10.23.6 db_change_nchar_charset Function

Change the application's NCHAR character set for this connection.

Syntax

```
unsigned int db_change_nchar_charset(
SQLCA * sqlca,
char * charset );
```

Parameters

sqlca

A pointer to a SQLCA structure.

charset

A string representing the character set.

Returns

1 if the change is successful; 0 otherwise.

Remarks

Data sent and fetched using DT_NFIXCHAR, DT_NVARCHAR, DT_LONGNVARCHAR, and DT_NSTRING host variable types are in the NCHAR character set.

If the db_change_nchar_charset function is not called, all data is sent and fetched using the CHAR character set. Typically, an application that wants to send and fetch Unicode data should set the NCHAR character set to UTF-8.

If this function is called, the charset parameter is usually "UTF-8". The NCHAR character set cannot be set to UTF-16.

In Embedded SQL, NCHAR, NVARCHAR and LONG NVARCHAR are described as DT_FIXCHAR, DT_VARCHAR, and DT_LONGVARCHAR, respectively, by default. If the db_change_nchar_charset function has been called, these types are described as DT_NFIXCHAR, DT_NVARCHAR, and DT_LONGNVARCHAR, respectively.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Recommended Character Sets and Collations

1.10.23.7 db_find_engine Function

Return status information about a local database server.

Syntax

```
unsigned short db_find_engine(
SQLCA * sqlca,
char * name );
```

Parameters

sqlca

A pointer to a SQLCA structure.

name

NULL or a string containing the server's name.

Returns

Server status as an unsigned short value, or 0 if no server can be found over shared memory.

Remarks

Returns an unsigned short value, which indicates status information about the local database server whose name is `name`. If no server can be found over shared memory with the specified name, the return value is 0. A non-zero value indicates that the local server is currently running.

If a null pointer is specified for `name`, information is returned about the default database server.

Each bit in the return value conveys some information. Constants that represent the bits for the various pieces of information are defined in the `sqldef.h` header file. Their meaning is described below.

DB_ENGINE

This flag is always set.

DB_CLIENT

This flag is always set.

DB_CAN_MULTI_DB_NAME

This flag is obsolete.

DB_DATABASE_SPECIFIED

This flag is always set.

DB_ACTIVE_CONNECTION

This flag is always set.

DB_CONNECTION_DIRTY

This flag is obsolete.

DB_CAN_MULTI_CONNECT

This flag is obsolete.

DB_NO_DATABASES

This flag is set if the server has no databases started.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.8 db_fini Function

Free resources used by the database interface or DLL.

Syntax

```
int db_fini( SQLCA * sqlca );
```

Parameters

sqlca

A pointer to a SQLCA structure.

Returns

Non-zero value for success; 0 otherwise.

Remarks

This function frees resources used by the database interface or DLL. You must not make any other library calls or execute any Embedded SQL statements after db_fini is called. If an error occurs during processing, the error code is set in SQLCA and the function returns 0. If there are no errors, a non-zero value is returned.

You must call db_fini once for each SQLCA being used.

The db_fini function should not be called directly or indirectly from the DllMain function in a Windows Dynamic Link Library. The DllMain entry point function is intended to perform only simple initialization and termination tasks. Calling db_fini can create deadlocks and circular dependencies.

For UltraLite applications, there is an equivalent db_fini method.

Related Information

The SQL Communication Area (SQLCA) [page 290]
db_fini Method

1.10.23.9 db_get_property Function

Obtain information about the database interface or the server to which you are connected.

Syntax

```
unsigned int db_get_property(
SQLCA * sqlca,
a_db_property property,
char * value_buffer,
int value_buffer_size );
```

Parameters

sqlca

A pointer to a SQLCA structure.

a_db_property

The property requested, either DB_PROP_CLIENT_CHARSET, DB_PROP_SERVER_ADDRESS, or DB_PROP_DBLIB_VERSION.

value_buffer

This argument is filled with the property value as a null-terminated string.

value_buffer_size

The maximum length of the string value_buffer, including room for the terminating null character.

Returns

1 if successful; 0 otherwise.

Remarks

This function is used to obtain information about the database interface or the server to which you are connected.

The following properties are supported:

DB_PROP_CLIENT_CHARSET

This property value gets the client character set (for example, "windows-1252").

DB_PROP_SERVER_ADDRESS

This property value gets the current connection's server network address as a printable string. The shared memory protocol always returns the empty string for the address. The TCP/IP protocol returns non-empty string addresses.

DB_PROP_DBLIB_VERSION

This property value gets the database interface library's version (for example, "17.0.11.1293").

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.10 db_init Function

Initialize the database interface library.

Syntax

```
int db_init( SQLCA * sqlca );
```

Parameters

sqlca

A pointer to a SQLCA structure.

Returns

Non-zero value if successful; 0 otherwise.

Remarks

This function initializes the database interface library. This function must be called before any other library call is made and before any Embedded SQL statement is executed. The resources the interface library required for your program are allocated and initialized on this call.

Use db_fini to free the resources at the end of your program. If there are any errors during processing, they are returned in the SQLCA and 0 is returned. If there are no errors, a non-zero value is returned and you can begin using Embedded SQL statements and functions.

Usually, this function should be called only once (passing the address of the global sqlca variable defined in the `sqlca.h` header file). If you are writing a DLL or an application that has multiple threads using Embedded SQL, call db_init once for each SQLCA that is being used.

For UltraLite applications, there is an equivalent db_init method.

Related Information

The SQL Communication Area (SQLCA) [page 290]
SQLCA Management for Multithreaded or Reentrant Code [page 293]
db_init Method

1.10.23.11 db_is_working Function

Indicates whether a database request is in progress.

Syntax

```
unsigned short db_is_working( SQLCA * sqlca );
```

Parameters

sqlca

A pointer to a SQLCA structure.

Returns

1 if your application has a database request in progress that uses the given `sqlca` and 0 if there is no request in progress that uses the given `sqlca`.

Remarks

This function can be called asynchronously. This function and db_cancel_request are the only functions in the database interface library that can be called asynchronously using a SQLCA that might be in use by another request.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.12 db_locate_servers Function

Obtain information about all database servers on the local network that are listening on TCP/IP.

Syntax

```
unsigned int db_locate_servers(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
```

```
void * callback_user_data );
```

Parameters

sqlca

A pointer to a SQLCA structure.

callback_address

The address of a callback function.

callback_user_data

The address of a user-defined area in which to store data.

Returns

1 if successful; 0 otherwise.

Remarks

Provides programmatic access to the information displayed by the dblocate utility, listing all the database servers on the local network that are listening on TCP/IP.

The callback function must have the following prototype:

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

The callback function is called for each server found. If the callback function returns 0, db_locate_servers stops iterating through servers.

The sqlca and callback_user_data passed to the callback function are those passed into db_locate_servers. The second parameter is a pointer to an a_server_address structure. a_server_address is defined in `sqlca.h`, with the following definition:

```
typedef struct a_server_address {
    a_sql_uint32 port_type;
    a_sql_uint32 port_num;
    char         *name;
    char         *address;
} a_server_address;
```

port_type

Is always PORT_TYPE_TCP at this time (defined to be 6 in `sqlca.h`).

port_num

Is the TCP port number on which this server is listening.

name

Points to a buffer containing the server name.

address

Points to a buffer containing the IP address of the server.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Server Enumeration Utility (dblocate)

1.10.23.13 db_locate_servers_ex Function

Obtain information about all database servers on the local network that are listening on TCP/IP.

Syntax

```
unsigned int db_locate_servers_ex(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data,
unsigned int bitmask);
```

Parameters

sqlca

A pointer to a SQLCA structure.

callback_address

The address of a callback function.

callback_user_data

The address of a user-defined area in which to store data.

bitmask

A mask composed of any of DB_LOOKUP_FLAG_NUMERIC, DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT, or DB_LOOKUP_FLAG_DATABASES.

Returns

1 if successful; 0 otherwise.

Remarks

Provides programmatic access to the information displayed by the dblocate utility, listing all the database servers on the local network that are listening on TCP/IP, and provides a mask parameter used to select addresses passed to the callback function.

The callback function must have the following prototype:

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

The callback function is called for each server found. If the callback function returns 0, db_locate_servers_ex stops iterating through servers.

The sqlca and callback_user_data passed to the callback function are those passed into db_locate_servers. The second parameter is a pointer to an a_server_address structure. a_server_address is defined in `sqlca.h`, with the following definition:

```
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

Is always PORT_TYPE_TCP at this time (defined to be 6 in `sqlca.h`).

port_num

Is the TCP port number on which this server is listening.

name

Points to a buffer containing the server name.

address

Points to a buffer containing the IP address of the server.

dbname

Points to a buffer containing the database name.

Three bitmask flags are supported:

- DB_LOOKUP_FLAG_NUMERIC
- DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
- DB_LOOKUP_FLAG_DATABASES

These flags are defined in `sqlca.h` and can be ORed together.

DB_LOOKUP_FLAG_NUMERIC ensures that addresses passed to the callback function are IP addresses, instead of host names.

DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT specifies that the address includes the TCP/IP port number in the a_server_address structure passed to the callback function.

DB_LOOKUP_FLAG_DATABASES specifies that the callback function is called once for each database found, or once for each database server found if the database server doesn't support sending database information (version 9.0.2 and earlier database servers).

Related Information

The SQL Communication Area (SQLCA) [page 290]
Server Enumeration Utility (dblocate)

1.10.23.14 db_register_a_callback Function

Register a callback function.

Syntax

```
void db_register_a_callback(
SQLCA * sqlca,
a_db_callback_index index,
( SQL_CALLBACK_PARM ) callback );
```

Parameters

sqlca

A pointer to a SQLCA structure.

index

An index value identifying the type of callback described below.

callback

The address of a user-defined callback function.

Remarks

This function registers callback functions.

If you do not register a DB_CALLBACK_WAIT callback, the default action is to do nothing. Your application blocks, waiting for the database response. You must register a callback for the MESSAGE TO CLIENT statement.

To remove a callback, pass a null pointer as the `callback` function.

The following values are allowed for the `index` parameter:

DB_CALLBACK_DEBUG_MESSAGE

The supplied function is called once for each debug message and is passed a null-terminated string containing the text of the debug message. A debug message is a message that is logged to the LogFile file. In order for a debug message to be passed to this callback, the LogFile connection parameter must be used. The string normally has a newline character (\n) immediately before the terminating null character. The prototype of the callback function is as follows:

```
void SQL_CALLBACK debug_message_callback(
SQLCA * sqlca,
char * message_string );
```

DB_CALLBACK_START

The prototype is as follows:

```
void SQL_CALLBACK start_callback( SQLCA * sqlca );
```

This function is called just before a database request is sent to the server. DB_CALLBACK_START is used only on Windows.

DB_CALLBACK_FINISH

The prototype is as follows:

```
void SQL_CALLBACK finish_callback( SQLCA * sqlca );
```

This function is called after the response to a database request has been received by the DBLIB interface DLL. DB_CALLBACK_FINISH is used only on Windows operating systems.

DB_CALLBACK_CONN_DROPPED

The prototype is as follows:

```
void SQL_CALLBACK conn_dropped_callback (
SQLCA * sqlca,
char * conn_name );
```

This function is called when the database server is about to drop a connection because of a liveness timeout, through a DROP CONNECTION statement, or because the database server is being shut down. The connection name `conn_name` is passed in to allow you to distinguish between connections. If the connection was not named, it has a value of NULL.

DB_CALLBACK_WAIT

The prototype is as follows:

```
void SQL_CALLBACK wait_callback( SQLCA * sqlca );
```

This function is called repeatedly by the interface library while the database server or client library is busy processing your database request.

DB_CALLBACK_MESSAGE

This is used to enable the application to handle messages received from the server during the processing of a request. Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

The callback prototype is as follows:

```
void SQL_CALLBACK message_callback(
SQLCA * sqlca,
unsigned char msg_type,
an_sql_code code,
unsigned short length,
char *  msg
);
```

The `msg_type` parameter states how important the message is. You can handle different message types in different ways. The following possible values for `msg_type` are defined in `sqldef.h`.

MESSAGE_TYPE_INFO

The message type was INFO.

MESSAGE_TYPE_WARNING

The message type was WARNING.

MESSAGE_TYPE_ACTION

The message type was ACTION.

MESSAGE_TYPE_STATUS

The message type was STATUS.

MESSAGE_TYPE_PROGRESS

The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

The `code` parameter may provide a SQLCODE associated with the message, otherwise the value is 0.

The `length` parameter tells you how long the message is.

The `msg` parameter points to the message text. The message is *not* null-terminated.

DBLIB, ODBC, and C API clients can use the DB_CALLBACK_MESSAGE callback to receive progress messages. For example, the Interactive SQL callback displays STATUS and INFO message on the *History* tab, while messages of type ACTION and WARNING go to a window. If an application does not register this callback, there is a default callback, which causes all messages to be written to the server logfile (if debugging is on and a logfile is specified). In addition, messages of type MESSAGE_TYPE_WARNING and MESSAGE_TYPE_ACTION are more prominently displayed, in an operating system-dependent manner.

When a message callback is not registered by the application, messages sent to the client are saved to the message log file when the LogFile connection parameter is specified. Also, ACTION or STATUS messages sent to the client appear in a window on Windows operating systems and are logged to stderr on UNIX and Linux operating systems.

DB_CALLBACK_VALIDATE_FILE_TRANSFER

This is used to register a file transfer validation callback function. Before allowing any transfer to take place, the client library invokes the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the client library will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below.

The callback prototype is as follows:

```
int SQL_CALLBACK file_transfer_callback(
SQLCA * sqlca,
char * file_name,
int is_write
);
```

The `file_name` parameter is the name of the file to be read or written. The `is_write` parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the Embedded SQL client library allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. If the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

Example

The following is an example of a MESSAGE callback and how to register it.

```
#include <stdio.h>
#include <stdlib.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;
EXEC SQL SET SQLCA "&sqlca";
void SQL_CALLBACK messages( SQLCA *sqlca,
    unsigned char   msg_type,
    an_sql_code     sqlcode,
    unsigned short  length,
    char *          msg )
{
    size_t  mlen;
    char    *mtype;
    char    mbuffer[80];
    switch( msg_type )
    {
        case MESSAGE_TYPE_INFO:
            mtype = "INFO";
            break;
        case MESSAGE_TYPE_WARNING:
            mtype = "WARNING";
            break;
        case MESSAGE_TYPE_ACTION:
            mtype = "ACTION";
            break;
        case MESSAGE_TYPE_STATUS:
            mtype = "STATUS";
            break;
        case MESSAGE_TYPE_PROGRESS:
```

```
            mtype = "PROGRESS";
            break;
    }
    mlen = __min( length, sizeof(mbuffer) );
    strncpy( mbuffer, msg, mlen );
    mbuffer[mlen] = '\0';
    printf( "Message was \"%s\", type %s, SQLCODE(%d)\n", mbuffer, mtype,
sqlcode );
}
int main( int argc, char *argv[] )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char *sqlstr = "MESSAGE 'All is well' TYPE INFO TO CLIENT";
    char  connectstr[80];
    EXEC SQL END DECLARE SECTION;

    if( argc <= 1 ) return 1; // no password
    db_init( &sqlca );
    strcpy( connectstr, "DSN=SQL Anywhere 17 Demo;PWD=" );
    strcat( connectstr, argv[1] );
    db_string_connect( &sqlca, connectstr );
    db_register_a_callback( &sqlca, DB_CALLBACK_MESSAGE,
(SQL_CALLBACK_PARM)messages );
    EXEC SQL EXECUTE IMMEDIATE :sqlstr;
    db_string_disconnect( &sqlca, "" );
    db_fini( &sqlca );
    return 0;
}
```

Related Information

The SQL Communication Area (SQLCA) [page 290]
LogFile (LOG) Connection Parameter
MESSAGE Statement
progress_messages Option

1.10.23.15 db_start_database Function

Start a database on a server.

Syntax

```
unsigned int db_start_database( SQLCA * sqlca, char * parms );
```

Parameters

sqlca

A pointer to a SQLCA structure.

parms

A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example:

```
"UID=DBA;PWD=password;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

The database is started on an existing server, if possible. Otherwise, a new server is started.

If the database was already running or was successfully started, the return value is true (non-zero) and SQLCODE is set to 0. Error information is returned in the SQLCA.

If a user ID and password are supplied in the parameters, they are ignored.

The privilege required to start and stop a database is set on the server command line using the -gd option.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Alphabetical List of Connection Parameters
Troubleshooting: How Database Servers Are Located
-gd Database Server Option

1.10.23.16 db_start_engine Function

Start a database server.

Syntax

```
unsigned int db_start_engine( SQLCA * sqlca, char * parms );
```

Parameters

sqlca

A pointer to a SQLCA structure.

parms

A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=`value`. For example:

```
"UID=DBA;PWD=password;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

Starts the database server if it is not running.

If the database server was already running or was successfully started, the return value is TRUE (non-zero) and SQLCODE is set to 0. Error information is returned in the SQLCA.

The following call to db_start_engine starts the database server, loads the specified database, and names the server demo.

```
db_start_engine( &sqlca, "DBF=demo.db;START=dbsrv17" );
```

Unless the ForceStart (FORCE) connection parameter is used and set to YES, the db_start_engine function attempts to connect to a server before starting one, to avoid attempting to start a server that is already running.

When the ForceStart connection is set to YES, there is no attempt to connect to a server before trying to start one. This enables the following pair of commands to work as expected:

1. Start a database server named server_1:

   ```
   dbsrv17 -n server_1 demo.db
   ```

2. Force a new server to start and connect to it:

   ```
   db_start_engine( &sqlda,
     "START=dbsrv17 -n server_2 mydb.db;ForceStart=YES" )
   ```

If ForceStart (FORCE) is not used and the ServerName (Server) parameter is not used, then the second command would have attempted to connect to server_1. The db_start_engine function does not pick up the server name from the -n option of the StartLine (START) parameter.

Related Information

The SQL Communication Area (SQLCA) [page 290]

Alphabetical List of Connection Parameters
Troubleshooting: How Database Servers Are Located

1.10.23.17 db_stop_database Function

Stop a database on a server.

Syntax

```
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

Parameters

sqlca

A pointer to a SQLCA structure.

parms

A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example:

```
"UID=DBA;PWD=password;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

Stop the database identified by DatabaseName (DBN) on the server identified by ServerName (Server). If ServerName is not specified, the default server is used.

By default, this function does not stop a database that has existing connections. If Unconditional (UNC) is set to **yes**, the database is stopped regardless of existing connections.

A return value of TRUE indicates that there were no errors.

The privilege required to start and stop a database is set on the server command line using the -gd option.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Alphabetical List of Connection Parameters
-gd Database Server Option

1.10.23.18 db_stop_engine Function

Stop a database server.

Syntax

```
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

Parameters

sqlca

A pointer to a SQLCA structure.

parms

A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example:

```
"UID=DBA;PWD=password;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

Stops execution of the database server. The steps carried out by this function are:

- Look for a local database server that has a name that matches the ServerName (Server) parameter. If no ServerName is specified, look for the default local database server.
- If no matching server is found, this function returns with success.
- Send a request to the server to tell it to checkpoint and shut down all databases.
- Unload the database server.

By default, this function does not stop a database server that has existing connections. If the Unconditional=yes connection parameter is specified, the database server is stopped regardless of existing connections.

A C program can use this function instead of spawning dbstop. A return value of TRUE indicates that there were no errors.

The use of db_stop_engine is subject to the privileges set with the -gk server option.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Alphabetical List of Connection Parameters
-gk Database Server Option

1.10.23.19 db_string_connect Function

Connect to a database on a database server.

Syntax

```
unsigned int db_string_connect( SQLCA * sqlca, char * parms );
```

Parameters

sqlca

A pointer to a SQLCA structure.

parms

A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example:

```
"UID=DBA;PWD=password;DBF=c:\\db\\mydatabase.db"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

Provides extra functionality beyond the Embedded SQL CONNECT statement.

The algorithm used by this function is described in the troubleshooting connections topic.

The return value is TRUE (non-zero) if a connection was successfully established and FALSE (zero) otherwise. Error information for starting the server, starting the database, or connecting is returned in the SQLCA.

Related Information

The SQL Communication Area (SQLCA) [page 290]

Alphabetical List of Connection Parameters
Troubleshooting: Connections

1.10.23.20 db_string_disconnect Function

Disconnect the current or other connection from a database on a database server.

Syntax

```
unsigned int db_string_disconnect(
    SQLCA * sqlca,
    char * parms );
```

Parameters

sqlca

A pointer to a SQLCA structure.

parms

A null-terminated string containing a semicolon-delimited list of connection parameters, each of the form keyword=`value`. For example:

```
"DSN=SQL Anywhere 17 Demo;ConnectionName=esql-con-20130228;PWD=sql"
```

Returns

Non-zero if successful; 0 otherwise.

Remarks

This function disconnects the connection identified by the ConnectionName (CON) connection parameter, if one is specified. All other parameters are ignored.

If no ConnectionName parameter is specified in the string, the unnamed connection is disconnected. This is equivalent to the Embedded SQL DISCONNECT statement. The return value is TRUE if a connection was successfully ended. Error information is returned in the SQLCA.

This function shuts down the database if it was started with the AutoStop=yes connection parameter and there are no other connections to the database. It also stops the server if it was started with the AutoStop=yes parameter and there are no other databases running.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Alphabetical List of Connection Parameters

1.10.23.21 db_string_ping_server Function

Determine if a server can be located, and optionally, if it a successful connection to a database can be made.

Syntax

```
unsigned int db_string_ping_server(
SQLCA * sqlca,
char * connect_string,
unsigned int connect_to_db );
```

Parameters

sqlca

A pointer to a SQLCA structure.

connect_string

The `connect_string` is a normal connection string that may or may not contain server and database information.

connect_to_db

If `connect_to_db` is non-zero (TRUE), then the function attempts to connect to a database on a server. It returns TRUE only if the connection string is sufficient to connect to the named database on the named server.

If `connect_to_db` is zero, then the function only attempts to locate a server. It returns TRUE only if the connection string is sufficient to locate a server. It makes no attempt to connect to the database.

Returns

TRUE (non-zero) if the server or database was successfully located; FALSE (zero) otherwise. Error information for locating the server or database is returned in the SQLCA.

Remarks

This function can be used to determine if a server can be located, and optionally, if it a successful connection to a database can be made.

Related Information

The SQL Communication Area (SQLCA) [page 290]
Alphabetical List of Connection Parameters

1.10.23.22 db_time_change Function

Notify the server that the time has changed on the client.

Syntax

```
unsigned int db_time_change(
SQLCA * sqlca);
```

Parameters

sqlca

A pointer to a SQLCA structure.

Returns

TRUE if successful; FALSE otherwise.

Remarks

This function permits clients to notify the server that the time has changed on the client. This function recalculates the time zone adjustment and sends it to the server. On Windows platforms, applications must call this function when they receive the WM_TIMECHANGE message. This will make sure that UTC timestamps are consistent over time changes, time zone changes, or daylight savings time changeovers.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.23 DBAlloc Function

Allocate memory for a SQLDA variable.

Syntax

```
void * DBAlloc( size_t size );
```

Parameters

size

The number of bytes to allocate.

Returns

DBAlloc returns a void pointer to the allocated space, or NULL if there is insufficient memory available.

Remarks

Use this function to allocate memory for Embedded SQL data areas.

Example

In this example, assume that the third column has sqltype DT_LONGVARCHAR. Since fill_sqlda allocates at most 32767 bytes of memory for the data area, the data area is released and a new area is allocated for a maximum size string of 100K bytes. The LONGVARCHAR data structure is initialized. A subsequent call to the free_filled_sqlda function will release this new data area.

```
#define MAXLEN (100*1024)
LONGVARCHAR *lvcdata;
fill_sqlda( sqlda );
DBFree( sqlda->sqlvar[ 2 ].sqldata );
sqlda->sqlvar[ 2 ].sqldata = DBAlloc( LONGVARCHARSIZE(MAXLEN) );
lvcdata = (LONGVARCHAR *)sqlda->sqlvar[ 2 ].sqldata;
lvcdata->array_len = MAXLEN;
lvcdata->stored_len = 0;
lvcdata->untrunc_len = 0;
```

1.10.23.24 DBFree Function

Free memory the was allocated using DBAlloc or DBRealloc.

Syntax

```
void DBFree( void * ptr );
```

Parameters

ptr

Pointer to the memory allocated by the DBAlloc or DBRealloc function.

Remarks

Use this function to free memory for Embedded SQL data areas.

Example

In this example, assume that the third column has sqltype DT_LONGVARCHAR. Since fill_sqlda allocates at most 32767 bytes of memory for the data area, the data area is released and reallocated for a maximum size

string of 100K bytes. The LONGVARCHAR data structure is initialized. A subsequent call to the free_filled_sqlda function will release this new data area.

```
#define MAXLEN (100*1024)
LONGVARCHAR *lvcdata;
fill_sqlda( sqlda );
DBFree( sqlda->sqlvar[ 2 ].sqldata );
sqlda->sqlvar[ 2 ].sqldata = DBAlloc( LONGVARCHARSIZE(MAXLEN) );
lvcdata = (LONGVARCHAR *)sqlda->sqlvar[ 2 ].sqldata;
lvcdata->array_len = MAXLEN;
lvcdata->stored_len = 0;
lvcdata->untrunc_len = 0;
```

1.10.23.25 DBRealloc Function

Reallocate memory for a SQLDA variable.

Syntax

```
void * DBRealloc( void * ptr, size_t size );
```

Parameters

ptr

Pointer to the memory allocated by a previous call to the DBAlloc or DBRealloc function.

size

The number of bytes to allocate.

Returns

DBRealloc returns a void pointer to the allocated space, or NULL if there is insufficient memory available.

Remarks

Use this function to reallocate memory for Embedded SQL data areas.

Example

In this example, assume that the third column has sqltype DT_LONGVARCHAR. Since fill_sqlda allocates at most 32767 bytes of memory for the data area, the data area is reallocated for a maximum size string of 100K

bytes. The LONGVARCHAR data structure is initialized. A subsequent call to the free_filled_sqlda function will release this new data area.

```
#define MAXLEN (100*1024)
LONGVARCHAR *lvcdata;
fill_sqlda( sqlda );
sqlda->sqlvar[ 2 ].sqldata = DBRealloc( sqlda->sqlvar[ 2 ].sqldata,
LONGVARCHARSIZE(MAXLEN) );
lvcdata = (LONGVARCHAR *)sqlda->sqlvar[ 2 ].sqldata;
lvcdata->array_len = MAXLEN;
lvcdata->stored_len = 0;
lvcdata->untrunc_len = 0;
```

1.10.23.26 fill_s_sqlda Function

Allocate space for each variable described in each descriptor of a SQLDA, changing all data types to DT_STRING.

Syntax

```
struct sqlda * fill_s_sqlda(
struct sqlda * sqlda,
unsigned int maxlen );
```

Parameters

sqlda

A pointer to a SQLDA structure.

maxlen

The maximum number of bytes to allocate for the string.

Returns

`sqlda` if successful and returns NULL if there is not enough memory available.

Remarks

This function is the same as fill_sqlda, except that it changes all the data types in `sqlda` to type DT_STRING. Enough space is allocated to hold the string representation of the type originally specified by the SQLDA, up to a maximum of `maxlen` - 1 bytes. The address of this memory is assigned to the `sqldata` field of the corresponding descriptor. The maximum value for `maxlen` is 32767.

For DT_STRING variable types, the `sqllen` is updated to include the null-terminator.

The SQLDA should be freed using the free_filled_sqlda function.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.27 fill_sqlda Function

Allocate space for each variable described in each descriptor of a SQLDA.

Syntax

```
struct sqlda * fill_sqlda( struct sqlda * sqlda );
```

Parameters

sqlda

A pointer to a SQLDA structure.

Returns

`sqlda` if successful and returns NULL if there is not enough memory available.

Remarks

Allocates space for each variable described in each descriptor of `sqlda`, and assigns the address of this memory to the `sqldata` field of the corresponding descriptor. Enough space is allocated for the database type and length indicated in the descriptor.

fill_sqlda converts DT_LONGVARCHAR, DT_LONGNVARCHAR and DT_LONGBINARY types to DT_VARCHAR, DT_NVARCHAR and DT_BINARY respectively.

For the DT_STRING variable type, the `sqllen` is updated to include the null-terminator. DT_STRING variables are always null-terminated. Other variable types are not null-terminated.

The SQLDA should be freed using the free_filled_sqlda function.

fill_sqlda(sqlda) is equivalent to fill_sqlda_ex(sqlda, 0).

Related Information

The SQL Communication Area (SQLCA) [page 290]
fill_sqlda_ex Function [page 378]
free_filled_sqlda Function [page 379]

1.10.23.28 fill_sqlda_ex Function

Allocate space for each variable described in each descriptor of a SQLDA, with special processing for LONG data types.

Syntax

```
struct sqlda * fill_sqlda_ex( struct sqlda * sqlda , unsigned int flags );
```

Parameters

sqlda

A pointer to a SQLDA structure.

flags

0 or FILL_SQLDA_FLAG_RETURN_DT_LONG

Returns

`sqlda` if successful and returns NULL if there is not enough memory available.

Remarks

Allocates space for each variable described in each descriptor of `sqlda`, and assigns the address of this memory to the `sqldata` field of the corresponding descriptor. Enough space is allocated for the database type and length indicated in the descriptor.

For the DT_STRING variable type, the `sqllen` is updated to include the null-terminator. DT_STRING variables are always null-terminated. Other variable types are not null-terminated.

The SQLDA should be freed using the free_filled_sqlda function.

One flag bit is supported: FILL_SQLDA_FLAG_RETURN_DT_LONG. This flag is defined in sqlca.h.

FILL_SQLDA_FLAG_RETURN_DT_LONG preserves DT_LONGVARCHAR, DT_LONGNVARCHAR and DT_LONGBINARY types in the filled descriptor. If this flag bit is not specified, fill_sqlda_ex converts DT_LONGVARCHAR, DT_LONGNVARCHAR and DT_LONGBINARY types to DT_VARCHAR, DT_NVARCHAR and DT_BINARY respectively. Using DT_LONG... types makes it possible to fetch 32767 bytes, not the 32765 bytes that DT_VARCHAR, DT_NVARCHAR and DT_BINARY are limited to.

fill_sqlda(sqlda) is equivalent to fill_sqlda_ex(sqlda, 0).

Related Information

The SQL Communication Area (SQLCA) [page 290]
fill_sqlda Function [page 377]
free_filled_sqlda Function [page 379]

1.10.23.29 free_filled_sqlda Function

Free memory allocated to each sqldata pointer and the space allocated for the SQLDA itself.

Syntax

```
void free_filled_sqlda( struct sqlda * sqlda );
```

Parameters

sqlda

A pointer to a SQLDA structure.

Remarks

Free the memory allocated to each sqldata pointer and the space allocated for the SQLDA itself. Any null pointer is not freed.

This should only be called if fill_sqlda, fill_sqlda_ex, or fill_s_sqlda was used to allocate the sqldata fields of the SQLDA.

Calling this function causes free_sqlda to be called automatically, and so any descriptors allocated by alloc_sqlda are freed.

Related Information

The SQL Communication Area (SQLCA) [page 290]
alloc_sqlda Function [page 342]
fill_s_sqlda Function [page 376]
fill_sqlda Function [page 377]
fill_sqlda_ex Function [page 378]
free_sqlda Function [page 380]

1.10.23.30 free_sqlda Function

Free memory allocated to a SQLDA.

Syntax

```
void free_sqlda( struct sqlda * sqlda );
```

Parameters

sqlda

A pointer to a SQLDA structure.

Remarks

Free space allocated to this `sqlda` and free the indicator variable space, as allocated in fill_sqlda. Do not free the memory referenced by each sqldata pointer.

Related Information

The SQL Descriptor Area (SQLDA) [page 300]
alloc_sqlda Function [page 342]
fill_s_sqlda Function [page 376]
fill_sqlda Function [page 377]
fill_sqlda_ex Function [page 378]

1.10.23.31 free_sqlda_noind Function

Free memory allocated to a SQLDA, ignoring indicator variable pointers.

Syntax

```
void free_sqlda_noind( struct sqlda * sqlda );
```

Parameters

sqlda

A pointer to a SQLDA structure.

Remarks

Free space allocated to this `sqlda`. Do not free the memory referenced by each sqldata pointer. The indicator variable pointers are ignored.

Related Information

The SQL Descriptor Area (SQLDA) [page 300]
alloc_sqlda Function [page 342]
fill_s_sqlda Function [page 376]
fill_sqlda Function [page 377]
fill_sqlda_ex Function [page 378]

1.10.23.32 sql_needs_quotes Function

Determine if quotes are needed for a SQL identifier.

Syntax

```
unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
```

Parameters

sqlca

A pointer to a SQLCA structure.

str

A string of characters that is a candidate for a SQL identifier.

Returns

TRUE or FALSE indicating whether the string requires double quotes around it when it is used as a SQL identifier.

Remarks

This function formulates a request to the database server to determine if quotes are needed. Relevant information is stored in the sqlcode field.

There are three cases of return value/code combinations:

return = FALSE, sqlcode = 0

The string does not need quotes.

return = TRUE

The sqlcode is always SQLE_WARNING, and the string requires quotes.

return = FALSE

If sqlcode is something other than 0 or SQLE_WARNING, the test is inconclusive.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.23.33 sqlda_storage Function

Determine the amount of storage required to store a value.

Syntax

```
a_sql_uint32 sqlda_storage( struct sqlda * sqlda, int varno );
```

Parameters

sqlda

A pointer to a SQLDA structure.

varno

An index for a sqlvar host variable, starting at 0.

Returns

An unsigned 32-bit integer value representing the amount of storage required to store any value for the variable described in sqlda->sqlvar[varno]. For example, this includes the null-termination characters for DT_STRING and DT_NSTRING, as well as the overhead for types such as DT_VARCHAR, DT_BINARY, DT_LONGVARCHAR, DT_LONGNVARCHAR, and so on. The calculation uses the current sqlda->sqlvar[varno].sqllen field value which is at most 32767.

Remarks

For some types, there are also macros defined in the `sqlca.h` and `sqlda.h` header files that return the amount of storage required to store a variable of the indicated size. Since sqlda_storage performs calculation based on the sqllen value which is limited to 32767, the macros for DT_LONGBINARY, DT_LONGVARCHAR, and DT_LONGNVARCHAR may be more appropriate for larger sizes.

_BINARYSIZE(n)

Returns the amount of storage required to store a DT_BINARY of the specified length. Use the BINARY struct to map this storage.

_VARCHARSIZE(n)

Returns the amount of storage required to store a DT_VARCHAR of the specified length. Use the VARCHAR struct to map this storage.

DECIMALSTORAGE(n)

Returns the amount of storage required to store a DT_DECIMAL of the specified length. Use the TYPE_DECIMAL struct to map this storage.

LONGBINARYSIZE(n)

Returns the amount of storage required to store a DT_LONGBINARY of the specified length. Use the LONGBINARY struct to map this storage.

LONGNVARCHARSIZE(n)

Returns the amount of storage required to store a DT_LONGNVARCHAR of the specified length. Use the LONGNVARCHAR struct to map this storage.

LONGVARCHARSIZE(n)

Returns the amount of storage required to store a DT_LONGVARCHAR of the specified length. Use the LONGVARCHAR struct to map this storage.

These can be used to determine how much storage to allocate for a variable.

Example

The following C code fragment allocates a 128K buffer for a DT_LONGVARCHAR and initializes the three headers on this buffer:

```
LONGVARCHAR *lvc;
int          maxlen = 128*1024;
lvc = (LONGVARCHAR *)DBAlloc( LONGVARCHARSIZE( maxlen ) );
if( lvc )
{
    lvc->array_len = maxlen;
    lvc->stored_len = 0;
    lvc->untrunc_len = 0;
}
```

Related Information

The SQL Descriptor Area (SQLDA) [page 300]

1.10.23.34 sqlda_string_length Function

Determine the amount of storage required to store a value as a C string.

Syntax

```
a_sql_uint32 sqlda_string_length( struct sqlda * sqlda, int varno );
```

Parameters

sqlda

A pointer to a SQLDA structure.

varno

An index for a sqlvar host variable.

Returns

An unsigned 32-bit integer value representing the length of the C string (type DT_STRING) that would be required to hold the variable sqlda->sqlvar[varno] (no matter what its type is). It includes the null termination character.

Example

In this example, all columns will be fetched as C strings (DT_STRING). Since sqlda_string_length includes the null termination character, the sqllen field is set to exclude this character. The new sqltype must be set after the call to sqlda_string_length.

```
for( col = 0; col < sqlda->sqld; col++ ) {
    sqlda->sqlvar[col].sqllen = sqlda_string_length( sqlda, col ) - 1;
    sqlda->sqlvar[col].sqltype = DT_STRING;
}
fill_sqlda( sqlda );
```

Related Information

The SQL Descriptor Area (SQLDA) [page 300]

1.10.23.35 sqlerror_message Function

Obtain the error message text.

Syntax

```
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

Parameters

sqlca

A pointer to a SQLCA structure.

buffer

The buffer in which to place the message (up to `max` characters).

max

The maximum length of the buffer.

Returns

A pointer to a string that contains an error message or NULL if no error was indicated.

Remarks

Returns a pointer to a string that contains an error message. The error message contains text for the error code in the SQLCA. If no error was indicated, a null pointer is returned. The error message is placed in the buffer supplied, truncated to length `max` if necessary.

Related Information

The SQL Communication Area (SQLCA) [page 290]

1.10.24 Embedded SQL Statement Summary

All Embedded SQL statements must be preceded with EXEC SQL and end with a semicolon (;).

There are two groups of Embedded SQL statements. Standard SQL statements are used by simply placing them in a C program enclosed with EXEC SQL and a semicolon (;). CONNECT, DELETE, SELECT, SET, and UPDATE have additional formats only available in Embedded SQL. The additional formats fall into the second category of Embedded SQL specific statements.

Several SQL statements are specific to Embedded SQL and can only be used in a C program.

Standard data manipulation and data definition statements can be used from Embedded SQL applications. In addition, the following statements are specifically for Embedded SQL programming:

CLOSE statement [ESQL] [SP]

Close a cursor.

CONNECT statement [ESQL] [Interactive SQL]

Connect to the database.

DEALLOCATE DESCRIPTOR statement [ESQL]

Reclaim memory for a descriptor.

Declaration section [ESQL]

Declare host variables for database communication.

DECLARE CURSOR statement [ESQL] [SP]

Declare a cursor.

DELETE statement (positioned) [ESQL] [SP]

Delete the row at the current position in a cursor.

DESCRIBE statement [ESQL]

Describe the host variables for a particular SQL statement.

DISCONNECT statement [ESQL] [Interactive SQL]

Disconnect from database server.

DROP STATEMENT statement [ESQL]

Free resources used by a prepared statement.

EXECUTE statement [ESQL]

Execute a particular SQL statement.

EXPLAIN statement [ESQL]

Explain the optimization strategy for a particular cursor.

FETCH statement [ESQL] [SP]

Fetch a row from a cursor.

GET DATA statement [ESQL]

Fetch long values from a cursor.

GET DESCRIPTOR statement [ESQL]

Retrieve information about a variable in a SQLDA.

GET OPTION statement [ESQL]

Get the setting for a particular database option.

INCLUDE statement [ESQL]

Include a file for SQL preprocessing.

OPEN statement [ESQL] [SP]

Open a cursor.

PREPARE statement [ESQL]

Prepare a particular SQL statement.

PUT statement [ESQL]

Insert a row into a cursor.

SET CONNECTION statement [Interactive SQL] [ESQL]

Change active connection.

SET DESCRIPTOR statement [ESQL]

Describe the variables in a SQLDA and place data into the SQLDA.

SET SQLCA statement [ESQL]

Set SQLCA to something other than the default global one.

UPDATE (positioned) statement [ESQL] [SP]

Update the row at the current location of a cursor.

WHENEVER statement [ESQL]

Specify actions to occur on errors in SQL statements.

Related Information

SQL Language Elements
ALLOCATE DESCRIPTOR Statement [ESQL]
CLOSE statement [ESQL] [SP]
CONNECT Statement [ESQL] [Interactive SQL]
DEALLOCATE DESCRIPTOR Statement [ESQL]
Declaration Section [ESQL]
DECLARE CURSOR Statement [ESQL] [SP]
DELETE Statement (Positioned) [ESQL] [SP]
DESCRIBE Statement [ESQL]
DISCONNECT Statement [ESQL] [Interactive SQL]
DROP STATEMENT Statement [ESQL]
EXECUTE Statement [ESQL]
EXPLAIN Statement [ESQL]
FETCH Statement [ESQL] [SP]
GET DATA Statement [ESQL]
GET DESCRIPTOR Statement [ESQL]
GET DESCRIPTOR Statement [ESQL]
GET OPTION Statement [ESQL]
INCLUDE Statement [ESQL]
OPEN Statement [ESQL] [SP]
PREPARE Statement [ESQL]
PUT Statement [ESQL]
SET CONNECTION Statement [Interactive SQL] [ESQL]
SET DESCRIPTOR Statement [ESQL]
SET SQLCA Statement [ESQL]
UPDATE (Positioned) Statement [ESQL] [SP]
WHENEVER Statement [ESQL]

1.11 SQL Anywhere Database API for C/C++

The SQL Anywhere C application programming interface (API) is a data access API for the C / C++ languages.

The C API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using the C API, your C / C++ applications have direct access to SQL Anywhere database servers.

In this section:

SQL Anywhere C API Support [page 389]
The SQL Anywhere C Application Programming Interface (API) is a data access API for the C / C++ languages. The C API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used.

SQL Anywhere C API Reference [page 390]
The specific C API elements are defined in header files.

1.11.1 SQL Anywhere C API Support

The SQL Anywhere C Application Programming Interface (API) is a data access API for the C / C++ languages. The C API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used.

Using the C API, your C / C++ applications have direct access to databases running on SQL Anywhere database servers.

The C API is layered on top of the DBLIB package and it is implemented with Embedded SQL. Although it is not a replacement for DBLIB, the C API simplifies the creation of applications using C and C++. You do not need an advanced knowledge of Embedded SQL to use the C API.

The C API also simplifies the creation of C and C++ wrapper drivers for several interpreted programming languages including PHP, Perl, Python, and Ruby.

API Distribution

The API is built as a dynamic link library (DLL) (`dbcapi.dll`) on Microsoft Windows systems and as a shared object (`libdbcapi.so`) on UNIX and Linux systems. The DLL is statically linked to the DBLIB package of the software version on which it is built. When the `dbcapi.dll` file is loaded, the corresponding `dblibX.dll` file is loaded by the operating system. Applications using `dbcapi.dll` can either link directly to it or load it dynamically.

Descriptions of the C API data types and entry points are provided in the `sacapi.h` header file which is located in the `sdk\dbcapi` directory of your software installation.

Threading Support

The C API library is thread-unaware; the library does not perform any tasks that require mutual exclusion. To allow the library to work in threaded applications, only one request is allowed on a single connection. With this rule, the application is responsible for doing mutual exclusion when accessing any connection-specific resource. This includes connection handles, prepared statements, and result set objects.

C API Examples

Examples that show how to use the C API can be found in the `sdk\dbcapi\examples` subdirectory of your database software installation.

callback.cpp

This is an example of how to create and use callbacks.

connecting.cpp

This is an example of how to create a connection object and use it to connect to a database.

dbcapi_isql.cpp

This example shows how to write an ISQL-like application.

fetching_a_result_set.cpp

This example shows how to fetch data from a result set.

fetching_multiple_from_sp.cpp

This example shows how to fetch multiple result sets from a stored procedure.

preparing_statements.cpp

This example shows how to prepare and execute a statement.

send_retrieve_full_blob.cpp

This example shows how to insert and retrieve a blob in one chunk.

send_retrieve_part_blob.cpp

This example shows how to insert a blob in chunks and how to retrieve it in chunks as well.

In this section:

Dynamic Loading of the Interface Library [page 390]
Use dynamic linking to access the methods in the C API library.

1.11.1.1 Dynamic Loading of the Interface Library

Use dynamic linking to access the methods in the C API library.

The code to dynamically load the DLL is contained in the `sacapidll.c` source file which is located in the `sdk\dbcapi` subdirectory of your software installation. Applications must use the `sacapidll.h` header file and include the source code in `sacapidll.c`. You can use the sqlany_initialize_interface method to dynamically load the DLL and look up the entry points. Examples are provided with the software installation.

1.11.2 SQL Anywhere C API Reference

The specific C API elements are defined in header files.

Header Files

- `sacapi.h`
- `sacapidll.h`

Remarks

The `sacapi.h` header file defines the SQL Anywhere C API entry points.

The `sacapidll.h` header file defines the C API library initialization and finalization functions. You must include `sacapidll.h` in your source files and include the source code from `sacapidll.c`.

The SQL Anywhere C API reference is available in the *SQL Anywhere- C API Reference* at https://help.sap.com/viewer/b86b137c54474b11b955a4d16358b208/LATEST/en-US.

1.12 External Call Interface

You can call a function in an external library from a stored procedure or function.

You can call functions in a DLL under Windows operating systems and in a shared object on Unix.

The material that follows describes how to use the external function call interface. Sample external stored procedures, plus the files required to build a DLL containing them, are located in the following folder: *%SQLANYSAMP17%*`\SQLAnywhere\ExternalProcedures`.

> ⚠ Caution
>
> External libraries called from procedures share the memory of the server. If you call an external library from a procedure and the external library contains memory-handling errors, you can crash the server or corrupt your database. Ensure that you thoroughly test your libraries before deploying them on production databases.

The interface described replaces an older interface, which has been deprecated. Libraries written to the older interface, used in versions before version 7.0.x, are still supported, but in any new development, the new interface is recommended. The new interface must be used for all Unix platforms and for all 64-bit platforms, including 64-bit Windows.

The database server includes a set of system procedures that make use of this capability, for example to send MAPI email messages.

In this section:

Procedures and Functions That Use External Calls [page 392]
You can create a SQL stored procedure that calls a C/C++ function in a library (a Dynamic Link Library (DLL) or shared object) as follows:

External Function Prototypes [page 393]
The interface that you use for functions written in C or C++ is defined by two header files named `dllapi.h` and `extfnapi.h`, in the `SDK\Include` subdirectory of your software installation directory. These header files handle the platform-dependent features of external function prototypes.

External Function Call Interface Methods [page 407]
The external function call interface methods are used to send and retrieve data to the database server.

Data Type Handling [page 411]
Most SQL data types can be passed to an external library.

How to Unload an External Library [page 413]
The system procedure sa_external_library_unload can be used to unload an external library when the library is not in use.

Related Information

MAPI and SMTP System Procedures

1.12.1 Procedures and Functions That Use External Calls

You can create a SQL stored procedure that calls a C/C++ function in a library (a Dynamic Link Library (DLL) or shared object) as follows:

```
CREATE PROCEDURE coverProc( parameter-list )
  EXTERNAL NAME 'myFunction@myLibrary'
  LANGUAGE C_ESQL32;
```

You must have the CREATE EXTERNAL REFERENCE system privilege to create procedures or functions that reference external libraries.

When you define a stored procedure or function in this way, you are creating a bridge to the function in the external DLL. The stored procedure or function cannot perform any other tasks.

Similarly, you can create a SQL stored function that calls a C/C++ function in a library as follows:

```
CREATE FUNCTION coverFunc( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'myFunction@myLibrary'
  LANGUAGE C_ESQL32;
```

In these statements, the EXTERNAL NAME clause indicates the function name and library in which it resides. In the example, `myFunction` is the exported name of a function in the library, and `myLibrary` is the name of the library (for example, `myLibrary.dll` or `myLibrary.so`).

The LANGUAGE clause indicates that the function is to be called in an external environment. The LANGUAGE clause can specify one of C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64. The 32 or 64 suffix indicates that the function is compiled as a 32-bit or 64-bit application. The ODBC designation indicates that the application uses the ODBC API. The ESQL designation indicates that the application could use the Embedded SQL API, the C API, any other non-ODBC API, or no API at all.

If the LANGUAGE clause is omitted, then the library containing the function is loaded into the address space of the database server. When called, the external function will execute as part of the server. In this case, if the function causes a fault, then the database server is terminated. For this reason, loading and executing functions in an external environment is recommended. If a function causes a fault in an external environment, the database server will continue to run.

The arguments in `parameter-list` must correspond in type and order to the arguments expected by the library function. The library function accesses the procedure arguments using a special interface.

Any value or result set returned by the external function can be returned by the stored procedure or function to the calling environment.

A stored procedure or function that references an external function can include no other statements: its sole purpose is to take arguments for a function, call the function, and return any value and returned arguments from the function to the calling environment. You can use IN, INOUT, or OUT parameters in the procedure call in the same way as for other procedures: the input values get passed to the external function, and any parameters modified by the function are returned to the calling environment in OUT or INOUT parameters or as the RETURNS result of the stored function.

You can specify operating-system dependent calls, so that a procedure calls one function when run on one operating system, and another function (presumably analogous) on another operating system. The syntax for such calls involves prefixing the function name with the operating system name. The operating system identifier must be *unix* for UNIX and Linux systems. An example follows.

```
CREATE FUNCTION func ( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'unix:function-name@library.so;function-name@library.dll';
```

If the list of functions does not contain an entry for the operating system on which the server is running, but the list does contain an entry without an operating system specified, the database server calls the function in that entry.

Related Information

External Environment Support [page 413]
External Function Prototypes [page 393]
CREATE FUNCTION Statement [Web Service]
CREATE FUNCTION Statement [External Call]

1.12.2 External Function Prototypes

The interface that you use for functions written in C or C++ is defined by two header files named `dllapi.h` and `extfnapi.h`, in the `SDK\Include` subdirectory of your software installation directory. These header files handle the platform-dependent features of external function prototypes.

Function Prototypes

The name of the function must match that referenced in the CREATE PROCEDURE or CREATE FUNCTION statement. Suppose the following CREATE FUNCTION statement had been executed.

```
CREATE FUNCTION cover-name ( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'function-name@library.dll';
```

The C/C++ function definition takes the following form:

```
#include "dllapi.h"
```

```
#include "extfnapi.h"
extern "C"
{
    void _entry function-name( an_extfn_api *api, void *argument_handle )
    {
        // ...
    }
}
```

The function must return void, and must take as arguments a pointer to a structure used to call a set of callback functions and a handle to the arguments provided by the SQL procedure.

Example

The following example implements a function called `upstring` that converts a string to uppercase letters and returns the result..

```
#include "dllapi.h"
#include "extfnapi.h"
extern "C"
{
    a_sql_uint32 _entry extfn_use_new_api(void)
    {
        return(EXTFN_API_VERSION);
    }
    void _callback extfn_cancel(void *cancel_handle)
    {
        *(short *)cancel_handle = 1;
    }
    void _entry upstring(an_extfn_api *api, void *arg_handle)
    {
        short               result;
        short               canceled;
        an_extfn_value      arg;
        an_extfn_value      retval;
        a_sql_data_type     data_type;
        unsigned            offset;
        char                *string;
        canceled = 0;
        api->set_cancel(arg_handle, &canceled);
        result = api->get_value(arg_handle, 1, &arg);
        if (canceled || result == 0 || arg.data == NULL)
        {
            return; // no parameter or parameter is NULL
        }
        data_type = arg.type & DT_TYPES;
        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;
            if (canceled) break;
            result = api->get_piece(arg_handle, 1, &arg, offset);
        }
        if (!canceled)
        {
            switch (data_type)
            {
```

```
            case DT_NSTRING:
            case DT_NFIXCHAR:
            case DT_NVARCHAR:
            case DT_LONGNVARCHAR:
            case DT_STRING:
            case DT_FIXCHAR:
            case DT_VARCHAR:
            case DT_LONGVARCHAR:
                _strupr_s(string, (size_t)offset + 1);
                retval.type = DT_LONGVARCHAR;
                retval.data = string;
                retval.piece_len = retval.len.total_len =
(a_sql_uint32)strlen(string);
                api->set_value(arg_handle, 0, &retval, 0);
            default:
                break;
            }
        }
        free(string);
        return;
    }
}
```

On Windows platforms, this C code must be compiled and linked such that the entry points are exported from the Dynamic Link Library (DLL) with undecorated names. This is usually accomplished with a linker EXPORTS file such as the following example:

```
EXPORTS
extfn_use_new_api
extfn_cancel
upstring
```

The following is an example of a SQL statement that calls the `upstring` function.

```
CREATE OR REPLACE FUNCTION upstring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'upstring@mystring.dll';
SELECT upstring('Hello world!');
```

In this section:

extfn_use_new_api Method [page 396]
To notify the database server that the external library is written using the external function call interface, your external library must export the extfn_use_new_api function.

extfn_cancel Method [page 397]
To notify the database server that the external library supports cancel processing, your external library must export the extfn_cancel function.

extfn_post_load_library Method [page 398]
If the extfn_post_load_library function is implemented and exposed in the external library, it is executed by the database server after the external library has been loaded and the version check has been performed, and before any other function defined in the external library is called.

extfn_pre_unload_library Method [page 399]
If the extfn_pre_unload_library function is implemented and exposed in the external library, it is executed by the database server immediately before unloading the external library.

an_extfn_api Structure [page 399]

The an_extfn_api structure is used to communicate with the calling SQL environment. This structure is defined by the header file named `extfnapi.h`, in the `SDK\Include` subdirectory of your software installation directory.

an_extfn_value Structure [page 402]
The an_extfn_value structure is used to access parameter data from the calling SQL environment.

an_extfn_result_set_info Structure [page 403]
The an_extfn_result_set_info structure facilitates the return of result sets to the calling SQL environment.

an_extfn_result_set_column_info Structure [page 404]
The an_extfn_result_set_column_info structure is used to describe a result set.

an_extfn_result_set_column_data Structure [page 406]
The an_extfn_result_set_column_data structure is used to return the data values for columns.

1.12.2.1 extfn_use_new_api Method

To notify the database server that the external library is written using the external function call interface, your external library must export the extfn_use_new_api function.

Syntax

```
extern "C" a_sql_uint32 SQL_CALLBACK extfn_use_new_api( void );
```

Returns

The function returns an unsigned 32-bit integer. The returned value must be the interface version number, EXTFN_API_VERSION, defined in `extfnapi.h`. A return value of 0 means that the old, deprecated interface is being used.

Remarks

If the function is not exported by the library, the database server assumes that the old interface is in use. The new interface must be used for all UNIX and Linux platforms and for all 64-bit platforms, including 64-bit Microsoft Windows.

A typical implementation of this function follows:

```
extern "C" a_sql_uint32 SQL_CALLBACK extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}
```

Related Information

an_extfn_api Structure [page 399]

1.12.2.2 extfn_cancel Method

To notify the database server that the external library supports cancel processing, your external library must export the extfn_cancel function.

Syntax

```
extern "C" void _callback extfn_cancel( void *cancel_handle );
```

Parameters

cancel_handle

A pointer to a variable to manipulate.

Remarks

This function is called asynchronously by the database server whenever the currently executing SQL statement is canceled.

The function uses the cancel_handle to set a flag indicating to the external library functions that the SQL statement has been canceled.

If the function is not exported by the library, the database server assumes that cancel processing is not supported.

A typical implementation of this function follows:

```
#include "dllapi.h"
extern "C"
{
    void _callback extfn_cancel(void *cancel_handle)
    {
        *(short *)cancel_handle = 1;
    }
}
```

Related Information

an_extfn_api Structure [page 399]

1.12.2.3 extfn_post_load_library Method

If the extfn_post_load_library function is implemented and exposed in the external library, it is executed by the database server after the external library has been loaded and the version check has been performed, and before any other function defined in the external library is called.

Syntax

```
extern "C" void _entry extfn_post_load_library( void );
```

Remarks

This function is required only if there is a library-specific requirement to do library-wide setup before any function within the library is called.

This function is called asynchronously by the database server after the external library has been loaded and the version check has been performed, and before any other function defined in the external library is called.

Example

```
#include "dllapi.h"
extern "C"
{
    void _entry extfn_post_load_library( void )
    {
        MessageBox(NULL, L"Library loaded", L"Application Notification", MB_OK |
MB_TASKMODAL);
    }
}
```

Related Information

an_extfn_api Structure [page 399]

1.12.2.4 extfn_pre_unload_library Method

If the extfn_pre_unload_library function is implemented and exposed in the external library, it is executed by the database server immediately before unloading the external library.

Syntax

```
extern "C" void _entry extfn_pre_unload_library( void );
```

Remarks

This function is required only if there is a library-specific requirement to do library-wide cleanup before the library is unloaded.

This function is called asynchronously by the database server immediately before unloading the external library.

Example

```
#include "dllapi.h"
extern "C"
{
    void _entry extfn_post_load_library( void )
    {
        MessageBox(NULL, L"Library unloading", L"Application Notification",
MB_OK | MB_TASKMODAL);
    }
}
```

Related Information

an_extfn_api Structure [page 399]

1.12.2.5 an_extfn_api Structure

The an_extfn_api structure is used to communicate with the calling SQL environment. This structure is defined by the header file named `extfnapi.h`, in the `SDK\Include` subdirectory of your software installation directory.

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;
```

Properties

get_value

Use this callback function to get the specified parameter's value. The following example gets the value for parameter 1.

```
result = api->get_value( arg_handle, 1, &arg )
if( result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}
```

get_piece

Use this callback function to get the next chunk of the specified parameter's value (if there are any). The following example gets the remaining pieces for parameter 1.

```
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 = api->get_piece( arg_handle, 1, &arg, offset );
}
```

set_value

Use this callback function to set the specified parameter's value. The following example sets the return value (parameter 0) for a RETURNS clause of a FUNCTION.

```
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 );
api->set_value( arg_handle, 0, &retval, 0 );
```

set_cancel

Use this callback function to establish a pointer to a variable that can be set by the extfn_cancel method. The following is example.

```
short               canceled = 0;
api->set_cancel( arg_handle, &canceled );
```

Remarks

A pointer to the an_extfn_api structure is passed by the caller to your external function. Here is an example.

```
#include "dllapi.h"
#include "extfnapi.h"
extern "C"
{
    void _entry upstring(an_extfn_api *api, void *arg_handle)
    {
        short               result;
        short               canceled;
        an_extfn_value      arg;

        canceled = 0;
        api->set_cancel( arg_handle, &canceled );

        result = api->get_value( arg_handle, 1, &arg );
        if( canceled || result == 0 || arg.data == NULL )
        {
            return; // no parameter or parameter is NULL
        }
        .
        .
        .
    }
}
```

Whenever you use any of the callback functions, you must pass back the argument handle that was passed to your external function as the second parameter.

Related Information

an_extfn_value Structure [page 402]
extfn_cancel Method [page 397]

1.12.2.6 an_extfn_value Structure

The an_extfn_value structure is used to access parameter data from the calling SQL environment.

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;
```

Properties

data

A pointer to the data for this parameter.

piece_len

The length of this segment of the parameter. This is less than or equal to total_len.

total_len

The total length of the parameter. For strings, this represents the length of the string and does not include a null terminator. This property is set after a call to the get_value callback function. This property is no longer valid after a call to the get_piece callback function.

remain_len

When the parameter is obtained in segments, this is the length of the part that has not yet been obtained. This property is set after each call to the get_piece callback function.

type

Indicates the type of the parameter. This is one of the Embedded SQL data types such as DT_INT, DT_FIXCHAR, or DT_BINARY.

Remarks

Suppose that your external function interface was described using the following SQL statement.

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mystring.dll';
```

The following code fragment shows how to access the properties for objects of type an_extfn_value. In the example, the input parameter 1 (`instr`) to this function (`mystring`) is expected to be a SQL LONGVARCHAR string.

```
an_extfn_value      arg;
result = api->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 = api->get_piece( arg_handle, 1, &arg, offset );
}
```

Related Information

Embedded SQL Data Types [page 277]
an_extfn_api Structure [page 399]

1.12.2.7 an_extfn_result_set_info Structure

The an_extfn_result_set_info structure facilitates the return of result sets to the calling SQL environment.

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;
```

Properties

number_of_columns

The number of columns in the result set.

column_infos

Link to a description of the result set columns.

column_data_values

Link to a description of the result set column data.

Remarks

The following code fragment shows how to set the properties for objects of this type.

```
int columns = 2;
an_extfn_result_set_info    rs_info;
rs_info.number_of_columns   = columns;
rs_info.column_infos        = col_info;
rs_info.column_data_values  = col_data;
```

Related Information

an_extfn_result_set_column_info Structure [page 404]
an_extfn_result_set_column_data Structure [page 406]

1.12.2.8 an_extfn_result_set_column_info Structure

The an_extfn_result_set_column_info structure is used to describe a result set.

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;
```

Properties

column_name

Points to the name of the column which is a null-terminated string.

column_type

Indicates the type of the column. This is one of the Embedded SQL data types such as DT_INT, DT_FIXCHAR, or DT_BINARY.

column_width

Defines the maximum width for char(n), varchar(n) and binary(n) declarations and is set to 0 for all other types.

column_index

The ordinal position of the column which starts at 1.

column_can_be_null

Set to 1 if the column is nullable; otherwise it is set to 0.

Remarks

The following code fragment shows how to set the properties for objects of this type and how to describe the result set to the calling SQL environment.

```
// set up column descriptions
an_extfn_result_set_column_info *col_info =
    (an_extfn_result_set_column_info *)
    malloc( columns * sizeof(an_extfn_result_set_column_info) );
// 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;
api->set_value( arg_handle,
                EXTFN_RESULT_SET_ARG_NUM,
                (an_extfn_value *)&rs_info,
                EXTFN_RESULT_SET_DESCRIBE );
```

Related Information

Embedded SQL Data Types [page 277]
an_extfn_result_set_info Structure [page 403]
an_extfn_result_set_column_data Structure [page 406]

1.12.2.9 an_extfn_result_set_column_data Structure

The an_extfn_result_set_column_data structure is used to return the data values for columns.

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;
```

Properties

column_index

The ordinal position of the column which starts at 1.

column_data

Pointer to a buffer containing the column data.

data_length

The actual length of the data.

append

Used to return the column value in chunks. Set to 1 when returning a partial column value; 0 otherwise.

Remarks

The following code fragment shows how to set the properties for objects of this type and how to return the result set row to the calling SQL environment.

```
int DeptNumber = 400;
char * DeptName = "Marketing";
an_extfn_result_set_column_data *col_data =
    (an_extfn_result_set_column_data *)
    malloc( columns * sizeof(an_extfn_result_set_column_data) );
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;
api->set_value(  arg_handle,
                EXTFN_RESULT_SET_ARG_NUM,
                (an_extfn_value *)&rs_info,
                EXTFN_RESULT_SET_NEW_ROW_FLUSH );
```

Related Information

an_extfn_result_set_info Structure [page 403]
an_extfn_result_set_column_info Structure [page 404]

1.12.3 External Function Call Interface Methods

The external function call interface methods are used to send and retrieve data to the database server.

get_value Callback

```
short (SQL_CALLBACK *get_value)
(
    void *         arg_handle,
    a_sql_uint32   arg_num,
    an_extfn_value *value
);
```

The get_value callback function can be used to obtain the value of a parameter that was passed to the stored procedure or function that acts as the interface to the external function. It returns 0 if not successful; otherwise it returns a non-zero result. After calling get_value, the total_len field of the an_extfn_value structure contains the length of the entire value. The piece_len field contains the length of the portion that was obtained as a result of calling get_value. The piece_len field will always be less than or equal to total_len. When it is less than, a second function get_piece can be called to obtain the remaining pieces. The total_len field is only valid after the initial call to get_value. This field is overlaid by the remain_len field which is altered by calls to get_piece. It is important to preserve the value of the total_len field immediately after calling get_value if you plan to use it later on.

get_piece Callback

```
short (SQL_CALLBACK *get_piece)
(
    void *         arg_handle,
    a_sql_uint32   arg_num,
    an_extfn_value *value,
    a_sql_uint32   offset
);
```

If the entire parameter value cannot be returned in one piece, then the get_piece function can be called iteratively to obtain the remaining pieces of the parameter value.

The sum of all the piece_len values returned by both calls to get_value and get_piece will add up to the initial value that was returned in the total_len field after calling get_value. After calling get_piece, the remain_len field, which overlays total_len, represents the amount not yet obtained.

Using get_value and get_piece Callbacks

The following example shows the use of get_value and get_piece to obtain the value of a string parameter such as a long varchar parameter.

Suppose that the wrapper to an external function was declared as follows:

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
  EXTERNAL NAME 'mystring@mystring.dll';
```

To call the external function from SQL, use a statement like the following.

```
CALL mystring('Hello world!');
```

A sample implementation for the Windows operating system of the mystring function, written in C, follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include "dllapi.h"
#include "extfnapi.h"
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved )
{
    return TRUE;
}
extern "C"
{
    a_sql_uint32 _entry extfn_use_new_api( void )
    {
        return( EXTFN_API_VERSION );
    }
    void _entry mystring( an_extfn_api *api, void *arg_handle )
    {
        short               result;
        an_extfn_value      arg;
        unsigned            offset;
        char                *string;
        result = api->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 = api->get_piece( arg_handle, 1, &arg, offset );
        }
        MessageBoxA( NULL, string,
              "Application Notification",
              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
);
```

The set_value callback function can be used to set the values of OUT parameters and the RETURNS result of a stored function. Use an arg_num value of 0 to set the RETURNS value. The following is an example.

```
an_extfn_value      retval;
retval.type = DT_LONGVARCHAR;
retval.data = result;
retval.piece_len = retval.len.total_len = (a_sql_uint32) strlen( result );
api->set_value( arg_handle, 0, &retval, 0 );
```

The append argument of set_value determines whether the supplied data replaces (false) or appends to (true) the existing data. You must call set_value with `append=FALSE` before calling it with `append=TRUE` for the same argument. The append argument is ignored for fixed-length data types.

To return NULL, set the data field of the an_extfn_value structure to NULL.

set_cancel Callback

```
void (SQL_CALLBACK *set_cancel)
(
    void *arg_handle,
    void *cancel_handle
);
```

External functions can get the values of IN or INOUT parameters and set the values of OUT parameters and the RETURNS result of a stored function. There is a case, however, where the parameter values obtained may no longer be valid or the setting of values is no longer necessary. This occurs when an executing SQL statement is canceled. This may occur as the result of an application abruptly disconnecting from the database server. To handle this situation, you can define a special entry point in the library called extfn_cancel. When this function is defined, the server will call it whenever a running SQL statement is canceled.

The extfn_cancel function is called with a handle that can be used in any way you consider suitable. A typical use of the handle is to indirectly set a flag to indicate that the calling SQL statement has been canceled.

The value of the handle that is passed can be set by functions in the external library using the set_cancel callback function. For example:

```
void _callback extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}

void _entry mystring( an_extfn_api *api, void *arg_handle )
{
.
.
```

```
.
    short  canceled = 0;
    api->set_cancel( arg_handle, &canceled );
.
.
.
    if( canceled )
```

Setting a static global "canceled" variable is inappropriate since that would be misinterpreted as all SQL statements on all connections being canceled which is usually not the case. This is why a set_cancel callback function is provided. Make sure to initialize the "canceled" variable before calling set_cancel.

It is important to check the setting of the "canceled" variable at strategic points in your external function. Strategic points would include before and after calling any of the external library call interface functions like get_value and set_value. When the variable is set (as a result of extfn_cancel having been called), then the external function can take appropriate termination action. A code fragment based on the earlier example follows:

```
if( canceled )
{
    free( string );
    return;
}
```

Notes

The get_piece function for any given argument can only be called immediately after the get_value function for the same argument.

Calling get_value on an OUT parameter returns the type field of the an_extfn_value structure set to the data type of the argument, and returns the data field of the an_extfn_value structure set to NULL.

The header file `extfnapi.h` in the database server software `SDK\Include` folder contains some additional notes.

The following table shows the conditions under which the functions defined in an_extfn_api return false:

Function	Returns 0 when the Following Is True; Else Returns 1
get_value()	• arg_num is invalid; for example, arg_num is greater than the number of arguments for the external function.
get_piece()	• arg_num is invalid; for example, arg_num does not correspond to the argument number used with the previous call to get_value. • The offset is greater than the total length of the value for the arg_num argument. • It is called before get_value has been called.

Function	Returns 0 when the Following Is True; Else Returns 1
set_value()	• arg_num is invalid; for example, arg_num is greater than the number of arguments for the external function. • Argument arg_num is input only. • The type of value supplied does not match that of argument arg_num.

1.12.4 Data Type Handling

Most SQL data types can be passed to an external library.

Data types

The following SQL data types can be passed to an external library:

SQL Data Type	sqldef.h	C Type
CHAR	DT_FIXCHAR	Character data, with a specified length
VARCHAR	DT_VARCHAR	Character data, with a specified length
LONG VARCHAR, TEXT	DT_LONGVARCHAR	Character data, with a specified length
UNIQUEIDENTIFIERSTR	DT_FIXCHAR	Character data, with a specified length
XML	DT_LONGVARCHAR	Character data, with a specified length
NCHAR	DT_NFIXCHAR	UTF-8 character data, with a specified length
NVARCHAR	DT_NVARCHAR	UTF-8 character data, with a specified length
LONG NVARCHAR, NTEXT	DT_LONGNVARCHAR	UTF-8 character data, with a specified length
UNIQUEIDENTIFIER	DT_BINARY	Binary data, 16 bytes long
BINARY	DT_BINARY	Binary data, with a specified length
VARBINARY	DT_BINARY	Binary data, with a specified length
LONG BINARY	DT_LONGBINARY	Binary data, with a specified length
TINYINT	DT_TINYINT	1-byte integer
[UNSIGNED] SMALLINT	DT_SMALLINT, DT_UNSSMALLINT	[Unsigned] 2-byte integer
[UNSIGNED] INT	DT_INT, DT_UNSINT	[Unsigned] 4-byte integer
[UNSIGNED] BIGINT	DT_BIGINT, DT_UNSBIGINT	[Unsigned] 8-byte integer
REAL, FLOAT(1-24)	DT_FLOAT	Single-precision floating-point number

SQL Data Type	sqldef.h	C Type
DOUBLE, FLOAT(25-53)	DT_DOUBLE	Double-precision floating-point number

You cannot use any of the date or time data types, and you cannot use the DECIMAL or NUMERIC data types (including the money types).

To provide values for INOUT or OUT parameters, use the set_value function. To read IN and INOUT parameters, use the get_value function.

Determining Data Types of Parameters

After a call to get_value, the type field of the an_extfn_value structure can be used to obtain data type information for the parameter. The following sample code fragment shows how to identify the type of the parameter.

```
an_extfn_value      arg;
a_sql_data_type     data_type;
api->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;
}
```

UTF-8 Types

The UTF-8 data types such as NCHAR, NVARCHAR, LONG NVARCHAR and NTEXT as passed as UTF-8 encoded strings. A function such as the Windows MultiByteToWideChar function can be used to convert a UTF-8 string to a wide-character (Unicode) string.

Passing NULL

You can pass NULL as a valid value for all arguments. Functions in external libraries can supply NULL as a return value for any data type.

Return Values

To set a return value in an external function, call the set_value function with an arg_num parameter value of 0. If set_value is not called with arg_num set to 0, the function result is NULL.

It is also important to set the data type of a return value for a stored function call. The following code fragment shows how to set the return data type.

```
an_extfn_value      retval;
retval.type = DT_LONGVARCHAR;
retval.data = result;
retval.piece_len = retval.len.total_len = (a_sql_uint32) strlen( result );
api->set_value( arg_handle, 0, &retval, 0 );
```

Related Information

Host Variables in Embedded SQL [page 281]

1.12.5 How to Unload an External Library

The system procedure sa_external_library_unload can be used to unload an external library when the library is not in use.

The procedure takes one optional parameter, a long varchar. The parameter specifies the library to be unloaded and must match the file path string used to load the library. If no parameter is specified, all external libraries not in use are unloaded.

The following example unloads an external function library.

```
CALL sa_external_library_unload( 'library.dll' );
```

This function is useful when developing a set of external functions because you do not have to shut down the database server to install a newer version of the library.

The following example shows how to use this system procedure.

```
CREATE OR REPLACE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@c:\\projects\\mystring\\debug\\mystring.dll';
CALL mystring('Hello world!');
CALL sa_external_library_unload( 'c:\\projects\\mystring\\debug\\mystring.dll' );
```

1.13 External Environment Support

Seven external runtime environments are supported. These include Embedded SQL and ODBC applications written in C/C++, and applications written in Java, JavaScript (Deprecated), Perl, PHP (Deprecated), or

languages such as Microsoft C# and Microsoft Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

While it is possible to call compiled native functions in a dynamic link library or shared object that is loaded by the database server into its own address space, the risk here is that if the native function causes a fault, then the database server crashes. Running compiled native functions outside the database server, in an external environment, eliminates these risks to the server.

The following is an overview of the external environment support.

- The START EXTERNAL ENVIRONMENT and STOP EXTERNAL ENVIRONMENT statements are used to start or stop an external environment on demand. These statements are optional since external environments are automatically started and stopped when needed.
- The ALTER EXTERNAL ENVIRONMENT statement is used to set or modify the location of an external environment.
- The COMMENT ON EXTERNAL ENVIRONMENT statement is used to add a comment for an external environment.
- Once an external environment is set up to be used on the database server, you can then install objects into the database and create stored procedures and functions that make use of these objects within the external environment.
- The INSTALL EXTERNAL OBJECT statement is used to install a JavaScript (Deprecated), Perl or PHP (Deprecated) external object (for example, a Perl script) from a file or an expression into the database. Once the external objects are installed in the database, they can be used within external stored procedure and function definitions.
- The COMMENT ON EXTERNAL ENVIRONMENT OBJECT statement is used to add a comment for an external environment object.
- To remove an installed JavaScript (Deprecated), Perl or PHP (Deprecated) external object from the database, you use the REMOVE EXTERNAL OBJECT statement.
- The CREATE PROCEDURE and CREATE FUNCTION statements are used to create external stored procedure and function definitions. They can be used like any other stored procedure or function in the database. The database server, when encountering an external environment stored procedure or function, automatically launches the external environment (if it has not already been started), and sends over whatever information is needed to get the external environment to fetch the external object from the database and execute it. Any result sets or return values resulting from the execution are returned as needed.
- All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

In this section:

The CLR External Environment [page 415]
CLR stored procedures and functions can be called from within the database in the same manner as SQL stored procedures.

The ESQL and ODBC External Environments [page 419]
External compiled native functions that use Embedded SQL or ODBC can be called from the database in the same manner as SQL stored procedures.

The Java External Environment [page 427]

Java methods can be called from the database in the same manner as SQL stored procedures.

The JavaScript External Environment (Deprecated) [page 432]
JavaScript stored procedures and functions can be called from the database can be called from the database in the same manner as SQL stored procedures.

The Perl External Environment [page 438]
Perl stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

The PHP External Environment (Deprecated) [page 442]
PHP stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

Related Information

ALTER EXTERNAL ENVIRONMENT Statement
COMMENT Statement
CREATE FUNCTION Statement [External Call]
CREATE PROCEDURE Statement [External Call]
INSTALL EXTERNAL OBJECT Statement
REMOVE EXTERNAL OBJECT Statement
START EXTERNAL ENVIRONMENT Statement
STOP EXTERNAL ENVIRONMENT Statement

1.13.1 The CLR External Environment

CLR stored procedures and functions can be called from within the database in the same manner as SQL stored procedures.

A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a Microsoft .NET language such as Microsoft C# or Microsoft Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate Microsoft .NET executable).

There is only one instance of this Microsoft .NET executable per database. All connections executing CLR functions and stored procedures use the same Microsoft .NET executable instance, but the namespaces for each connection are separate. Statics persist for the duration of the connection, but are not shareable across connections.

By default, the database server uses one CLR external environment for each database running on the database server. The -sclr command-line option or the SingleCLRInstanceVersion database server option can be used to request that one CLR external environment be used for all databases running on the database server. This option must be specified before starting the CLR external environment on any database.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

Note that the ADO.NET data provider is used in the CLR external environment and it operates in automatic commit mode by default using the connection of the caller. This will affect uncommitted transactions in the client application. If this is an issue, then consider using a second connection for CLR external environment calls.

To call an external CLR function or procedure, you define a corresponding stored procedure or function with an EXTERNAL NAME string defining which DLL to load and which function within the assembly to call. You must also specify LANGUAGE CLR when defining the stored procedure or function. An example declaration follows:

```
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 this example, the stored procedure called `clr_stored_proc`, when executed, loads the DLL `MyCLRTest.dll` and calls the function MyCLRTest.Run. The clr_stored_proc procedure takes three SQL parameters, two IN parameters, one of type INT and one of type UNSIGNED SMALLINT, and one OUT parameter of type LONG VARCHAR. On the Microsoft .NET side, these three parameters translate to input arguments of type int and ushort and an output argument of type string. In addition to out arguments, the CLR function can also have ref arguments. A user must declare a ref CLR argument if the corresponding stored procedure has an INOUT parameter.

The following table lists the various CLR argument types and the corresponding suggested SQL data type:

CLR Type	Recommended SQL Data Type
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

If you see the error message *Object reference not set to an instance of an object*, check that you have specified the path to the DLL.

The declaration of the DLL can be either a relative or absolute path. If the specified path is relative, then the external Microsoft .NET executable searches the path, and other locations, for the DLL. The executable does not search the Global Assembly Cache (GAC) for the DLL.

Like the existing Java stored procedures and functions, CLR stored procedures and functions can make server-side requests back to the database, and they can return result sets. Also, like Java, any information output to Console.Out and Console.Error is automatically redirected to the database server messages window.

To use CLR in the database, make sure the database server is able to locate and start the CLR external environment executable. There are several versions of this executable.

Table 2: .NET CLR versions supported

.NET Version	File Name
3.5	`dbextclr[VER_MAJOR]`
4.x	`dbextclr[VER_MAJOR]_v4.5`

Make sure that the selected CLR external environment module is identified using the ALTER EXTERNAL ENVIRONMENT statement. The following is an example.

```
ALTER EXTERNAL ENVIRONMENT CLR LOCATION 'dbextclr[VER_MAJOR]_v4.5'
```

For portability to newer versions of the software, it is recommended that you use [VER_MAJOR] in the LOCATION string rather than the current software release version number. The database server will replace [VER_MAJOR] with the appropriate version number.

Make sure that the file corresponding to the .NET version you are using is present in the *%SQLANY17%*`\Bin32` or *%SQLANY17%*`\Bin64` folder. By default, the 64-bit database server will use the 64-bit version of the module and the 32-bit database server will use the 32-bit version of the module. Ideally, your .NET application should be targeted for the Any CPU platform.

You can verify if the database server is able to locate and start the selected CLR external environment executable by executing the following statement:

```
START EXTERNAL ENVIRONMENT CLR;
```

If the database server fails to start CLR, then the database server is likely not able to locate the CLR external environment executable. If you see a message that SAClrClassLoader has stopped working, then run SetupVSPackage to install the current version of the .NET data provider.

The START EXTERNAL ENVIRONMENT CLR statement is not necessary other than to verify that the database server can launch CLR executables. In general, making a CLR stored procedure or function call starts CLR automatically.

Similarly, the STOP EXTERNAL ENVIRONMENT CLR statement is not necessary to stop an instance of CLR since the instance automatically goes away when the connection terminates. The STOP EXTERNAL ENVIRONMENT CLR statement releases the CLR instance for your connection. There are a few reasons why you might do this.

- You are completely done with CLR and you want to free up some resources.
- You change CLR versions using the ALTER EXTERNAL ENVIRONMENT statement and the CLR instance is currently running.
- You want to reload/restart the assemblies that have been loaded by the CLR.

Unlike the Perl, PHP, and Java external environments, the CLR environment does not require the installation of anything in the database. As a result, you do not need to execute any INSTALL statements before using the CLR external environment.

Here is an example of a function written in C# that can be run within an external environment.

```
public class StaticTest
{
    private static int val = 0;
    public static int GetValue()
    {
        val += 1;
        return val;
    }
}
```

When compiled into a dynamic link library, this function can be called from an external environment. An executable image called `dbextclr`*17*`.exe` is started by the database server and it loads the dynamic link library for you. Different versions of this executable are provided. For example, on Windows you may have both 32-bit and 64-bit executables. One is for use with the 32-bit version of the database server and the other for the 64-bit version of the database server.

To build this application into a dynamic link library using the Microsoft C# compiler, use a command like the following. The source code for the above example is assumed to reside in a file called `StaticTest.cs`.

```
csc /target:library /out:clrtest.dll StaticTest.cs
```

This command places the compiled code in a DLL called `clrtest.dll`. To call the compiled Microsoft C# function, GetValue, a wrapper is defined as follows using Interactive SQL:

```
CREATE FUNCTION stc_get_value()
RETURNS INT
EXTERNAL NAME 'clrtest.dll::StaticTest.GetValue() int'
LANGUAGE CLR;
```

For CLR, the EXTERNAL NAME string is specified in a single line of SQL. You may be required to include the path to the DLL as part of the EXTERNAL NAME string so that it can be located. For dependent assemblies (for example, if `myLib.dll` has code that calls functions in, or in some way depends on, `myOtherLib.dll`) then it is up to the .NET Framework to load the dependencies. The CLR external environment will take care of loading the specified assembly, but extra steps might be required to ensure that dependent assemblies are loaded. One solution is to register all dependencies in the Global Assembly Cache (GAC) by using the Microsoft gacutil utility installed with the Microsoft .NET Framework. For custom-developed libraries, gacutil requires that these be signed with a strong name key before they can be registered in the GAC.

To execute the sample compiled C# function, execute the following statement.

```
SELECT stc_get_value();
```

Each time the Microsoft C# function is called, a new integer result is produced. The sequence of values returned is 1, 2, 3, and so on.

The following example illustrates how to make a server-side connection using C#. You create a connection object in the usual way and then assign it a value using the `SAServerSideConnection.Connection` object.

```
using Sap.Data.SQLAnywhere;
using Sap.SQLAnywhere.Server;
class test
{
    private static SAConnection _conn;
    private static void GetConnection()
    {
        if( _conn == null )
```

```
        {
            _conn = SAServerSideConnection.Connection;
        }
    }
}
```

For more information and examples on using the CLR in the database support, refer to the examples located in the *%SQLANYSAMP17%*`\SQLAnywhere\ExternalEnvironments\CLR` directory.

For more information and examples on using the CLR in the database support, how to make server-side requests, and how to return result sets from a CLR function or stored procedure, refer to the samples located in the *%SQLANYSAMP17%*`\SQLAnywhere\ExternalEnvironments\CLR` directory.

Related Information

SQL Functions
User-defined Functions
System Procedures
Procedures
Named Parameters

1.13.2 The ESQL and ODBC External Environments

External compiled native functions that use Embedded SQL or ODBC can be called from the database in the same manner as SQL stored procedures.

The database server software has long supported the loading of dynamic link libraries or shared objects containing compiled C or C++ code into its address space, and the subsequent calling of functions in these libraries. While being able to call native functions is very efficient, there can be serious consequences if the native function misbehaves. In particular, if the native function enters an infinite loop, then the database server can hang, and if the native function causes a fault, then the database server crashes.

Because of these consequences, it is much better to run compiled native functions outside of the database server's address space, in an external environment. There are some key benefits to running a compiled native function in an external environment:

- The database server does not hang or crash if the compiled native function misbehaves.
- The native function can be written to use ODBC, Embedded SQL (ESQL), or the C API and can make server-side calls back into the database server without having to make a connection.
- The native function can return a result set to the database server.
- In the external environment, a 32-bit database server can communicate with a 64-bit compiled native function and vice versa. This is not possible when the compiled native functions are loaded directly into the address space of the database server. A 32-bit library can only be loaded by a 32-bit server and a 64-bit library can only be loaded by a 64-bit server.

Running a compiled native function in an external environment instead of within the database server results in a small performance penalty.

Also, the compiled native function must use the external call interface to pass information to and return information from the native function.

To run a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute specifying one of C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

Unlike the Perl, PHP, JavaScript, and Java external environments, you do not install any source code or compiled objects in the database. As a result, you do not need to execute any INSTALL statements before using the ESQL and ODBC external environments.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

Here is an example of a function written in C++ that can be run within the database server or in an external environment.

```
#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;
}
```

When compiled into a dynamic link library or shared object, this function can be called from an external environment. An executable image called `dbexternc`*17* is started by the database server and this executable image loads the dynamic link library or shared object for you. Different versions of this executable are provided. For example, on Windows you may have both 32-bit and 64-bit executables.

Either the 32-bit or 64-bit version of the database server can be used and either version can start 32-bit or 64-bit versions of `dbexternc`*17*. This is one of the advantages of using the external environment. Once `dbexternc`*17* is started by the database server, it does not terminate until the connection has been terminated or a STOP EXTERNAL ENVIRONMENT statement (with the correct environment name) is executed. Each connection that does an external environment call will get its own copy of `dbexternc`*17*.

To call the compiled native function, SimpleCFunction, a wrapper is defined as follows:

```
CREATE FUNCTION SimpleCDemo(
  IN arg1 INT,
  IN arg2 INT,
  IN arg3 INT,
  IN arg4 INT )
RETURNS INT
EXTERNAL NAME 'SimpleCFunction@c:\\c\\extdemo.dll'
LANGUAGE C_ODBC32;
```

This is almost identical to the way a compiled native function is described when it is to be loaded into the database server's address space. The one difference is the use of the LANGUAGE C_ODBC32 clause. This clause indicates that SimpleCDemo is a function running in an external environment and that it is using 32-bit ODBC calls. The language specification of C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64 tells the database server whether the external C function issues 32-bit or 64-bit ODBC, ESQL, or C API calls when making server-side requests.

When the native function uses none of the ODBC, ESQL, or C API calls to make server-side requests, then either C_ODBC32 or C_ESQL32 can be used for 32-bit applications and either C_ODBC64 or C_ESQL64 can be used for 64-bit applications. This is the case in the external C function shown above. It does not use any of these APIs.

To execute the sample compiled native function, execute the following statement.

```
SELECT SimpleCDemo(1,2,3,4);
```

To use server-side ODBC, the C/C++ code must use the default database connection. To get a handle to the database connection, call get_value with an EXTFN_CONNECTION_HANDLE_ARG_NUM argument. The argument tells the database server to return the current external environment connection rather than opening a new one.

```
#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;
}
```

If the above ODBC code is stored in the file `extodbc.cpp`, it can be built for Windows using the following commands (assuming that the database server software is installed in the folder `c:\sa17` and that Microsoft Visual C++ is installed).

```
cl extodbc.cpp /LD /Ic:\sa17\sdk\include odbc32.lib
```

The following example creates a table, defines the stored procedure wrapper to call the compiled native function, and then calls the native function to populate the table.

```
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;
```

Similarly, to use server-side ESQL, the C/C++ code must use the default database connection. To get a handle to the database connection, call get_value with an EXTFN_CONNECTION_HANDLE_ARG_NUM argument. The

argument tells the database server to return the current external environment connection rather than opening a new one.

```
#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 );
}
```

If the above Embedded SQL statements are stored in the file `extesql.sqc`, it can be built for Windows using the following commands (assuming that the database server software is installed in the folder `c:\sa17` and that Microsoft Visual C++ is installed).

```
sqlpp extesql.sqc extesql.cpp
cl extesql.cpp /LD /Ic:\sa17\sdk\include c:\sa17\sdk\lib\x86\dblibtm.lib
```

The following example creates a table, defines the stored procedure wrapper to call the compiled native function, and then calls the native function to populate the table.

```
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;
```

As in the previous examples, to use server-side C API calls, the C/C++ code must use the default database connection. To get a handle to the database connection, call get_value with an EXTFN_CONNECTION_HANDLE_ARG_NUM argument. The argument tells the database server to return the current external environment connection rather than opening a new one. The following example shows the framework for obtaining the connection handle, initializing the C API environment, and transforming the connection handle into a connection object (a_sqlany_connection) that can be used with the C API.

```
#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;
}
```

If the above C code is stored in the file `extcapi.c`, it can be built for Windows using the following commands (assuming that the database server software is installed in the folder `c:\sa17` and that Microsoft Visual C++ is installed).

```
cl /LD /Tp extcapi.c /Tp c:\sa17\SDK\C\sacapidll.c
      /Ic:\sa17\SDK\Include c:\sa17\SDK\Lib\X86\dbcapi.lib
```

The following example defines the stored procedure wrapper to call the compiled native function, and then calls the native function.

```
CREATE FUNCTION ServerSideC()
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extcapi.dll'
LANGUAGE C_ESQL32;
SELECT ServerSideC();
```

The LANGUAGE attribute in the above example specifies C_ESQL32. For 64-bit applications, you would use C_ESQL64. You must use the Embedded SQL language attribute since the C API is built on the same layer (library) as ESQL.

As mentioned earlier, each connection that does an external environment call will start its own copy of `dbexternc17`. This executable application is loaded automatically by the server the first time an external environment call is made. However, you can use the START EXTERNAL ENVIRONMENT statement to preload `dbexternc17`. This is useful to avoid the slight delay that is incurred when an external environment call is executed for the first time. Here is an example of the statement.

```
START EXTERNAL ENVIRONMENT C_ESQL32
```

Another case where preloading `dbexternc17` is useful is when you want to debug your external function. You can use the debugger to attach to the running `dbexternc17` process and set breakpoints in your external function.

The STOP EXTERNAL ENVIRONMENT statement is useful when updating a dynamic link library or shared object. It will terminate the native library loader, `dbexternc17`, for the current connection thereby releasing access to the dynamic link library or shared object. If multiple connections are using the same dynamic link library or shared object then each of their copies of `dbexternc17` must be terminated. The appropriate external environment name must be specified in the STOP EXTERNAL ENVIRONMENT statement. Here is an example of the statement.

```
STOP EXTERNAL ENVIRONMENT C_ESQL32
```

To return a result set from an external function, the compiled native function must use the external call interface.

The following code fragment shows how to set up a result set information structure. It contains a column count, a pointer to an array of column information structures, and a pointer to an array of column data value structures. The example also uses the 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;
```

The following code fragment shows how to describe the result set. It uses the C API to obtain column information for a SQL query that was executed previously by the C API. The information that is obtained from the C API for each column is transformed into a column name, type, width, index, and null value indicator that are used to describe the result set.

```
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;
}
```

Once the result set has been described, the result set rows can be returned. The following code fragment shows how to return the rows of the result set. It uses the C API to fetch the rows for a SQL query that was executed previously by the C API. The rows returned by the C API are sent back, one at a time, to the calling environment. The array of column data value structures must be filled in before returning each row. The column data value structure consists of a column index, a pointer to a data value, a data length, and an append flag.

```
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;
    }
}
```

For more information about how to make server-side requests and how to return result sets from an external function, refer to the samples in *%SQLANYSAMP17%*`\SQLAnywhere\ExternalEnvironments\ExternC`.

Related Information

External Call Interface [page 391]

1.13.3 The Java External Environment

Java methods can be called from the database in the same manner as SQL stored procedures.

A Java method behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the Java method takes place outside the database server (that is, within a Java VM environment).

There can be one instance of the Java VM for each database or there can be one instance of the Java VM for each database server (that is, all databases use the same instance).

Java stored procedures can return result sets.

There are a few prerequisites to using Java in the database support:

- A copy of the Java Runtime Environment must be installed on the database server computer.
- The database server must be able to locate the Java executable (the Java VM).
- All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In

this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

To use Java in the database, make sure that the database server is able to locate and start the Java executable. Verify that this can be done by executing:

```
START EXTERNAL ENVIRONMENT JAVA;
```

If the database server fails to start Java then the problem probably occurs because the database server is not able to locate the Java executable. In this case, execute an ALTER EXTERNAL ENVIRONMENT statement to explicitly set the location of the Java executable. Make sure to include the executable file name.

```
ALTER EXTERNAL ENVIRONMENT JAVA
  LOCATION 'java-path';
```

For example:

```
ALTER EXTERNAL ENVIRONMENT JAVA
  LOCATION 'c:\\jdk1.8.0\\jre\\bin\\java.exe';
```

You can query the location of the Java VM that the database server will use by executing the following SQL query:

```
SELECT db_property('JAVAVM');
```

The START EXTERNAL ENVIRONMENT JAVA statement is not necessary other than to verify that the database server can start the Java VM. In general, making a Java stored procedure or function call starts the Java VM automatically.

Similarly, the STOP EXTERNAL ENVIRONMENT JAVA statement is not necessary to stop an instance of Java since the instance automatically goes away when the all connections to the database have terminated. However, if you are completely done with Java and you want to make it possible to free up some resources, then the STOP EXTERNAL ENVIRONMENT JAVA statement decrements the usage count for the Java VM.

Once you have verified that the database server can start the Java VM executable, the next thing to do is to install the necessary Java class code into the database. Do this by using the INSTALL JAVA statement. For example, you can execute the following statement to install a Java class from a file into the database.

```
INSTALL JAVA
  NEW
  FROM FILE 'java-class-file';
```

You can also install a Java JAR file into the database.

```
INSTALL JAVA
  NEW
  JAR 'jar-name'
  FROM FILE 'jar-file';
```

Java classes can be installed from a variable, as follows:

```
CREATE VARIABLE JavaClass LONG VARCHAR;
SET JavaClass = xp_read_file('java-class-file');
INSTALL JAVA
  NEW
  FROM JavaClass;
```

Java JAR files can be installed from a variable, as follows:

```
CREATE VARIABLE JavaJar LONG VARCHAR;
SET JavaJar = xp_read_file('jar-file');
INSTALL JAVA
  NEW
  JAR 'jar-name'
  FROM JavaJar;
```

To remove a Java class from the database, use the REMOVE JAVA statement, as follows:

```
REMOVE JAVA CLASS java-class
```

To remove a Java JAR from the database, use the REMOVE JAVA statement, as follows:

```
REMOVE JAVA JAR 'jar-name'
```

To modify existing Java classes, you can use the UPDATE clause of the INSTALL JAVA statement, as follows:

```
INSTALL JAVA
  UPDATE
  FROM FILE 'java-class-file'
```

You can also update existing Java JAR files in the database.

```
INSTALL JAVA
  UPDATE
  JAR 'jar-name'
  FROM FILE 'jar-file';
```

Java classes can be updated from a variable, as follows:

```
CREATE VARIABLE JavaClass LONG VARCHAR;
SET JavaClass = xp_read_file('java-class-file');
INSTALL JAVA
  UPDATE
  FROM JavaClass;
```

Java JAR files can be updated from a variable, as follows:

```
CREATE VARIABLE JavaJar LONG VARCHAR;
SET JavaJar = xp_read_file('jar-file');
INSTALL JAVA
  UPDATE
  FROM JavaJar;
```

Once the Java class is installed in the database, you can then create stored procedures and functions to interface to the Java methods. The EXTERNAL NAME string contains the information needed to call the Java method and to return OUT parameters and return values. The LANGUAGE attribute of the EXTERNAL NAME clause must specify JAVA. The format of the EXTERNAL NAME clause is:

```
EXTERNAL NAME 'java-call' LANGUAGE JAVA
```

```
java-call :
[package-name.]class-name.method-name method-signature
```

```
method-signature :
( [ field-descriptor, ... ] ) return-descriptor
```

```
field-descriptor and return-descriptor :
Z
| B
| S
| I
| J
| F
| D
| C
| V
| [descriptor
| Lclass-name;
```

A Java method signature is a compact character representation of the types of the parameters and the type of the return value. If the number of parameters is less than the number indicated in the method-signature, then the difference must equal the number specified in DYNAMIC RESULT SETS, and each parameter in the method signature that is more than those in the procedure parameter list must have a method signature of `[Ljava/sql/ResultSet;`.

The `field-descriptor` and `return-descriptor` have the following meanings:

Field Type	Java Data Type
B	byte
C	char
D	double
F	float
I	int
J	long
L `class-name`;	an instance of the class `class-name`. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, java/lang/String
S	short
V	void
Z	Boolean
[	use one for each dimension of an array

For example:

```
double some_method(
  boolean a,
  int b,
  java.math.BigDecimal c,
  byte [][] d,
  java.sql.ResultSet[] rs )
{
}
```

would have the following signature:

```
'(ZILjava/math/BigDecimal;[[B[Ljava/sql/ResultSet;)D'
```

The following procedure creates an interface to a Java method. The Java method does not return any value (V).

```
CREATE PROCEDURE insertfix()
  EXTERNAL NAME 'JDBCExample.InsertFixed()V'
  LANGUAGE JAVA;
```

The following procedure creates an interface to a Java method that has a String ([Ljava/lang/String;) input argument. The Java method does not return any value (V).

```
CREATE PROCEDURE InvoiceMain( IN arg1 CHAR(50) )
  EXTERNAL NAME 'Invoice.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
```

The following procedure creates an interface to a Java method Invoice.init which takes a string argument (Ljava/lang/String;), a double (D), another string argument (Ljava/lang/String;), and another double (D), and returns no value (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;
```

The following Java example contains the function `main` which takes a string argument and writes it to the database server messages window. It also contains the function `whoAreYou` that returns a Java String.

```
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 an external Java method." );
    }
}
```

The Java code above is placed in the file `Hello.java` and compiled using the Java compiler. The class file that results is loaded into the database as follows.

```
INSTALL JAVA
  NEW
  FROM FILE 'Hello.class';
```

Using Interactive SQL, the stored procedure that will interface to the method `main` in the class `Hello` is created as follows:

```
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
  EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
```

The argument to `main` is described as an array of java.lang.String. Using Interactive SQL, test the interface by executing the following SQL statement.

```
CALL HelloDemo('This is a test');
```

If you check the database server messages window, you will find the message written there. All output to System.out is redirected to the server messages window.

Using Interactive SQL, the function that will interface to the method `whoAreYou` in the class `Hello` is created as follows:

```
CREATE FUNCTION WhoAreYou()
  RETURNS LONG VARCHAR
  EXTERNAL NAME 'Hello.whoAreYou(V)Ljava/lang/String;'
  LANGUAGE JAVA;
```

The function `whoAreYou` is described as returning a java.lang.String. Using Interactive SQL, test the interface by executing the following SQL statement.

```
SELECT WhoAreYou();
```

In attempting to troubleshoot why a Java external environment did not start, that is, if the application gets a "main thread not found" error when a Java call is made, the DBA should check the following:

- If the Java VM is a different bitness than the database server, then ensure that the client libraries with the same bitness as the VM are installed on the database server computer.
- Ensure that the `sajdbc4.jar` and `dbjdbc`*17*`/libdbjdbc`*17* shared objects are from the same software build.
- If more than one `sajdbc4.jar` are on the database server computer, make sure they are all synchronized to the same software version.
- If the database server computer is very busy, then there is a chance the error is being reported due to a timeout.

Related Information

How to Return Result Sets from Java Methods [page 215]
Java in the Database [page 199]
Lesson 4: Calling Methods in a Java Class [page 207]

1.13.4 The JavaScript External Environment (Deprecated)

JavaScript stored procedures and functions can be called from the database can be called from the database in the same manner as SQL stored procedures.

A JavaScript function behaves the same as a SQL procedure or function except that the code for the procedure or function is written in JavaScript and the execution of the procedure or function takes place outside the database server (that is, within a Node.js executable instance). There is a separate instance of the JavaScript execution environment for each connection that uses JavaScript functions. This behavior is different from Java

stored procedures and functions. For Java, there is one instance of the Java VM for each database rather than one instance per connection. The other major difference between JavaScript and Java is that JavaScript functions do not return result sets, whereas Java stored procedures can return result sets.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

To use JavaScript in the database, you must install Node.js, the XS JavaScript driver, and JavaScript External Environment module provided with the database server software.

The following Node.js versions are supported: 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x.

In this section:

Installing the JavaScript External Environment Module (Deprecated) [page 433]
Install the external environment module to support JavaScript in the database.

How to Use JavaScript from the Database (Deprecated) [page 434]
Execute JavaScript code from the database using SQL statements.

Related Information

XS JavaScript Application Programming (Deprecated) [page 219]

1.13.4.1 Installing the JavaScript External Environment Module (Deprecated)

Install the external environment module to support JavaScript in the database.

Prerequisites

- Node.js must be installed on the database server computer.
- The database server must be able to locate the Node.js executable (`C:\Program Files\nodejs\node.exe` on Microsoft Windows), for example, by including the Node.js binaries folder in the PATH.
- The SQL Anywhere XS JavaScript driver and the JavaScript External Environment support module supplied with the database server software must be installed on the database server computer. These modules are located in *%SQLANY17%*`\Node`.

Procedure

1. Add the path to the XS JavaScript driver and the JavaScript External Environment support module. The following is an example for Microsoft Windows:

   ```
   SET NODE_PATH=%SQLANY17%\Node
   ```

2. Verify that the database server is able to locate and start the Node.js executable. Start and connect to the database server. Execute the following SQL statement:

   ```
   START EXTERNAL ENVIRONMENT JS;
   ```

3. If the database server fails to start Node.js, then the database server is probably not able to locate the Node.js executable. In this case, execute an ALTER EXTERNAL ENVIRONMENT statement to explicitly set the location of the Node.js executable. Make sure to include the executable file name.

   For example:

   ```
   ALTER EXTERNAL ENVIRONMENT JS
     LOCATION 'c:\\Program Files\\nodejs\\node.exe';
   ```

   The START EXTERNAL ENVIRONMENT JS statement verifies that the database server can start Node.js. In general, making a JavaScript function call starts Node.js automatically.

   Similarly, the STOP EXTERNAL ENVIRONMENT JS statement is not necessary to stop an instance of Node.js since the instance automatically goes away when the connection terminates. However, if you are completely done with JavaScript and you want to free up some resources, then the STOP EXTERNAL ENVIRONMENT JS statement releases the Node.js instance for your connection.

Results

The JavaScript external environment module is installed.

1.13.4.2 How to Use JavaScript from the Database (Deprecated)

Execute JavaScript code from the database using SQL statements.

Once you have verified that the database server can start the Node.js executable, you can use the INSTALL statement to install the necessary JavaScript code into the database. For example, you can execute the following statement to install a JavaScript script from a file into the database.

```
INSTALL EXTERNAL OBJECT 'javascript-object'
  NEW
  FROM FILE 'javascript-file'
  ENVIRONMENT JS;
```

JavaScript code also can be built and installed from an expression, as follows:

```
INSTALL EXTERNAL OBJECT 'javascript-object'
  NEW
  FROM VALUE 'javascript-statements'
  ENVIRONMENT JS;
```

JavaScript code also can be built and installed from a variable, as follows:

```
CREATE VARIABLE javascript-variable LONG VARCHAR;
SET javascript-variable = 'javascript-statements';
INSTALL EXTERNAL OBJECT 'javascript-object'
  NEW
  FROM VALUE javascript-variable
  ENVIRONMENT JS;
```

To remove JavaScript code from the database, use the REMOVE statement, as follows:

```
REMOVE EXTERNAL OBJECT 'javascript-object';
```

To modify existing JavaScript code, you can use the UPDATE clause of the INSTALL EXTERNAL OBJECT statement, as follows:

```
INSTALL EXTERNAL OBJECT 'javascript-object'
  UPDATE
  FROM FILE 'javascript-file'
  ENVIRONMENT JS;
```

```
INSTALL EXTERNAL OBJECT 'javascript-object'
  UPDATE
  FROM VALUE 'javascript-statements'
  ENVIRONMENT JS;
```

```
SET javascript-variable = 'javascript-statements';
INSTALL EXTERNAL OBJECT 'javascript-object'
  UPDATE
  FROM VALUE javascript-variable
  ENVIRONMENT JS;
```

Once the JavaScript code is installed in the database, you can then create the SQL stored procedures and functions that will call them. When creating SQL stored procedures and functions, the LANGUAGE is always JS and the EXTERNAL NAME string contains the information needed to call the JavaScript functions and to return OUT parameters and return values.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect.

A SQL stored procedure or function can be created with any set of data types for arguments. However, the parameters are converted to and from a number or string for use inside the JavaScript function. The return value of a JavaScript function is converted to the type of the user-defined SQL function.

The return type of the JavaScript function must be specified at the beginning of the EXTERNAL NAME string inside angle brackets (<>). Valid characters are S, B, D, I, and U for String, Boolean, Double, Integer, and Unsigned Integer respectively. The types of the JavaScript parameters are listed as a sequence of characters S, B, D, I, and U inside parenthesis after the JavaScript function name.

Since JavaScript does not allow pass-by-reference for simple variables inside functions, the left bracket character ([) can proceed the S, B, D, I, or U characters to indicate that a one element array will be passed to the JavaScript stored procedure. This is done to allow for INOUT and OUT parameters in stored procedures.

The following example demonstrates a user-defined SQL function that calls a JavaScript function:

```
INSTALL EXTERNAL OBJECT 'SimpleJSExample'
  NEW
  FROM VALUE 'function SimpleJSFunction(
    thousand, hundred, ten, one )
    { return (thousand * 1000) +
      (hundred * 100) +
      (ten * 10) +
      one;
    }'
  ENVIRONMENT JS;
CREATE FUNCTION SimpleJSDemo(
    IN thousands INT,
    IN hundreds INT,
    IN tens INT,
    IN ones INT)
  RETURNS INT
  EXTERNAL NAME '<I><file=SimpleJSExample> SimpleJSFunction(IIII)'
  LANGUAGE JS;
// The number 1234 should appear
SELECT SimpleJSDemo(1,2,3,4);
```

file= references the JavaScript object and not a true file. The user-defined SQL function indicates that the JavaScript function takes 4 Integer arguments and returns an Integer value.

The following example demonstrates a SQL stored procedure with INOUT and OUT parameters that calls a JavaScript procedure:

```
INSTALL EXTERNAL OBJECT 'JSInOutParam'
  NEW
  FROM VALUE 'function JSFunctionPlusOne( number1, number2 )
    {
         number1[0] = number1[0] + 1;
         number2[0] = number1[0] * 2;
    }'
  ENVIRONMENT JS;
CREATE PROCEDURE JSInOutDemo( INOUT num1 INT, OUT num2 INT )
  EXTERNAL NAME '<file=JSInOutParam> JSFunctionPlusOne([I[I)'
  LANGUAGE JS;
BEGIN
  DECLARE @x INT;
  DECLARE @y INT;
  SET @x = 5;
  CALL JSInOutDemo( @x, @y );
  SELECT @x, @y;
END
```

To use server-side JavaScript, the JavaScript code must use the special `sa_dbcapi_handle` variable to connect to the database server. The following example creates an empty table using SQL and then calls a JavaScript function to populate the table:

```
CREATE OR REPLACE TABLE JSTab(c1 INT, c2 CHAR(128));
INSTALL EXTERNAL OBJECT 'ServerSideJS'
  NEW
  FROM VALUE 'function sjs()
    {
      var sqla = require( ''sqlanywhere-xs'' );
      var conn = sqla.createConnection();
```

```
      conn.connect( sa_dbcapi_handle );
      conn.prepareStatement(
        "MESSAGE ''JavaScript says hello'' TO CONSOLE" )
        .execute();
      conn.prepareStatement( "DELETE FROM JSTab" )
        .execute();
      conn.prepareStatement(
        "INSERT INTO JSTab SELECT table_id, table_name FROM SYS.SYSTAB" )
        .execute();
      conn.disconnect();
  }'
  ENVIRONMENT JS;
CREATE PROCEDURE JavaScriptPopulateTable()
  EXTERNAL NAME '<file=ServerSideJS> sjs()'
  LANGUAGE JS;
CALL JavaScriptPopulateTable();
// The following should return 2 identical rows
SELECT count(*) FROM JSTab
UNION ALL
SELECT count(*) FROM SYS.SYSTAB;
```

The JavaScript code connects to the database server using the caller's connection handle, executes a SQL MESSAGE statement on that connection that writes a message to the database server messages window, executes SQL DELETE and INSERT statements, and then disconnects from the database server.

Assume that the following JavaScript code resides in a file called `JSLogger.js`. It logs a message to a file in the `temp` folder.

```
function JSLogger( message )
{
    var fs = require('fs');
    fs.appendFileSync( '/temp/javascript.log', message + "\r\n" );
}
```

The following example loads the JavaScript code in this file into the database using the INSTALL EXTERNAL OBJECT ... FROM FILE clause. It defines a cover procedure in SQL that is the interface to the JavaScript code. It then calls this SQL procedure.

```
INSTALL EXTERNAL OBJECT 'JSLoggerFile'
  NEW
  FROM FILE '\\temp\\JSLogger.js'
  ENVIRONMENT JS;
CREATE OR REPLACE PROCEDURE JSLogger( IN msg LONG VARCHAR )
  EXTERNAL NAME '<file=JSLoggerFile> JSLogger(S)'
  LANGUAGE JS;
CALL JSLogger( 'Hello world!' );
```

The following example is similar but the JavaScript code is embedded as a string in the SQL statement. Care must be taken to properly handle apostrophes and backslash characters in the embedded JavaScript code.

```
INSTALL EXTERNAL OBJECT 'JSLoggerObject'
  NEW
  FROM VALUE 'function JSLogger( message )
    {
        var fs = require(''fs'');
        fs.appendFileSync( ''/temp/javascript.log'', message + "\\r\\n" );
    }'
  ENVIRONMENT JS;
CREATE OR REPLACE PROCEDURE JSLogger( IN msg LONG VARCHAR )
  EXTERNAL NAME '<file=JSLoggerObject> JSLogger(S)'
  LANGUAGE JS;
CALL JSLogger( 'Hello world!' );
```

For more information and examples on using the JavaScript in the database support, refer to the examples located in the *%SQLANYSAMP17%*`\SQLAnywhere\ExternalEnvironments\JavaScript` directory.

1.13.5 The Perl External Environment

Perl stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance). There is a separate instance of the Perl executable for each connection that uses Perl stored procedures and functions. This behavior is different from Java stored procedures and functions. For Java, there is one instance of the Java VM for each database rather than one instance per connection. The other major difference between Perl and Java is that Perl stored procedures do not return result sets, whereas Java stored procedures can return result sets.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

In this section:

Installing the Perl External Environment Module [page 438]
Install the external environment module to support Perl in the database.

How to Use Perl from the Database [page 439]
Execute Perl code from the database using SQL statements.

Related Information

Perl DBI Support [page 448]

1.13.5.1 Installing the Perl External Environment Module

Install the external environment module to support Perl in the database.

Prerequisites

1. Perl must be installed on the database server computer and the database server must be able to locate the Perl executable.

2. The DBD::SQLAnywhere driver must be installed on the database server computer.
3. On Microsoft Windows, Microsoft Visual Studio must be installed. This is a prerequisite since it is necessary for installing the DBD::SQLAnywhere driver.

In addition to the above prerequisites, the database administrator must also install the Perl External Environment module supplied with the database server software.

Procedure

To install the Perl external environment module, choose one of the following:

Option	Description

Windows	Run the following commands from the `SDK\PerlEnv` subdirectory of your database server software:
	`perl Makefile.PL`
	`nmake`
	`nmake install`

UNIX and Linux	Run the following commands from the `sdk/perlenv` subdirectory of your database server software:
	`perl Makefile.PL`
	`make`
	`make install`

Results

The Perl external environment module is installed.

1.13.5.2 How to Use Perl from the Database

Execute Perl code from the database using SQL statements.

To use Perl in the database, make sure that the database server is able to locate and start the Perl executable. Verify this by executing:

```
START EXTERNAL ENVIRONMENT PERL;
```

If the database server fails to start Perl, then the problem probably occurs because the database server is not able to locate the Perl executable. In this case, execute an ALTER EXTERNAL ENVIRONMENT statement to explicitly set the location of the Perl executable. Make sure to include the executable file name.

```
ALTER EXTERNAL ENVIRONMENT PERL
  LOCATION 'perl-path';
```

For example:

```
ALTER EXTERNAL ENVIRONMENT PERL
  LOCATION 'c:\\Perl\\bin\\perl.exe';
```

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect.

The START EXTERNAL ENVIRONMENT PERL statement is not necessary other than to verify that the database server can start Perl. In general, making a Perl stored procedure or function call starts Perl automatically.

Similarly, the STOP EXTERNAL ENVIRONMENT PERL statement is not necessary to stop an instance of Perl since the instance automatically goes away when the connection terminates. However, if you are completely done with Perl and you want to free up some resources, then the STOP EXTERNAL ENVIRONMENT PERL statement releases the Perl instance for your connection.

Once you have verified that the database server can start the Perl executable, the next thing to do is to install the necessary Perl code into the database. Do this by using the INSTALL statement. For example, you can execute the following statement to install a Perl script from a file into the database.

```
INSTALL EXTERNAL OBJECT 'perl-script'
  NEW
  FROM FILE 'perl-file'
  ENVIRONMENT PERL;
```

Perl code also can be built and installed from an expression, as follows:

```
INSTALL EXTERNAL OBJECT 'perl-script'
  NEW
  FROM VALUE 'perl-statements'
  ENVIRONMENT PERL;
```

Perl code also can be built and installed from a variable, as follows:

```
CREATE VARIABLE PerlVariable LONG VARCHAR;
SET PerlVariable = 'perl-statements';
INSTALL EXTERNAL OBJECT 'perl-script'
  NEW
  FROM VALUE PerlVariable
  ENVIRONMENT PERL;
```

To remove Perl code from the database, use the REMOVE statement, as follows:

```
REMOVE EXTERNAL OBJECT 'perl-script';
```

To modify existing Perl code, you can use the UPDATE clause of the INSTALL EXTERNAL OBJECT statement, as follows:

```
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;
```

Once the Perl code is installed in the database, you can then create the necessary Perl stored procedures and functions. When creating Perl stored procedures and functions, the LANGUAGE is always PERL and the EXTERNAL NAME string contains the information needed to call the Perl subroutines and to return OUT parameters and return values. The following global variables are available to the Perl code on each call:

Global Variable	Description
$sa_perl_return	This is used to set the return value for a function call.
$sa_perl_argN	N is a positive integer [0 .. n]. This is used for passing the SQL arguments down to the Perl code. For example, `$sa_perl_arg0` refers to argument 0, `$sa_perl_arg1` refers to argument 1, and so on.
$sa_perl_default_connection	This is used for making server-side Perl calls.
$sa_output_handle	This is used for sending output from the Perl code to the database server messages window.

A Perl stored procedure can be created with any set of data types for input and output arguments, and for the return value. However, all non-binary data types are mapped to strings when making the Perl call while binary data is mapped to an array of numbers. A simple Perl example follows:

```
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);
```

The following Perl example takes a string and writes it to the database server messages window:

```
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' );
```

To use server-side Perl, the Perl code must use the $sa_perl_default_connection variable. The following example creates a table and then calls a Perl stored procedure to populate the table:

```
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;
```

For more information and examples on using the Perl in the database support, refer to the examples located in the *%SQLANYSAMP17%*`\SQLAnywhere\ExternalEnvironments\Perl` directory.

1.13.6 The PHP External Environment (Deprecated)

PHP stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance). There is a separate instance of the PHP executable for each connection that uses PHP stored procedures and functions. This behavior is quite different from Java stored procedures and functions. For Java, there is one instance of the Java VM for each database rather than one instance per connection. The other major difference between PHP and Java is that PHP stored procedures do not return result sets, whereas Java stored procedures can return result sets. PHP only returns an object of type LONG VARCHAR, which is the output of the PHP script.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect. In this case, make sure to include the -es option with the other database server start options to enable shared memory connections.

To use PHP in the database, you must first install PHP. See http://php.net for more information.

Then you must install the PHP driver and PHP External Environment module which can be downloaded from The SAP SQL Anywhere PHP Module and you must configure PHP to locate and load these components. These steps are described next.

In this section:

Installing the PHP External Environment Module on Windows or Linux (Deprecated) [page 443]
Install the external environment module to support PHP in the database.

How to Use PHP from the Database (Deprecated) [page 444]
Execute PHP code from the database using SQL statements.

1.13.6.1 Installing the PHP External Environment Module on Windows or Linux (Deprecated)

Install the external environment module to support PHP in the database.

Prerequisites

1. A copy of PHP must be installed on the database server computer and the database server must be able to locate the PHP executable.
2. The SQL Anywhere PHP extension must be installed on the database server computer and PHP must be configured to use it. Either thread-safe or non-thread-safe extension can be used.

Context

In addition to the above two prerequisites, the database administrator must also install the PHP External Environment module. To install prebuilt modules, copy the appropriate module to your PHP extensions directory (which can be found in php.ini). On Unix, you can also use a symbolic link. The following steps describe how to do this.

Procedure

1. Download prebuilt versions of the PHP extension and the external environment module from The SAP SQL Anywhere PHP Module.
2. Locate the `php.ini` file for your PHP installation and open it in a text editor. Locate the line that specifies the location of the extension_dir directory. If extension_dir is not set to any specific directory, set it to point to an isolated directory for better system security.
3. Copy the desired external environment PHP module to your PHP extensions directory.

   Both 32-bit and 64-bit versions of the module are available for different versions of PHP. The file name portion of the external environment module has the following pattern (where `x.y` corresponds to the PHP version you are using):

   ```
   php-x.y.0_sqlanywhere_extenv17
   ```

4. Add a line to the Dynamic Extensions section of the `php.ini` file to load the external environment PHP module automatically. Change the `x.y` to reflect the version you have selected.

   For Windows, add the following line:

   ```
   extension=php-x.y.0_sqlanywhere_extenv17.dll
   ```

   For Unix, add the following line:

   ```
   extension=php-x.y.0_sqlanywhere_extenv17.so
   ```

5. Save and close `php.ini`.
6. Make sure that you have also installed the SQL Anywhere PHP extension into your PHP extensions directory.

   Instructions for installing the PHP extension are provided elsewhere.

Results

The external environment module is installed.

Related Information

PHP Client Deployment [page 849]
PHP Client Deployment [page 849]

1.13.6.2 How to Use PHP from the Database (Deprecated)

Execute PHP code from the database using SQL statements.

To use PHP in the database, the database server must be able to locate and start the PHP executable. You can verify if the database server is able to locate and start the PHP executable by executing the following statement:

```
START EXTERNAL ENVIRONMENT PHP;
```

If you see a message that states that 'external executable' could not be found, then the problem is that the database server is not able to locate the PHP executable. In this case, execute an ALTER EXTERNAL ENVIRONMENT statement to explicitly set the location of the PHP executable including the executable name, or ensure that the PATH environment variable includes the directory containing the PHP executable.

```
ALTER EXTERNAL ENVIRONMENT PHP
  LOCATION 'php-path';
```

For example:

```
ALTER EXTERNAL ENVIRONMENT PHP
```

```
  LOCATION 'c:\\php\\php.exe';
```

To restore the default setting, execute the following statement:

```
ALTER EXTERNAL ENVIRONMENT PHP
  LOCATION 'php';
```

If you see a message that states that 'main thread' could not be found, then check for the following:

- Make sure that both the `php-x.y.0_sqlanywhere` and `php-x.y.0_sqlanywhere_extenv`*17* modules are located in the directory indicated by extension_dir in the `php.ini` file. Note that `x.y` identifies the PHP version that you are using. Check the installation steps described in the previous section.
- Make sure that both the `php-x.y.0_sqlanywhere` and `php-x.y.0_sqlanywhere_extenv`*17* modules are listed in the Dynamic Extensions section of the `php.ini` file. Check the installation steps described in the previous section.
- Make sure that `phpenv.php` can be located. Check that the database server `bin32` folder is in your PATH.
- For Microsoft Windows, make sure that the 64-bit or 32-bit DLLs (`dbcapi.dll`, `dblib`*17*`.dll`, `dbicu`*17*`.dll`, `dbicudt`*17*`.dll`, `dblgen`*17*`.dll`, `dbextenv`*17*`.dll`, and `dbrsa`*17*`.dll`) can be located. Check that the database server `bin32` or `bin64` folder is in your PATH.
- For UNIX and Linux operating systems, make sure that the 32-bit or 64-bit shared objects (`libdbcapi_r`, `libdblib`*17*`_r`, `libdbicu`*17*`_r`, `libdbicudt`*17*, `dblgen`*17*`.res`, `libdbextenv`*17*`_r`, and `libdbrsa`*17*`_r`) can be located. Check that the database server `bin32` or bin64 folder is in your PATH.
- Make sure that the environment variable PHPRC is not set, or make sure that it points to the version of PHP that you intend to use.

All external environments connect back to the database server over shared memory. If the database server is configured to accept only encrypted connections, then the external environment will fail to connect.

The START EXTERNAL ENVIRONMENT PHP statement is not necessary other than to verify that the database server can start PHP. In general, making a PHP stored procedure or function call starts PHP automatically.

Similarly, the STOP EXTERNAL ENVIRONMENT PHP statement is not necessary to stop an instance of PHP since the instance automatically goes away when the connection terminates. However, if you are completely done with PHP and you want to free up some resources, then the STOP EXTERNAL ENVIRONMENT PHP statement releases the PHP instance for your connection.

Once you have verified that the database server can start the PHP executable, the next thing to do is to install the PHP code into the database. Do this by using the INSTALL statement. For example, you can execute the following statement to install a particular PHP script into the database.

```
INSTALL EXTERNAL OBJECT 'php-script'
  NEW
  FROM FILE 'php-file'
  ENVIRONMENT PHP;
```

PHP code can also be built and installed from an expression as follows:

```
INSTALL EXTERNAL OBJECT 'php-script'
  NEW
  FROM VALUE 'php-statements'
  ENVIRONMENT PHP;
```

PHP code can also be built and installed from a variable as follows:

```
CREATE VARIABLE PHPVariable LONG VARCHAR;
```

```
SET PHPVariable = 'php-statements';
INSTALL EXTERNAL OBJECT 'php-script'
  NEW
  FROM VALUE PHPVariable
  ENVIRONMENT PHP;
```

To remove PHP code from the database, use the REMOVE statement as follows:

```
REMOVE EXTERNAL OBJECT 'php-script';
```

To modify existing PHP code, you can use the UPDATE clause of the INSTALL statement as follows:

```
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;
```

Once the PHP code is installed in the database, you can then go ahead and create the necessary PHP stored procedures and functions. When creating PHP stored procedures and functions, the LANGUAGE is always PHP and the EXTERNAL NAME string contains the information needed to call the PHP subroutines and for returning OUT parameters.

The arguments are passed to the PHP script in the $argv array, similar to the way PHP would take arguments from the command line (that is, $argv[1] is the first argument). To set an output parameter, assign it to the appropriate $argv element. The return value is always the output from the script (as a LONG VARCHAR).

A PHP stored procedure can be created with any set of data types for input or output arguments. However, the parameters are converted to and from a boolean, integer, double, or string for use inside the PHP script. The return value is always an object of type LONG VARCHAR. A simple PHP example follows:

```
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);
```

For PHP, the EXTERNAL NAME string is specified in a single line of SQL.

To use server-side PHP, the PHP code can use the default database connection. To get a handle to the database connection, call sasql_pconnect with an empty string argument ('' or ""). The empty string argument tells the PHP driver to return the current external environment connection rather than opening a new one. The following example creates a table and then calls a PHP stored procedure to populate the table:

```
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;
```

For PHP, the EXTERNAL NAME string is specified in a single line of SQL. In the above example, the single quotes are doubled-up because of the way quotes are parsed in SQL. If the PHP source code was in a file, then the single quotes would not be doubled-up.

To return an error back to the database server, throw a PHP exception. The following example shows how to do this.

```
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();
```

The above example should terminate with error SQLE_UNHANDLED_EXTENV_EXCEPTION indicating that the table `phpTabNoExist` could not be found.

For more information and examples on using the PHP in the database support, refer to the examples located in the *%SQLANYSAMP17%*`\SQLAnywhere\ExternalEnvironments\PHP` directory.

1.14 Perl DBI Support

DBD::SQLAnywhere is a database driver for the Perl Database Independent Interface (DBI), which is a data access API for the Perl language.

The Perl DBI API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using Perl DBI and DBD::SQLAnywhere, your Perl scripts have direct access to SQL Anywhere database servers.

In this section:

DBD::SQLAnywhere [page 448]
DBD::SQLAnywhere is a driver for the Database Independent Interface (DBI) module for Perl. Once you have installed the Perl DBI module and DBD::SQLAnywhere, you can access and change the information in SQL Anywhere databases from Perl.

Installing DBD::SQLAnywhere on Windows [page 449]
Install the DBD::SQLAnywhere interface on the supported Windows platform to use Perl to access databases.

Installing DBD::SQLAnywhere on UNIX/Linux [page 451]
Install the DBD::SQLAnywhere interface on the supported UNIX and Linux platforms to use Perl to access databases.

Perl Scripts That Use DBD::SQLAnywhere [page 453]
You can write Perl scripts that use the DBD::SQLAnywhere interface. DBD::SQLAnywhere is a driver for the Perl DBI module.

1.14.1 DBD::SQLAnywhere

DBD::SQLAnywhere is a driver for the Database Independent Interface (DBI) module for Perl. Once you have installed the Perl DBI module and DBD::SQLAnywhere, you can access and change the information in SQL Anywhere databases from Perl.

The DBD::SQLAnywhere driver is thread-safe when using Perl with ithreads.

Requirements

The DBD::SQLAnywhere interface requires the following components.

- Perl 5.6.0 or later. On Windows, ActivePerl 5.6.0 build 616 or later is required.
- Perl DBI 1.34 or later.
- A C compiler. On Windows, only the Microsoft Visual C++ compiler is supported.

1.14.2 Installing DBD::SQLAnywhere on Windows

Install the DBD::SQLAnywhere interface on the supported Windows platform to use Perl to access databases.

Prerequisites

- ActivePerl 5.6.0 or later.
- (Optional) If you want to build the DBD::SQLAnywhere driver for ActivePerl 5.16 or earlier, then Microsoft Visual Studio is required. It is not required for ActivePerl 5.18 or later.

Procedure

1. If ActiveState builds are working correctly, you can install the driver from their repository. At a command prompt, change to the `bin` subdirectory of your ActivePerl installation directory and run the following command.

   ```
   ppm install DBD-SQLAnywhere
   ```

   If the driver installed correctly, then you are done.
2. To build the driver yourself using ActivePerl 5.18 or later (see below for earlier versions of ActivePerl):
   a. At a command prompt, make a copy of the `SDK\Perl` directory of your SQL Anywhere installation in a new directory that is writable and change to this directory.
   b. Install the MinGW Perl package.

      ```
      ppm install MinGW
      ```

   c. Add your Perl `site\bin` directory to your PATH if it is not already there.
   d. At the command prompt, run the following commands.

      ```
      perl Makefile.PL
      dmake
      ```

   e. To test DBD::SQLAnywhere, copy the sample database file to your current directory and make the tests.

      ```
      demo.db
      dmake test
      ```

      If the tests do not run, ensure that the `bin32` or `bin64` subdirectory of the software installation is in your path.
   f. To complete the installation, run the following command at the same prompt.

      ```
      dmake install
      ```

      If for any reason you must start over, you can run the command **`dmake clean`** to remove any partially built targets.

3. To build the driver yourself using ActivePerl 5.16 or earlier:

   Note: Microsoft C is required.

   a. Add your Perl `site\bin` directory to your PATH if it is not already there.
   b. At a command prompt, make a copy of the `SDK\Perl` directory of your SQL Anywhere installation in a new directory that is writable and change to this directory.
   c. Set your PATH, LIB, and INCLUDE environment variables correctly for Microsoft Visual Studio. Microsoft provides a batch file for this purpose. For 32-bit builds, a batch file called `vcvars32.bat` is included in the `vc\bin` subdirectory of the Microsoft Visual Studio installation. For 64-bit builds, look for a 64-bit version of this batch file such as `vcvars64.bat` or `vcvarsamd64.bat`.
   d. If Perl DBI is not installed, you must install it. To do so, enter the following command:

      ```
      ppm install dbi
      ```

   e. At the command prompt, run the following commands.

      ```
      perl Makefile.PL
      nmake
      ```

      If you are using an old version of Microsoft Visual Studio and/or an old version of ActivePerl, and you see errors about manifests, uncomment the following line in Makefile.PL and try again starting with 'perl Makefile.PL':

      ```
      #    $opts{LD} = "\$(PERL) dolink.pl \$@";
      ```

      Do not hand-edit the generated Makefile as those changes will be lost the next time 'perl Makefile.PL' is invoked. Always try to make changes via the Makefile.PL command line and/or editing Makefile.PL.
   f. To test DBD::SQLAnywhere, copy demo.db from the SQL Anywhere installation directory to the current directory.

      ```
      demo.db
      nmake test
      ```

      If the tests do not run, ensure that the `bin32` or `bin64` subdirectory of the software installation is in your path.
   g. To complete the installation, run the following command at the same prompt.

      ```
      nmake install
      ```

      If for any reason you must start over, you can run the command **`nmake clean`** to remove any partially built targets.

Results

The Perl DBI module and the DBD::SQLAnywhere interface are now ready to use.

Related Information

How to: Enable a 64-Bit Visual C++ Toolset on the Command Line
Perl DBI

1.14.3 Installing DBD::SQLAnywhere on UNIX/Linux

Install the DBD::SQLAnywhere interface on the supported UNIX and Linux platforms to use Perl to access databases.

Prerequisites

- Build, test, and install Perl 5 (at least 5.6.0 is recommended).

  > **Note**
  >
  > It is very important to install and test it.

- Build, test, and install the DBI module (at least DBI 1.51 is required) if it is not already included in your Perl distribution.
- No SQL Anywhere software needs to be installed to build this module. However, a SQL Anywhere client installation is required in order to use the module.
- A full SQL Anywhere client and server installation is recommended for testing this module, otherwise the tests will be skipped.

Context

These steps apply to Linux, Solaris, AIX, HP-UX, and macOS.

Procedure

1. If it is not already included in your Perl distribution, download the Perl DBI module source from the Comprehensive Perl Archive Network (CPAN).
   a. Extract the contents of this file into a new directory.
   b. At a command prompt, change to the new directory and run the following commands to build the Perl DBI module.

   ```
   perl Makefile.PL
   make
   ```

c. Copy the following from the SQL Anywhere installation directory to the current directory.

```
demo.db
```

d. Use the following command to test the Perl DBI module.

```
make test
```

e. To complete the installation, run the following command at the same prompt.

```
make install
```

If for any reason you must start over, you can use the command **`make clean`** to remove any partially built targets.

2. Make sure the environment is set up to run the database server. Depending on which shell you are using, enter the appropriate command to source the SQL Anywhere configuration script from the software installation directory:

Bourne shell and its derivatives

Under Bourne shell and its derivatives, the name of this command is `.` (a single period). For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following statement sources `sa_config.sh` (use `bin32` for 32-bit environments):

```
. /opt/sqlanywhere17/bin64/sa_config.sh
```

For example, on macOS, run the following command to source `sa_config.sh`:

```
. /Applications/SQLAnywhere17/System/bin64/sa_config.sh
```

C-shell and its derivatives

Under C-shell and its derivatives, the command is `source`. For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following command sources `sa_config.csh` (use `bin32` for 32-bit environments):

```
source  /opt/sqlanywhere17/bin64/sa_config.csh
```

> **Note**
>
> On OS X 10.11, set the SQLANY_API_DLL environment variable to the full path to `libdbcapi_r.dylib`.

3. At a shell prompt, make a copy of the `sdk/perl` directory of your SQL Anywhere installation in a new directory that is writable and change to this directory.
4. Run the following commands to build DBD::SQLAnywhere.

```
perl Makefile.PL
make
```

5. To test DBD::SQLAnywhere, copy the sample database file to your current directory and make the tests.

```
newdemo.sh
make test
```

If the tests do not run, ensure that the `bin32` or `bin64` subdirectory of the software installation is in your path.

6. To complete the installation, run the following command at the same prompt.

   ```
   make install
   ```

   If for any reason you must start over, you can use the command **`make clean`** to remove any partially built targets.

Results

The Perl DBI module and the DBD::SQLAnywhere interface are ready for use.

Next Steps

Optionally, you can delete the Perl DBI source tree and your copy of the `sdk/perl` directory. They are no longer required.

Related Information

The Comprehensive Perl Archive Network

1.14.4 Perl Scripts That Use DBD::SQLAnywhere

You can write Perl scripts that use the DBD::SQLAnywhere interface. DBD::SQLAnywhere is a driver for the Perl DBI module.

In this section:

The Perl DBI Module [page 454]
To use the DBD::SQLAnywhere interface from a Perl script, you must first tell Perl that you plan to use the Perl DBI module. To do so, include the following line at the top of the file.

How to Open and Close a Database Connection Using Perl DBI [page 454]
Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements.

How to Obtain Result Sets Using Perl DBI [page 455]
Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

How to Process Multiple Result Sets Using Perl DBI [page 456]
The method for handling multiple result sets from a query involves wrapping the fetch loop within another loop that moves between result sets.

How to Insert Rows Using Perl DBI [page 457]

Inserting rows requires a handle to an open connection. The simplest method is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values.

1.14.4.1 The Perl DBI Module

To use the DBD::SQLAnywhere interface from a Perl script, you must first tell Perl that you plan to use the Perl DBI module. To do so, include the following line at the top of the file.

```
use DBI;
```

In addition, it is highly recommended that you run Perl in strict mode. This statement, which makes explicit variable definitions mandatory, is likely to greatly reduce the chance that you will run into mysterious errors due to such common mistakes as typographical errors.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
```

The Perl DBI module automatically loads the DBD drivers, including DBD::SQLAnywhere, as required.

1.14.4.2 How to Open and Close a Database Connection Using Perl DBI

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements.

To open a connection, you use the *connect* method. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The parameters to the connect method are as follows:

1. "DBI:SQLAnywhere:" and additional connection parameters separated by semicolons.
2. A user name. Unless this string is blank, ";UID=`value`" is appended to the connection string.
3. A password value. Unless this string is blank, ";PWD=`value`" is appended to the connection string.
4. A pointer to a hash of default values. Settings such as AutoCommit, RaiseError, and PrintError may be set in this manner.

The following code sample opens and closes a connection to the sample database. You must start the database server and sample database before running this script.

```
#!/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      = "password";
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__
```

Optionally, you can append the user name or password value to the data-source string instead of supplying them as separate parameters. If you do so, supply a blank string for the corresponding argument. For example, in the above script may be altered by replacing the statement that opens the connections with these statements:

```
$data_src .= ";UID=$uid";
$data_src .= ";PWD=$password";
my $dbh = DBI->connect($data_src, '', '', \%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr\n";
```

1.14.4.3 How to Obtain Result Sets Using Perl DBI

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

SQL statements that return row sets must be prepared before being executed. The *prepare* method returns a handle to the statement. You use the handle to execute the statement, then retrieve meta information about the result set and the rows of the result set.

```
#!/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      = "password";
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__
```

Prepared statements are not dropped from the database server until the Perl statement handle is destroyed. To destroy a statement handle, reuse the variable or set it to undef. Calling the finish method does not drop the handle. In fact, the finish method should not be called, except when you have decided not to finish reading a result set.

To detect handle leaks, the database server limits the number of cursors and prepared statements permitted to a maximum of 50 per connection by default. The resource governor automatically generates an error if these limits are exceeded. If you get this error, check for undestroyed statement handles. Use prepare_cached sparingly, as the statement handles are not destroyed.

If necessary, you can alter these limits by setting the max_cursor_count and max_statement_count options.

Related Information

max_cursor_count Option
max_statement_count Option
max_cursor_count Option
max_statement_count Option

1.14.4.4 How to Process Multiple Result Sets Using Perl DBI

The method for handling multiple result sets from a query involves wrapping the fetch loop within another loop that moves between result sets.

SQL statements that return multiple result sets must be prepared before being executed. The *prepare* method returns a handle to the statement. You use the handle to execute the statement, then retrieve meta information about the result set and the rows of each of the result sets.

```
#!/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      = "password";
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__
```

1.14.4.5 How to Insert Rows Using Perl DBI

Inserting rows requires a handle to an open connection. The simplest method is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values.

The statement is first prepared, and then executed once per new row. The new row values are supplied as parameters to the execute method.

The following sample program inserts two new customers. Although the row values appear as literal strings, you could read the values from a file.

```
#!/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      = "password";
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__
```

1.15 Python and Database Access

The Python extension modules sqlanydb, sqlany-django, and sqlalchemy-sqlany support the use of Python when connecting to databases, issuing SQL queries, and obtaining result sets.

The Python Database API Specification v2.0 (PEP 249) defines a set of methods that provides a consistent database interface independent of the actual database being used. The Python extension module sqlanydb implements PEP 249. Once you have installed the sqlanydb module, you can access and change the information in databases from Python.

For information, see PEP 249 -- Python Database API Specification v2.0 .

The Python extension module sqlany-django allows customers to use the database server as a back end for a Django-based web site.

The Python extension module sqlalchemy-sqlany allows users to create SQLAlchemy applications that can communicate with the database server.

In this section:

Installing sqlanydb on Windows [page 459]
Set up Python support on Windows by running the applicable setup script from the `SDK\Python` subdirectory of your software installation.

Installing sqlanydb on UNIX/Linux [page 460]
Set up Python support on UNIX and Linux by running the applicable setup script from the `sdk/python` subdirectory of your software installation.

Installing the Django Driver (sqlany-django) [page 462]
Django is a Python-based framework for creating web sites. The sqlany-django driver allows customers to use the SQL Anywhere database server as a back end for a Django-based web site.

Installing the SQLAlchemy Dialect (sqlalchemy-sqlany) [page 462]
SQLAlchemy is a Python-based toolkit and object relational mapper. The sqlalchemy-sqlany dialect allows users to create SQLAlchemy applications that communicate with a SQL Anywhere database server.

Python Scripts That Use sqlanydb [page 462]
The following material provides an overview of how to write Python scripts that use the sqlanydb interface.

Related Information

SQL Anywhere Client Interfaces
PEP 249 -- Python Database API Specification v2.0

1.15.1 Installing sqlanydb on Windows

Set up Python support on Windows by running the applicable setup script from the `SDK\Python` subdirectory of your software installation.

Prerequisites

Ensure that Python is installed.

The ctypes module is required. To test if the ctypes module is present, run Python. At the Python prompt, enter the following statement.

```
import ctypes
```

If you see an error message, then ctypes is not present. The following is an example.

```
>>> import ctypes
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named ctypes
```

If ctypes is not included in your Python installation, then install it. Installs can be found at the SourceForge.net website.

Peak EasyInstall also installs ctypes.

Procedure

1. Change to the `SDK\Python` subdirectory of your software installation.
2. Run the following command to install sqlanydb.

   ```
   pip install sqlanydb
   ```
3. Run the following commands to make a copy of the sample database file in your current directory and test sqlanydb.

   ```
   newdemo
   dbsrv17 demo.db
   python Scripts\test.py
   ```

The test script makes a connection to the database server and executes a SQL query. If the test is successful, then it displays the message `sqlanydb successfully installed.`

If the tests do not run, then ensure that the `bin32` or `bin64` subdirectory of the software installation is in your path.

You can delete the sample database and log file from your current directory.

Results

The sqlanydb module is now ready to use.

Related Information

SQL Anywhere Client Interfaces
PEAK EasyInstall

1.15.2 Installing sqlanydb on UNIX/Linux

Set up Python support on UNIX and Linux by running the applicable setup script from the `sdk/python` subdirectory of your software installation.

Prerequisites

Ensure that Python is installed.

The ctypes module is required. To test if the ctypes module is present, run Python. At the Python prompt, enter the following statement.

```
import ctypes
```

If you see an error message, then ctypes is not present. The following is an example.

```
>>> import ctypes
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named ctypes
```

If ctypes is not included in your Python installation, then install it. Installs can be found at the SourceForge.net website.

Peak EasyInstall also installs ctypes.

Procedure

1. Make sure the environment is set up to run the database server. Depending on which shell you are using, enter the appropriate command to source the SQL Anywhere configuration script from the software installation directory:

   Bourne shell and its derivatives

   Under Bourne shell and its derivatives, the name of this command is `.` (a single period). For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following statement sources `sa_config.sh` (use `bin32` for 32-bit environments):

   ```
   .  /opt/sqlanywhere17/bin64/sa_config.sh
   ```

   For example, on a Mac, run the following command to source `sa_config.sh`:

   ```
   . /Applications/SQLAnywhere17/System/bin64/sa_config.sh
   ```

   C-shell and its derivatives

   Under C-shell and its derivatives, the command is `source`. For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following command sources `sa_config.csh` (use `bin32` for 32-bit environments):

   ```
   source  /opt/sqlanywhere17/bin64/sa_config.csh
   ```

   > **Note**
   >
   > If you are using macOS 10.11 or a later version, then you must set the SQLANY_API_DLL environment variable to the full path to `libdbcapi_r.dylib`.

2. At a shell prompt, change to the `sdk/python` subdirectory of your software installation.
3. Enter the following command to install sqlanydb.

   ```
   pip install sqlanydb
   ```

4. Run the following commands to make a copy of the sample database file in your current directory and test sqlanydb.

   ```
   newdemo
   dbsrv17 demo.db
   python scripts/test.py
   ```

   The test script makes a connection to the database server and executes a SQL query. If the test is successful, then it displays the message `sqlanydb successfully installed.`

   If the test does not run, then ensure that the `bin32` or `bin64` subdirectory of the software installation is in your path.

   You can delete the sample database and log file from your current directory.

Results

The sqlanydb module is now ready to use.

Related Information

SQL Anywhere Client Interfaces
PEAK EasyInstall

1.15.3 Installing the Django Driver (sqlany-django)

Django is a Python-based framework for creating web sites. The sqlany-django driver allows customers to use the SQL Anywhere database server as a back end for a Django-based web site.

The most current software and documentation for the SQL Anywhere Django driver is available from the PyPI (Python Package Index) web site (https://pypi.python.org/pypi). Search for `sqlany-django`.

The driver is also available from https://github.com/sqlanywhere/sqlany-django/ . The Django driver requires the SQL Anywhere Python driver, which is included in the SQL Anywhere install, but it is also available from https://github.com/sqlanywhere/sqlanydb .

1.15.4 Installing the SQLAlchemy Dialect (sqlalchemy-sqlany)

SQLAlchemy is a Python-based toolkit and object relational mapper. The sqlalchemy-sqlany dialect allows users to create SQLAlchemy applications that communicate with a SQL Anywhere database server.

The most current software and documentation for the SQL Anywhere SQLAlchemy dialect is available from the PyPI (Python Package Index) web site (https://pypi.python.org/pypi). Search for `sqlalchemy-sqlany`.

The dialect is also available from https://github.com/sqlanywhere/sqlalchemy-sqlany . The SQLAlchemy driver requires the SQL Anywhere Python driver, which is included in the SQL Anywhere install, but it is also available from https://github.com/sqlanywhere/sqlanydb .

1.15.5 Python Scripts That Use sqlanydb

The following material provides an overview of how to write Python scripts that use the sqlanydb interface.

In this section:

The sqlanydb Module [page 463]
To use the sqlanydb module from a Python script, you must first load it by including an *import* statement at the top of the file.

How to Open and Close a Database Connection Using Python [page 464]
Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements.

How to Obtain Result Sets Using Python [page 465]
Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

How to Insert Rows Using Python [page 465]
The simplest way to insert rows into a table is to use a non-parameterized INSERT statement, meaning that values are specified as part of the SQL statement.

Database Type Conversion in Python [page 467]
To control how database types are mapped into Python objects when results are fetched from the database server, conversion callbacks can be registered.

Related Information

Variables Supplied to Web Services [page 665]

1.15.5.1 The sqlanydb Module

To use the sqlanydb module from a Python script, you must first load it by including an *import* statement at the top of the file.

```
import sqlanydb
```

The sqlanydb module is thread-safe when using Python with threads.

> **Note**
>
> The Python driver now supports Python 3.9. The prerequisites to use the Python driver are:
>
> - A full installation of the client software.
> - Keep the environment variables created by the installation program on Windows.
> - If you do not have the environment variable `SQLANY17`, append the directory that contains the `dbcapi.dll` file to the environment variable `PYTHONPATH`.
> - If you have a 32-bit Python executable, append the full path of the directory that contains the 32-bit `dbcapi.dll` file to `PYTHONPATH`.
> - If you have a 64-bit Python executable, append the full path of the directory that contains the 64-bit `dbcapi.dll` file to `PYTHONPATH`.

1.15.5.2 How to Open and Close a Database Connection Using Python

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements.

To open a connection, you use the *connect* method. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The parameters to the connect method are specified as a series of keyword=value pairs delimited by commas. Keywords are case-insensitive.

```
sqlanydb.connect( keyword=value, ... )
```

Some common connection parameters are as follows:

DataSourceName="dsn"

A short form for this connection parameter is DSN="`dsn`". An example is DataSourceName="SQL Anywhere 17 Demo".

UserID="user-id"

A short form for this connection parameter is UID="`user-id`". An example is UserID="DBA".

Password="passwd"

A short form for this connection parameter is PWD="`passwd`". An example is Password="sql".

DatabaseFile="db-file"

A short form for this connection parameter is DBF="`db-file`". An example is DatabaseFile="demo.db".

The following code sample opens and closes a connection to the sample database. You must start the database server with the sample database before running this script.

```
import sqlanydb
try: input = raw_input
except NameError: pass
myuid = input("Enter your user ID: ")
mypwd = input("Enter your password: ")
# Create a connection object
con = sqlanydb.connect( userid=myuid, password=mypwd )
# Close the connection
con.close()
```

To avoid starting the database server manually, you could use a data source that is configured to start the server with the database. This is shown in the following example.

```
import sqlanydb
try: input = raw_input
except NameError: pass
mydsn = "SQL Anywhere 17 Demo"
myuid = input("Enter your user ID: ")
mypwd = input("Enter your password: ")
# Create a connection object
con = sqlanydb.connect( dsn=mydsn, userid=myuid, password=mypwd )
# Close the connection
con.close()
```

Related Information

Alphabetical List of Connection Parameters
Python Database Programming

1.15.5.3 How to Obtain Result Sets Using Python

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

The *cursor* method is used to create a cursor on the open connection. The *execute* method is used to create a result set. The *fetchall* method is used to obtain the rows in this result set.

```
import sqlanydb
try: input = raw_input
except NameError: pass
myuid = input("Enter your user ID: ")
mypwd = input("Enter your password: ")
# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid=myuid, password=mypwd )
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( "Number of columns %d" % (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()
```

1.15.5.4 How to Insert Rows Using Python

The simplest way to insert rows into a table is to use a non-parameterized INSERT statement, meaning that values are specified as part of the SQL statement.

A new statement is constructed and executed for each new row. As in the previous example, a cursor is required to execute SQL statements.

The following sample program inserts two new customers into the sample database. Before disconnecting, it commits the transactions to the database.

```
import sqlanydb
try: input = raw_input
except NameError: pass
myuid = input("Enter your user ID: ")
```

```
mypwd = input("Enter your password: ")
# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid=myuid, pwd=mypwd )
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])
con.commit()
sql = "SELECT * FROM Customers WHERE ID > 800"
cursor.execute(sql)
# Get a cursor description which contains column names
desc = cursor.description
print( "Number of columns %d" % (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()
```

An alternate technique is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The executemany method is used to execute an INSERT statement for each member of the set of rows. The new row values are supplied as a single argument to the executemany method.

```
import sqlanydb
try: input = raw_input
except NameError: pass
myuid = input("Enter your user ID: ")
mypwd = input("Enter your password: ")
# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid=myuid, pwd=mypwd )
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)
con.commit()
sql = "SELECT * FROM Customers WHERE ID > 800"
cursor.execute(sql)
# Get a cursor description which contains column names
desc = cursor.description
print( "Number of columns %d" % (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()
```

Although both examples may appear to be equally suitable techniques for inserting row data into a table, the latter example is superior for a couple of reasons. If the data values are obtained by prompts for input, then the first example is susceptible to injection of rogue data including SQL statements. In the first example, the execute method is called for each row to be inserted into the table. In the second example, the executemany method is called only once to insert all the rows into the table.

1.15.5.5 Database Type Conversion in Python

To control how database types are mapped into Python objects when results are fetched from the database server, conversion callbacks can be registered.

Callbacks are registered using the module level register_converter method. This method is called with the database type as the first parameter and the conversion function as the second parameter. For example, to request that sqlanydb create Decimal objects for data in any column described as having type DT_DECIMAL, you would use the following example:

```
import sqlanydb
import decimal
def convert_to_decimal(num):
    return decimal.Decimal(num)
sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_decimal)
```

By default columns of type DT_BIT are converted to an integer type. To request that sqlanydb create boolean objects for data in any column described as having type DT_BIT, you would use the following example:

```
import sqlanydb
def convert_to_boolean(num):
    return num != 0
sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_decimal)
```

Converters may be registered for the following database types:

```
DT_DATE
DT_TIME
DT_TIMESTAMP
DT_VARCHAR
DT_FIXCHAR
DT_LONGVARCHAR
DT_STRING
DT_DOUBLE
DT_FLOAT
DT_DECIMAL
DT_INT
DT_SMALLINT
DT_BINARY
DT_LONGBINARY
DT_TINYINT
DT_BIGINT
DT_UNSINT
DT_UNSSMALLINT
DT_UNSBIGINT
DT_BIT
```

The following example demonstrates how to convert decimal results to integer resulting in the truncation of any digits after the decimal point. The salary amount displayed when the application is run is an integral value.

```
import sqlanydb
def convert_to_int(num):
    return int(float(num))
sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_int)
try: input = raw_input
except NameError: pass
myuid = input("Enter your user ID: ")
mypwd = input("Enter your password: ")
# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid=myuid, password=mypwd )
cursor = con.cursor()
# Execute a SQL string
sql = "SELECT * FROM Employees WHERE EmployeeID=105"
cursor.execute(sql)
# Get a cursor description which contains column metadata
desc = cursor.description
print ("Number of columns = %d\n" % len(desc))
# Fetch all results from the cursor into a sequence and
# display the column metadata and values as name:value pairs
rowset = cursor.fetchall()
for row in rowset:
    col = 0
    for column in cursor.description:
        name, type_code, display_size, internal_size, precision, scale, null_ok
= column

        print ("Column name: %s" % name)
        print ("Type code: %s" % type_code)
        print ("Display size: %s" % display_size)
        print ("Internal size: %d" % internal_size)
        print ("Precision: %d" % precision)
        print ("Scale: %d" % scale)
        print ("Null OK: %d" % null_ok)
        print ("Value: %s" % row[col])
        if type_code == sqlanydb.BINARY:
            print ("Python type: BINARY")
        if type_code == sqlanydb.DATE:
            print ("Python type: DATE")
        if type_code == sqlanydb.DATETIME:
            print ("Python type: DATETIME")
        if type_code == sqlanydb.NUMBER:
            print ("Python type: NUMBER")
        if type_code == sqlanydb.STRING:
            print ("Python type: STRING")
        if type_code == sqlanydb.TIME:
            print ("Python type: TIME")
        if type_code == sqlanydb.TIMESTAMP:
            print ("Python type: TIMESTAMP")
        print ("")
        col = col + 1
cursor.close()
con.close()
```

1.16 PHP Support (Deprecated)

You can use PHP to develop database applications.

In this section:

SQL Anywhere PHP Extension (Deprecated) [page 469]
PHP, which stands for PHP: Hypertext Preprocessor, is an open source scripting language. Although it can be used as a general-purpose scripting language, it was designed to be a convenient language in which to write scripts that could be embedded with HTML documents.

SQL Anywhere PHP API Reference (Deprecated) [page 478]
The methods of the PHP API can be sorted into several categories: connections, queries, result sets, transactions, statements, and miscellaneous.

1.16.1 SQL Anywhere PHP Extension (Deprecated)

PHP, which stands for PHP: Hypertext Preprocessor, is an open source scripting language. Although it can be used as a general-purpose scripting language, it was designed to be a convenient language in which to write scripts that could be embedded with HTML documents.

Unlike scripts written in JavaScript, which are frequently executed by the client, PHP scripts are processed by the web server, and the resulting HTML output sent to the clients. The syntax of PHP is derived from that of other popular languages, such as Java and Perl.

To make it a convenient language in which to develop dynamic web pages, PHP provides the ability to retrieve information from databases. An extension is included with the database software that provides access to SQL Anywhere databases from PHP. You can use the PHP extension and the PHP language to write standalone scripts and create dynamic web pages that rely on information stored in databases.

The PHP extension provides a native means of accessing your databases from PHP. You might prefer it to other PHP data access techniques because it is simple, and it helps to avoid system resource leaks that can occur with other techniques.

Download prebuilt versions of the PHP extension for Windows and Linux from The SAP SQL Anywhere PHP Module. Source code for the PHP extension is installed in the `sdk\php` subdirectory of your software installation.

In this section:

Testing the PHP Extension [page 470]
Perform a quick check to verify that the PHP extension is working correctly.

Creating and Running PHP Test Pages [page 471]
Create and run several web pages that test whether PHP is set up properly.

PHP Script Development [page 473]
You can write PHP scripts that use the PHP extension to access databases.

Building the PHP Extension on UNIX/Linux [page 477]
You can build the PHP extension using the steps described here.

Related Information

PHP Client Deployment [page 849]
The SAP SQL Anywhere PHP Module

1.16.1.1 Testing the PHP Extension

Perform a quick check to verify that the PHP extension is working correctly.

Prerequisites

All of the required PHP components should be installed on your system.

Procedure

1. Make sure that the `bin32` subdirectory of your software installation is in your path. The PHP extension requires the `bin32` directory to be in your path.
2. At a command prompt, run the following command to start the sample database.

   ```
   dbsrv17 "%SQLANYSAMP17%\demo.db"
   ```

   The command starts a database server using the sample database.
3. At a command prompt, change to the `SDK\PHP\Examples` subdirectory of your software installation. Make sure that the php executable directory is included in your path. Enter the following command:

   ```
   php test.php
   ```

   Messages similar to the following should appear. If the PHP command is not recognized, verify that PHP is in your path.

   ```
   Installation successfulUsing php-5.2.11_sqlanywhere.dllConnected successfully
   ```

   If the PHP extension does not load, you can use the command "php -i" for helpful information about your PHP setup. Search for extension_dir and sqlanywhere in the output from this command.
4. When you are done, stop the database server by clicking *Shut Down* in the database server messages window.

Results

The tests should succeed, indicating that the PHP extension is working correctly.

Related Information

Creating and Running PHP Test Pages [page 471]

1.16.1.2 Creating and Running PHP Test Pages

Create and run several web pages that test whether PHP is set up properly.

Prerequisites

You must install PHP.

Context

This procedure applies to all configurations.

Procedure

1. Create a file in your root web content directory named `info.php`.

   If you are not sure which directory to use, then check your web server's configuration file. In Apache installations, the content directory is often called `htdocs`.

   If you are using macOS, then the web content directory name may depend on which account you are using:

   - If you are the System Administrator on a macOS system, use `/Library/WebServer/Documents`.
   - If you are a macOS user, place the file in `/Users/your-user-name/Sites/`.
2. Insert the following code into this file:

   ```
   <?php phpinfo(); ?>
   ```

   The PHP function, phpinfo, generates a page of system setup information. This confirms that your installation of PHP and your web server are working together properly.
3. Copy the file `connect.php` from the `sdk\php\examples` directory to your root web content directory. This confirms that your installation of PHP, the PHP extension, and the database server are working together properly.
4. Create a file in your root web content directory named `sa_test.php` and insert the following code into this file:

   ```
   <?php  $conn = sasql_connect( "UID=DBA;PWD=password" );
   $result = sasql_query( $conn, "SELECT * FROM
   ```

```
Employees" );  sasql_result_all( $result );  sasql_free_result( $result );
sasql_disconnect( $conn );?>
```

The sa_test page displays the contents of the Employees table.

5. Start your web server if it is required.

   For example, to start the Apache web server, run the following command from the `bin` subdirectory of your Apache installation:

   ```
   apachectl start
   ```

6. Make sure the environment is set up to run the database server. Depending on which shell you are using, enter the appropriate command to source the SQL Anywhere configuration script from the software installation directory:

   Bourne shell and its derivatives

   Under Bourne shell and its derivatives, the name of this command is `.` (a single period). For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following statement sources `sa_config.sh` (use `bin32` for 32-bit environments):

   ```
   .  /opt/sqlanywhere17/bin64/sa_config.sh
   ```

   For example, on a macOS system, run the following command to source `sa_config.sh`:

   ```
   . /Applications/SQLAnywhere17/System/bin64/sa_config.sh
   ```

   C-shell and its derivatives

   Under C-shell and its derivatives, the command is `source`. For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following command sources `sa_config.csh` (use `bin32` for 32-bit environments):

   ```
   source  /opt/sqlanywhere17/bin64/sa_config.csh
   ```

7. At a command prompt, run the following command to start the sample database (if you have not already done so):

   ```
   dbsrv17 "%SQLANYSAMP17%\demo.db"
   ```

8. To test that PHP and your web server are working correctly with the database server, access the test pages from a browser that is running on the same computer as the server:

Test Page	URL
`info.php`	`http://localhost/info.php`
`connect.php`	`http://localhost/connect.php`
`sa_test.php`	`http://localhost/sa_test.php`

Results

The info page displays the output from the `phpinfo()` call.

The connect page displays the message `Connected successfully`.

The sa_test page displays the contents of the Employees table.

1.16.1.3 PHP Script Development

You can write PHP scripts that use the PHP extension to access databases.

The source code for the following examples and others is located in the `SDK\PHP\Examples` subdirectory of your software installation.

In this section:

How to Connect to a Database Using PHP [page 473]
To make a connection to a database, pass a standard database connection string to the database server as a parameter to the sasql_connect function.

How to Retrieve Data from a Database Using PHP [page 474]
One use of PHP scripts in web pages is to retrieve and display information contained in a database.

Web Forms [page 476]
PHP can take user input from a web form, pass it to the database server as a SQL query, and display the result that is returned.

BLOBs in PHP Applications [page 476]
A binary large object (BLOB) can be used to store any type of data. If that data is of a type readable by a web browser, a PHP script can easily retrieve it from the database and display it on a dynamically generated page.

1.16.1.3.1 How to Connect to a Database Using PHP

To make a connection to a database, pass a standard database connection string to the database server as a parameter to the sasql_connect function.

The <?php and ?> tags tell the web server that it should let PHP execute the code that lies between them and replace it with the PHP output.

The source code for this example is contained in your software installation in a file called `connect.php`.

```
<?php  # Connect to the sample database   $conn =
sasql_connect( "UID=DBA;PWD=password" );  if( ! $conn ) {      echo
"Connection failed\n";  } else {      echo "Connected successfully\n";
sasql_close( $conn );  }?>
```

This script attempts to make a connection to a database on a local server. For this code to succeed, the sample database or one with identical credentials must be started on a local server.

1.16.1.3.2 How to Retrieve Data from a Database Using PHP

One use of PHP scripts in web pages is to retrieve and display information contained in a database.

The following examples demonstrate some useful techniques.

Simple Select Query

The following PHP code demonstrates a convenient way to include the result set of a SELECT statement in a web page. This sample is designed to connect to the sample database and return a list of customers.

This code can be embedded in a web page, provided your web server is configured to execute PHP scripts.

The source code for this sample is contained in your software installation in a file called `query.php`.

```
<?php  # Connect to the sample database using the default user ID
and password DBA/sql  $conn = sasql_connect( "UID=DBA;PWD=password" );
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 );  }?>
```

The sasql_result_all function fetches all the rows of the result set and generates an HTML output table to display them. The sasql_free_result function releases the resources used to store the result set.

Fetching by Column Name

In certain cases, you do not want to display all the data from a result set, or you want to display the data in a different manner. The following sample illustrates how you can exercise greater control over the output format of the result set. PHP allows you to display as much information as you want in whatever manner you choose.

The source code for this sample is contained in your software installation in a file called `fetch.php`.

```
<?php  # Connect to the sample database using the user ID and password DBA/sql
$conn = sasql_connect( "UID=DBA;PWD=password" );  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 );?>
```

The sasql_fetch_array function returns a single row from the table. The data can be retrieved by column names and column indexes.

The sasql_fetch_assoc function returns a single row from the table as an associative array. The data can be retrieved by using the column names as indexes. The following is an example.

The source code for this sample is contained in your software installation in a file called `fetch_assoc.php`.

```
<?php  # Connect to the sample database using the user ID
and password DBA/sql  $conn = sasql_connect("UID=DBA;PWD=password");    /*
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);?>
```

Two other similar methods are provided in the PHP interface: sasql_fetch_row returns a row that can be searched by column indexes only, while sasql_fetch_object returns a row that can be searched by column names only.

For an example of the sasql_fetch_object function, see the `fetch_object.php` example script.

Nested Result Sets

When a SELECT statement is sent to the database, a result set is returned. The sasql_fetch_row and sasql_fetch_array functions retrieve data from the individual rows of a result set, returning each row as an array of columns that can be queried further.

The source code for this sample is contained in your software installation in a file called `nested.php`.

```
<?php  $conn = sasql_connect( "UID=DBA;PWD=password" );  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 );  }?>
```

In the above sample, the SQL statement selects the table ID and name for each table from SYSTAB. The sasql_query function returns an array of rows. The script iterates through the rows using the sasql_fetch_array function to retrieve the rows from an array. An inner iteration goes through the columns of each row and prints their values.

1.16.1.3.3 Web Forms

PHP can take user input from a web form, pass it to the database server as a SQL query, and display the result that is returned.

The following example demonstrates a simple web form that gives the user the ability to query the sample database using SQL statements and display the results in an HTML table.

The source code for this sample is contained in your software installation in a file called `webisql.php`.

```
<?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";?>
```

This design could be extended to handle complex web forms by formulating customized SQL queries based on the values entered by the user.

1.16.1.3.4 BLOBs in PHP Applications

A binary large object (BLOB) can be used to store any type of data. If that data is of a type readable by a web browser, a PHP script can easily retrieve it from the database and display it on a dynamically generated page.

BLOB fields are often used for storing non-text data, such as images in GIF or JPG format. Numerous types of data can be passed to a web browser without any need for third-party software or data type conversion. The following sample illustrates the process of adding an image to the database and then retrieving it again to be displayed in a web browser.

This sample is similar to the sample code in the files `image-insert.php` and `image-retrieve.php` of your software installation. These samples also illustrate the use of a BLOB column for storing images.

```
<?php  $conn = sasql_connect( "UID=DBA;PWD=password" )        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('SAP_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);?>
```

To send the binary data from the database directly to a web browser, the script must set the data's MIME type using the header function. In this case, the browser is told to expect a GIF image so it can display it correctly.

1.16.1.4 Building the PHP Extension on UNIX/Linux

You can build the PHP extension using the steps described here.

Prerequisites

The following is a list of software you must have on your system to build the PHP extension on UNIX/Linux:

- The SQL Anywhere PHP extension source code. This code is included in the SQL Anywhere software install under `sdk/php`.
- The PHP development package (php5-dev) which can be downloaded and installed from the PHP web site.

Context

You must have the same access privileges as the person who installed PHP to perform certain steps of the installation. Most UNIX and Linux systems offer a *sudo* command that allows users with insufficient permissions to execute certain commands as a user with the right to execute them.

Procedure

1. Source the SQL Anywhere environment variables using one of the supplied scripts.

   Bourne shell and its derivatives

   Under Bourne shell and its derivatives, the name of this command is `.` (a single period). For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following statement sources `sa_config.sh` (use `bin32` for 32-bit environments):

   ```
   .  /opt/sqlanywhere17/bin64/sa_config.sh
   ```

   For example, on a Mac, run the following command to source `sa_config.sh`:

   ```
   . /Applications/SQLAnywhere17/System/bin64/sa_config.sh
   ```

   C-shell and its derivatives

   Under C-shell and its derivatives, the command is `source`. For example, if SQL Anywhere is installed in */opt/sqlanywhere17*, the following command sources `sa_config.csh` (use `bin32` for 32-bit environments):

   ```
   source  /opt/sqlanywhere17/bin64/sa_config.csh
   ```

2. Change to the SQL Anywhere `sdk/php` directory containing the SQL Anywhere PHP extension software.
3. Build the extension using the following commands:

   ```
   $ phpize$ ./configure --with-sqlanywhere$ make$ make test$ sudo make install
   ```

4. Edit the `php.ini` file in your PHP software install directory and make sure that the `enable_dl` option is enabled as follows.

```
enable_dl = On
```

5. Verify that PHP is aware of the extension by listing the contents of the PHP extensions directory and looking for `sqlanywhere.so`.

```
$ php -i | grep extension_dir$ ls <extension_dir>
```

Results

If you are unsuccessful in building the PHP extension, keep track of the output of these commands and post it to the SQL Anywhere Forum for assistance.

Related Information

The SAP SQL Anywhere PHP Module
SQL Anywhere Client Interfaces
Apache HTTP Server Project
SQL Anywhere Forum

1.16.2 SQL Anywhere PHP API Reference (Deprecated)

The methods of the PHP API can be sorted into several categories: connections, queries, result sets, transactions, statements, and miscellaneous.

The PHP API supports the following functions:

Connections

- sasql_close
- sasql_connect
- sasql_disconnect
- sasql_error
- sasql_errorcode
- sasql_insert_id
- sasql_message
- sasql_pconnect

- sasql_set_option

Queries

- sasql_affected_rows
- sasql_next_result
- sasql_query
- sasql_real_query
- sasql_store_result
- sasql_use_result

Result Sets

- 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

Transactions

- sasql_commit
- sasql_rollback

Statements

- 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_errno
- sasql_stmt_error
- 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

Miscellaneous

- sasql_escape_string
- sasql_get_client_info

In this section:

sasql_affected_rows [page 483]
Returns the number of rows affected by the last SQL statement.

sasql_commit [page 484]
Ends a transaction on the database and makes any changes made during the transaction permanent.

sasql_close [page 485]
Closes a previously opened database connection.

sasql_connect [page 485]
Establishes a connection to a database.

sasql_data_seek [page 486]
Positions the cursor on row.

sasql_disconnect [page 487]
Closes a connection that has been opened with sasql_connect or sasql_pconnect.

sasql_error [page 488]
Returns the error text of the most recently executed function.

sasql_errorcode [page 489]
Returns the error code of the most-recently executed function.

sasql_escape_string [page 490]
Escapes all special characters in the supplied string.

sasql_fetch_array [page 490]
Fetches one row from the result set.

sasql_fetch_assoc [page 491]
Fetches one row from the result set as an associative array.

sasql_fetch_field [page 492]
Returns an object that contains information about a specific column.

sasql_fetch_object [page 493]
Fetches one row from the result set as an object.

sasql_fetch_row [page 494]
Fetches one row from the result set. This row is returned as an array that can be indexed by the column indexes only.

sasql_field_count [page 495]
Returns the number of columns (fields) the last result contains.

sasql_field_seek [page 495]
Sets the field cursor to the given offset.

sasql_free_result [page 496]
Frees database resources associated with a result resource returned from sasql_query, sasql_store_result, or sasql_use_result.

sasql_get_client_info [page 497]
Returns the version information of the client.

sasql_insert_id [page 497]
Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

sasql_message [page 498]
Writes a message to the server messages window.

sasql_multi_query [page 499]
Prepares and executes one or more SQL queries.

sasql_next_result [page 500]
Prepares the next result set from the last executed query.

sasql_num_fields [page 500]
Returns the number of fields in a row

sasql_num_rows [page 501]
Returns the number of rows in a result.

sasql_pconnect [page 502]
Establishes a persistent connection to a database.

sasql_prepare [page 503]
Prepares the supplied SQL string.

sasql_query [page 504]
Prepares and executes a SQL query.

sasql_real_escape_string [page 505]
Escapes all special characters in the supplied string.

sasql_real_query [page 506]
Executes a query against the database using the supplied connection resource.

sasql_result_all [page 507]
Fetches all results and generates an HTML output table with an optional formatting string.

sasql_rollback [page 508]
Ends a transaction on the database and discards any changes made during the transaction.

sasql_set_option [page 509]
Sets the value of the specified option on the specified connection.

sasql_stmt_affected_rows [page 510]
Returns the number of rows affected by executing the statement.

sasql_stmt_bind_param [page 511]
Binds PHP variables to statement parameters.

sasql_stmt_bind_param_ex [page 512]
Binds a PHP variable to a statement parameter.

sasql_stmt_bind_result [page 513]
Binds one or more PHP variables to result columns of a statement that was executed, and returns a result set.

sasql_stmt_close [page 514]
Closes the supplied statement resource and frees any resources associated with it.

sasql_stmt_data_seek [page 514]
Seeks to the specified offset in the result set.

sasql_stmt_errno [page 515]
Returns the error code for the most recently executed statement function using the specified statement resource.

sasql_stmt_error [page 516]
Returns the error text for the most recently executed statement function using the specified statement resource.

sasql_stmt_execute [page 517]
Executes the prepared statement. The sasql_stmt_result_metadata can be used to check whether the statement returns a result set.

sasql_stmt_fetch [page 517]
Fetches one row out of the result for the statement and places the columns in the variables that were bound using sasql_stmt_bind_result.

sasql_stmt_field_count [page 518]
Returns the number of columns in the result set of the statement.

sasql_stmt_free_result [page 519]
Frees the cached result set of the statement.

sasql_stmt_insert_id [page 519]
Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

sasql_stmt_next_result [page 520]
Advances to the next result from the statement.

sasql_stmt_num_rows [page 521]
Returns the number of rows in the result set.

sasql_stmt_param_count [page 522]
Returns the number of parameters in the supplied prepared statement resource.

sasql_stmt_reset [page 522]
Resets the statement object to the state just after the describe.

sasql_stmt_result_metadata [page 523]
Returns a result set object for the supplied statement.

sasql_stmt_send_long_data [page 524]
Allows the user to send parameter data in chunks.

sasql_stmt_store_result [page 525]
Allows the client to cache the whole result set of the statement.

sasql_store_result [page 525]
Transfers the result set from the last query.

sasql_sqlstate [page 527]
Returns the most recent SQLSTATE string.

sasql_use_result [page 528]
Initiates a result set retrieval for the last query that executed on the connection.

1.16.2.1 sasql_affected_rows

Returns the number of rows affected by the last SQL statement.

Syntax

```
int sasql_affected_rows( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

The number of rows affected.

Remarks

This function is typically used for INSERT, UPDATE, or DELETE statements. For SELECT statements, use the sasql_num_rows function.

Related Information

sasql_num_rows [page 501]

1.16.2.2 sasql_commit

Ends a transaction on the database and makes any changes made during the transaction permanent.

Syntax

```
bool sasql_commit( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on failure.

Remarks

Useful only when the auto_commit option is Off.

Related Information

sasql_rollback [page 508]

sasql_set_option [page 509]
sasql_pconnect [page 502]
sasql_disconnect [page 487]

1.16.2.3 sasql_close

Closes a previously opened database connection.

Syntax

```
bool sasql_close( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on failure.

1.16.2.4 sasql_connect

Establishes a connection to a database.

Syntax

```
sasql_conn sasql_connect( string $con_str )
```

Parameters

$con_str

A valid connection string.

Returns

A positively valued connection resource on success, or an error and 0 on failure.

Related Information

Alphabetical List of Connection Parameters
Database Connections
sasql_pconnect [page 502]
sasql_disconnect [page 487]

1.16.2.5 sasql_data_seek

Positions the cursor on row.

Syntax

```
bool sasql_data_seek( sasql_result $result, int row_num )
```

Parameters

$result

The result resource returned by the sasql_query function.

row_num

An integer that represents the new position of the cursor within the result resource. For example, specify 0 to move the cursor to the first row of the result set or 5 to move it to the sixth row. Negative numbers represent rows relative to the end of the result set. For example, -1 moves the cursor to the last row in the result set and -2 moves it to the second-last row.

Returns

TRUE on success or FALSE on error.

Remarks

Positions the cursor on row `row_num` on the `$result` that was opened using sasql_query.

Related Information

sasql_fetch_field [page 492]
sasql_fetch_array [page 490]
sasql_fetch_assoc [page 491]
sasql_fetch_row [page 494]
sasql_fetch_object [page 493]
sasql_query [page 504]

1.16.2.6 sasql_disconnect

Closes a connection that has been opened with sasql_connect or sasql_pconnect.

Syntax

```
bool sasql_disconnect( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on error.

Related Information

sasql_connect [page 485]
sasql_pconnect [page 502]

1.16.2.7 sasql_error

Returns the error text of the most recently executed function.

Syntax

```
string sasql_error( [ sasql_conn $conn ] )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

A string describing the error.

Remarks

Error messages are stored per connection. If no $conn is specified, then sasql_error returns the last error message where no connection was available. For example, if you call sasql_connect and the connection fails, then call sasql_error with no parameter for $conn to get the error message. To obtain the corresponding error code value, use the sasql_errorcode function.

Related Information

SQL Anywhere Error Messages
sasql_errorcode [page 489]
sasql_sqlstate [page 527]
sasql_set_option [page 509]
sasql_stmt_errno [page 515]
sasql_stmt_error [page 516]

1.16.2.8 sasql_errorcode

Returns the error code of the most-recently executed function.

Syntax

```
int sasql_errorcode( [ sasql_conn $conn ] )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

A signed integer representing an error code. An error code of 0 means success. A positive error code indicates success with warnings. A negative error code indicates failure.

Remarks

Error codes are stored per connection. If no `$conn` is specified, then sasql_errorcode returns the last error code where no connection was available. For example, if you are calling sasql_connect and the connection fails, then call sasql_errorcode with no parameter for the `$conn` to get the error code. To get the corresponding error message use the sasql_error function.

Related Information

sasql_connect [page 485]
sasql_pconnect [page 502]
sasql_error [page 488]
sasql_sqlstate [page 527]
sasql_set_option [page 509]
sasql_stmt_errno [page 515]
sasql_stmt_error [page 516]
SQL Anywhere Error Messages Sorted by SQLCODE

1.16.2.9 sasql_escape_string

Escapes all special characters in the supplied string.

Syntax

```
string sasql_escape_string( sasql_conn $conn, string $str )
```

Parameters

$conn

The connection resource returned by a connect function.

$string

The string to be escaped.

Returns

The escaped string.

Remarks

The special characters that are escaped are \r, \n, ', ", ;, \, and the NULL character. This function is an alias of sasql_real_escape_string.

Related Information

sasql_real_escape_string [page 505]
sasql_connect [page 485]

1.16.2.10 sasql_fetch_array

Fetches one row from the result set.

Syntax

```
array sasql_fetch_array( sasql_result $result [, int $result_type ])
```

Parameters

$result

The result resource returned by the sasql_query function.

$result_type

This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants SASQL_ASSOC, SASQL_NUM, or SASQL_BOTH. It defaults to SASQL_BOTH.

By using the SASQL_ASSOC constant, this function will behave identically to the sasql_fetch_assoc function, while SASQL_NUM will behave identically to the sasql_fetch_row function. The final option SASQL_BOTH will create a single array with the attributes of both.

Returns

An array that represents a row from the result set, or FALSE when no rows are available.

Remarks

The row is returned as an array that can be indexed by the column names or by the column indexes.

Related Information

sasql_data_seek [page 486]
sasql_fetch_assoc [page 491]
sasql_fetch_field [page 492]
sasql_fetch_row [page 494]
sasql_fetch_object [page 493]

1.16.2.11 sasql_fetch_assoc

Fetches one row from the result set as an associative array.

Syntax

```
array sasql_fetch_assoc( sasql_result $result )
```

Parameters

$result

The result resource returned by the sasql_query function.

Returns

An associative array of strings representing the fetched row in the result set, where each key in the array represents the name of one of the result set's columns or FALSE if there are no more rows in resultset.

Related Information

sasql_data_seek [page 486]
sasql_fetch_field [page 492]
sasql_fetch_field [page 492]
sasql_fetch_row [page 494]
sasql_fetch_object [page 493]

1.16.2.12 sasql_fetch_field

Returns an object that contains information about a specific column.

Syntax

```
object sasql_fetch_field( sasql_result $result [, int $field_offset ] )
```

Parameters

$result

The result resource returned by the sasql_query function.

$field_offset

An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

Returns

An object that has the following properties:

id

contains the field's number.

name

contains the field's name.

numeric

indicates whether the field is a numeric value.

length

returns the field's native storage size, or display size for fields with native_type equal to DT_DECIMAL.

type

returns the field's type.

native_type

returns the field's native type. These are values like DT_FIXCHAR, DT_DECIMAL or DT_DATE.

precision

returns the field's numeric precision. This property is only set for fields with native_type equal to DT_DECIMAL.

scale

returns the field's numeric scale. This property is only set for fields with native_type equal to DT_DECIMAL.

Related Information

Embedded SQL Data Types [page 277]
sasql_data_seek [page 486]
sasql_fetch_array [page 490]
sasql_fetch_assoc [page 491]
sasql_fetch_row [page 494]
sasql_fetch_object [page 493]

1.16.2.13 sasql_fetch_object

Fetches one row from the result set as an object.

Syntax

```
object sasql_fetch_object( sasql_result $result )
```

Parameters

$result

The result resource returned by the sasql_query function.

Returns

An object representing the fetched row in the result set where each property name matches one of the result set column names, or FALSE if there are no more rows in result set.

Related Information

sasql_data_seek [page 486]
sasql_fetch_field [page 492]
sasql_fetch_array [page 490]
sasql_fetch_assoc [page 491]
sasql_fetch_row [page 494]

1.16.2.14 sasql_fetch_row

Fetches one row from the result set. This row is returned as an array that can be indexed by the column indexes only.

Syntax

```
array sasql_fetch_row( sasql_result $result )
```

Parameters

$result

The result resource returned by the sasql_query function.

Returns

An array that represents a row from the result set, or FALSE when no rows are available.

Related Information

sasql_data_seek [page 486]
sasql_fetch_field [page 492]
sasql_fetch_array [page 490]
sasql_fetch_assoc [page 491]
sasql_fetch_object [page 493]

1.16.2.15 sasql_field_count

Returns the number of columns (fields) the last result contains.

Syntax

```
int sasql_field_count( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

A positive number of columns, or FALSE if `$conn` is not valid.

1.16.2.16 sasql_field_seek

Sets the field cursor to the given offset.

Syntax

```
bool sasql_field_seek( sasql_result $result, int $field_offset )
```

Parameters

$result

The result resource returned by the sasql_query function.

$field_offset

An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

Returns

TRUE on success or FALSE on error.

Remarks

The next call to sasql_fetch_field will retrieve the field definition of the column associated with that offset.

1.16.2.17 sasql_free_result

Frees database resources associated with a result resource returned from sasql_query, sasql_store_result, or sasql_use_result.

Syntax

```
bool sasql_free_result( sasql_result $result )
```

Parameters

$result

The result resource returned by the sasql_query function.

Returns

TRUE on success or FALSE on error.

Related Information

sasql_query [page 504]
sasql_store_result [page 525]
sasql_use_result [page 528]

1.16.2.18 sasql_get_client_info

Returns the version information of the client.

Syntax

```
string sasql_get_client_info( )
```

Parameters

None

Returns

A string that represents the client software version. The returned string is of the form X.Y.Z.W where X is the major version number, Y is the minor version number, Z is the patch number, and W is the build number (for example, 10.0.1.3616).

1.16.2.19 sasql_insert_id

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

Syntax

```
int sasql_insert_id( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

The ID generated for an AUTOINCREMENT column by a previous INSERT statement or zero if last insert did not affect an AUTOINCREMENT column. The function can return FALSE if the `$conn` is not valid.

Remarks

The sasql_insert_id function is provided for compatibility with MySQL databases.

1.16.2.20 sasql_message

Writes a message to the server messages window.

Syntax

```
bool sasql_message( sasql_conn $conn, string $message )
```

Parameters

$conn

The connection resource returned by a connect function.

$message

A message to be written to the server messages window.

Returns

TRUE on success or FALSE on failure.

1.16.2.21 sasql_multi_query

Prepares and executes one or more SQL queries.

Syntax

```
bool sasql_multi_query( sasql_conn $conn, string $sql_str )
```

Parameters

$conn

The connection resource returned by a connect function.

$sql_str

One or more SQL statements separated by semicolons.

Returns

TRUE on success or FALSE on failure.

Remarks

Prepares and executes one or more SQL queries specified by `$sql_str` using the supplied connection resource. Each query is separated from the other using semicolons.

The first query result can be retrieved or stored using sasql_use_result or sasql_store_result. sasql_field_count can be used to check if the query returns a result set or not.

All subsequent query results can be processed using sasql_next_result and sasql_use_result/ sasql_store_result.

Related Information

SQL Statements
sasql_store_result [page 525]
sasql_use_result [page 528]
sasql_field_count [page 495]

1.16.2.22 sasql_next_result

Prepares the next result set from the last executed query.

Syntax

```
bool sasql_next_result( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

FALSE if there is no other result set to be retrieved. TRUE if there is another result to be retrieved. Call sasql_use_result or sasql_store_result to retrieve the next result set.

Remarks

Prepares the next result set from the last query that executed on `$conn`.

Related Information

sasql_use_result [page 528]
sasql_store_result [page 525]

1.16.2.23 sasql_num_fields

Returns the number of fields in a row

Syntax

```
int sasql_num_fields( sasql_result $result )
```

Parameters

$result

The result resource returned by the sasql_query function.

Returns

Returns the number of fields in the specified result set.

Remarks

Returns the number of fields that a row in the `$result` contains.

Related Information

sasql_num_rows [page 501]
sasql_query [page 504]

1.16.2.24 sasql_num_rows

Returns the number of rows in a result.

Syntax

```
int sasql_num_rows( sasql_result $result )
```

Parameters

$result

The result resource returned by the sasql_query function.

Returns

A positive number if the number of rows is exact, or a negative number if it is an estimate. To get the exact number of rows, the database option row_counts must be set permanently on the database, or temporarily on the connection.

Remarks

Returns the number of rows that the `$result` contains.

Related Information

sasql_num_fields [page 500]
sasql_query [page 504]
sasql_set_option [page 509]

1.16.2.25 sasql_pconnect

Establishes a persistent connection to a database.

Syntax

```
sasql_conn sasql_pconnect( string $con_str )
```

Parameters

$con_str

A valid connection string.

Returns

A positively valued persistent connection resource on success, or an error and 0 on failure.

Remarks

Because of the way Apache creates child processes, you may observe a performance gain when using sasql_pconnect instead of sasql_connect. Persistent connections may provide improved performance in a similar fashion to connection pooling. If your database server has a limited number of connections (for example, the personal database server is limited to 10 concurrent connections), caution should be exercised when using persistent connections. Persistent connections could be attached to each of the child processes, and if you have more child processes in Apache than there are available connections, you will receive connection errors.

Related Information

Alphabetical List of Connection Parameters
Database Connections
sasql_connect [page 485]
sasql_disconnect [page 487]

1.16.2.26 sasql_prepare

Prepares the supplied SQL string.

Syntax

```
sasql_stmt sasql_prepare( sasql_conn $conn, string $sql_str )
```

Parameters

$conn

The connection resource returned by a connect function.

$sql_str

The SQL statement to be prepared. The string can include parameter markers by embedding question marks at the appropriate positions.

Returns

A statement object or FALSE on failure. To determine the cause of failure, use the sasql_error and sasql_errorcode functions.

Related Information

SQL Statements
sasql_stmt_param_count [page 522]
sasql_stmt_bind_param [page 511]
sasql_stmt_bind_param_ex [page 512]
sasql_error [page 488]
sasql_errorcode [page 489]
sasql_stmt_execute [page 517]
sasql_connect [page 485]
sasql_pconnect [page 502]

1.16.2.27 sasql_query

Prepares and executes a SQL query.

Syntax

```
mixed sasql_query( sasql_conn $conn, string $sql_str [, int $result_mode ] )
```

Parameters

$conn

The connection resource returned by a connect function.

$sql_str

A SQL statement supported by the database server.

$result_mode

Either SASQL_USE_RESULT, or SASQL_STORE_RESULT (the default).

Returns

FALSE on failure; TRUE on success for INSERT, UPDATE, DELETE, CREATE; sasql_result for SELECT.

Remarks

Prepares, describes, and executes the SQL query `$sql_str` on the connection identified by `$conn` that has already been opened using sasql_connect or sasql_pconnect.

The sasql_query function is equivalent to calling two functions, sasql_real_query and one of sasql_store_result or sasql_use_result.

If a warning is generated at the prepare and describe steps, it will be overwritten if the execute succeeds. If you need to obtain any warnings from each step, then use the sasql_prepare, sasql_error, and sasql_errorcode functions.

Related Information

SQL Statements
sasql_real_query [page 506]
sasql_free_result [page 496]
sasql_fetch_array [page 490]
sasql_fetch_field [page 492]
sasql_fetch_object [page 493]
sasql_fetch_row [page 494]

1.16.2.28 sasql_real_escape_string

Escapes all special characters in the supplied string.

Syntax

```
string sasql_real_escape_string( sasql_conn $conn, string $str )
```

Parameters

$conn

The connection resource returned by a connect function.

$string

The string to be escaped.

Returns

The escaped string or FALSE on error.

Remarks

The special characters that are escaped are \r, \n, ', ", ;, \, and the NULL character.

Related Information

sasql_escape_string [page 490]
sasql_connect [page 485]

1.16.2.29 sasql_real_query

Executes a query against the database using the supplied connection resource.

Syntax

```
bool sasql_real_query( sasql_conn $conn, string $sql_str )
```

Parameters

$conn

The connection resource returned by a connect function.

$sql_str

A SQL statement supported by the database server.

Returns

TRUE on success or FALSE on failure.

Remarks

The query result can be retrieved or stored using sasql_store_result or sasql_use_result. The sasql_field_count function can be used to check if the query returns a result set or not.

The sasql_query function is equivalent to calling this function and one of sasql_store_result or sasql_use_result.

Related Information

sasql_query [page 504]
sasql_store_result [page 525]
sasql_use_result [page 528]
sasql_field_count [page 495]

1.16.2.30 sasql_result_all

Fetches all results and generates an HTML output table with an optional formatting string.

Syntax

```
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
] ] ] ] )
```

Parameters

$result

The result resource returned by the sasql_query function.

$html_table_format_string

A format string that applies to HTML tables. For example, **"Border=1; Cellpadding=5"**. The special value none does not create an HTML table. This is useful to customize your column names or scripts. To avoid specifying an explicit value for this parameter, use NULL for the parameter value.

$html_table_header_format_string

A format string that applies to column headings for HTML tables. For example, **"bgcolor=#FF9533"**. The special value none does not create an HTML table. This is useful to customize your column names or scripts. To avoid specifying an explicit value for this parameter, use NULL for the parameter value.

$html_table_row_format_string

A format string that applies to rows within HTML tables. For example, `"onclick='alert('this')'"`. If you would like different formats that alternate, use the special token ><. The left side of the token indicates which format to use on odd rows and the right side of the token is used to format even rows. If you do not place this token in your format string, all rows have the same format. If you do not want to specify an explicit value for this parameter, use NULL for the parameter value.

$html_table_cell_format_string

A format string that applies to cells within HTML table rows. For example, `"onclick='alert('this')'"`. If you do not want to specify an explicit value for this parameter, use NULL for the parameter value.

Returns

TRUE on success or FALSE on failure.

Remarks

Fetches all results of the `$result` and generates an HTML output table with an optional formatting string.

Related Information

sasql_query [page 504]

1.16.2.31 sasql_rollback

Ends a transaction on the database and discards any changes made during the transaction.

Syntax

```
bool sasql_rollback( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on failure.

Remarks

This function is only useful when the auto_commit option is Off.

Related Information

sasql_commit [page 484]
sasql_set_option [page 509]

1.16.2.32 sasql_set_option

Sets the value of the specified option on the specified connection.

Syntax

```
bool sasql_set_option( sasql_conn $conn, string $option, mixed $value )
```

Parameters

$conn

The connection resource returned by a connect function.

$option

The name of the option you want to set.

$value

The new option value.

Returns

TRUE on success or FALSE on failure.

Remarks

You can set the value for the following options:

Name	Description	Default
auto_commit	When this option is set to on, the database server commits after executing each statement.	on
row_counts	When this option is set to FALSE, the sasql_num_rows function returns an estimate of the number of rows affected. To obtain an exact count, set this option to TRUE.	FALSE
verbose_errors	When this option is set to TRUE, the PHP driver returns verbose errors. When this option is set to FALSE, you must call the sasql_error or sasql_errorcode functions to get further error information.	TRUE

You can change the default value for an option by including the following line in the `php.ini` file. In this example, the default value is set for the auto_commit option.

```
sqlanywhere.auto_commit=0
```

Related Information

sasql_commit [page 484]
sasql_error [page 488]
sasql_errorcode [page 489]
sasql_num_rows [page 501]
sasql_rollback [page 508]

1.16.2.33 sasql_stmt_affected_rows

Returns the number of rows affected by executing the statement.

Syntax

```
int sasql_stmt_affected_rows( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource that was executed by sasql_stmt_execute.

Returns

The number of rows affected or FALSE on failure.

Related Information

sasql_stmt_execute [page 517]

1.16.2.34 sasql_stmt_bind_param

Binds PHP variables to statement parameters.

Syntax

```
bool sasql_stmt_bind_param( sasql_stmt $stmt, string $types, mixed &$var_1 [,
mixed &$var_2 .. ] )
```

Parameters

$stmt

A prepared statement resource that was returned by the sasql_prepare function.

$types

A string that contains one or more characters specifying the types of the corresponding bind. This can be any of: *s* for string, *i* for integer, *d* for double, *b* for blobs. The length of the $types string must match the number of parameters that follow the $types parameter ($var_1, $var_2, ...). The number of characters should also match the number of parameter markers (question marks) in the prepared statement.

$var_n

The variable references.

Returns

TRUE if binding the variables was successful or FALSE otherwise.

Related Information

sasql_prepare [page 503]
sasql_stmt_param_count [page 522]
sasql_stmt_bind_param_ex [page 512]
sasql_stmt_execute [page 517]
sasql_stmt_errno [page 515]
sasql_stmt_error [page 516]

1.16.2.35 sasql_stmt_bind_param_ex

Binds a PHP variable to a statement parameter.

Syntax

```
bool sasql_stmt_bind_param_ex( sasql_stmt $stmt, int $param_number, mixed &$var,
string $type [, bool $is_null [, int $direction ] ] )
```

Parameters

$stmt

A prepared statement resource that was returned by the sasql_prepare function.

$param_number

The parameter number. This should be a number between 0 and (sasql_stmt_param_count($stmt) - 1).

$var

A PHP variable. Only references to PHP variables are allowed.

$type

Type of the variable. This can be one of: *s* for string, *i* for integer, *d* for double, *b* for blobs.

$is_null

Whether the value of the variable is NULL or not.

$direction

Can be SASQL_D_INPUT, SASQL_D_OUTPUT, or SASQL_INPUT_OUTPUT.

Returns

TRUE if binding the variable was successful or FALSE otherwise.

Related Information

sasql_prepare [page 503]
sasql_stmt_param_count [page 522]
sasql_stmt_bind_param [page 511]
sasql_stmt_execute [page 517]

1.16.2.36 sasql_stmt_bind_result

Binds one or more PHP variables to result columns of a statement that was executed, and returns a result set.

Syntax

```
bool sasql_stmt_bind_result( sasql_stmt $stmt, mixed &$var1 [, mixed &$var2 .. ] )
```

Parameters

$stmt

A statement resource that was executed by sasql_stmt_execute.

$var1

References to PHP variables that are bound to result set columns returned by the sasql_stmt_fetch.

Returns

TRUE on success or FALSE on failure.

Related Information

sasql_stmt_execute [page 517]
sasql_stmt_fetch [page 517]

1.16.2.37 sasql_stmt_close

Closes the supplied statement resource and frees any resources associated with it.

Syntax

```
bool sasql_stmt_close( sasql_stmt $stmt )
```

Parameters

$stmt

A prepared statement resource that was returned by the sasql_prepare function.

Returns

TRUE for success or FALSE on failure.

Remarks

This function will also free any result objects that were returned by the sasql_stmt_result_metadata.

Related Information

sasql_stmt_result_metadata [page 523]
sasql_prepare [page 503]

1.16.2.38 sasql_stmt_data_seek

Seeks to the specified offset in the result set.

Syntax

```
bool sasql_stmt_data_seek( sasql_stmt $stmt, int $offset )
```

Parameters

$stmt

A statement resource.

$offset

The offset in the result set. This is a number between 0 and (sasql_stmt_num_rows($stmt) - 1).

Returns

TRUE on success or FALSE failure.

Related Information

sasql_stmt_num_rows [page 521]

1.16.2.39 sasql_stmt_errno

Returns the error code for the most recently executed statement function using the specified statement resource.

Syntax

```
int sasql_stmt_errno( sasql_stmt $stmt )
```

Parameters

$stmt

A prepared statement resource that was returned by the sasql_prepare function.

Returns

An integer error code.

Related Information

sasql_stmt_error [page 516]
sasql_error [page 488]
sasql_errorcode [page 489]
sasql_prepare [page 503]
sasql_stmt_result_metadata [page 523]
SQL Anywhere Error Messages Sorted by SQLCODE

1.16.2.40 sasql_stmt_error

Returns the error text for the most recently executed statement function using the specified statement resource.

Syntax

```
string sasql_stmt_error( sasql_stmt $stmt )
```

Parameters

$stmt

A prepared statement resource that was returned by the sasql_prepare function.

Returns

A string describing the error.

Related Information

SQL Anywhere Error Messages
sasql_stmt_errno [page 515]
sasql_error [page 488]
sasql_errorcode [page 489]
sasql_prepare [page 503]
sasql_stmt_result_metadata [page 523]

1.16.2.41 sasql_stmt_execute

Executes the prepared statement. The sasql_stmt_result_metadata can be used to check whether the statement returns a result set.

Syntax

```
bool sasql_stmt_execute( sasql_stmt $stmt )
```

Parameters

$stmt

A prepared statement resource that was returned by the sasql_prepare function. Variables should be bound before calling execute.

Returns

TRUE for success or FALSE on failure.

Related Information

sasql_prepare [page 503]
sasql_stmt_param_count [page 522]
sasql_stmt_bind_param [page 511]
sasql_stmt_bind_param_ex [page 512]
sasql_stmt_result_metadata [page 523]
sasql_stmt_bind_result [page 513]

1.16.2.42 sasql_stmt_fetch

Fetches one row out of the result for the statement and places the columns in the variables that were bound using sasql_stmt_bind_result.

Syntax

```
bool sasql_stmt_fetch( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource.

Returns

TRUE on success or FALSE on failure.

Related Information

sasql_stmt_bind_result [page 513]

1.16.2.43 sasql_stmt_field_count

Returns the number of columns in the result set of the statement.

Syntax

```
int sasql_stmt_field_count( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource.

Returns

The number of columns in the result of the statement. If the statement does not return a result, it returns 0.

Related Information

sasql_stmt_result_metadata [page 523]

1.16.2.44 sasql_stmt_free_result

Frees the cached result set of the statement.

Syntax

```
bool sasql_stmt_free_result( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource that was executed using sasql_stmt_execute.

Returns

TRUE on success or FALSE on failure.

Related Information

sasql_stmt_execute [page 517]
sasql_stmt_store_result [page 525]

1.16.2.45 sasql_stmt_insert_id

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

Syntax

```
int sasql_stmt_insert_id( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource that was executed by sasql_stmt_execute.

Returns

The ID generated for an IDENTITY column or a DEFAULT AUTOINCREMENT column by a previous INSERT statement, or zero if the last insert did not affect an IDENTITY or DEFAULT AUTOINCREMENT column. The function can return FALSE (0) if $stmt is not valid.

Related Information

sasql_stmt_execute [page 517]

1.16.2.46 sasql_stmt_next_result

Advances to the next result from the statement.

Syntax

```
bool sasql_stmt_next_result( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource.

Returns

TRUE on success or FALSE failure.

Remarks

If there is another result set, the currently cashed results are discarded and the associated result set object deleted (as returned by sasql_stmt_result_metadata).

Related Information

sasql_stmt_result_metadata [page 523]

1.16.2.47 sasql_stmt_num_rows

Returns the number of rows in the result set.

Syntax

```
int sasql_stmt_num_rows( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource that was executed by sasql_stmt_execute and for which sasql_stmt_store_result was called.

Returns

The number of rows available in the result or 0 on failure.

Remarks

The actual number of rows in the result set can only be determined after the sasql_stmt_store_result function is called to buffer the entire result set. If the sasql_stmt_store_result function has not been called, 0 is returned.

Related Information

sasql_stmt_execute [page 517]
sasql_stmt_store_result [page 525]

1.16.2.48 sasql_stmt_param_count

Returns the number of parameters in the supplied prepared statement resource.

Syntax

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource returned by the sasql_prepare function.

Returns

The number of parameters or FALSE on error.

Related Information

sasql_prepare [page 503]
sasql_stmt_bind_param [page 511]
sasql_stmt_bind_param_ex [page 512]

1.16.2.49 sasql_stmt_reset

Resets the statement object to the state just after the describe.

Syntax

```
bool sasql_stmt_reset( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource.

Returns

TRUE on success or FALSE on failure.

Remarks

This function resets the `$stmt` object to the state just after the describe. Any variables that were bound are unbound and any data sent using sasql_stmt_send_long_data are dropped.

Related Information

sasql_stmt_send_long_data [page 524]

1.16.2.50 sasql_stmt_result_metadata

Returns a result set object for the supplied statement.

Syntax

```
sasql_result sasql_stmt_result_metadata( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource that was prepared and executed.

Returns

sasql_result object or FALSE if the statement does not return any results.

1.16.2.51 sasql_stmt_send_long_data

Allows the user to send parameter data in chunks.

Syntax

```
bool sasql_stmt_send_long_data( sasql_stmt $stmt, int $param_number, string $data )
```

Parameters

$stmt

A statement resource that was prepared using sasql_prepare.

$param_number

The parameter number. This must be a number between 0 and (sasql_stmt_param_count($stmt) - 1).

$data

The data to be sent.

Returns

TRUE on success or FALSE on failure.

Remarks

The user must first call sasql_stmt_bind_param or sasql_stmt_bind_param_ex before attempting to send any data. The bind parameter must be of type string or blob. Repeatedly calling this function appends on to what was previously sent.

Related Information

sasql_stmt_bind_param [page 511]
sasql_stmt_bind_param_ex [page 512]
sasql_prepare [page 503]
sasql_stmt_param_count [page 522]

1.16.2.52 sasql_stmt_store_result

Allows the client to cache the whole result set of the statement.

Syntax

```
bool sasql_stmt_store_result( sasql_stmt $stmt )
```

Parameters

$stmt

A statement resource that was executed using sasql_stmt_execute.

Returns

TRUE on success or FALSE on failure.

Remarks

You can use the function sasql_stmt_free_result to free the cached result.

Related Information

sasql_stmt_free_result [page 519]
sasql_stmt_execute [page 517]

1.16.2.53 sasql_store_result

Transfers the result set from the last query.

Syntax

```
sasql_result sasql_store_result( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

FALSE if the query does not return a result object, or a result set object, that contains all the rows of the result. The result is cached at the client.

Remarks

This function is called after executing sasql_real_query. After a successful execution of sasql_real_query, use sasql_store_result to retrieve a handle to the result set returned from the query. The result set handle represents data that has been fetched from the server.

The sasql_store_result function caches the result set on the client side. To prevent caching of the result set on the client side, use sasql_use_result instead.

Example

```
<?php
    $conn = sasql_connect( "UID=DBA;PWD=sql" )
        or die( "Cannot connect to database" );
    if( sasql_real_query( $conn, "SELECT * FROM Customers" ) )
    {
        $num_cols = sasql_field_count( $conn );
        $result = sasql_store_result( $conn );
        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";
        }
        sasql_free_result( $result );
        echo "$curr_row rows.\n";
    }
    sasql_close( $conn );
?>
```

Related Information

sasql_data_seek [page 486]
sasql_real_query [page 506]
sasql_free_result [page 496]
sasql_use_result [page 528]

1.16.2.54 sasql_sqlstate

Returns the most recent SQLSTATE string.

Syntax

```
string sasql_sqlstate( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

Returns a string of five characters containing the current SQLSTATE code. The result "00000" means no error.

Remarks

SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition. SQLSTATE codes consists of five characters with "00000" representing no error. The values are defined by the ISO/ANSI SQL standard.

Related Information

sasql_error [page 488]
sasql_errorcode [page 489]
SQL Anywhere Error Messages Sorted by SQLSTATE

1.16.2.55 sasql_use_result

Initiates a result set retrieval for the last query that executed on the connection.

Syntax

```
sasql_result sasql_use_result( sasql_conn $conn )
```

Parameters

$conn

The connection resource returned by a connect function.

Returns

FALSE if the query does not return a result object or a result set object. The result is not cached on the client.

Remarks

This function is called after executing sasql_real_query. After a successful execution of sasql_real_query, use sasql_use_result to retrieve a handle to the result set returned from the query. The result set handle represents data that has not been fetched from the server yet. Each fetch using this result set handle sends a request to the server to retrieve a row of the result set.

The sasql_use_result function prevents caching of the result set on the client side. To cache the result set on the client side, use sasql_store_result instead.

Example

```
<?php
    $conn = sasql_connect( "UID=DBA;PWD=sql" )
        or die( "Cannot connect to database" );
    if( sasql_real_query( $conn, "SELECT * FROM Customers" ) )
    {
        $num_cols = sasql_field_count( $conn );
        $result = sasql_use_result( $conn );
        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";
        }
        sasql_free_result( $result );
        echo "$curr_row rows.\n";
    }
    sasql_close( $conn );
?>
```

Related Information

sasql_data_seek [page 486]
sasql_real_query [page 506]
sasql_free_result [page 496]
sasql_store_result [page 525]

1.17 Ruby Support

You can use Ruby to develop database applications.

In this section:

Ruby Programming [page 529]
There are three different Ruby Application Programming Interfaces supported by SQL Anywhere.

SQL Anywhere Ruby API Reference [page 539]
The Ruby extension API permits the rapid development of SQL applications. This extension is built on a low-level interface called the SQL Anywhere C API.

1.17.1 Ruby Programming

There are three different Ruby Application Programming Interfaces supported by SQL Anywhere.

First, there is the Native Ruby API. This API provides a Ruby wrapping over the interface exposed by the SQL Anywhere C API.

Second, there is support for ActiveRecord, an object-relational mapper popularized by being part of the Ruby on Rails web development framework.

Third, there is support for Ruby DBI. A Ruby Database Driver (DBD) is provided which can be used with DBI.

There are three separate packages available in the SQL Anywhere for Ruby project. The simplest way to install any of these packages is to use RubyGems.

> **Note**
>
> If you are using macOS 10.11 or a later version, then set the SQLANY_API_DLL environment variable to the full path to `libdbcapi_r.dylib`.

Native Ruby Driver

sqlanywhere

This package is a low-level driver that allows Ruby code to interface with SQL Anywhere databases. This package provides a Ruby wrapping over the interface exposed by the SQL Anywhere C API. This package is written in C and is available as source, or as pre-compiled gems, for Windows and Linux. If you have RubyGems installed, this package can be obtained by running the following command:

```
gem install sqlanywhere
```

This package is a prerequisite for any of the other Ruby interfaces.

ActiveRecord Adapter

activerecord-sqlanywhere-adapter

This package is an adapter that allows ActiveRecord to communicate with the database server. ActiveRecord is an object-relational mapper, popularized by being part of the Ruby on Rails web development framework. This package is written in pure Ruby, and available in source, or gem format. This adapter uses (and has a dependency on) the sqlanywhere gem. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install activerecord-sqlanywhere-adapter
```

Ruby/DBI Driver

dbi

This package is a DBI driver for Ruby. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install dbi
```

dbd-sqlanywhere

This package is a driver that allows Ruby/DBI to communicate with the database server. Ruby/DBI is a generic database interface modeled after the popular Perl DBI module. This package is written in pure Ruby, and available in source, or gem format. This driver uses (and has a dependency on) the sqlanywhere

gem. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install dbd-sqlanywhere
```

In this section:

Ruby on Rails Support [page 531]
Rails is a web development framework written in the Ruby language. Its strength is in web application development. A familiarity with the Ruby programming language is highly recommended before you attempt Rails development.

Ruby-DBI Driver [page 535]
You can write Ruby applications that use the SQL Anywhere DBI driver.

Related Information

SQL Anywhere Ruby API Reference [page 539]
Ruby Programming
Ruby Gems
SQL Anywhere Ruby Driver
Ruby/DBI, a database abstraction layer and interface
Ruby on Rails
SAP SQL Anywhere Forum

1.17.1.1 Ruby on Rails Support

Rails is a web development framework written in the Ruby language. Its strength is in web application development. A familiarity with the Ruby programming language is highly recommended before you attempt Rails development.

Before beginning Rails development, Rails must be configured.

Once you have configured Rails, consider trying the tutorial on the Ruby on Rails website. Instructions for adapting the tutorial to SQL Anywhere are presented below.

In this section:

Configuring Rails Support [page 532]
Add SQL Anywhere to the set of database management systems supported by Rails.

Learning Rails [page 534]
Use the tutorial on the Ruby on Rails website to learn about Rails development.

Related Information

SQL Anywhere Ruby API Reference [page 539]

1.17.1.1.1 Configuring Rails Support

Add SQL Anywhere to the set of database management systems supported by Rails.

Procedure

1. Install the Ruby interpreter on your system.
2. Install RubyGems. It simplifies the installation of Ruby packages. The Ruby on Rails download page recommends which version to install.
3. Install Rails and its dependencies by running the following command:

   ```
   gem install rails
   ```
4. Install the correct Ruby Development Kit (DevKit).
5. Install the ActiveRecord support (activerecord-sqlanywhere-adapter) by running the following command:

   ```
   gem install activerecord-sqlanywhere-adapter
   ```
6. Add SQL Anywhere to the set of database management systems supported by Rails. At the time of writing, Rails 3.1.13 was the current released version.
   a. Configure a database by creating a `sqlanywhere.yml` file in the Rails `configs\databases` directory. If you have installed Ruby in the `\Ruby` directory and you have installed version 3.1.13 of Rails, then the path to this file would be `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.13\lib\rails\generators\rails\app\templates\config\databases`. The contents of this file should be:

      ```
      #
      # 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: password
      # 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: password
production:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_production
  username: DBA
  password: password
```

The `sqlanywhere.yml` file provides a template for creating `database.yml` files in Rails projects. The following database options can be specified:

adapter

(required, no default). This option must be set to sqlanywhere to use the ActiveRecord adapter.

database

(required, no default) This option corresponds to DatabaseName in a connection string.

server

(optional, defaults to the *database* option) This option corresponds to ServerName in a connection string.

username

This option corresponds to UserID in a connection string.

password

This option corresponds to Password in a connection string.

encoding

(optional, defaults to the OS character set) This option corresponds to CharSet in a connection string.

commlinks

(optional) This option corresponds to CommLinks in a connection string.

connection_name

(optional) This option corresponds to ConnectionName in connection string.

b. Update the Rails `app_base.rb` file. Using the same assumptions in the previous step, this file is located in the path `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.2.13\lib\rails\generators\app_base.rb`. Edit the `app_base.rb` file and locate the following line:

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase ibm_db
sqlserver )
```

Add sqlanywhere to the list as follows:

```
DATABASES = %w( sqlanywhere mysql oracle postgresql sqlite3 frontbase
ibm_db sqlserver )
```

Results

You have successfully configured Rails support.

Related Information

Ruby on Rails
Downloads
Ruby Development Kit (DevKit)

1.17.1.1.2 Learning Rails

Use the tutorial on the Ruby on Rails website to learn about Rails development.

Prerequisites

Rails must be configured for use with SQL Anywhere.

Use the tutorial on the Ruby on Rails website in conjunction with the steps below to become familiar with Rails development.

Procedure

1. In the tutorial, you are shown the command to initialize the **`blog`** project. Here is the command to initialize the **`blog`** project for use with SQL Anywhere:

   ```
   rails new blog -d sqlanywhere
   ```

2. After you create the **`blog`** application, switch to its folder to continue work directly in that application:

   ```
   cd blog
   ```

3. Edit the `Gemfile` file to include a *gem* directive for the SQL Anywhere ActiveRecord adapter. Add the new directive following the indicated line below:

   ```
   gem 'sqlanywhere'
   gem 'activerecord-sqlanywhere-adapter'
   ```

4. The `config\database.yml` file references the development, test, and production databases. Instead of using a *rake* command to create the databases as indicated by the tutorial, change to the `db` directory of the project and create three databases as follows.

   ```
   cd db
   dbinit -dba DBA,password blog_development
   dbinit -dba DBA,password blog_test
   dbinit -dba DBA,password blog_production
   ```

5. Start the database server and the three databases and then change to the `blog` directory as follows.

   ```
   dbsrv17 -n blog blog_development.db blog_production.db blog_test.db
   cd ..
   ```

   The database server name in the command line (`blog`) must match the name specified by the *server:* tags in the `config\database.yml` file. The `sqlanywhere.yml` template file is configured to ensure that the database server name matches the project name in all generated `database.yml` files.

6. Follow the remaining steps in the tutorial.

Results

You are now prepared for Ruby on Rails software development.

Related Information

Configuring Rails Support [page 532]
Ruby on Rails Guides

1.17.1.2 Ruby-DBI Driver

You can write Ruby applications that use the SQL Anywhere DBI driver.

Loading the DBI Module

To use the DBI:SQLAnywhere interface from a Ruby application, you must first tell Ruby that you plan to use the Ruby DBI module. To do so, include the following line near the top of the Ruby source file.

```
require 'dbi'
```

The DBI module automatically loads the database driver (DBD) interface as required.

Opening and Closing a Connection

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the connect function. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The call to the connect function takes the general form:

```
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password, options)
```

server-name

is the name of the database server that you want to connect to. Alternately, you can specify a connection string in the format "option1=value1;option2=value2;...".

user-id

is a valid user ID. Unless this string is empty, ";UID=value" is appended to the connection string.

password

is the corresponding password for the user ID. Unless this string is empty, ";PWD=value" is appended to the connection string.

options

is a hash of additional connection parameters such as DatabaseName, DatabaseFile, and ConnectionName. These are appended to the connection string in the format "option1=value1;option2=value2;...".

To demonstrate the connect function, start the database server and sample database before running the sample Ruby scripts.

```
dbsrv17 -n myserver "%SQLANYSAMP17%\demo.db"
```

The following code sample opens and closes a connection to the sample database. The string "myserver" in the example below is the server name.

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', 'DBA', 'sql') do |dbh|
    if dbh.ping
       print "Successfully Connected\n"
       dbh.disconnect()
    end
end
```

Optionally, you can specify a connection string in place of the server name. For example, in the above script may be altered by replacing the first parameter to the connect function as follows:

```
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
```

The user ID and password cannot be specified in the connection string. Ruby DBI automatically fills in the username and password with defaults if these arguments are omitted, so never include a UID or PWD connection parameter in your connection string. If you do, an exception is thrown.

The following example shows how additional connection parameters can be passed to the connect function as a hash of keyword and value pairs.

```
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
```

Selecting Data

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

A SQL statement must be executed first. If the statement returns a result set, you use the resulting statement handle to retrieve meta information about the result set and the rows of the result set. The following example obtains the column names from the metadata and displays the column names and values for each row fetched.

```
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
```

The first few lines of output that appear are reproduced below.

```
# 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
```

It is important to call finish to release the statement handle when you are done. If you do not, then you may get an error like the following:

```
Resource governor for 'prepared statements' exceeded
```

To detect handle leaks, the database server limits the number of cursors and prepared statements permitted to a maximum of 50 per connection by default. The resource governor automatically generates an error if these limits are exceeded. If you get this error, check for undestroyed statement handles. Use prepare_cached sparingly, as the statement handles are not destroyed.

If necessary, you can alter these limits by setting the max_cursor_count and max_statement_count options.

Inserting Rows

Inserting rows requires a handle to an open connection. The simplest way to insert rows is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The statement is first prepared, and then executed once per new row. The new row values are supplied as parameters to the execute method.

```
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
```

Related Information

max_cursor_count Option
max_statement_count Option
Ruby/DBI, a database abstraction layer and interface

1.17.2 SQL Anywhere Ruby API Reference

The Ruby extension API permits the rapid development of SQL applications. This extension is built on a low-level interface called the SQL Anywhere C API.

To demonstrate the power of Ruby application development, consider the following sample Ruby program. It loads the Ruby extension, connects to the sample database, lists column values from the Products table, disconnects, and terminates.

```
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()
puts "Enter user ID"
myuid = STDIN.gets.chomp
puts "Enter password"
mypwd = STDIN.gets.chomp
api.sqlany_connect( conn, "DSN=SQL Anywhere 17
Demo;UserID="+myuid+";Password="+mypwd )
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
    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 )
```

The first two rows of the result set output from this Ruby program are shown below:

```
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
```

In this section:

sqlany_affected_rows Function [page 542]
Returns the number of rows affected by execution of the prepared statement.

sqlany_bind_param Function [page 543]
Binds a user-supplied buffer as a parameter to the prepared statement.

sqlany_clear_error Function [page 544]
Clears the last stored error code.

sqlany_client_version Function [page 544]
Returns the current client version.

sqlany_commit Function [page 545]
Commits the current transaction.

sqlany_connect Function [page 546]
Creates a connection to a database server using the specified connection object and connection string.

sqlany_describe_bind_param Function [page 547]
Describes the bind parameters of a prepared statement.

sqlany_disconnect Function [page 548]
Disconnects a database connection. All uncommitted transactions are rolled back.

sqlany_error Function [page 549]
Returns the last error code and message stored in the connection object.

sqlany_execute Function [page 550]
Executes a prepared statement.

sqlany_execute_direct Function [page 551]
Executes the SQL statement specified by the string argument.

sqlany_execute_immediate Function [page 552]
Executes the specified SQL statement immediately without returning a result set. It is useful for statements that do not return result sets.

sqlany_fetch_absolute Function [page 553]
Moves the current row in the result set to the row number specified and then fetches the data at that row.

sqlany_fetch_next Function [page 554]
Returns the next row from the result set. This function first advances the row pointer and then fetches the data at the new row.

sqlany_fini Function [page 555]
Frees resources allocated by the API.

sqlany_free_connection Function [page 556]
Frees the resources associated with a connection object.

sqlany_free_stmt Function [page 557]
Frees resources associated with a statement object.

sqlany_get_bind_param_info Function [page 558]
Retrieves information about the parameters that were bound using sqlany_bind_param.

sqlany_get_column Function [page 559]
Returns the value fetched for the specified column.

sqlany_get_column_info Function [page 560]
Gets column information for the specified result set column.

sqlany_get_next_result Function [page 561]
Advances to the next result set in a multiple result set query.

sqlany_init Function [page 562]
Initializes the interface.

sqlany_new_connection Function [page 563]
Creates a connection object.

sqlany_num_cols Function [page 564]
Returns number of columns in the result set.

sqlany_num_params Function [page 564]
Returns the number of parameters that are expected for a prepared statement.

sqlany_num_rows Function [page 565]
Returns the number of rows in the result set.

sqlany_prepare Function [page 566]
Prepares the supplied SQL string.

sqlany_rollback Function [page 568]
Rolls back the current transaction.

sqlany_sqlstate Function [page 568]
Retrieves the current SQLSTATE.

Column Types [page 569]

The following Ruby class defines the column types returned by some Ruby extension functions.

Native Column Types [page 570]
The following table defines the native column types returned by some Ruby extension functions.

1.17.2.1 sqlany_affected_rows Function

Returns the number of rows affected by execution of the prepared statement.

Syntax

```
sqlany_affected_rows ( $stmt )
```

Parameters

$stmt

A statement that was prepared and executed successfully in which no result set was returned. For example, an INSERT, UPDATE or DELETE statement was executed.

Returns

Returns a scalar value that is the number of rows affected, or -1 on failure.

Example

```
affected = api.sqlany_affected( stmt )
```

Related Information

sqlany_execute Function [page 550]

1.17.2.2 sqlany_bind_param Function

Binds a user-supplied buffer as a parameter to the prepared statement.

Syntax

```
sqlany_bind_param ( $stmt, $index, $param )
```

Parameters

$stmt

A statement object returned by the successful execution of sqlany_prepare.

$index

The index of the parameter. The number must be between 0 and sqlany_num_params() - 1.

$param

A filled bind object retrieved from sqlany_describe_bind_param.

Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

Example

```
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 )
```

Related Information

sqlany_describe_bind_param Function [page 547]

1.17.2.3 sqlany_clear_error Function

Clears the last stored error code.

Syntax

```
sqlany_clear_error ( $conn )
```

Parameters

$conn

A connection object returned from sqlany_new_connection.

Returns

Returns nil.

Example

```
api.sqlany_clear_error( conn )
```

Related Information

sqlany_new_connection Function [page 563]

1.17.2.4 sqlany_client_version Function

Returns the current client version.

Syntax

```
sqlany_client_version ( )
```

Returns

Returns a scalar value that is the client version string.

Example

```
buffer = api.sqlany_client_version()
```

1.17.2.5 sqlany_commit Function

Commits the current transaction.

Syntax

```
sqlany_commit ( $conn )
```

Parameters

$conn

The connection object on which the commit operation is to be performed.

Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

Example

```
rc = api.sqlany_commit( conn )
```

Related Information

sqlany_rollback Function [page 568]

1.17.2.6 sqlany_connect Function

Creates a connection to a database server using the specified connection object and connection string.

Syntax

```
sqlany_connect ( $conn, $str )
```

Parameters

$conn

The connection object created by sqlany_new_connection.

$str

A valid connection string.

Returns

Returns a scalar value that is 1 if the connection is established successfully or 0 when the connection fails. Use sqlany_error to retrieve the error code and message.

Example

```
require 'sqlanywhere'
api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
# Create a connection
conn = api.sqlany_new_connection()
puts "Enter user ID"
myuid = STDIN.gets.chomp
puts "Enter password"
mypwd = STDIN.gets.chomp
# Establish a connection
status = api.sqlany_connect( conn, "UID="+myuid+";PWD="+mypwd )
print "Connection status = #{status}\n"
```

Related Information

Alphabetical List of Connection Parameters
Database Connections

sqlany_new_connection Function [page 563]
sqlany_error Function [page 549]

1.17.2.7 sqlany_describe_bind_param Function

Describes the bind parameters of a prepared statement.

Syntax

```
sqlany_describe_bind_param ( $stmt, $index )
```

Parameters

$stmt

A statement prepared successfully using sqlany_prepare.

$index

The index of the parameter. The number must be between 0 and sqlany_num_params() - 1.

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and a described parameter as the second element.

Remarks

This function allows the caller to determine information about prepared statement parameters. The type of prepared statement (stored procedure or a DML), determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

Example

```
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 )
```

Related Information

sqlany_bind_param Function [page 543]
sqlany_prepare Function [page 566]

1.17.2.8 sqlany_disconnect Function

Disconnects a database connection. All uncommitted transactions are rolled back.

Syntax

```
sqlany_disconnect ( $conn )
```

Parameters

$conn

A connection object with a connection established using sqlany_connect.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}\n"
```

Related Information

sqlany_connect Function [page 546]

sqlany_new_connection Function [page 563]

1.17.2.9 sqlany_error Function

Returns the last error code and message stored in the connection object.

Syntax

```
sqlany_error ( $conn )
```

Parameters

$conn

A connection object returned from sqlany_new_connection.

Returns

Returns a 2-element array that contains the SQL error code as the first element and an error message string as the second element.

For the error code, positive values are warnings, negative values are errors, and 0 is success.

Example

```
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

Related Information

sqlany_connect Function [page 546]
SQL Anywhere Error Messages Sorted by SQLCODE

1.17.2.10 sqlany_execute Function

Executes a prepared statement.

Syntax

```
sqlany_execute ( $stmt )
```

Parameters

$stmt

A statement prepared successfully using sqlany_prepare.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Remarks

You can use sqlany_num_cols to verify if the statement returned a result set.

Example

```
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 )
```

Related Information

sqlany_prepare Function [page 566]

1.17.2.11 sqlany_execute_direct Function

Executes the SQL statement specified by the string argument.

Syntax

```
sqlany_execute_direct ( $conn, $sql )
```

Parameters

$conn

A connection object with a connection established using sqlany_connect.

$sql

A SQL string. The SQL string should not have parameters such as *?*.

Returns

Returns a statement object or nil on failure.

Remarks

Use this function to prepare and execute a statement in one step. Do not use this function to execute a SQL statement with parameters.

Example

```
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"
```

Related Information

sqlany_fetch_absolute Function [page 553]
sqlany_fetch_next Function [page 554]
sqlany_num_cols Function [page 564]
sqlany_get_column Function [page 559]

1.17.2.12 sqlany_execute_immediate Function

Executes the specified SQL statement immediately without returning a result set. It is useful for statements that do not return result sets.

Syntax

```
sqlany_execute_immediate ( $conn, $sql )
```

Parameters

$conn

A connection object with a connection established using sqlany_connect.

$sql

A SQL string. The SQL string should not have parameters such as ?.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= 50" )
```

Related Information

sqlany_error Function [page 549]

1.17.2.13 sqlany_fetch_absolute Function

Moves the current row in the result set to the row number specified and then fetches the data at that row.

Syntax

```
sqlany_fetch_absolute ( $stmt, $row_num )
```

Parameters

$stmt

A statement object that was executed by sqlany_execute or sqlany_execute_direct.

$row_num

The row number to be fetched. The first row is 1, the last row is -1.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
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"
```

Related Information

sqlany_error Function [page 549]
sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]
sqlany_fetch_next Function [page 554]

1.17.2.14 sqlany_fetch_next Function

Returns the next row from the result set. This function first advances the row pointer and then fetches the data at the new row.

Syntax

```
sqlany_fetch_next ( $stmt )
```

Parameters

$stmt

A statement object that was executed by sqlany_execute or sqlany_execute_direct.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
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"
```

Related Information

sqlany_error Function [page 549]
sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]
sqlany_fetch_absolute Function [page 553]

1.17.2.15 sqlany_fini Function

Frees resources allocated by the API.

Syntax

```
sqlany_fini ( )
```

Returns

Returns nil.

Example

```
# 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 )
```

Related Information

sqlany_init Function [page 562]

1.17.2.16 sqlany_free_connection Function

Frees the resources associated with a connection object.

Syntax

```
sqlany_free_connection ( $conn )
```

Parameters

$conn

A connection object created by sqlany_new_connection.

Returns

Returns nil.

Example

```
# 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 )
```

Related Information

sqlany_new_connection Function [page 563]

1.17.2.17 sqlany_free_stmt Function

Frees resources associated with a statement object.

Syntax

```
sqlany_free_stmt ( $stmt )
```

Parameters

$stmt

A statement object returned by the successful execution of sqlany_prepare or sqlany_execute_direct.

Returns

Returns nil.

Example

```
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 )
```

Related Information

sqlany_prepare Function [page 566]
sqlany_execute_direct Function [page 551]

1.17.2.18 sqlany_get_bind_param_info Function

Retrieves information about the parameters that were bound using sqlany_bind_param.

Syntax

```
sqlany_get_bind_param_info ( $stmt, $index )
```

Parameters

$stmt

A statement successfully prepared using sqlany_prepare.

$index

The index of the parameter. The number must be between 0 and sqlany_num_params() - 1.

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and a described parameter as the second element.

Example

```
# 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"
```

Related Information

sqlany_bind_param Function [page 543]
sqlany_describe_bind_param Function [page 547]
sqlany_prepare Function [page 566]

1.17.2.19 sqlany_get_column Function

Returns the value fetched for the specified column.

Syntax

```
sqlany_get_column ( $stmt, $col_index )
```

Parameters

$stmt

A statement object that was executed by sqlany_execute or sqlany_execute_direct.

$col_index

The number of the column to be retrieved. A column number is between 0 and sqlany_num_cols() - 1.

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and the column value as the second element.

Example

```
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"
```

Related Information

sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]
sqlany_fetch_absolute Function [page 553]
sqlany_fetch_next Function [page 554]

1.17.2.20 sqlany_get_column_info Function

Gets column information for the specified result set column.

Syntax

```
sqlany_get_column_info ( $stmt, $col_index )
```

Parameters

$stmt

A statement object that was executed by sqlany_execute or sqlany_execute_direct.

$col_index

The column number between 0 and sqlany_num_cols() - 1.

Returns

Returns a 9-element array of information describing a column in a result set. The first element contains 1 on success or 0 on failure. The array elements are described in the following table.

Element Number	Type	Description
0	Integer	1 on success or 0 on failure.
1	Integer	Column index (0 to sqlany_num_cols() - 1).
2	String	Column name.
3	Integer	Column type.
4	Integer	Column native type.
5	Integer	Column precision (for numeric types).
6	Integer	Column scale (for numeric types).
7	Integer	Column size.
8	Integer	Column nullable (1=nullable, 0=not nullable).

Example

```
# 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 )
```

Related Information

Column Types [page 569]
Native Column Types [page 570]
sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]
sqlany_prepare Function [page 566]

1.17.2.21 sqlany_get_next_result Function

Advances to the next result set in a multiple result set query.

Syntax

```
sqlany_get_next_result ( $stmt )
```

Parameters

$stmt

A statement object executed by sqlany_execute or sqlany_execute_direct.

Returns

Returns a scalar value that is 1 on success or 0 on failure.

Example

```
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 )
```

Related Information

sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]

1.17.2.22 sqlany_init Function

Initializes the interface.

Syntax

```
sqlany_init ( )
```

Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and the Ruby interface version as the second element.

Example

```
# 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()
```

Related Information

sqlany_fini Function [page 555]

1.17.2.23 sqlany_new_connection Function

Creates a connection object.

Syntax

```
sqlany_new_connection ()
```

Returns

Returns a scalar value that is a connection object.

Remarks

A connection object must be created before a database connection is established. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time.

Example

```
require 'sqlanywhere'
api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
# Create a connection
conn = api.sqlany_new_connection()
puts "Enter user ID"
myuid = STDIN.gets.chomp
puts "Enter password"
mypwd = STDIN.gets.chomp
# Establish a connection
status = api.sqlany_connect( conn, "UID="+myuid+";PWD="+mypwd )
print "Connection status = #{status}\n"
```

Related Information

sqlany_connect Function [page 546]
sqlany_disconnect Function [page 548]

1.17.2.24 sqlany_num_cols Function

Returns number of columns in the result set.

Syntax

```
sqlany_num_cols ( $stmt )
```

Parameters

$stmt

A statement object executed by sqlany_execute or sqlany_execute_direct.

Returns

Returns a scalar value that is the number of columns in the result set, or -1 on a failure.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of result set columns
num_cols = api.sqlany_num_cols( stmt )
```

Related Information

sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]
sqlany_prepare Function [page 566]

1.17.2.25 sqlany_num_params Function

Returns the number of parameters that are expected for a prepared statement.

Syntax

```
sqlany_num_params ( $stmt )
```

Parameters

$stmt

A statement object returned by the successful execution of sqlany_prepare.

Returns

Returns a scalar value that is the number of parameters in a prepared statement, or -1 on a failure.

Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
      SET Contacts.ID = Contacts.ID + 1000
      WHERE Contacts.ID >= ?" )
num_params = api.sqlany_num_params( stmt )
```

Related Information

sqlany_prepare Function [page 566]

1.17.2.26 sqlany_num_rows Function

Returns the number of rows in the result set.

Syntax

```
sqlany_num_rows ( $stmt )
```

Parameters

$stmt

A statement object executed by sqlany_execute or sqlany_execute_direct.

Returns

Returns a scalar value that is the number of rows in the result set. If the number of rows is an estimate, the number returned is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

Remarks

By default, this function only returns an estimate. To return an exact count, set the ROW_COUNTS option on the connection.

A count of the number of rows in a result set can be returned only for the first result set in a statement that returns multiple result sets. If sqlany_get_next_result is used to move to the next result set, sqlany_num_rows will still return the number of rows in the first result set.

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of rows in result set
num_rows = api.sqlany_num_rows( stmt )
```

Related Information

sqlany_execute Function [page 550]
sqlany_execute_direct Function [page 551]
row_counts Option

1.17.2.27 sqlany_prepare Function

Prepares the supplied SQL string.

Syntax

```
sqlany_prepare ( $conn, $sql )
```

Parameters

$conn

A connection object with a connection established using sqlany_connect.

$sql

The SQL statement to be prepared.

Returns

Returns a scalar value that is the statement object, or nil on failure.

Remarks

The statement associated with the statement object is executed by sqlany_execute. You can use sqlany_free_stmt to free the resources associated with the statement object.

Example

```
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 )
```

Related Information

sqlany_execute Function [page 550]
sqlany_free_stmt Function [page 557]
sqlany_num_params Function [page 564]
sqlany_describe_bind_param Function [page 547]
sqlany_bind_param Function [page 543]

1.17.2.28 sqlany_rollback Function

Rolls back the current transaction.

Syntax

```
sqlany_rollback ( $conn )
```

Parameters

$conn

The connection object on which the rollback operation is to be performed.

Returns

Returns a scalar value that is 1 on success, 0 on failure.

Example

```
rc = api.sqlany_rollback( conn )
```

Related Information

sqlany_commit Function [page 545]

1.17.2.29 sqlany_sqlstate Function

Retrieves the current SQLSTATE.

Syntax

```
sqlany_sqlstate ( $conn )
```

Parameters

$conn

A connection object returned from sqlany_new_connection.

Returns

Returns a scalar value that is the current five-character SQLSTATE.

Example

```
sql_state = api.sqlany_sqlstate( conn )
```

Related Information

sqlany_error Function [page 549]
SQL Anywhere Error Messages Sorted by SQLSTATE

1.17.2.30 Column Types

The following Ruby class defines the column types returned by some Ruby extension functions.

```
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
```

1.17.2.31 Native Column Types

The following table defines the native column types returned by some Ruby extension functions.

Native Type Value	Native Type
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
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

1.18 SAP Open Client Support

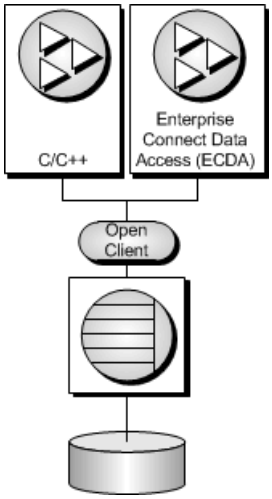
SAP Open Client provides customer applications, third-party products, and other SAP products with the interfaces needed to communicate with Open Servers.

You can develop applications in C or C++, and then connect those applications to an Open Server using the Open Client API.

The SQL Anywhere database server can act as an Open Server to Open Client applications. This feature enables SAP Open Client applications to connect natively to SQL Anywhere databases.

When to Use Open Client

Consider using the Open Client interface if you are concerned with SAP Adaptive Server Enterprise compatibility or if you are using other SAP products that support the Open Client interface.



Enterprise Connect Data Access Support

Because the database server can act as an Open Server, SAP Enterprise Connect Data Access (ECDA) is supported.

Enterprise Connect Data Access provides a unified view of disparate data within an organization, allowing users to access multiple data sources without having to know what the data looks like or where to find it. In addition, ECDA performs heterogeneous joins of data across the enterprise, enabling cross-platform table joins of targets such as SAP Adaptive Server Enterprise, SAP IQ, SAP SQL Anywhere, IBM DB2, IBM VSAM, and Oracle.

SAP Open Client and Open Server

If you simply want to use an existing Open Client application with the database server, you do not need to know any details of SAP Open Client and Open Server. However, an understanding of how these components fit together may be helpful for configuring your database and setting up applications.

This database server and other members of the SAP Database Management Server family such as SAP Adaptive Server Enterprise act as **Open Servers**. This means you can develop client applications using the **SAP Open Client** libraries available from SAP. Open Client includes both the Client Library (CT-Library) and the older DB-Library interfaces.

Tabular Data Stream (TDS)

SAP Open Client and Open Server exchange information using an application protocol called the **Tabular Data Stream** (TDS). All applications built using the SAP Open Client libraries are also TDS applications because the Open Client libraries handle the TDS interface. However, some applications (such as jConnect) are TDS applications even though they do not use the Open Client libraries. They communicate directly using the TDS protocol.

While many Open Servers use the SAP Open Server libraries to handle the interface to TDS, some servers have a direct interface to TDS of their own. SAP Adaptive Server Enterprise, SAP IQ, and SAP SQL Anywhere are examples of Open Servers that have native built-in support for TDS.

Programming Interfaces and Application Protocols

Two client application protocols are supported by the database servers.

SAP Open Client applications and other SAP applications such as Enterprise Connect Data Access (ECDA) use TDS.

The .NET, ODBC, JDBC, Embedded SQL, OLE DB, and other application interfaces use a separate application protocol called **Command Sequence** (CmdSeq).

TDS Uses TCP/IP

Application protocols such as TDS and CmdSeq sit on top of lower-level communications protocols that handle network traffic. TDS and CmdSeq use the TCP/IP network protocol to communicate between computers. In addition, SQL Anywhere also supports a shared memory protocol designed for same-computer communication.

In this section:

Open Client Architecture [page 573]
The following information describes some SAP Open Client features that are specific to .

What You Need to Build Open Client Applications [page 574]
To run Open Client applications, you must install and configure SAP Open Client components on the computer where the application is running.

Open Client Data Type Mappings [page 574]
SAP Open Client has its own internal data types, which differ in some details from those available in SQL Anywhere.

SQL in Open Client Applications [page 575]
The use of SQL in Open Client applications is briefly introduced here. Focus is on SQL Anywhere-specific issues.

Known Open Client Limitations of SQL Anywhere [page 578]
With the Open Client interface, you can use SQL Anywhere databases in much the same way as you would Adaptive Server Enterprise databases.

System Requirements for Using SQL Anywhere as an Open Server [page 579]
There are separate requirements at the client and server for using SQL Anywhere as an Open Server.

Database Server as an Open Server Startup [page 580]
The TCP/IP communication protocol is required when the database server is used as an Open Server. This protocol is started by default for the network server.

1.18.1 Open Client Architecture

The following information describes some SAP Open Client features that are specific to .

SAP Open Client has two components: programming interfaces and network services.

DB-Library and Client Library

SAP Open Client provides two core programming interfaces for writing client applications: DB-Library and Client-Library.

Open Client DB-Library provides support for older Open Client applications, and is a completely separate programming interface from Client-Library. DB-Library is documented in the *DB-Library/C Reference Manual*.

Client-Library programs also depend on CS-Library, which provides routines that are used in both Client-Library and Server-Library applications. Client-Library applications can also use routines from Bulk-Library to help high-speed data transfer. Client-Library is documented in the *Client-Library/C Reference Manual*.

Both CS-Library and Bulk-Library are included in Open Client, which is available separately.

See the *SDK for SAP Adaptive Server Enterprise* on the SAP Help Portal for the latest documentation.

Network Services

Open Client network services include Net-Library, which provides support for specific network protocols such as TCP/IP and DECnet. The Net-Library interface is invisible to application developers. However, on some

platforms, an application may need a different Net-Library driver for different system network configurations. Depending on your host platform, the Net-Library driver is specified either by the system's Adaptive Server Enterprise configuration or when you compile and link your programs.

Instructions for driver configuration can be found in the *SAP Open Server and Open Client Configuration Guide for Windows* or the *SAP Open Server and Open Client Configuration Guide for UNIX*.

Instructions for building Client-Library programs can be found in the *Client-Library/C Programmers Guide*.

See the *SDK for SAP Adaptive Server Enterprise* on the SAP Help Portal for the latest documentation.

1.18.2 What You Need to Build Open Client Applications

To run Open Client applications, you must install and configure SAP Open Client components on the computer where the application is running.

You may have these components present as part of your installation of other SAP products or you can optionally install these libraries with SQL Anywhere, subject to the terms of your license agreement.

Open Client applications do not need any Open Client components on the computer where the database server is running.

To build Open Client applications, you need the development version of Open Client, available from SAP.

By default, SQL Anywhere databases are created as case-insensitive, while Adaptive Server Enterprise databases are case sensitive.

1.18.3 Open Client Data Type Mappings

SAP Open Client has its own internal data types, which differ in some details from those available in SQL Anywhere.

For this reason, SQL Anywhere internally maps some data types between those used by Open Client applications and those available in SQL Anywhere.

To build Open Client applications, you need the development version of Open Client. To use Open Client applications, the Open Client run-times must be installed and configured on the computer where the application runs.

The SQL Anywhere server does not require any external communications runtime to support Open Client applications.

Each Open Client data type is mapped onto the equivalent SQL Anywhere data type. All Open Client data types are supported.

In this section:

Range Limitations in Open Client Data Type Mapping [page 575]
Some data types have different value ranges in SQL Anywhere than in Open Client. In such cases, overflow errors can occur during retrieval or insertion of data.

1.18.3.1 Range Limitations in Open Client Data Type Mapping

Some data types have different value ranges in SQL Anywhere than in Open Client. In such cases, overflow errors can occur during retrieval or insertion of data.

The following table lists Open Client application data types that can be mapped to SQL Anywhere data types, but with some restriction in the range of possible values.

The Open Client data type is usually mapped to a SQL Anywhere data type with a greater range of possible values. As a result, it is possible to store values in a database that are too large to be fetched by an Open Client application.

Data Type	Open Client Lower Range	Open Client Upper Range	SQL Anywhere Lower Range	SQL Anywhere Upper Range
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]	January 1, 1753	December 31, 9999	January 1, 0001	December 31, 9999
SMALLDATETIME	January 1, 1900	June 6, 2079	January 1, 0001	December 31, 9999

[1] The full range of dates from 0001-01-01 to 9999-12-31 is supported.

Example

For example, the Open Client MONEY and SMALLMONEY data types do not span the entire numeric range of their underlying SQL Anywhere implementations. Therefore, it is possible to have a value in a column which exceeds the boundaries of the Open Client data type MONEY. When the client fetches any such offending values from the database, an error is generated.

Timestamps

No restriction will apply to TIMESTAMP values.

1.18.4 SQL in Open Client Applications

The use of SQL in Open Client applications is briefly introduced here. Focus is on SQL Anywhere-specific issues.

In this section:

Open Client SQL Statement Execution [page 576]
You send SQL statements to a database server by including them in Client Library function calls.

Open Client Prepared Statements [page 576]

The ct_dynamic function is used to manage prepared statements.

Open Client Cursor Management [page 577]
The ct_cursor function is used to manage cursors.

Open Client Result Sets [page 578]
Open Client handles result sets in a different way than some other SQL Anywhere interfaces.

Related Information

Application Development Using SQL [page 11]
Open Server

1.18.4.1 Open Client SQL Statement Execution

You send SQL statements to a database server by including them in Client Library function calls.

For example, the following pair of calls executes a DELETE statement:

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

1.18.4.2 Open Client Prepared Statements

The ct_dynamic function is used to manage prepared statements.

This function takes a `type` parameter that describes the action you are taking.

Perform the following tasks to use a prepared statement in Open Client:

1. Prepare the statement using the ct_dynamic function, with a CS_PREPARE `type` parameter.
2. Set statement parameters using ct_param.
3. Execute the statement using ct_dynamic with a CS_EXECUTE `type` parameter.
4. Free the resources associated with the statement using ct_dynamic with a CS_DEALLOC `type` parameter.

For more information about using prepared statements in Open Client, see your Open Client documentation.

1.18.4.3 Open Client Cursor Management

The ct_cursor function is used to manage cursors.

This function takes a `type` parameter that describes the action you are taking.

Supported Cursor Types

Not all the types of cursor that SQL Anywhere supports are available through the Open Client interface. You cannot use scroll cursors, dynamic scroll cursors, or insensitive cursors through Open Client.

Uniqueness and updatability are two properties of cursors. Cursors can be unique (each row carries primary key or uniqueness information, regardless of whether it is used by the application) or not. Cursors can be read-only or updatable. If a cursor is updatable and not unique, performance may suffer, as no prefetching of rows is done in this case, regardless of the CS_CURSOR_ROWS setting.

The Steps in Using Cursors

In contrast to some other interfaces, such as Embedded SQL, Open Client associates a cursor with a SQL statement expressed as a string. Embedded SQL first prepares a statement and then the cursor is declared using the statement handle.

Perform the following tasks to use cursors in Open Client:

1. To declare a cursor in Open Client, use ct_cursor with CS_CURSOR_DECLARE as the `type` parameter.
2. After declaring a cursor, you can control how many rows are prefetched to the client side each time a row is fetched from the server by using ct_cursor with CS_CURSOR_ROWS as the `type` parameter.
   Storing prefetched rows at the client side reduces the number of calls to the server and this improves overall throughput and turnaround time. Prefetched rows are not immediately passed on to the application; they are stored in a buffer at the client side ready for use.
   The setting of the prefetch database option controls prefetching of rows for other interfaces. It is ignored by Open Client connections. The CS_CURSOR_ROWS setting is ignored for non-unique, updatable cursors.
3. To open a cursor in Open Client, use ct_cursor with CS_CURSOR_OPEN as the `type` parameter.
4. To fetch each row in to the application, use ct_fetch.
5. To close a cursor, you use ct_cursor with CS_CURSOR_CLOSE.
6. In Open Client, you must also deallocate the resources associated with a cursor. You do this by using ct_cursor with CS_CURSOR_DEALLOC. You can also use CS_CURSOR_CLOSE with the additional parameter CS_DEALLOC to perform these operations in a single step.

In this section:

Open Client Row Modification Through a Cursor [page 578]
With Open Client, you can delete or update rows in a cursor, as long as the cursor is for a single table. The user must have permissions to update the table and the cursor must be marked for update.

1.18.4.3.1 Open Client Row Modification Through a Cursor

With Open Client, you can delete or update rows in a cursor, as long as the cursor is for a single table. The user must have permissions to update the table and the cursor must be marked for update.

Instead of carrying out a fetch, you can delete or update the current row of the cursor using ct_cursor with CS_CURSOR_DELETE or CS_CURSOR_UPDATE, respectively.

You cannot insert rows through a cursor in Open Client applications.

1.18.4.4 Open Client Result Sets

Open Client handles result sets in a different way than some other SQL Anywhere interfaces.

In Embedded SQL and ODBC, you **describe** a query or stored procedure to set up the proper number and types of variables to receive the results. The description is done on the statement itself.

In Open Client, you do not need to describe a statement. Instead, each row returned from the server can carry a description of its contents. If you use ct_command and ct_send to execute statements, you can use the ct_results function to handle all aspects of rows returned in queries.

If you do not want to use this row-by-row method of handling result sets, you can use ct_dynamic to prepare a SQL statement and use ct_describe to describe its result set. This corresponds more closely to the describing of SQL statements in other interfaces.

1.18.5 Known Open Client Limitations of SQL Anywhere

With the Open Client interface, you can use SQL Anywhere databases in much the same way as you would Adaptive Server Enterprise databases.

There are some limitations, including the following:

- SQL Anywhere does not support the Adaptive Server Enterprise Commit Service.
- A client/server connection's **capabilities** determine the types of client requests and server responses permitted for that connection. The following capabilities are not supported:
  - 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

- Command encryption is not supported. However, password encryption and connections encrypted with TLS are supported.
- Open Client applications can connect to the database server using TCP/IP.
  For more information about capabilities, see the *Open Server Server-Library C Reference Manual*.
- When the CS_DATAFMT is used with the CS_DESCRIBE_INPUT, it does not return the data type of a column when a parameterized variable is sent to the database server as input.

1.18.6 System Requirements for Using SQL Anywhere as an Open Server

There are separate requirements at the client and server for using SQL Anywhere as an Open Server.

Server-side Requirements

You must have the following elements at the server side to use SQL Anywhere as an Open Server:

SQL Anywhere server components

You must use the network server (*dbsrv17*`.exe`) to access an Open Server over a network. You can use the personal server (*dbeng17*`.exe`) as an Open Server only for connections from the same computer.

TCP/IP

You must have a TCP/IP protocol stack to use SQL Anywhere as an Open Server, even if you are not connecting over a network.

Client-Side Requirements

You need the following elements to use client applications to connect to an Open Server (including SQL Anywhere):

SAP Open Client components

The SAP Open Client libraries provide the network libraries your application needs to communicate via TDS if your application uses SAP Open Client.

jConnect

If your application uses JDBC, you need jConnect and a Java Runtime Environment.

DSEdit

You need DSEdit, the directory services editor, to make server names available to your Open Client application. On UNIX and Linux platforms, this utility is called sybinit.

The database server automatically sets relevant database options to values that are compatible with Open Client and jConnect applications. These options are set temporarily, for the duration of the connection only. The client application can override these options at any time.

Related Information

Open Client/jConnect TDS Compatibility Options
Downloading jConnect

1.18.7 Database Server as an Open Server Startup

The TCP/IP communication protocol is required when the database server is used as an Open Server. This protocol is started by default for the network server.

You can use the personal database server as an Open Server for communications on the same computer because it supports the TCP/IP protocol. You must request the protocol when starting the personal server. For example, the following command starts the TCP/IP communication protocol:

```
dbeng17 -x tcpip -n myserver c:\mydata.db
```

The database server can serve other applications through the TCP/IP protocol at the same time as serving Open Client applications over TDS.

SAP Open Client Settings

To connect to the database server, the interfaces file at the client computer must contain an entry specifying the computer name on which the database server is running, and the TCP/IP port it uses.

1.19 OData Support

OData (Open Data Protocol) enables data services over RESTful HTTP. It allows you to perform operations through URIs (Uniform Resource Identifiers) to access and modify information.

You should be familiar with OData protocol concepts before you configure SQL Anywhere as an OData server.

In this section:

OData Server Architecture [page 581]
An OData server consists of OData Producers and an HTTP server.

OData Protocol Limitations [page 583]
OData Producers comply with OData protocol version 2 specifications with some limitations and exceptions.

OData Server Security Considerations [page 584]
Security measures must be considered before setting up an OData server.

How to Set Up an OData Server [page 586]

Deployment and set up of an OData server.

OData Server Samples [page 588]
OData server samples are available in the *%SQLANYSAMP17%*`\SQLAnywhere\` directory. The samples illustrate how to start the database server, set up an OData client, and send requests and responses between them.

How to Set Up Repeatable Requests [page 589]
Repeatable requests allow OData clients to resend data modification requests, such as UPDATE or DELETE, without risking database integrity or causing unintended side effects in cases where the clients are unsure of whether a request made it to the OData Producer.

How to Protect Against Cross-site Request Forgery Attacks [page 590]
CSRF tokens protect OData Producers from cross-site request forgery attacks.

How to Create an OData Producer Service Model [page 591]
An optional OData Service Definition Language (OSDL) model file can specify an OData Producer service model to expose specific tables, views, stored procedures, and functions.

How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Producer options for a third-party HTTP server are specified in a configuration file.

OSDL Statement Reference [page 596]
OSDL statements create an OData Producer service model to expose specific tables, views, stored procedures, and functions.

Related Information

OData Version 2.0

1.19.1 OData Server Architecture

An OData server consists of OData Producers and an HTTP server.

OData Producers

In SQL Anywhere, an OData Producer is a Java Servlet that uses the JDBC API to connect to a database server. An OData Producer processes OData requests and interfaces with the database.

The following table illustrates how an OData Producer maps OData concepts to relational database concepts:

OData Concept	Database Equivalent
Entity	Row
Entity type	Table or view schema
Entity Set	Table rows, where each row is an entity type instance

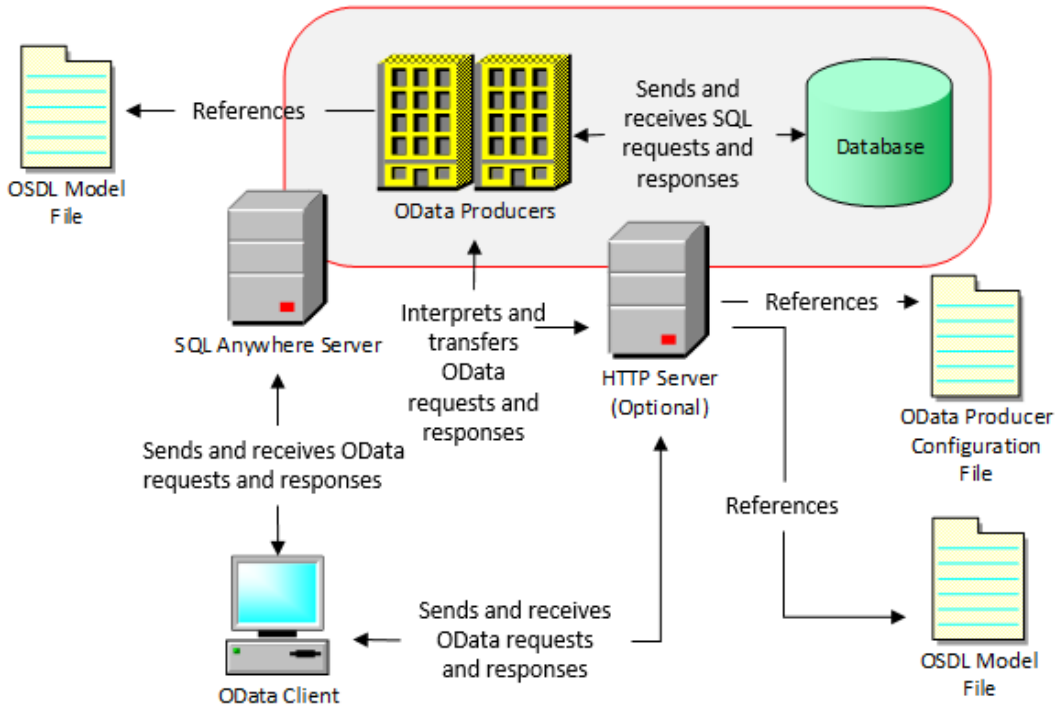
OData Concept	Database Equivalent
Key	Primary key
Association and Navigation Properties	Foreign key
Property	Column or column value

An HTTP Server

An HTTP server handles OData requests from web clients.

The database server uses the Jetty WebServer as its HTTP server for OData. This HTTP server acts as a Java Servlet container, which is required to host OData Producers.

Alternatively, you can use a third-party HTTP server to handle OData requests, provided that your solution can host Java Servlets. For example, you can set up an IIS or Apache server to forward requests to a Tomcat or Jetty server.



OData client requests are sent to an HTTP server through URIs and are processed by an OData Producer, which then interfaces with the database server to issue database requests and retrieve content for the OData responses.

The OData schema for each client is based on the client's database connection permissions. Clients are not able to view or modify database objects that they do not have permission to view.

Client access to the database can be granted by using either a pre-configured connection string or Basic HTTP authentication.

For more information about the OData protocol, see http://www.odata.org/documentation/odata-version-2-0 .

Related Information

OData Server Security Considerations [page 584]
OData Protocol Limitations [page 583]
How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Server Samples [page 588]
OData Version 2.0

1.19.2 OData Protocol Limitations

OData Producers comply with OData protocol version 2 specifications with some limitations and exceptions.

These limitations are not explicitly defined by OData protocol specifications.

For more information about the OData protocol, see http://www.odata.org/documentation/odata-version-2-0 .

Schema changes

Restart the OData Producers and modify the model when you make changes to the database schema so that the changes can take effect and become visible to OData clients.

Update OData Producers to use different service roots in production environments because OData clients cache metadata.

Search string filters

When using OData filters (such as substringof and indexof) on long search strings, searches are performed on the first 254 bytes only.

substringof changes

The substringof(s1, s2) filter returns whether the s1 string is a substring of s2.

$orderby queries

Sort by entity properties only. Ordering by direction is supported, but sorting by expressions is not.

Proxy tables

When creating entities in entity sets that are proxy tables in a database, ensure that all key properties are explicitly specified.

DefaultValue attributes

The DefaultValue attribute is not exposed because it is not appropriate for all possible default values that an underlying column in the database can have.

A missing DefaultValue attribute does not imply that the default value for the property is null.

Stored procedures

Stored procedures that return either multiple result sets or variable results (different data types based on invocation conditions) are not supported as service operations.

Service operations must be explicitly exposed using an OData Producer Service model file.

INOUT parameters are not supported.

Unsupported OData Protocol Features

The following OData protocol features are not supported by OData Producers:

- Dynamic properties
- Complex types (except as return values from service operations)
- Media types
- If-Match (except If-Match: *) or If-None-Match HTTP headers with Retrieve Entity Set requests

Related Information

OData Server Security Considerations [page 584]
How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Server Samples [page 588]
OData Version 2.0

1.19.3 OData Server Security Considerations

Security measures must be considered before setting up an OData server.

CSRF Tokens and Service Operations

When using CSRF tokens in OData to defend against cross-site request forgery, OData Producers assume that service operations invoked using GET requests do not modify the database. All service operations that modify the database should follow OData conventions and be restricted to POST requests only.

Repeatable Requests

The odata_sys_repeatable_request system table contains a copy of the request URL and body response for each repeatable request. This data should not be exposed to unauthorized users.

The OData Administrator user assigned in the database must be secure and used exclusively for the OData Producers. The user must have SELECT access to the odata_sys_repeatable_request table, and be able to create tables, indexes, and events.

Database Authentication

Use database authentication for production systems. Not using authentication is only recommended for testing or with read-only Producers that limit exposed database tables by using an OData Service model file.

Traffic Without Use of the TLS Protocol

Always use TLS (SSL) in production deployments.

When setting up the database server to handle OData requests, use the SSLKeyStore (KEYSTORE) protocol option and configure OData Producers to be SecureOnly. When deploying OData Producers to a third-party HTTP server, configure the server to only send secure traffic to the OData servlets.

Traffic between OData Producers and clients is transmitted in plain text, including user IDs and passwords, if the SSLKeyStore option is not specified when starting the database server. This option is not specified in default configurations.

HTTPS Certification

Use a secure server certificate with a strong validation chain to a known trusted signing authority and fully qualified host names.

The following Java keytool command creates a Java Key Store with a self-signed certificate (recommended for testing only):

```
keytool -genkeypair -keyalg RSA -keysize 2048
    -keypass sample -validity 1825 -keystore mystore.jks
    -storepass sample -v -alias localhost -sigalg SHA256withRSA
```

When using the database server for OData, use the **`-xs odata`** server option to pass the certificate information. When using a third-party HTTP server for OData, see the server documentation for more information about passing certificate information.

SQL Anywhere Version 12 and Earlier Databases

When using OData Producers in an HTTP server to query a SQL Anywhere version 12 or earlier database, all database objects are exposed. They are not filtered based on user credentials.

The database server returns an error when you attempt to access an object without the appropriate database privilege.

Related Information

How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Protocol Limitations [page 583]
-xs Database Server Option
OData Version 2.0
Jetty

1.19.4 How to Set Up an OData Server

Deployment and set up of an OData server.

Deploying an OData Server

The following steps are required to deploy an OData server:

1. Deploy the database server. Database server deployment is described in a separate topic.
2. Deploy support for Java external environment calls. External environment support deployment is described in a separate topic.
3. Deploy the SQL Anywhere JDBC driver. This includes the JDBC driver JAR file and supporting library files. JDBC driver deployment is described in a separate topic.
4. Deploy the following JAR files from the *%SQLANY17%*`\Java` folder.
   1. `dbodata.jar`
   2. `jetty-all-server-9.4.7.jar`
   3. `servlet-api-3.1.0.jar`
   4. `sajdbc4.jar` (this file is included as part of the previous step where you deploy the JDBC client)
5. Deploy a Java Runtime Environment (JRE).

Database Server Set Up

The database server automatically loads previously defined and enabled OData Producers when started with the **`-xs odata`** server option.

The database server cannot be manually configured or used to serve other non-OData content, such as HTML files, on the same port as the OData server. However, some HTTP server options can be specified with the **`-xs odata`** server option.

The following steps are required to set up and launch the database server for OData operations:

1. Start the database server with the **`-xs odata`** server option with the appropriate server settings. For example, the following statement starts the database server to listen to OData requests on port 8080:
   dbsrv17 **`-xs odata(ServerPort=8080) mydatabase.db`**

2. If you see a message indicating that the OData server cannot be started, then ensure that a Java VM can be located by setting the JAVA_HOME or JAVAHOME environment variable. The following is an example for Microsoft Windows:

   ```
   set JAVA_HOME=%SQLANY17%\Bin64\jre180
   ```

3. Use the CREATE ODATA PRODUCER statement to set up new OData Producers.
4. (Optional) Set up repeatable requests.
5. (Optional) Protect against cross-site request forgery (CSRF) attacks.

> **Note**
>
> The -xs odata database server option is supported on Windows and Linux (but not ARM platforms).

Third-Party HTTP Server Set Up

To use OData Producers with a third-party HTTP server, such as Apache or IIS, deploy the OData Producers to the server. Run OData Producers as Java Servlets that can be loaded into the HTTP server that support version 3.1 of the Java Servlet API. For example, Tomcat can be used as a Java Servlet container and paired with an Apache HTTP server.

> **Note**
>
> IIS cannot execute Java Servlets, but you can configure a connector that redirects Servlet requests from an IIS server to another server that is able to run them.

The process of deploying OData Producers differs depending on your HTTP server. For more information, see your HTTP server documentation.

The following steps are required to deploy OData Producers:

1. Create an OData Producer configuration file.
   OData Producers respect the logging configuration of the HTTP server. For more information about HTTP server logging, see your HTTP server documentation.
2. (Optional) Set up repeatable requests.
3. (Optional) Protect against cross-site request forgery (CSRF) attacks.
4. Deploy the OData Producer configuration file, along with *%SQLANY17%*`\Java\dbodata.jar`.
5. Deploy the JDBC client. This includes the JDBC driver and supporting files.
6. Configure your HTTP server to include the com.sap.odata.producer.servlets.ODataServlet class from `dbodata.jar` such that it performs the following:
   1. Receives requests to the subpath as specified as the ServiceRoot in your configuration file.
   2. Finds the OData Producer configuration file within the context of the servlet.
   3. Gets initialized with the *odataConfigFile* servlet init parameter value set to the name of the OData Producer configuration file.
   4. (Optional) Sets the *odataProducerName* servlet init parameter value to the name of an OData Producer. This step is required if multiple OData producers share the same configuration file.
   5. (Optional) The OData Producer receives requests only from HTTPS traffic or the specified interface/port (if possible). Alternatively, the SecureOnly producer option may be required.

> **Note**
>
> OData Producers as Java Servlets are officially supported on Windows and Linux (but not ARM platforms).

Related Information

JDBC Client Deployment [page 847]
How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Server Security Considerations [page 584]
How to Set Up Repeatable Requests [page 589]
How to Protect Against Cross-site Request Forgery Attacks [page 590]
OData Server Samples [page 588]
-xs Database Server Option
Network Protocol Options
CREATE ODATA PRODUCER Statement

1.19.5 OData Server Samples

OData server samples are available in the *%SQLANYSAMP17%*`\SQLAnywhere\` directory. The samples illustrate how to start the database server, set up an OData client, and send requests and responses between them.

Create backup copies of the original samples before modifying file contents.

All samples require Java SE version 7 or later.

Connecting to the OData Server with a .NET Client

The OData SalesOrder sample illustrates how to use a Microsoft .NET OData Client to send OData requests to the database server.

For more information, see *%SQLANYSAMP17%*`\SQLAnywhere\ODataSalesOrders\readme.txt`.

Connecting to the OData Server with a Java client

The OData Security sample illustrates how to use an OData4J Java client to send OData requests over HTTPS to a database server that connects to the sample database.

This sample illustrates how to use an OData Producer service model and database authentication.

For more information, see *%SQLANYSAMP17%*`\SQLAnywhere\ODataSecurity\readme.txt`.

Related Information

How to Configure OData Producers for a Third-party HTTP server [page 592]

1.19.6 How to Set Up Repeatable Requests

Repeatable requests allow OData clients to resend data modification requests, such as UPDATE or DELETE, without risking database integrity or causing unintended side effects in cases where the clients are unsure of whether a request made it to the OData Producer.

For a request to be repeatable, a client must include a valid RepeatabilityCreation HTTP header with a valid HTTP timestamp and a RequestID HTTP header whose value is globally unique and case-insensitive. RequestID values must not be null, must consist of ASCII non-control characters only, and have a maximum length of 256.

The following HTTP response header is included in the response when an OData Producer accepts a request as repeatable:

```
RepeatabilityResult: accepted
```

When an OData Producer receives a request with the same creation time, request ID, URL, and requesting user, the Producer does not execute the request; it responds with the same response as was generated by the original request, including any errors.

To enable a repeatable requests, perform the following tasks:

- Create an OData administrator user in the database.
  This user must have the AUTHENTICATE ODATA system privilege if the OData Producer is embedded in the database server.
- Reference the new user with the ADMIN USER clause in the CREATE ODATA PRODUCER statement or the ODataAdmin option in the OData Producer configuration file.
  The user must be secure and used exclusively for the OData Producer. The user must have SELECT access to the odata_sys_repeatable_request table, and be able to create tables, indexes, and events.
- (Optional) Specify the RepeatRequestForDays option with the USING clause in the CREATE ODATA PRODUCER statement in the OData Producer configuration file.
  This option indicates the number of days that repeatable requests are valid.

> **Note**
>
> Repeatable Requests are not supported with GET requests. Repeatable service operations must be POST requests. OData Producers assume that service operations invoked using GET requests do not modify the database. All service operations that modify the database should follow OData conventions and be restricted to POST requests only.

Developers may control how long repeatable requests are valid by specifying the number of days with the RepeatRequestForDays database option in the Producer configuration file. The default is 2, minimum is 1, maximum is 31, value must be an integer.

Repeatable requests may be used with POST service operations and may be used within changesets of batch requests. A batch request may not be marked as repeatable in its entirety.

Support Tables and InfraStructure

When an OData producer first starts up with repeatable requests enabled, the producer uses the OData administrator user to create tables, indexes, procedures and events required to support repeatable requests (if they do not already exist). These actions take place when the first request is handled.

The following database objects are created:

odata_sys_repeatable_request TABLE

This table is owned by the OData administrator user. The administrator user grants insert on this table to PUBLIC.

odata_sys_repeatable_request INDEX

An index is created on the odata_sys_repeatable_request table.

odata_sys_option TABLE

This table owned by the OData administrator user.

odata_sys_repeatable_request_cleanup() PROCEDURE

This procedure is owned by the OData administrator user and cleans up old repeatable requests in the odata_sys_repeatable_request table based on the RepeatRequestForDays option.

odata_sys_repeatable_request_cleanup EVENT

This daily event is owned by the OData administrator user and invokes the odata_sys_repeatable_request_cleanup procedure.

Advanced developers may choose to alter the odata_sys_repeatable_request_cleanup event schedule and the odata_sys_repeatable_request_cleanup procedure to clean up in hours instead of days or to add size restriction on the odata_sys_repeatable_request table.

If developers choose to change the OData administrator user, the above objects (owned by the old Administrator user) should be removed from the database.

Related Information

How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Server Samples [page 588]

1.19.7 How to Protect Against Cross-site Request Forgery Attacks

CSRF tokens protect OData Producers from cross-site request forgery attacks.

To enable CSRF tokens, specify the CSRFTokenTimeout option in the USING clause of the CREATE ODATA PRODUCER statement or in the OData Producer configuration file.

> **Note**
>
> CSRF Tokens are only available when database authentication is in use and the service is not read-only.

When CSRF tokens are enabled, clients must retrieve a CSRF token from the OData Producer prior to performing modification requests. To retrieve the token, clients must perform a GET request with the X-CSRF-Token HTTP request header set to Fetch (the word Fetch is case sensitive, thus the request header is X-CSRF-Token: Fetch).

The OData Producer responds to a token retrieval request by sending a new token in the X-CSRF-Token header response, and in a cookie of the form sap-XSRF_`SID_client`. This token must be sent with all modification requests until it expires.

> **Note**
>
> An X-CSRF-Token request header must only have either a single valid token or the fetch directive. Multiple X-CSRF-Token request headers are not supported.

To make a modification request, the client must send both the cookie and the token in a X-CSRF-Token HTTP request header. Individual modification requests within a BATCH request should not specify cookies or the X-CSRF-Token HTTP header; these cookies should be specified for the entire BATCH request. You cannot request a CSRF token in a batch operation and use it in the same BATCH request.

Related Information

How to Configure OData Producers for a Third-party HTTP server [page 592]
CREATE ODATA PRODUCER Statement
OData Server Samples [page 588]

1.19.8 How to Create an OData Producer Service Model

An optional OData Service Definition Language (OSDL) model file can specify an OData Producer service model to expose specific tables, views, stored procedures, and functions.

The model is applied to an OData Producer when you reference the ODSL model file by using the MODEL clause in the CREATE ODATA PRODUCER statement or the *Model* option in the OData Producer configuration file.

Entity, service operation, and association exposure is based solely on user privileges. The following table illustrates how the OData Producer chooses which entities and associations to expose based on the contents of the model:

Model Contents	Entities	Associations
No model	From the database	From the database
Service operations	From the database	From the database
Entities with or without service operations	Constrained by the model	From the database

Model Contents	Entities	Associations
Associations,entities, and service operations	Constrained by the model	From the model

Many-to-many relationships can only be defined using a model file.

Only service operations explicitly defined in the model and accessible to the user are exposed. If no model is provided, no service operations are exposed.

Related Information

How to Configure OData Producers for a Third-party HTTP server [page 592]
OData Server Architecture [page 581]
OSDL Statement Reference [page 596]
service OSDL Statement [page 597]
OData Server Samples [page 588]

1.19.9 How to Configure OData Producers for a Third-party HTTP server

OData Producer options for a third-party HTTP server are specified in a configuration file.

A configuration file is required when using OData Producers with a third-party HTTP server.

> **Note**
>
> The configuration file must use ISO 8859-1 character encoding, where each byte is one Latin1 character. Special characters and characters that are not in Latin1 are represented in keys and elements by using Unicode escape characters. Only a single 'u' character is allowed in an escape sequence. Use the native2ascii Java tool to convert configuration files to and from other character encodings.

The following OData Producer options and connection parameter settings can be specified in the configuration file.

A single configuration file may be used to configure multiple producers. In this case, all producers must be named and producer names must only include ASCII letters, ASCII numbers or the underscore character ([a-zA-Z0-9_]).

To apply an option to an specific OData Producer, use the [`producer-name.`] prefix (exemplified in the table below), where `producer-name` is the name of the OData Producer that the setting applies to. All options that are not prefixed are considered global. Individual OData Producer options supersede any specified global options.

Option	Description
[producer-name.]*Authentication* = { *none* | *database* }	Specifies the credentials that this OData Producer uses to connect to the database. The default setting, *database*, indicates that each user connects with personalized credentials that are appended to the DbConnectionString option to form their own complete database connection string. These credentials are requested using Basic HTTP authentication. The *none* setting indicates that all users connect using the same connection string as indicated by the DbConnectionString option. This setting is recommended for testing and read-only producers. See OData server security considerations for more information.
[producer-name.]*ConnectionPoolMaximum* = num-max-connections	Indicates the maximum number of simultaneous connections that this OData Producer keeps open for use in the connection pool. Fewer connections might be used by the connection pool depending on the server load. By default, the connection pool size is limited to half of the maximum number of simultaneous connections that the database server supports.
[producer-name.]*CSRFTokenTimeout* = num-seconds-valid	Enables CSRF token checking and specifies the number of seconds that a token is valid for. By default, this value is 0, which disables CSRF token checking. Otherwise, the number of seconds must be a valid integer value from 1 to 1800.
[producer-name.]*DbConnectionString* = connection-string	Specifies the connection string used to connect to the database. The connecting string should exclude the *UID* and *PWD* parameters when the Authentication option is set to the database.
[producer-name.]*DbProduct* = *sqlanywhere*	Indicates that the OData Producer handles content for a SQL Anywhere database.

Option	Description
[producer-name.]*Model* = path-and-filename	Specifies the path and file name of an OData Service Definition Language (OSDL) file that contains the OData Producer service model that indicates which tables and views are exposed in the OData service (subject to user privileges). The path is relative to the servlet's context and then the OData server's current directory. Path and file name are processed for environment variable references in the *${variable-name}* format. The default behavior is to expose all tables and views that the user has privileges to and not expose any stored procedures or functions. Tables and views without primary keys are not exposed.
[producer-name.]*ModelConnectionString* = connection-string	Specifies a connection string that this OData Producer uses to validate the OSDL file during startup. OSDL validation ensures that enlisted tables and columns exist, that key lists are used appropriately, and that the file is semantically correct. The connection string should include the *UID* and *PWD* parameters unless it includes a *DSN* that provides those parameters. The default behavior is to assume that the OSDL file is valid (the model will be validated when the first request is handled).
[producer-name.]*ODataAdmin* = admin-username:admin-password	References the login information for the database user acting as the OData Administrator. The OData Producer uses this user to create tables, a procedure, and an event to manage repeatable requests. The database administrator can modify the event used to clean up repeatable requests. This user should not be used for accessing OData services.
[producer-name.]*PageSize* = num-max-entities	Specifies the maximum number of entities to include in a retrieve entity set response before issuing a next link. The default setting is 100.
[producer-name.]*ReadOnly* = { *true* | *false* }	Indicates whether modification requests should be ignored. The default setting is false.
[producer-name.]*SecureOnly* = { *true* | *false* }	Indicates whether the Producer should only listen for requests on the HTTPS port. The default setting is false.

Option	Description
[producer-name.]*SecureOnly* = { days-number }	Specifies how long repeatable requests are valid. This value must be an integer ranging from 1 to 31. The default setting is 2.
[producer-name.]*ServiceOperationColumnNames* = { *generate* | *database* }	Specifies whether the column names in the metadata should be generated or taken from the result set columns in the database when naming the properties of the ComplexType used in the ReturnType. The default setting is generate.
[producer-name.]*ServiceRoot* = / path-prefix	Specifies the root of the OData service on the OData Server. Each OData Producer must have a unique service root, and each service root must not be a subpath of another service root. The service root of each producer must match the third party HTTP server's configuration for that producer. The default setting is */odata*. OData Producer resources are accessed using URIs of the following format: scheme:host:port/path-prefix/resource-path[query-options]

Example

The following sample illustrates how a configuration file is structured for two OData producers:

```
# Shared OData Producer options
# -----------------------------
Authentication = none
ConnectionPoolMaximum = 10
CSRFTokenTimeout = F00
DbProduct = sqlanywhere
ODataAdmin = SuperODataAdmin:SecretPassword
PageSize = 100
ReadOnly = false
RepeatRequestForDays = 3
ServiceOperationColumnNames = generate
# OrderEntryProducer options
# --------------------------
OrderEntryProducer.Model = ../../ordermodel.osdl
OrderEntryProducer.ModelConnectionString =
uid=dba;pwd=passwd;server=orderentry;dbf=orderentry.db
OrderEntryProducer.ServiceRoot = /orders/
OrderEntryProducer.DbConnectionString =
uid=dba;pwd=passwd;server=orderentry;dbf=orderentry.db
# StaffingProducer options
# ------------------------
StaffingProducer.Model = ../../staffingmodel.osdl
StaffingProducer.ModelConnectionString =
uid=dba;pwd=passwd;server=staffing;dbf=staffing.db
```

```
StaffingProducer.ServiceRoot = /staffing/
StaffingProducer.DbConnectionString =
uid=dba;pwd=passwd;server=staffing;dbf=staffing.db
```

Related Information

How to Set Up an OData Server [page 586]
How to Create an OData Producer Service Model [page 591]
OData Server Security Considerations [page 584]
OData Server Samples [page 588]
CREATE ODATA PRODUCER Statement
-xs Database Server Option

1.19.10 OSDL Statement Reference

OSDL statements create an OData Producer service model to expose specific tables, views, stored procedures, and functions.

Prefix a line with the # character to create a comment.

> **Note**
>
> All entity types, properties, service operations, and their parameters must be valid OData identifiers. OSDL files must be UTF8 encoded.

In this section:

service OSDL Statement [page 597]
Creates a service namespace.

Related Information

How to Create an OData Producer Service Model [page 591]
How to Configure OData Producers for a Third-party HTTP server [page 592]
CREATE ODATA PRODUCER Statement
OData Server Samples [page 588]

1.19.10.1 service OSDL Statement

Creates a service namespace.

Syntax

```
service [ namespace "namespace-name" ] {
    { entity-statement | association-statement | serviceop-statement }
    ...
}
```

Parameters

namespace-name

When this parameter is specified, *_Container* is appended to `namespace-name` to generate a container name.

entity-statement

A qualified entity statement.

association-statement

A qualified association statement.

serviceop-statement

A qualified service operation statement.

Remarks

Services can contain references to multiple entities, associations, or service operations.

In this section:

association OSDL Statement [page 598]
Creates associations between entities, including complex associations that use an underlaying association table.

entity OSDL Statement [page 600]
Creates an entity.

serviceop OSDL Statement [page 602]
Exposes a service as an HTTP GET or POST operation.

1.19.10.1.1 association OSDL Statement

Creates associations between entities, including complex associations that use an underlaying association table.

Syntax

```
association "association-name"
    principal "principal-entityset-name" ("column-name"[ , ... ])
  multiplicity "[ 1 | 0...1 | 1..* | * ]"
    dependent "dependent-entityset-name" ("column-name"[ , ... ])
  multiplicity "[ 1 | 0...1 | 1..* | * ]"
    over "owner"."object-name" principal ("column-name"[ , ... ])
  dependent ("column-name"[ , ... ])
```

Parameters

association-name

The name of an association.

principal-entityset-name

The name of a principal entity set.

dependent-entityset-name

The name of a dependent entity set.

object-name

The name of the database table or view.

Remarks

Associations are created with the foreign keys that are defined in the database if no associations are specified in the model.

In an association that does not have an association table, the number of join properties for the principal end must match the number of join properties for the dependent end.

In an association that has an association table, the number of join properties for the principal end must match the number of association table principal end join properties, and the number of join properties of the dependent end must match the number of association table dependent end properties.

When defining an association between two tables, the dependent table should reference the principal table.

The ordering of join properties for principal and dependent ends is relevant. The first column of the principal end is matched with the first column of the dependent end, and so on. The matching columns must also be the same type.

> **Note**
>
> The OData Producers do not enforce the uniqueness of key columns or the cardinality constraints. The Producers rely on the underlying database to enforce these constraints.

Example

Consider the following tables owned by the user DBA.

```
CREATE OR REPLACE TABLE ThingHolder(
   ID INTEGER PRIMARY KEY DEFAULT AUTOINCREMENT,
   Name VARCHAR(50)
);
CREATE OR REPLACE TABLE Things(
    ID INTEGER PRIMARY KEY DEFAULT AUTOINCREMENT,
    Holder INTEGER,
    Description VARCHAR(50),
    FOREIGN KEY ThingHolderRef(Holder) REFERENCES ThingHolder
);
```

The following is a simple association for these tables:

```
service {
    entity "dba"."ThingHolder" as "ThingHolder"
        navigates( "ThingHolderRef" as "MyThings" );
    entity "dba"."Things" as "Things"
        without( "Holder" )
        navigates( "ThingHolderRef" as "MyThingHolder" );
    association "ThingHolderRef"
        principal "ThingHolder" ("ID") multiplicity "0..1"
        dependent "Things" ("holder") multiplicity "*";
}
```

Consider the following tables owned by the user DBA.

```
CREATE OR REPLACE TABLE ThingHolder(
    ID INTEGER,
    Name VARCHAR(50) NULL,
    PRIMARY KEY (ID)
);
CREATE OR REPLACE TABLE Things(
    ID INTEGER,
    Description VARCHAR(50),
    PRIMARY KEY (ID)
);
CREATE OR REPLACE TABLE AssocTable (
    Holder      INTEGER,
    Thing       INTEGER,
    PRIMARY KEY (Holder, Thing),
    FOREIGN KEY ThingHolderRef( Holder ) REFERENCES ThingHolder,
    FOREIGN KEY ThingsRef( Thing ) REFERENCES Things
);
```

The following service model exposes the first two tables that are related using the association table:

```
service {
    entity "dba"."ThingHolder" as "ThingHolder"
        navigates( "ThingHolderRef" as "Things" );
    entity "dba"."Things" as "Things"
```

```
        navigates( "ThingHolderRef" as "ThingHolderRef" );
    association "ThingHolderRef"
        principal "ThingHolder" ("ID") multiplicity "0..1"
        dependent "Things" ("ID") multiplicity "*"
        over "dba"."AssocTable" principal ("holder") dependent ("thing");
}
```

1.19.10.1.2 entity OSDL Statement

Creates an entity.

Syntax

```
entity "owner"."object-name" [ as "entityset-name" ] [ { with | without } ("included-
or-excluded-column-name"[ , ... ]) ]
    [ keys { ("key-column-name"[ , ... ]) | generate local "key-column-name" } ]
    [ concurrencytoken ("token-column-name" [ , ... ] ) ]
    [ navigates ("association-name" as "navprop-name" [ from { principal | dependent
} ] [, ...] ) ]
```

Parameters

owner

The name of the database user who owns the database table or view to expose in the OData metadata.

object-name

The name of the database table or view to expose in the OData metadata.

entityset-name

The name of the exposed table or view defined by `object-name`. This name must be unique within the namespace and should not conflict with other service operations or with any entity set names. By default the OData producer will attempt to use `object-name` then `owner_object-name` as the entity set name.

included-or-excluded-column-name

When used in conjunction with the *with* clause, this parameter specifies a column name to include in the entity set defined by the `entityset-name`.

When used in conjunction with the *without* clause, this parameter specifies a column name to exclude in the entity set defined by the `entityset-name`.

You can reference multiple column names.

key-column-name

When used in conjunction with the *keys* clause, this parameter specifies a column name to use as a primary key when one is not specified in the table or view defined by `object-name`.

When used in conjunction with the *generate local* clause, this parameter specifies a generated key defined as the Edm.Int64 data type. Results are numbered with a starting index of 1.

Use generated keys to provide unique IDs in results. They cannot be used for retrieving or dereferencing the entity.

Generated keys are only valid for the duration of the current session. Generated properties cannot be used in $filter or $orderby. Entity types with generated properties cannot be used in associations.

Keys may have multiple columns.

token-column-name

The name of the column (property) that is always modified when an entity instance is modified, such as a DEFAULT CURRENT TIMESTAMP column or an INTEGER column that is incremented by an update trigger. You can reference multiple column names.

association-name

A reference to an association defined in the OSDL file. You can define multiple associations.

navprop-name

The name of a navigation property name.

Remarks

The following restrictions apply to the *keys* clause:

- Specify a key list when referencing a table or view that does not contain a primary key.
- Do not specify a key list when referencing a table that contains a primary key.

Primary key properties are ignored when included in a property list associated with the *with* and *without* keywords. The primary key columns are always included.

Specify the *from* clause if both ends of the association are of the same type.

The *concurrencytoken* clause generates ETags that identify the state of an entity instance at the time the instance is requested. The SQL used to generate an ETag uses SHA256 hash functions for each concurrency token property (column) and can be complex given the types of properties (columns) and number of properties included in the concurrency token. Concurrency tokens should comprise of a small number of properties, none of which should be BLOB types.

Example

The following service model exposes a table and a view:

```
service namespace "DBServer" {
    "dba"."TableWithPrimaryKey";
    "dba"."ViewWithoutPrimaryKey" keys("primary_key1", "primary_key2");
}
```

The following service model generates ETags:

```
service {
    entity "dba"."T1" as "T1" concurrencytoken("version") ;
    entity "dba"."T2" as "T2" concurrencytoken("ver_maj", "ver_min", "ver_bld");
    entity "dba"."T3" as "T3" concurrencytoken("ver_ts");
}
```

1.19.10.1.3 serviceop OSDL Statement

Exposes a service as an HTTP GET or POST operation.

Syntax

```
serviceop { get | post } "owner"."procedure-or-function-name
    [ as service-name ]
    [ returns multiplicity " { 0 | 1 | * } " ]
```

Parameters

owner

The name of the database user who owns the database stored procedure or function to expose in the OData metadata.

procedure-or-function-name

The name of the database stored procedure or function to expose in the OData metadata.

service-name

The name to expose the OData service as. This name must be unique within the namespace and should not conflict with other service operations or with any entity set names.

By default the OData producer will attempt to use `procedure-or-function-name` then `owner_procedure-or-function-name` as the service name.

returns multiplicity clause

The following values are used to indicate the multiplicity:

"0"

Use this value for service operations that have no output.

"1"

Use this value for service operations whose return type is either a single simple type or a single complex type.

"*"

Use this value for service operations whose return type is either a collection of simple types or a collection of complex types.

Remarks

Expose a service operation as both a GET and POST operation by declaring it twice with a unique name each time.

The OData Producer creates complex types for the result as needed. The complex type for a service operation includes any output parameters, as well as function returns or store procedure results.

OData Producers assume that service operations invoked using GET requests do not modify the database. All service operations that modify the database should follow OData conventions and be restricted to POST requests only.

Example

The following example declares several service operations:

```
service {
    serviceop post "dba"."P0R0" as "POST0R0" returns multiplicity "0";
    serviceop get "dba"."P0R0" returns multiplicity "0";
    serviceop get "dba"."P1R0" returns multiplicity "0";
    serviceop get "dba"."P0R3" returns multiplicity "*";
    serviceop get "dba"."P1R1" returns multiplicity "1";
}
```

1.20 HTTP Web Services

You can create HTTP web servers using SQL Anywhere as well as send requests and receive replies from other web servers.

In this section:

The Database Server as an HTTP Web Server [page 603]
The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

Access to Web Services Using Web Clients [page 647]
The database server can be used as a web client to access web services hosted by a SQL Anywhere HTTP web server or third party web servers such as Apache or IIS.

Web Service Error Code Reference [page 682]
The HTTP server generates standard web service errors when requests fail. These errors are assigned numbers consistent with protocol standards.

HTTP Web Service Examples [page 684]
Several sample implementations of web services are located in the *%SQLANYSAMP17%* `\SQLAnywhere\HTTP` directory.

1.20.1 The Database Server as an HTTP Web Server

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

You can handle HTTP and SOAP over HTTP requests sent by web browsers and client applications. The web server performance is optimized because web services are embedded in the database.

Web services provide client applications with an alternative to traditional interfaces, such as JDBC and ODBC. They are easily deployed because additional components are not needed, and can be accessed from multi-platform client applications written in a variety of languages, including scripting languages, such as Perl and Python.

In addition to providing web services over an HTTP web server, the database server can also function as a SOAP or HTTP client application to access standard web services available over the Internet, including those provided by SQL Anywhere HTTP web servers.

In this section:

Quick Start to Using the Database Server as an HTTP Web Server [page 604]
You can start a SQL Anywhere HTTP web server, create a web service, and access it from a web browser.

How to Start an HTTP Web Server [page 606]
The HTTP web server that is built into the SQL Anywhere database server is started with the -xs server option or the sp_start_listener system procedure.

What Are Web Services [page 609]
Web services refer to software that assists inter-computer data transfer and interoperability.

How to Develop Web Service Applications in an HTTP Web Server [page 621]
Customized web pages can be produced by the stored procedures that are called from web services. A very elementary web page example is shown below.

How to Browse a SQL Anywhere HTTP Web Server [page 643]
How your web services are named and designed dictates what constitutes a valid URL.

Related Information

Access to Web Services Using Web Clients [page 647]

1.20.1.1 Quick Start to Using the Database Server as an HTTP Web Server

You can start a SQL Anywhere HTTP web server, create a web service, and access it from a web browser.

Perform the following tasks to create a SQL Anywhere HTTP web server and HTTP web service:

1. Start the HTTP web server and a database. For example, run the following command at a command prompt:

```
dbsrv17 -xs http(port=8082) "%SQLANYSAMP17%\demo.db"
```

> ⓘ Note
>
> Use the dbsrv17 command to start a database server that can be accessed on a network. Use the dbeng17 command to start a personal database server that can be accessed by the local host only. In production systems, HTTPS should be used.

The -xs http(port=8082) option instructs the server to listen for HTTP requests on port 8082. Use a different port number if a web server is already running on port 8082.

2. Use the CREATE SERVICE statement to create a web service that responds to incoming web browser requests.
   1. Connect to the *demo.db* database using Interactive SQL by running the following command:

      ```
      dbisql -c "dbf=%SQLANYSAMP17%\demo.db;uid=DBA;pwd=sql"
      ```

   2. Create a new web service in the database.
      Execute the following SQL statement in Interactive SQL:

      ```
      CREATE SERVICE SampleWebService
          TYPE 'web-service-type-clause'
          AUTHORIZATION OFF
          USER DBA
          AS SELECT 'Hello world!';
      ```

      Replace `web-service-type-clause` with the desired web service type. The HTML type clause is recommended for web browser compatibility. Other general HTTP web service type clauses include XML, RAW, and JSON.
      The CREATE SERVICE statement creates the SampleWebService web service, which returns the result set of the SELECT statement. In this example, the statement returns "Hello world!"
      The AUTHORIZATION OFF clause indicates that authorization is not required to access the web service.
      The USER DBA statement indicates that the service statement should be run under the DBA login name.
      The AS SELECT clause allows the service to select from a table or function, or view data directly. Use AS CALL as an alternative clause to call a stored procedure.
3. View the web service in a web browser.
   On the computer running the SQL Anywhere HTTP web server, open a web browser, such as Microsoft Internet Explorer or Firefox, and go to the following URL:

   ```
   http://localhost:8082/demo/SampleWebService
   ```

   This URL directs your web browser to the HTTP web server on port 8082. SampleWebService prints "Hello world". The result set output is displayed in the format specified by the `web-service-type-clause` from step 2.

Other Sample Resources

Samples are included in the *%SQLANYSAMP17%*`\SQLAnywhere\http` directory.

Related Information

How to Start an HTTP Web Server [page 606]
Web Service Types [page 610]
CREATE SERVICE Statement [HTTP Web Service]

1.20.1.2 How to Start an HTTP Web Server

The HTTP web server that is built into the SQL Anywhere database server is started with the -xs server option or the sp_start_listener system procedure.

The HTTP web server starts automatically when you launch the database server with the -xs server option.

An HTTP web server can also be started using the sp_start_listener system procedure.

Both methods allow you to perform the following tasks:

- Enable a web service protocol to listen for web service requests.
- Configure network protocol options, such as server port, logging, timeout criteria, and the maximum request size.

The general format of the command line is:

```
dbsrv17 -xs protocol-type(protocol-options) your-database-name.db
```

The general format for the stored procedure call is:

```
CALL sp_start_listener(protocol-type,address-port,protocol-options);
```

Replace `protocol-type` and `protocol-options` with one of the following supported protocols and options:

HTTP

Use this protocol to listen for HTTP connections. Here are examples.

```
dbsrv17 -xs HTTP(PORT=8082) services.db
```

```
CALL sp_start_listener('HTTP','127.0.0.1:8082','LOG=c:\\temp\\http.log');
```

HTTPS

Use this protocol to listen for HTTPS connections. TLS versions 1.0 or later are supported. Here are examples.

```
dbsrv17 -xs "HTTPS(FIPS=N;PORT=443;IDENTITY=C:\Users\Public\Documents\SQL
Anywhere
        17\Samples\Certificates\rsaserver.id;IDENTITY_PASSWORD=test)"
services.db
```

```
CALL sp_start_listener('HTTPS','127.0.0.1:443','LOG=c:\
\temp\\https.log;FIPS=N;IDENTITY=C:\\SQLA\\Samples\\Certificates\
\rsaserver.id;IDENTITY_PASSWORD=test');
```

Network protocol options are available for each supported protocol. These options allow you to control protocol behavior.

In this section:

Configuration of Network Protocol Options [page 607]
Network protocol options are optional settings that provide control over a specified web service protocol.

How to Start Multiple HTTP Web Servers [page 609]

A multiple HTTP web server configuration allows you to create web services across databases and have them appear as part of a single web site.

Related Information

What Are Web Services [page 609]
-xs Database Server Option
sp_start_listener System Procedure

1.20.1.2.1 Configuration of Network Protocol Options

Network protocol options are optional settings that provide control over a specified web service protocol.

These settings are configured at the command line when you launch your database server with the -xs database server option. For example, the following command line configures an HTTPS listener with the PORT, FIPS, Identity, and Identity_Password network protocol options specified:

```
dbsrv17 -xs https(PORT=544;FIPS=YES;
    IDENTITY=rsaserver.id;IDENTITY_PASSWORD=test) your-database-name.db
```

This command starts a database server that enables the HTTPS web service protocol for the `your-database-name.db` database.

These settings are also configured by calling the sp_start_listener system procedure. For example, the following SQL CALL configures an HTTPS listener on localhost using port 544 with the FIPS, Identity, and Identity_Password network protocol options specified:

```
CALL
sp_start_listener('HTTPS','127.0.0.1:544','FIPS=YES;IDENTITY=rsaserver.id;IDENTIT
Y_PASSWORD=test');
```

The network protocol options indicate that the web server should perform the following tasks:

- Listen on port 544 instead of the default HTTPS port (443).
- Enable FIPS-certified security algorithms to encrypt communications.
- Locate the specified identity file, `rsaserver.id`, which contains a public certificate and its private key.
- Use the specified identity password, **`test`**, to decrypt the private key.

The following list identifies the network protocol options that are commonly used for web service protocols:

Network Protocol Option	Available Web Service Protocols	Description
DatabaseName (DBN)	HTTP, HTTPS	Specifies the name of a database to use when processing web requests, or uses the REQUIRED or AUTO keyword to specify whether database names are required as part of the URL.

Network Protocol Option	Available Web Service Protocols	Description
FIPS	HTTPS	Enables FIPS-certified (FIPS PUB 140-2) security algorithms to encrypt the communications for database client/server communication and web services.
Identity	HTTPS	Specifies the name of an identity file to use for secure HTTPS connections.
Identity_Password	HTTPS	Specifies the password for the identity file.
LocalOnly (LO)	HTTP, HTTPS	Allows a database server to restrict connections to the local computer only.
LogFile (LOG)	HTTP, HTTPS	Specifies the name of the file where the database server writes information about web requests.
LogFormat (LF)	HTTP, HTTPS	Controls the format of messages written to the message log file where the database server writes information about web requests, and specifies which fields appear in the messages.
LogOptions (LOPT)	HTTP, HTTPS	Specifies the types of messages that are recorded in the log where the database server writes information about web requests.
ServerPort (PORT)	HTTP, HTTPS	Specifies the port on which the database server listens for requests.

The ServerPort (PORT) protocol option cannot be used with sp_start_listener.

Related Information

Network Protocol Options
sp_start_listener System Procedure
-xs Database Server Option
DatabaseName (DBN) Protocol Option
FIPS Protocol Option
identity Protocol Option
identity_password Protocol Option
LocalOnly (LO) Protocol Option
LogFile (LOG) Protocol Option
LogFormat (LF) Protocol Option
LogOptions (LOPT) Protocol Option
ServerPort (PORT) Protocol Option

1.20.1.2.2 How to Start Multiple HTTP Web Servers

A multiple HTTP web server configuration allows you to create web services across databases and have them appear as part of a single web site.

You can start multiple HTTP web servers by using multiple instances of the -xs database server option. This task is performed by specifying a unique port number for each HTTP web server.

Example

In this example, the following command line starts two HTTP web services: one for `your-first-database.db` and one for `your-second-database.db`:

```
dbsrv17 -xs http(port=80;dbn=your-first-database),http(port=8800;dbn=your-second-database)
    your-first-database.db your-second-database.db
```

The following example is equivalent to the previous one.

```
dbsrv17 -xs http(port=80;dbn=your-first-database) -xs http(port=8800;dbn=your-second-database)
    your-first-database.db your-second-database.db
```

Related Information

-xs Database Server Option
DatabaseName (DBN) Protocol Option

1.20.1.3 What Are Web Services

Web services refer to software that assists inter-computer data transfer and interoperability.

They make segments of business logic available over the Internet. URLs become available to clients when managing web services in an HTTP web server. The conventions used when specifying a URL determine how the server should communicate with web clients.

Web service management involves the following tasks:

- Choosing the types of web services that you want to manage.
- Creating and maintaining those web services

Web services can be created and stored in the database.

In this section:

Web Service Types [page 610]

The SQL Anywhere HTTP web server supports HTML, XML, RAW, JSON, as well as SOAP/DISH web service types.

Web Service Maintenance [page 612]
SQL statements are used to create, alter, drop, and enable or disable web services.

Connection Pooling for Web Services [page 620]
Each database that exposes web services has access to a pool of database connections. The pool is grouped by user name such that all services defined under a given USER clause share the same connection pool group.

1.20.1.3.1 Web Service Types

The SQL Anywhere HTTP web server supports HTML, XML, RAW, JSON, as well as SOAP/DISH web service types.

When a web browser or client application makes a web service request to a SQL Anywhere HTTP web server, the request is processed and a result set is returned in the response. Several web service types are provided that give you control over the result set format and how result sets are returned. You specify the web service type with the TYPE clause of the CREATE SERVICE or ALTER SERVICE statement after choosing an appropriate web service type.

The following web service types are supported:

HTML

The result set of a statement, function, or procedure is formatted into an HTML document that contains a table. Web browsers display the body of the HTML document.

XML

The result set of a statement, function, or procedure is returned as an XML document. Non-XML formatted result sets are automatically formatted into XML. Web browsers display the raw XML code, including tags and attributes.

The XML formatting is the equivalent of using the FOR XML RAW clause in a SELECT statement, such as in the following SQL statement example:

```
SELECT * FROM table-name FOR XML RAW
```

RAW

The result set of a statement, function, or procedure is returned without automatic formatting.

This service type provides the most control over the result set. However, you must generate the response by writing the necessary markup (HTML, XML) explicitly within your stored procedure. You can use the SA_SET_HTTP_HEADER system procedure to set the HTTP Content-Type header to specify the MIME type, allowing web browsers to correctly display the result set.

Web pages can be customized using the RAW web service type.

JSON

The result set of a statement, function, or procedure is returned in JSON (JavaScript Object Notation). JavaScript Object Notation (JSON) is a language-independent, text-based data interchange format developed for the serialization of JavaScript data. JSON represents four basic types: strings, numbers, booleans, and NULL. JSON also represents two structured types: objects and arrays.

This service is used by AJAX to make HTTP calls to web applications. For an example of the JSON type, see *%SQLANYSAMP17%*`\SQLAnywhere\HTTP\json_sample.sql`.

SOAP

The result set of a statement, function, or procedure is returned as a SOAP response. SOAP services provide a common data interchange standard to provide data access to disparate client applications that support SOAP. SOAP request and response envelopes are transported as an XML payload using HTTP (SOAP over HTTP). A request to a SOAP service must be a valid SOAP request, not a general HTTP request. The output of SOAP services can be adjusted using the FORMAT and DATATYPE attributes of the CREATE or ALTER SERVICE statement.

DISH

A DISH service (Determine SOAP Handler) is a SOAP endpoint. The DISH service exposes the WSDL (Web Services Description Language) document that describes all SOAP Operations (SOAP services) accessible through it. A SOAP client toolkit builds the client application with interfaces based on the WSDL. The SOAP client application directs all SOAP requests to the SOAP endpoint (the DISH service).

Example

The following example illustrates the creation of a general HTTP web service that uses the RAW service type:

```
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 );
```

Related Information

How to Customize Web Pages [page 622]
How to Create or Alter a Web Service [page 613]
HTTP and SOAP Request Structures [page 680]
How to Create DISH Services [page 615]
How to Create SOAP over HTTP Services [page 614]
Web Services System Procedures
CREATE SERVICE Statement [HTTP Web Service]
Introducing JSON

1.20.1.3.2 Web Service Maintenance

SQL statements are used to create, alter, drop, and enable or disable web services.

Web service maintenance involves the following tasks:

Creating or altering web services

Create or alter web services to provide web applications supporting a web browser interface and provide data interchange over the web using REST and SOAP methodologies.

Dropping web services

Dropping a web service causes the subsequent requests made for that service to return a `404 Not Found` HTTP status message. All unresolved requests, intended or unintended, are processed if a **`root`** web service exists.

Commenting on web services

Commenting is optional and allows you to provide documentation for your web services.

Creating and customizing a root web service

You can create a **`root`** web service to handle HTTP requests that do not match any other web service requests.

Enabling and disabling web services

A disabled web service returns a `404 Not Found` HTTP status message. The METHODS clause of the CREATE SERVICE statement specifies the HTTP methods that can be called for a particular web service.

In this section:

How to Create or Alter a Web Service [page 613]
Creating or altering a web service requires use of the CREATE SERVICE or ALTER SERVICE statement, respectively. These statements can be executed with Interactive SQL to create different kinds of web services.

How to Drop a Web Service [page 616]
Dropping a web service causes the subsequent requests made for that service to return a `404 Not Found` HTTP status message. All unresolved requests, intended or unintended, are processed if a **`root`** web service exists.

How to Comment a Web Service [page 617]
Providing documentation for a web service requires use of the COMMENT ON SERVICE statement.

How to Create and Customize a Root Web Service [page 617]
An HTTP client request that does not match any web service request is processed by the **`root`** web service if a **`root`** web service is defined.

Web Service SQL Statements [page 619]
SQL statements are used to implement web services in a database.

Related Information

Quick Start to Using the Database Server as an HTTP Web Server [page 604]

CREATE SERVICE Statement [HTTP Web Service]

1.20.1.3.2.1 How to Create or Alter a Web Service

Creating or altering a web service requires use of the CREATE SERVICE or ALTER SERVICE statement, respectively. These statements can be executed with Interactive SQL to create different kinds of web services.

In this section:

How to Create HTTP Web Services [page 613]
HTTP web services are classified as HTML, XML, RAW or JSON. All HTTP web services can be created or altered using the same CREATE SERVICE and ALTER SERVICE statement syntax.

How to Create SOAP over HTTP Services [page 614]
SOAP is a data interchange standard supported by many development environments.

How to Create DISH Services [page 615]
DISH services act as SOAP endpoints for groups of SOAP services. DISH services also automatically construct WSDL (Web Services Description Language) documents that allow SOAP client toolkits to generate the interfaces necessary to interchange data with the SOAP services described by the WSDL.

1.20.1.3.2.1.1 How to Create HTTP Web Services

HTTP web services are classified as HTML, XML, RAW or JSON. All HTTP web services can be created or altered using the same CREATE SERVICE and ALTER SERVICE statement syntax.

Example

Execute the following statement in Interactive SQL to create a sample general HTTP web service in the HTTP web server:

```
CREATE SERVICE SampleWebService
    TYPE 'web-service-type-clause'
    URL OFF
    USER DBA
    AUTHORIZATION OFF
    AS sql-statement;
```

The CREATE SERVICE statement creates a new web service named SampleWebService and returns the result set of `sql-statement`. You can replace `sql-statement` with either a SELECT statement to select data from a table or view directly, or a CALL statement to call a stored procedure in the database.

Replace `web-service-type-clause` with the desired web service type. Valid clauses for HTTP web services include HTML, XML, RAW and JSON.

You can view the generated result set for the SampleWebService service by accessing the service in a web browser.

Related Information

How to Develop Web Service Applications in an HTTP Web Server [page 621]
Web Service Types [page 610]
CREATE SERVICE Statement [HTTP Web Service]
ALTER SERVICE Statement [HTTP Web Service]
How to Browse a SQL Anywhere HTTP Web Server [page 643]

1.20.1.3.2.1.2 How to Create SOAP over HTTP Services

SOAP is a data interchange standard supported by many development environments.

A SOAP payload consists of an XML document, known as a SOAP envelope. A SOAP request envelope contains the SOAP operation (a SOAP service) and specifies all appropriate parameters. The SOAP service parses the request envelope to obtain the parameters and calls or selects a stored procedure or function just as any other service does. The presentation layer of the SOAP service streams the result set back to the client within a SOAP envelope in a predefined format as specified by the DISH service's WSDL.

By default, SOAP service parameters and result data are typed as XmlSchema string parameters. DATATYPE ON specifies that input parameters and response data should use TRUE types. Specifying DATATYPE changes the WSDL specification accordingly, so that client SOAP toolkits generate interfaces with the appropriate type of parameters and response objects.

The FORMAT clause is used to target specific SOAP toolkits with varying capabilities. DNET provides Microsoft .NET client applications to consume a SOAP service response as a System.Data.DataSet object. CONCRETE exposes a more general structure that allows an object-oriented application, such as .NET or Java, to generate response objects that package rows and columns. XML returns the entire response as an XML document, exposing it as a string. Clients can further process the data using an XML parser. The FORMAT clause of the CREATE SERVICE statement supports multiple client application types.

> **Note**
>
> The DATATYPE clause only pertains to SOAP services (there is no data typing in HTML) The FORMAT clause can be specified for either a SOAP or DISH service. A SOAP service FORMAT specification overrides that of the DISH service.

Example

Execute the following statement in Interactive SQL to create a SOAP over HTTP service:

```
CREATE SERVICE SampleSOAPService
    TYPE 'SOAP'
    DATATYPE ON
    FORMAT 'CONCRETE'
    USER DBA
    AUTHORIZATION OFF
    AS sql-statement;
```

Related Information

Variables Accessed from Result Sets [page 669]
HTTP Web Service Examples [page 684]
Web Service Types [page 610]
How to Create DISH Services [page 615]
CREATE SERVICE Statement [SOAP Web Service]
ALTER SERVICE Statement [SOAP Web Service]
Simple Object Access Protocol (SOAP)

1.20.1.3.2.1.3 How to Create DISH Services

DISH services act as SOAP endpoints for groups of SOAP services. DISH services also automatically construct WSDL (Web Services Description Language) documents that allow SOAP client toolkits to generate the interfaces necessary to interchange data with the SOAP services described by the WSDL.

SOAP services can be added and removed without requiring maintenance to the DISH services because the current working set of SOAP over HTTP services are always exposed.

Example

Execute the following SQL statements in Interactive SQL to create sample SOAP and DISH services in the HTTP web server:

```
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';
```

The first CREATE SERVICE statement creates a new SOAP service named Samples/TestSoapOp.

The second CREATE SERVICE statement creates a new DISH service named dnet_endpoint. The Samples portion of the GROUP clause identifies the group of SOAP services to expose. You can view the WSDL document generated by the DISH service. When running your SQL Anywhere HTTP web server on a computer, you can access the service using the `http://localhost:port-number/dnet_endpoint` URL, where `port-number` is the port number that the server is running on.

In this example, the SOAP service does not contain a FORMAT clause to indicate a SOAP response format. Therefore, the SOAP response format is dictated by the DISH service, which does not override the FORMAT

clause of the SOAP service. This feature allows you to create homogeneous DISH services where each DISH endpoint can serve SOAP clients with varying capabilities.

Creating Homogeneous DISH Services

When SOAP service definitions defer the specification of the FORMAT clause to the DISH service, a set of SOAP services can be grouped together within a DISH service that defines the format. Multiple DISH services can then expose the same group of SOAP services with a different FORMAT specifications. If you expand on the TestSoapOp example, you can create another DISH service named java_endpoint using the following SQL statement:

```
CREATE SERVICE "java_endpoint"
    TYPE 'DISH'
    GROUP "Samples"
    FORMAT 'CONCRETE';
```

In this example, the SOAP client receives a response object named TestSoapOp_Dataset when it makes a web service request for the TestSoapOp operation through the java_endpoint DISH service. The WSDL can be inspected to compare the differences between dnet_endpoint and java_endpoint. Using this technique, a SOAP endpoint can quickly be constructed to meet the needs of a particular SOAP client toolkit.

Related Information

Web Service Types [page 610]
HTTP Web Service Examples [page 684]
How to Create SOAP over HTTP Services [page 614]
How to Browse a SQL Anywhere HTTP Web Server [page 643]
CREATE SERVICE Statement [SOAP Web Service]
ALTER SERVICE Statement [SOAP Web Service]

1.20.1.3.2.2 How to Drop a Web Service

Dropping a web service causes the subsequent requests made for that service to return a `404 Not Found` HTTP status message. All unresolved requests, intended or unintended, are processed if a **`root`** web service exists.

Example

Execute the following SQL statement to drop a web service named SampleWebService:

```
DROP SERVICE SampleWebService;
```

Related Information

How to Create and Customize a Root Web Service [page 617]
DROP SERVICE Statement

1.20.1.3.2.3 How to Comment a Web Service

Providing documentation for a web service requires use of the COMMENT ON SERVICE statement.

A comment can be removed by setting the `statement` clause to null.

Example

For example, execute the following SQL statement to create a new comment on a web service named SampleWebService:

```
COMMENT ON SERVICE SampleWebService
    IS "This is a comment on my web service.";
```

Related Information

COMMENT Statement

1.20.1.3.2.4 How to Create and Customize a Root Web Service

An HTTP client request that does not match any web service request is processed by the **`root`** web service if a **`root`** web service is defined.

The **`root`** web service provides you with an easy and flexible method to handle arbitrary HTTP requests whose URLs are not necessarily known at the time when you build your application, and to handle unrecognized requests.

Example

This example illustrates how to use a **`root`** web service, which is stored in a table within the database, to provide content to web browsers and other HTTP clients. It assumes that you have started a local HTTP web server on a single database and listening on port 80. All scripts are run on the web server.

Connect to the database server through Interactive SQL and execute the following SQL statement to create a **root** web service that passes the url host variable, which is supplied by the client, to a procedure named PageContent:

```
CREATE SERVICE root
    TYPE 'RAW'
    AUTHORIZATION OFF
    SECURE OFF
    URL ON
    USER DBA
    AS CALL PageContent(:url);
```

The URL ON portion specifies that the full path component is made accessible by an HTTP variable named URL.

Execute the following SQL statement to create a table for storing page content. In this example, the page content is defined by its URL, MIME-type, and the content itself.

```
CREATE TABLE Page_Content (
    url          VARCHAR(1024) NOT NULL PRIMARY KEY,
    content_type VARCHAR(128)  NOT NULL,
    image        LONG VARCHAR  NOT NULL
);
```

Execute the following SQL statements to populate the table. In this example, the intent is to define the content to be provided to the HTTP client when the index.html page is requested.

```
INSERT INTO Page_Content
VALUES(
    'index.html',
    'text/html',
    '<html><body><h1>Hello World</h1></body></html>'
);
COMMIT;
```

Execute the following SQL statements to implement the PageContent procedure, which accepts the url host variable that is passed through to the **root** web service:

```
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;
```

The **root** web service calls the PageContent procedure when a request to the HTTP server does not match any other defined web service URL. The procedure checks if the client-supplied URL matches a url in the Page_Content table. The SELECT statement sends a response to the client. If the client-supplied URL was not found in the table, a generic `404 - Page Not Found` HTML page is built and sent to the client.

Some browsers will respond to the 404 status with their own page, so there is no guarantee that the generic page is displayed.

In the error message, the HTML_ENCODE function is used to encode the special characters in the client-supplied URL.

The @HttpStatus header is used to set the status code returned with the request. A 404 status indicates a Not Found error, and a 200 status indicates OK. The 'Content-Type' header is used to set the content type returned with the request. In this example, the content (MIME) type of the index.html page is text/html.

Related Information

How to Customize Web Pages [page 622]
How to Browse a SQL Anywhere HTTP Web Server [page 643]
HTML_ENCODE Function [Miscellaneous]
CREATE SERVICE Statement [HTTP Web Service]
sa_set_http_header System Procedure

1.20.1.3.2.5 Web Service SQL Statements

SQL statements are used to implement web services in a database.

The following SQL statements are available.

Web Service Related SQL Statements	Description
CREATE SERVICE	Creates a new HTTP web service or a new SOAP over HTTP or DISH service.
ALTER SERVICE	Alters an existing HTTP web service or SOAP over HTTP or DISH service.
COMMENT	Stores a comment for a database object in the system tables. Use the following syntax to comment on a web service: `COMMENT ON SERVICE 'web-service-name' IS 'your comments'`
DROP SERVICE	Drops a web service.

Related Information

CREATE SERVICE Statement [HTTP Web Service]
CREATE SERVICE Statement [SOAP Web Service]
ALTER SERVICE Statement [HTTP Web Service]
ALTER SERVICE Statement [SOAP Web Service]
COMMENT Statement
DROP SERVICE Statement

1.20.1.3.3 Connection Pooling for Web Services

Each database that exposes web services has access to a pool of database connections. The pool is grouped by user name such that all services defined under a given USER clause share the same connection pool group.

A service request executing a query for the first time must go through an optimization phase to establish an execution plan. If the plan can be cached and reused, then the optimization phase can be skipped for subsequent executions. HTTP connection pooling leverages the plan caching in database connections by reusing them whenever possible. Each service maintains its own list of connections to optimize reuse. However, during peak loads, a service can steal least used connections from within the same user group of connections.

Over time, a given connection may acquire cached plans that can optimize performance for the execution of a number of services. The number of cached plans can be controlled using the max_plans_cached option.

In a connection pool, an HTTP request for a given service tries to acquire a database connection from a pool. Unlike HTTP sessions, pooled connections are sanitized by resetting the connection scope environment, such as connection scope variables and temporary tables.

Database connections that are pooled within the HTTP connection pool are not counted as connections in use for the purposes of licensing. Connections are counted as licensed connections when they are acquired from the pool. A `503 Service Temporarily Unavailable` status is returned when an HTTP request exceeds the licensing restrictions while acquiring a connection from the pool.

Web services can only use a connection pool when they are defined with AUTHORIZATION OFF.

A database connection within a pool is not updated when changes occur to database and connection options.

Related Information

max_plans_cached Option
CREATE SERVICE Statement [HTTP Web Service]
CREATE SERVICE Statement [SOAP Web Service]
http_connection_pool_basesize Option
http_connection_pool_timeout Option
Web Services Connection Properties [page 642]

1.20.1.4 How to Develop Web Service Applications in an HTTP Web Server

Customized web pages can be produced by the stored procedures that are called from web services. A very elementary web page example is shown below.

```
CREATE SERVICE ROOT
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL MyHomePage;
CREATE OR REPLACE PROCEDURE MyHomePage()
BEGIN
  CALL sa_set_http_header ( 'Content-Type', 'text/html' );
  SELECT '<html><body><p>Welcome to my web page</p></body></html>';
END
```

When you browse to the root web page of your web server (for example, http://localhost:8082/), you are greeted with a "welcome" message.

You can develop more sophisticated web pages by creating additional services and using stored procedures to produce dynamic web page content.

For detailed examples of web service applications, see the *%SQLANYSAMP17%*`\SQLAnywhere\HTTP` directory.

In this section:

How to Customize Web Pages [page 622]
You must first evaluate the format of the web service invoked by the HTTP web server to customize your web pages.

How to Access Client-supplied HTTP Variables and Headers [page 624]
Variables and headers in an HTTP client request can be accessed by web services.

How to Access Client-supplied SOAP Request Headers [page 628]
Headers in SOAP requests can be obtained using a combination of the NEXT_SOAP_HEADER and SOAP_HEADER functions.

HTTP Session Management on an HTTP Server [page 630]
A web application can support sessions in various ways.

Character Set Conversion Considerations [page 640]
By default, character-set conversion is performed automatically on outgoing result sets consisting of text. Result sets of other types, such as binary objects, are not affected.

Cross Site Scripting Considerations [page 640]
When developing your web application, ensure that it is not vulnerable to cross-site scripting (XSS). This type of vulnerability occurs when an attacker attempts to inject a script into your web page.

Web Services System Procedures [page 641]
Use these system procedures when working with web services.

Web Services Functions [page 641]
Web service functions assist the handling of HTTP and SOAP requests within web services.

Web Services Connection Properties [page 642]
Web service connection properties are accessed using the CONNECTION_PROPERTY function.

Web Services Options [page 642]
Web service options control various aspects of HTTP server behavior.

Related Information

How to Start an HTTP Web Server [page 606]
What Are Web Services [page 609]

1.20.1.4.1 How to Customize Web Pages

You must first evaluate the format of the web service invoked by the HTTP web server to customize your web pages.

For example, web pages are formatted in HTML when the web service specifies the HTML type.

The RAW web service type provides the most customization because web service procedures and functions must explicitly generate the required markup such as HTML or XML. The following tasks must be performed to customize web pages when using the RAW type:

- Set the HTTP Content-Type header field to the appropriate MIME type, such as text/html, in the called stored procedure.
- Apply appropriate markup for the MIME type when generating web page output from the called stored procedure.

Example

The following example illustrates how to create a new web service with the RAW type specified:

```
CREATE SERVICE WebServiceName
    TYPE 'RAW'
    AUTHORIZATION OFF
    URL ON
    USER DBA
    AS CALL HomePage( :url );
```

In this example, the web service calls the HomePage stored procedure, which is required to define a single URL parameter that receives the PATH component of the URL.

Setting the Content-Type Header Field

Use the sa_set_http_header system procedure to define the HTTP Content-Type header to ensure that web browsers correctly render the content.

The following example illustrates how to format web page output in HTML using the text/html MIME-type with the sa_set_http_header system procedure:

```
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
```

Applying Tagging Conventions of the MIME-Type

You must apply the tagging conventions of the MIME-type specified by the Content-Type header in your stored procedure. Several functions are provided that allow you to create tags.

The following example illustrates how to use the XMLCONCAT and XMLELEMENT functions to generate HTML content, assuming that the sa_set_http_header system procedure is used to set the Content-Type header to the text/html MIME-type:

```
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!')
        )
    )
)
```

Since element content is always escaped unless the data type is XML, the above example uses the CAST function. Otherwise, special characters are escaped (for example, < for <).

Related Information

Web Service Types [page 610]
How to Create and Customize a Root Web Service [page 617]
Functions
sa_set_http_header System Procedure
Web Services Functions [page 641]
XMLCONCAT Function [String]
XMLELEMENT Function [String]
How to Browse a SQL Anywhere HTTP Web Server [page 643]

1.20.1.4.2 How to Access Client-supplied HTTP Variables and Headers

Variables and headers in an HTTP client request can be accessed by web services.

- You can use the web service statement declaration to pass URL variables as parameters to a stored function or procedure.
- You can call the HTTP_VARIABLE, NEXT_HTTP_VARIABLE, HTTP_HEADER, NEXT_HTTP_HEADER functions in a stored function or procedure to access variables and headers in an HTTP client request.

In this section:

How to Access HTTP Variables Using Host Parameters [page 624]
You can get the value of client-supplied variables when you pass the variable names as parameters to a function or procedure call.

How to Access HTTP Variables and Headers Using Web Service Functions [page 625]
The HTTP_VARIABLE, NEXT_HTTP_VARIABLE, HTTP_HEADER, and NEXT_HTTP_HEADER functions can be used to iterate through the variables and headers supplied by the web service client.

1.20.1.4.2.1 How to Access HTTP Variables Using Host Parameters

You can get the value of client-supplied variables when you pass the variable names as parameters to a function or procedure call.

Example

The following example illustrates how to access the parameters used in a web service named ShowTable:

```
CREATE SERVICE ShowTable
    TYPE 'RAW'
    AUTHORIZATION ON
    AS CALL ShowTable( :user_name, :table_name );
CREATE PROCEDURE ShowTable(IN username VARCHAR(128), IN tablename VARCHAR(128))
BEGIN
    -- write SQL code utilizing the username and tablename variables here.
END;
```

Service host parameters are mapped in the declaration order of procedure parameters. In the above example, the user_name and table_name host parameters map to the username and tblname parameters, respectively.

The following is an example of a URL that accesses this service.

```
http://localhost:8082/ShowTable?user_name=Groupo&table_name=Customers
```

Related Information

How to Browse a SQL Anywhere HTTP Web Server [page 643]

1.20.1.4.2.2 How to Access HTTP Variables and Headers Using Web Service Functions

The HTTP_VARIABLE, NEXT_HTTP_VARIABLE, HTTP_HEADER, and NEXT_HTTP_HEADER functions can be used to iterate through the variables and headers supplied by the web service client.

Accessing Variables Using HTTP_VARIABLE and NEXT_HTTP_VARIABLE

You can iterate through all client-supplied variables using NEXT_HTTP_VARIABLE and HTTP_VARIABLE functions within your stored procedures.

The HTTP_VARIABLE function allows you to get the value of a variable name.

The NEXT_HTTP_VARIABLE function allows you to iterate through all variables sent by the client. Pass the NULL value when calling it for the first time to get the first variable name. Use the returned variable name as a parameter to an HTTP_VARIABLE function call to get its value. Passing the previous variable name to the next_http_variable call gets the next variable name. Null is returned when the last variable name is passed.

Iterating through the variable names guarantees that each variable name is returned exactly once but the variable name order may not be the same as the order they appear in the client request.

The following example illustrates how to use the HTTP_VARIABLE function to retrieve values from parameters supplied in a client request that accesses the ShowDetail service:

```
CREATE OR REPLACE SERVICE ShowDetail
    TYPE 'HTML'
    URL PATH OFF
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowDetail();
CREATE OR REPLACE 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;
```

The following is an example of a URL that accesses this service.

```
http://localhost:8082/ShowDetail?customer_id=103&product_id=300
```

The following example illustrates how to retrieve three attributes from header-field values associated with the **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' );
```

Supplying an integer as the second parameter allows you to retrieve additional values. The third parameter allows you to retrieve header-field values from multi-part requests. Supply the name of a header field to retrieve its value.

For more examples, see the *%SQLANY17%*`\SQLAnywhere\HTTP\gallery.sql` project.

Accessing Headers Using HTTP_HEADER and NEXT_HTTP_HEADER

HTTP request headers can be obtained from a request using the NEXT_HTTP_HEADER and HTTP_HEADER functions.

The HTTP_HEADER function returns the value of the named HTTP header field.

The NEXT_HTTP_HEADER function iterates through the HTTP headers and returns the next HTTP header name. Calling this function with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the function. NULL is returned when the last header name is called.

The following table lists some common HTTP request headers and sample values:

Header Name	Header Value
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 (compatible; MSIE 7.0; Windows NT 5.2; WOW64; SV1; .NET CLR 2.0.50727)
Host	localhost:8080
Connection	Keep-Alive

The following table lists special headers and sample values:

Header Name	Header Value
@HttpMethod	GET
@HttpURI	/demo/ShowHTTPHeaders
@HttpVersion	HTTP/1.1
@HttpQueryString	id=-123&version=109&lang=en

You can use the @HttpStatus special header to set the status code of the request being processed.

The following example illustrates how to format header names and values into an HTML table.

Create the ShowHTTPHeaders web service:

```
CREATE OR REPLACE SERVICE ShowHTTPHeaders
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL HTTPHeaderExample();
```

Create a HTTPHeaderExample procedure that uses the NEXT_HTTP_HEADER function to get the name of the header, then uses the HTTP_HEADER function to retrieve the values of all instances of the header:

```
CREATE OR REPLACE PROCEDURE HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
    DECLARE instance INTEGER;
    DECLARE header_name LONG VARCHAR;
    DECLARE header_value LONG VARCHAR;
    DECLARE header_query LONG VARCHAR;
    DECLARE table_rows XML;
    SET table_rows = NULL;
    SET header_name = NULL;
  header_loop:
    LOOP
        SET header_name = NEXT_HTTP_HEADER( header_name );
        IF header_name IS NULL THEN
            LEAVE header_loop
        END IF;
        SET instance = 0;
      value_loop:
        LOOP
            SET instance = instance + 1;
            SET header_value = HTTP_HEADER( header_name, instance );
            IF header_value IS NULL THEN
                LEAVE value_loop;
            END IF;
            -- Format header name, value, and instance number 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", instance ) );
        END LOOP;
    END LOOP;
    CALL sa_set_http_header( 'Content-Type', 'text/html' );
    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" ),
                                   'Header Instance' ),
                table_rows );
END;
```

Access the ShowHTTPHeaders service in a web browser to see the request headers arranged in an HTML table. The following is an example of a URL that accesses this service.

```
http://localhost:8082/ShowHTTPHeaders?customer_id=103&product_id=300
```

Related Information

Variables Supplied to Web Services [page 665]
HTTP_VARIABLE Function [Web Service]
NEXT_HTTP_VARIABLE Function [Web Service]
HTTP_HEADER Function [Web Service]
NEXT_HTTP_HEADER Function [Web Service]

1.20.1.4.3 How to Access Client-supplied SOAP Request Headers

Headers in SOAP requests can be obtained using a combination of the NEXT_SOAP_HEADER and SOAP_HEADER functions.

The NEXT_SOAP_HEADER function iterates through the SOAP headers included within a SOAP request envelope and returns the next SOAP header name. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the NEXT_SOAP_HEADER function. This function returns NULL when called with the name of the last header.

The following example illustrates the SOAP header retrieval.

```
SET hd_key = NEXT_SOAP_HEADER( hd_key );
    IF hd_key IS NULL THEN
      -- no more header entries
      LEAVE header_loop;
    END IF;
```

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order they appear in the SOAP request.

The SOAP_HEADER function returns the value of the named SOAP header field, or NULL if not called from an SOAP service. It is used when processing an SOAP request via a web service. If a header for the given field-name does not exist, the return value is NULL.

The example searches for a SOAP header named Authentication. When it finds this header, it extracts the value for entire SOAP header and the values of the @namespace and mustUnderstand attributes. The SOAP header value might look something like this XML string:

```
<Authentication xmlns="CustomerOrderURN" mustUnderstand="1">
  <userName pwd="none">
    <first>John</first>
    <last>Smith</last>
  </userName>
</Authentication>
```

For this header, the @namespace attribute value would be CustomerOrderURN.

Also, the mustUnderstand attribute value would be 1

The interior of this XML string is parsed with the OPENXML operator using an XPath string set to `/*:Authentication/*:userName`.

```
SELECT * FROM OPENXML( hd_entry, xpath )
    WITH ( pwd LONG VARCHAR '@*:pwd',
        first_name LONG VARCHAR '*:first/text()',
        last_name LONG VARCHAR '*:last/text()' );
```

Using the sample SOAP header value shown above, the SELECT statement would create a result set as follows:

pwd	first_name	last_name
none	John	Smith

A cursor is declared on this result set and the three column values are fetched into three variables. At this point, you have all the information of interest that was passed to the web service.

Example

The following example illustrates how a web server can process SOAP requests containing parameters, and SOAP headers. The example implements an addItem SOAP operation that takes two parameters: *amount* of type int and *item* of type string. The sp_addItems procedure processes an Authentication SOAP header extracting the first and last name of the user. The values are used to populate a SOAP response Validation header via the sa_set_soap_header system procedure. The response is a result of three columns: *quantity*, *item* and *status* with types INT, LONG VARCHAR and LONG VARCHAR respectively.

```
// 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;
```

Related Information

SOAP Request Header Management [page 659]

1.20.1.4.4 HTTP Session Management on an HTTP Server

A web application can support sessions in various ways.

Hidden fields within HTML forms can be used to preserve client/server data across multiple requests. Alternatively, Web 2.0 techniques, such as an AJAX enabled client-side JavaScript, can make asynchronous HTTP requests based on client state. A database connection can be preserved for exclusive use of sessioned HTTP requests.

Any connection scope variables and temporary tables created and altered within the HTTP session are accessible to subsequent HTTP requests that specify the given SessionID. The SessionID can be specified by a GET or POST HTTP request method or specified within an HTTP cookie header. When an HTTP request is received with a SessionID variable, the server checks its session repository for a matching context. If it finds a session, the server uses its database connection for processing the request. If the session is in use, it queues the HTTP request and activates it when the session is freed.

The sa_set_http_option can be used to create, delete and change session ids.

HTTP sessions require special handling for management of the session criteria. Only one database connection exists for use by a given SessionID, so consecutive client requests for that SessionID are serialized by the server. Up to 16 requests can be queued for a given SessionID. Subsequent requests for the given SessionID are rejected with a `503 Service Unavailable` status when the session queue is full.

When creating a SessionID for the first time, the SessionID is immediately registered by the system. Subsequent requests that modify or delete the SessionID are only applied when the given HTTP request terminates. This approach promotes consistent behavior if the request processing results in a roll-back or if the application deletes and resets the SessionID.

The current session is deleted and replaced with the pending session when an HTTP request changes the SessionID. The database connection cached by the session is effectively moved to the new session context, and all state data is preserved, such as temporary tables and created variables.

For a complete example of HTTP session usage, see *%SQLANYSAMP17%* `\SQLAnywhere\HTTP\session.sql`.

> **Note**
>
> Stale sessions should be deleted and an appropriate timeout should be set to minimize the number of outstanding connections because each client application connection holds a license seat. Connections associated with HTTP sessions maintain their hold on the server database for the duration of the connection.

In this section:

How to Create an HTTP Session [page 632]
: Sessions can be created using the SessionID option in the sa_set_http_option system procedure. The session ID can be defined by any non-null string.

How to Detect an Inactive HTTP Session [page 636]
: The SessionCreateTime and SessionLastTime connection properties can be used to determine if the current connection is within a session context. The HTTP request is not running within a session context when either connection property query returns an empty string.

How to Delete an HTTP Session or Change the Session ID [page 637]
: There are a number of ways to delete an HTTP session or change the session ID.

HTTP Session Administration [page 638]
: A session created by an HTTP request is immediately instantiated so that any subsequent HTTP requests requiring that session context are queued by the session.

HTTP Session Error Codes [page 639]
: There are some common errors that occur during HTTP session management.

Related Information

sa_set_http_option System Procedure
sa_set_http_header System Procedure
CONNECTION_PROPERTY Function [System]

List of Connection Properties

1.20.1.4.4.1 How to Create an HTTP Session

Sessions can be created using the SessionID option in the sa_set_http_option system procedure. The session ID can be defined by any non-null string.

Example

The following example illustrates unique session ID creation within a user-defined SQL function called from HTTP web services.

```
CREATE OR REPLACE FUNCTION create_session()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' || CONNECTION_PROPERTY('Number') || '_'
        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
```

The sa_set_http_option system procedure returns an error if the session ID is owned by another HTTP request, thereby guaranteeing uniqueness.

Use CONNECTION_PROPERTY('SessionID') to obtain the session ID value set using the sa_set_http_option system procedure. The result is an empty string if the session ID is not defined for the connection, indicating this is a session-less connection.

A non-empty session ID string returned by this function can be used in a URL or cookie.

In this section:

How to Use URLs to Manage a Session [page 633]
In a URL session state management system, the client application or web browser provides the session ID by including SessionID=`value` in a URL.

How to Use Cookies to Manage a Session [page 634]
In a cookie session state management system, the client application or web browser provides the session ID in an HTTP cookie header instead of a URL.

1.20.1.4.4.1.1 How to Use URLs to Manage a Session

In a URL session state management system, the client application or web browser provides the session ID by including SessionID=`value` in a URL.

Session state management can be accomplished using URLs or cookies. HTTP session information can be stored in the URL of a GET request or from within the body of a POST (x-www-form-urlencoded) request, or it can be stored in HTTP cookies.

The following URL demonstrates session management using a URL. It contains the keyword SessionID and specifies the session identifier XXX:

```
http://localhost/mysession?SessionID=XXX
```

When an HTTP request is received with a SessionID variable, the server checks its session repository for a matching context. If it finds a session, the server utilizes its database connection for processing the request.

If no session with identifier XXX exists, then the request is processed as a standard session-less request (as if SessionID=XXX had not been specified).

Example

The following example illustrates how to create a RAW web service that creates and deletes sessions. A connection scope variable named request_count is incremented each time an HTTP request is made while specifying a valid session ID. This example illustrates HTTP session management using a URL only.

```
CREATE OR REPLACE FUNCTION create_session()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' || CONNECTION_PROPERTY('Number') || '_'
        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
CREATE SERVICE mysession
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL mysession_proc();
CREATE OR REPLACE PROCEDURE mysession_proc()
BEGIN
    DECLARE sesid LONG VARCHAR;
    DECLARE svcname LONG VARCHAR;
    DECLARE hostname LONG VARCHAR;
    DECLARE body LONG VARCHAR;

    SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
    SELECT CONNECTION_PROPERTY('HttpServiceName') INTO svcname;
    SELECT HTTP_HEADER( 'Host' ) INTO hostname;
    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    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>';
    ELSE IF sesid = '' THEN
        SET sesid = create_session();
        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;
    END IF;
    SELECT body;
END
```

The result of CONNECTION_PROPERTY('SessionID') is an empty string if the session ID is not defined for the connection. In this case, a new session is created by the mysession_proc procedure.

Related Information

How to Detect an Inactive HTTP Session [page 636]
How to Delete an HTTP Session or Change the Session ID [page 637]
sa_set_http_option System Procedure
sa_set_http_header System Procedure
CONNECTION_PROPERTY Function [System]
List of Connection Properties
Web Service Error Code Reference [page 682]

1.20.1.4.4.1.2 How to Use Cookies to Manage a Session

In a cookie session state management system, the client application or web browser provides the session ID in an HTTP cookie header instead of a URL.

Cookie session management is supported with the 'Set-Cookie' HTTP response header of the sa_set_http_header system procedure.

> **Note**
>
> You cannot rely on cookie state management when cookies can be disabled in the client application or web browser. Support for both URL and cookie state management is recommended. When session IDs are provided by both the URL and a cookie, then the URL-supplied session ID is used.

Example

The following example illustrates how to create a RAW web service that creates and deletes sessions. A connection scope variable named request_count is incremented each time an HTTP request is made while specifying a valid SessionID. This example illustrates HTTP session management using a cookie only. It is a variation of the mysession web service introduced earlier.

```
CREATE OR REPLACE FUNCTION set_session_cookie( session_id LONG VARCHAR, max_age
INTEGER )
RETURNS LONG VARCHAR
BEGIN
    SET cookie_str = 'sessionid=' || session_id || ';' ||
        ' expires=' || F_COOKIE_DATE( DATEADD( SECOND, max_age, CURRENT UTC
TIMESTAMP ) )|| ';' ||
        ' path=' || HTTP_ENCODE('/') || ';';
END
CREATE OR REPLACE FUNCTION f_cookie_date( in @expires timestamp )
RETURNS LONG VARCHAR
BEGIN
    DECLARE @dow ARRAY( 7 ) OF CHAR(3) = ARRAY( 'Sun', 'Mon', 'Tue', 'Wed',
'Thu', 'Fri', 'Sat' );
    DECLARE @moy ARRAY( 12 ) OF CHAR(3) = ARRAY( 'Jan', 'Feb', 'Mar', 'Apr',
'May', 'Jun',
    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' );
    RETURN @dow[[DATEPART( weekday, @expires ) ]] ||','
    || REPLACE(
    DATEFORMAT( @expires, 'DD-zzz-YY hh:nn:ss' ),
    'zzz',
    @moy[[ MONTH( @expires ) ]]
    )
END;
CREATE SERVICE mysession2
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL mysession2_proc();
CREATE OR REPLACE PROCEDURE mysession2_proc()
BEGIN
    DECLARE sesid LONG VARCHAR;
    DECLARE svcname LONG VARCHAR;
    DECLARE hostname LONG VARCHAR;
    DECLARE body LONG VARCHAR;
    SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
    SELECT CONNECTION_PROPERTY('HttpServiceName') INTO svcname;
    SELECT HTTP_HEADER( 'Host' ) INTO hostname;
    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    IF HTTP_VARIABLE('delete') IS NOT NULL THEN
        CALL sa_set_http_option( 'SessionID', NULL );
        CALL set_session_cookie( sesid, 0 );
        SET body = '<html><body>Deleted ' || sesid
            || '</BR><a href="http://' || hostname || '/' || svcname || '">Start
Again</a>';
    ELSE IF sesid = '' THEN
        SET sesid = create_session();
        CALL set_session_cookie(sesid, 1*60*60);
        CREATE VARIABLE request_count INT;
        SET request_count = 0;
        SET body = '<html><body> Created session ID ' || sesid
            || '</br><a href="http://' || hostname || '/' || svcname
            || '"> 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
            || '">Enter into Session</a></br>'
            || '<a href="http://' || hostname || '/' || svcname
            || '?delete">Delete Session</a>';
        END IF;
    END IF;
    SELECT body;
END
```

Related Information

How to Detect an Inactive HTTP Session [page 636]
How to Delete an HTTP Session or Change the Session ID [page 637]
sa_set_http_option System Procedure
sa_set_http_header System Procedure
CONNECTION_PROPERTY Function [System]
List of Connection Properties

1.20.1.4.4.2 How to Detect an Inactive HTTP Session

The SessionCreateTime and SessionLastTime connection properties can be used to determine if the current connection is within a session context. The HTTP request is not running within a session context when either connection property query returns an empty string.

The SessionCreateTime connection property provides a metric of when a given session was created. It is initially defined when the sa_set_http_option system procedure is called to establish the SessionID.

The SessionLastTime connection property provides the time when the last processed session request released the database connection upon termination of the previous request. It is returned as an empty string when the session is first created until the creator request releases the connection.

> **Note**
>
> You can adjust the session timeout duration using the http_session_timeout option.

Example

The following example illustrates session detection using the SessionCreateTime and SessionLastTime connection properties:

```
SELECT CONNECTION_PROPERTY( 'SessionCreateTime' ) INTO ses_create;
SELECT CONNECTION_PROPERTY( 'SessionLastTime' ) INTO ses_last;
```

Related Information

How to Create an HTTP Session [page 632]
How to Delete an HTTP Session or Change the Session ID [page 637]
CONNECTION_PROPERTY Function [System]
List of Connection Properties
http_session_timeout Option

1.20.1.4.4.3 How to Delete an HTTP Session or Change the Session ID

There are a number of ways to delete an HTTP session or change the session ID.

Explicitly dropping a database connection that is cached within a session context causes the session to be deleted. Session deletion in this manner is a cancel operation; any requests released from the session queue are in a canceled state. This action ensures that any outstanding requests waiting on the session are terminated. Similarly, a server or database shutdown cancels all database connections.

A session can be deleted by setting the SessionID option in the sa_set_http_option system procedure to null or an empty string.

The following code can be used for session deletion:

```
CALL sa_set_http_option( 'SessionID', null );
```

When an HTTP session is deleted or the SessionID is changed, any pending HTTP requests that are waiting on the session queue are released and allowed to run outside of a session context. The pending requests do not reuse the same database connection.

A session ID cannot be set to an existing session ID. Pending requests referring to the old SessionID are released to run as session-less requests when a SessionID has changed. Subsequent requests referring to the new SessionID reuse the same database connection instantiated by the old SessionID.

The following conditions are applied when deleting or changing an HTTP session:

- The behavior differs depending on whether the current request had inherited a session whereby a database connection belonging to a session was acquired, or whether a session-less request had instantiated a new session. If the request began as session-less, then the act of creating or deleting a session occurs immediately. If the request has inherited a session, then a change in the session state, such as deleting the session or changing the SessionID, only occurs after the request terminates and its changes have been committed. The difference in behavior addresses processing anomalies that may occur if a client makes simultaneous requests using the same SessionID.
- Changing a session to a SessionID of the current session (has no pending session) is not an error and has no substantial effect.
- Changing a session to a SessionID in use by another HTTP request is an error.
- Changing a session when a change is already pending results in the pending session being deleted and new pending session being created. The pending session is only activated once the request successfully terminates.

- Changing a session with a pending session back to its original SessionID results in the pending session being deleted without any change to the current session.

Related Information

How to Detect an Inactive HTTP Session [page 636]
CONNECTION_PROPERTY Function [System]
List of Connection Properties
sa_set_http_option System Procedure

1.20.1.4.4.4 HTTP Session Administration

A session created by an HTTP request is immediately instantiated so that any subsequent HTTP requests requiring that session context are queued by the session.

In this example, a local host client can access the session with the specified session ID, session_63315422814117, running within the database, `dbname`, running the service session_service with the following URL once the session is created on the server with the sa_set_http_option procedure.

```
http://localhost/dbname/session_service?sessionid=session_63315422814117
```

A web application can require a means to track active session usage within the HTTP web server. Session data can be found using the NEXT_CONNECTION function call to iterate through the active database connections and checking for session related properties such as SessionID.

The following SQL statements illustrate how to track an active session:

```
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';
```

If you examine the database server messages window, you see data that is similar to the following output:

```
conn_id = 30
conn_id = 29, SessionID = session_63315442223323
conn_id = 28, SessionID = session_63315442220088
conn_id = 25, SessionID = session_63315441867629
```

Explicitly dropping a connection that belongs to a session causes the connection to be closed and the session to be deleted. If the connection being dropped is currently active in servicing an HTTP request, the request is marked for deletion and the connection is sent a cancel signal to terminate the request. When the request terminates, the session is deleted and the connection closed. Deleting the session causes any pending requests on that session's queue to be re-queued.

In the event the connection is currently inactive, the session is marked for deletion and re-queued to the beginning of the session timeout queue. The session and the connection are deleted in the next timeout cycle (normally within 5 seconds). Any session marked for deletion cannot be used by a new HTTP request.

All sessions are lost when the database is stopped.

Related Information

How to Create an HTTP Session [page 632]
How to Delete an HTTP Session or Change the Session ID [page 637]
sa_set_http_option System Procedure
NEXT_CONNECTION Function [System]

1.20.1.4.4.5 HTTP Session Error Codes

There are some common errors that occur during HTTP session management.

The `503 Service Unavailable` error occurs when a new request tries to access a session where more than 16 requests are pending on that session, or an error occurred while queuing the session.

The `403 Forbidden` error occurs when the client IP address or host name does not match that of the creator of the session.

A request stipulating a session that does not exist does not implicitly generate an error. It is up to the web application to detect this condition (by checking SessionID, SessionCreateTime, or SessionLastTime connection properties) and do the appropriate action.

Related Information

Web Service Error Code Reference [page 682]

1.20.1.4.5 Character Set Conversion Considerations

By default, character-set conversion is performed automatically on outgoing result sets consisting of text. Result sets of other types, such as binary objects, are not affected.

The character set of the request is converted to the HTTP web server character set, and the result set is converted to the client application character set. The server uses the first suitable character set listed in the request when multiple sets are listed.

Character-set conversion can be enabled or disabled by setting the HTTP option 'CharsetConversion' option of the sa_set_http_option system procedure.

The following example illustrates how to turn off automatic character-set conversion:

```
CALL sa_set_http_option('CharsetConversion', 'OFF');
```

You can use the 'AcceptCharset' option of the sa_set_http_option system procedure to specify the character-set encoding preference when character-set conversion is enabled.

The following example illustrates how to specify the web service character set encoding preference to ISO-8859-5, if supported; otherwise, set it to UTF-8:

```
CALL sa_set_http_option('AcceptCharset', 'iso-8859-5, utf-8');
```

Character sets are prioritized by server preference but the selection also considers the client's Accept-Charset criteria. The most favored character set according to the client that is also specified by this option is used.

Related Information

sa_set_http_option System Procedure

1.20.1.4.6 Cross Site Scripting Considerations

When developing your web application, ensure that it is not vulnerable to cross-site scripting (XSS). This type of vulnerability occurs when an attacker attempts to inject a script into your web page.

It is highly recommended that application developers and database administrators review their web application code for possible security vulnerabilities before it is put into production. The Open Web Application Security Project (https://www.owasp.org) contains more information about how to secure your web application.

Here is an example of a cross-site scripting vulnerability. If your web application supports page redirection, it should validate the URL before a redirect is performed. For example, if your web application supports a login function that returns the user to the original page, it should check that the page to which the user is being redirected does not take the user off-site. The following is an example of a malicious redirect.

```
https://www.mysite.com/login?referer=http://www.badsite.com/index.html
```

1.20.1.4.7 Web Services System Procedures

Use these system procedures when working with web services.

- sa_http_header_info system procedure
- sa_http_php_page system procedure
- sa_http_php_page_interpreted system procedure
- sa_http_variable_info system procedure
- sa_set_http_header system procedure
- sa_set_http_option system procedure
- sa_set_soap_header system procedure

Related Information

Web Services Functions
The Database Server as an HTTP Web Server [page 603]
-xs Database Server Option
sa_http_header_info System Procedure
sa_http_php_page System Procedure
sa_http_php_page_interpreted System Procedure
sa_http_variable_info System Procedure
sa_set_http_header System Procedure
sa_set_http_option System Procedure
sa_set_soap_header System Procedure

1.20.1.4.8 Web Services Functions

Web service functions assist the handling of HTTP and SOAP requests within web services.

Refer to the SQL Anywhere Server - SQL Reference documentation for a list of available functions.

There are also many system procedures available for web services.

Related Information

Web Services Functions
The Database Server as an HTTP Web Server [page 603]
-xs Database Server Option
Web Services System Procedures

1.20.1.4.9 Web Services Connection Properties

Web service connection properties are accessed using the CONNECTION_PROPERTY function.

Use the following syntax to store a connection property value from the HTTP server to a local variable in a SQL function or procedure.

```
SELECT CONNECTION_PROPERTY('connection-property-name') INTO variable_name;
```

The following is a list of useful runtime HTTP request connection properties that are commonly used for web service applications:

HttpServiceName

Returns the service name origin for a web application.

AuthType

Returns the type of authentication used when connecting.

ServerPort

Returns the database server's TCP/IP port number or 0.

ClientNodeAddress

Returns the node for the client in a client/server connection.

ServerNodeAddress

Returns the node for the server in a client/server connection.

BytesReceived

Returns the number of bytes received during client/server communications.

SessionID

Returns the session ID for the connection if it exists, otherwise, returns an empty string.

SessionCreateTime

Returns the time the HTTP session was created.

SessionLastTime

Returns the time of the last request for the HTTP session.

Related Information

List of Connection Properties

1.20.1.4.10 Web Services Options

Web service options control various aspects of HTTP server behavior.

Use the following syntax to set a public option in an HTTP server:

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

The following is a list of options that are commonly used in HTTP servers for application configuration:

http_connection_pool_basesize

Specifies the nominal threshold size of database connections.

http_connection_pool_timeout

Specifies the maximum duration that an unused connection can be retained in the connection pool.

http_session_timeout

Specifies the default timeout duration, in minutes, that the HTTP session persists during inactivity.

request_timeout

Controls the maximum time a single request can run.

webservice_namespace_host

Specifies the hostname to be used as the XML namespace within specification for DISH services.

Related Information

Database Options
http_connection_pool_basesize Option
http_connection_pool_timeout Option
http_session_timeout Option
request_timeout Option
webservice_namespace_host Option

1.20.1.5 How to Browse a SQL Anywhere HTTP Web Server

How your web services are named and designed dictates what constitutes a valid URL.

URLs uniquely specify resources such as HTML content available through HTTP or secured HTTPS requests. URL syntax permits access to the web services defined by your HTTP web server.

Syntax

```
{http|https}://host-name[:port-number][/dbn]/service-name[/path-name|?url-query]
```

Parameters

host-name and port-number

Specifies the location of the web server and, optionally, the port number if it is not defined as the default HTTP or HTTPS port numbers. The `host-name` can be the IP address of the computer running the web server. The `port-number` must match the port number used when you started the web server.

dbn

Specifies the name of a database. This database must be running on the web server and contain web services.

You do not need to specify `dbn` if the web server is running only one database or if the database name was specified for the given HTTP/HTTPS listener of the protocol option.

service-name

Specifies the name of the web service to access. This web service must exist in the database specified by `dbn`. Slash characters (/) are permitted when you create or alter a web service, so you can use them as part of the `service-name`. The remainder of the URL is matched with the defined service.

The client request is processed if a `service-name` is not specified and the **`root`** web service is defined. A `404 Not Found` error is returned if the server cannot identify an applicable service to process the request. As a side-effect, if the **`root`** web service does exist and cannot process the request based on the URL criteria, then it is responsible for generating the `404 Not Found` error.

path-name

After resolving the service name, the remaining slash delimited path can be accessed by a web service procedure. If the service was created with URL ON, then the whole path is accessible using a designated URL HTTP variable. If the service was created with URL ELEMENTS, then each path element can be accessed using designated HTTP variables URL1 to URL10.

Path element variables can be defined as host variables within the parameter declaration of the service statement definition. Alternatively, or additionally, HTTP variables can be accessed from within a stored procedure using the HTTP_VARIABLE function call.

The following example illustrates the SQL statement used to create a web service where the URL clause is set to ELEMENTS:

```
CREATE SERVICE TestWebService
    TYPE 'HTML'
    URL ELEMENTS
    AUTHORIZATION OFF
    USER DBA
    AS CALL TestProcedure ( :url1, :url2 );
```

This TestWebService web service calls a procedure that explicitly references the url1 and url2 host variables.

You can access this web service using the following URL, assuming that TestWebService is running on the database demo from localhost through the default port:

```
http://localhost/demo/TestWebService/Assignment1/Assignment2/Assignment3
```

This URL accesses TestWebService, which runs TestProcedure and assigns the Assignment1 value to url1, and the Assignment2 value to url2. Optionally, TestProcedure can access other path elements using the HTTP_VARIABLE function. For example, the HTTP_VARIABLE('url3') function call returns Assignment3.

url-query

An HTTP GET request may follow a path with a query component that specifies HTTP variables. Similarly, the body of a POST request using a standard application/x-www-form-urlencoded Content-Type can pass HTTP variables within the request body. In either case, HTTP variables are passed as name/value pairs where the variable name is delimited from its value with an equals signs. Variables are delimited with an ampersand.

HTTP variables can be explicitly declared as host variables within the parameter list of the service-statement, or accessed using the HTTP_VARIABLE function from within the stored procedure of the service statement.

For example, the following SQL statement creates a web service that requires two host variables. Host variables are identified with a colon (:) prefix.

```
CREATE SERVICE ShowSalesOrderDetail
    TYPE 'HTML'
    URL OFF
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

Assuming that the ShowSalesOrderDetail web service is available at the web server on localhost through the default port, you can access the web service using the following URL:

```
http://localhost/demo/ShowSalesOrderDetail?customer_id=101&product_id=300
```

This URL accesses ShowSalesOrderDetail and assigns a value of 101 to customer_id, and a value of 300 to product_id. The resultant output is displayed in your web browser in HTML format.

Remarks

Each web service provides its own set of web content. This content is typically generated by custom functions and procedures in your database, but content can also be generated with a URL that specifies a SQL statement. Alternatively, or in conjunction, you can define the **`root`** web service, which processes all HTTP requests that are not processed by a dedicated service. The **`root`** web service would typically inspect the request URL and headers to determine how to process the request.

The web browser prompts for user name and password when required to connect to the server. The browser then base64 encodes the user input within an Authorization request header and resends the request.

If your web service URL clause is set to ON or ELEMENTS, the URL syntax properties of `path-name` and `url-query` can be used simultaneously so that the web service is accessible using one of several different formatting options. When using these syntax properties simultaneously, the `path-name` format must be used first followed by the `url-query` format.

In the following example, this SQL statement creates a web service where the URL clause is set to ON, which defines the url variable:

```
CREATE SERVICE ShowSalesOrderDetail
    TYPE 'HTML'
    URL ON
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowSalesOrderDetail( :product_id, :url );
```

The following is a sample list of acceptable URLs that assign a url value of 101 and a product_id value of 300:

- `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`

When a host variable name is assigned more than once in the context of `path-name` and `url-query`, the last assignment always takes precedence. For example, the following sample URLs assign a url value of 101 and a product_id value of 300:

- `http://localhost:80/demo/ShowSalesOrderDetail2/302?url=101&product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2/String?product_id=300&url=101`

> **Note**
>
> The information presented here applies to HTTP web services that use general HTTP web service types, such as RAW, XML, and HTML. You cannot use a browser to issue SOAP requests. JSON services return result sets for consumption by web service applications using AJAX.

Example

The following URL syntax is used to access a web service named gallery_image that is running in a database named demo on a local HTTP server through the default port, assuming that the gallery_image service is defined with URL ON:

```
http://localhost/demo/gallery_image/sunset.jpg
```

The URL appears to request a graphic file in a directory from a traditional web server, but it accesses the gallery_image service with `sunset.jpg` specified as an input parameter for an HTTP web server.

The following SQL statement illustrates how the gallery service could be defined on the HTTP server to accomplish this behavior:

```
CREATE SERVICE gallery_image
    TYPE 'RAW'
    URL ON
    AUTHORIZATION OFF
    USER DBA
    AS CALL gallery_image ( :url );
```

The gallery_image service calls a procedure with the same name, passing the client-supplied URL. For a sample implementation of a gallery_image procedure that can be accessed by this web service definition, see *%SQLANYSAMP17%*`\SQLAnywhere\HTTP\gallery.sql`.

Related Information

How to Access Client-supplied HTTP Variables and Headers [page 624]
How to Create and Customize a Root Web Service [page 617]
-xs Database Server Option
ServerPort (PORT) Protocol Option
DatabaseName (DBN) Protocol Option
CREATE SERVICE Statement [HTTP Web Service]
HTTP_VARIABLE Function [Web Service]

1.20.2 Access to Web Services Using Web Clients

The database server can be used as a web client to access web services hosted by a SQL Anywhere HTTP web server or third party web servers such as Apache or IIS.

In addition to using the database server as a web client, web services provide client applications with an alternative to traditional interfaces, such as JDBC and ODBC. They are easily deployed because additional components are not needed, and can be accessed from multi-platform client applications written in a variety of languages, including scripting languages, such as Perl and Python.

In this section:

Using the Database Server as a Web Client [page 647]
Run the database server as a web client application to connect to an HTTP server and access a general HTTP web service.

Accessing a SQL Anywhere HTTP Web Server [page 650]
SQL Anywhere HTTP web server access is illustrated using two different types of client application: Python and C#.

Web Client Application Development [page 652]
SQL Anywhere databases can act as web client applications to access web services hosted on a SQL Anywhere web server or on third party web servers.

HTTP and SOAP Request Structures [page 680]
All parameters to a function or procedure, unless used during parameter substitution, are passed as part of the web service request. The format in which they are passed depends on the type of the web service request.

How to Log Web Client Requests [page 681]
Web service client information, including HTTP requests and transport data, can be logged to the web service client log file.

1.20.2.1 Using the Database Server as a Web Client

Run the database server as a web client application to connect to an HTTP server and access a general HTTP web service.

Prerequisites

You can develop web client applications that connect to any type of online web server, but this topic assumes that you have started a local SQL Anywhere HTTP web server on port 8082 and want to connect to a web service named SampleHTMLService that was created with the following SQL statements:

```
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;
```

Context

A web client procedure cannot make an outbound HTTP request to itself; it cannot call a SQL Anywhere web service running on the same database. In the following steps, you create a new database, start a second database server, and connect to the new database.

Perform the following tasks to create a web client application:

Procedure

1. Run the following command to create a client database if one does not already exist:

   ```
   dbinit -dba DBA,passwd client-database-name
   ```

   Replace `client-database-name` with a new name for your client database.
2. Run the following command to start the client database:

   ```
   dbsrv17 client-database-name.db
   ```
3. Run the following command to connect to the client database through Interactive SQL:

   ```
   dbisql -c "UID=DBA;PWD=password;SERVER=client-database-name"
   ```
4. Create a new client procedure that connects to the SampleHTMLService web service using the following SQL statement:

   ```
   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. Execute the following SQL statement to call the client procedure and send an HTTP request to the web server:

   ```
   CALL client_post(3.14, 9, 's varchar', 'x varchar');
   ```

   The HTTP POST request created by client_post looks similar to the following output:

   ```
   POST /SampleHTMLService HTTP/1.0
   ASA-Id: ea1746b01cd0472eb4f0729948db60a2
   User-Agent: SATest
   Accept-Charset: windows-1252, UTF-8, *
   Date: Wed, 8 Jun 2016 21:55:01 GMT
   Host: localhost:8082
   Connection: close
   Content-Type: application/x-www-form-urlencoded; charset=windows-1252
   Content-Length: 58
   ```

```
&f=3.14000010490417 48&i=9&s=s%20varchar&x=x%20varchar
```

Results

The web service SampleHTMLService running on the web server extracts the parameter values for i, f, and s from the POST request and passes them as parameters to the sp_echo procedure. Parameter value x is ignored. The sp_echo procedure creates a result set which is returned to the web service. Agreement in parameter names between the client and the web server is essential for proper mapping.

The web service creates the response which is sent back to the client. The output displayed in Interactive SQL should be similar to the following output:

Attribute	Value	Instance
Status	HTTP/1.1 200 OK	1
Body	`<html><head><title>/SampleHTMLService</title></head><body><h3>/SampleHTMLService</h3><table border=1><tr class="header"><th><b>ret_i</b></th><th><b>ret_f</b></th><th><b>ret_s</b></th></tr><tr><td>9</td><td>3.140000104904174 8</td><td>s varchar</td>`	1
Date	Wed, 08 Jun 2016 21:55:01 GMT	1
Connection	close	1
Expires	Wed, 08 Jun 2016 21:55:01 GMT	1
Content-Type	text/html; charset=windows-1252	1
Server	SQLAnywhere/17.0.11.1293	1

Related Information

Quick Start to Using the Database Server as an HTTP Web Server [page 604]
How to Develop Web Service Applications in an HTTP Web Server [page 621]
CREATE PROCEDURE Statement [Web Service]
CREATE FUNCTION Statement [Web Service]

1.20.2.2 Accessing a SQL Anywhere HTTP Web Server

SQL Anywhere HTTP web server access is illustrated using two different types of client application: Python and C#.

Context

You can develop web client applications that connect to any type of online web server, but this topic assumes that you have started a local SQL Anywhere HTTP web server on port 8082 and want to connect to a web service named SampleXMLService, created with the following SQL statements:

```
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;
```

Perform the following tasks to access an XML web service using C# or Python:

Procedure

1. Create a procedure that connects to a web service on an HTTP server.

   Write code that accesses the SampleXMLService web service.

Option	Description

For C#, use the following 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") + " ");`

Option	Description

	``
	                        Console.Write(reader.GetAttribute("ret_s") +
	" ");
	
	Console.WriteLine(reader.GetAttribute("ret_f"));
	                    }
	                    break;
	                }
	            }
	            reader.Close();
	        }
	}
	``
	Save the code to a file named `DocHandler.cs`.
	To compile the program, run the following command at a command prompt:
	``
	csc /out:DocHandler.exe DocHandler.cs
	``

For Python, use the following 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')
	``
	Save the code to a file named `DocHandler.py`.

2. Perform operations on the result set sent by the HTTP server.

Option	Description
For C#, run the following command:	`DocHandler`
For Python, run the following command:	`python DocHandler.py`

Results

The application displays the following output:

```
5 hello 3.14
```

Related Information

How to Develop Web Service Applications in an HTTP Web Server [page 621]
Quick Start to Using the Database Server as an HTTP Web Server [page 604]

1.20.2.3 Web Client Application Development

SQL Anywhere databases can act as web client applications to access web services hosted on a SQL Anywhere web server or on third party web servers.

SQL Anywhere web client applications are created by writing stored procedures and functions using configuration clauses, such as the URL clause that specifies the web service target endpoint. Web client procedures do not have a body, but in every other way are used as any other stored procedure. When called, a web client procedure makes an outbound HTTP or SOAP request. A web client procedure is restricted from making an outbound HTTP request to itself; it cannot call a SQL Anywhere web service running on the same database.

For detailed examples of web service applications, see the *%SQLANYSAMP17%*`\SQLAnywhere\HTTP` directory.

In this section:

Web Client Function and Procedure Requirements and Recommendations [page 653]
Web service client procedures and functions require the definition of a URL clause to identify the web service endpoint. A web service client procedure or function has specialized clauses for configuration but is used like any other stored procedure or function in every other respect.

Variables Supplied to Web Services [page 665]
Variables can be supplied to a web service in various ways depending on the web service type.

Variables Accessed from Result Sets [page 669]
Web service client calls can be made with stored functions or procedures.

Substitution Parameters Used for Clause Values [page 679]
Declared parameters to a stored procedure or function are automatically substituted for placeholders within a clause definition each time the stored procedure or function is run.

Related Information

Web Client SQL Statements [page 665]

1.20.2.3.1 Web Client Function and Procedure Requirements and Recommendations

Web service client procedures and functions require the definition of a URL clause to identify the web service endpoint. A web service client procedure or function has specialized clauses for configuration but is used like any other stored procedure or function in every other respect.

You can use the CREATE PROCEDURE and CREATE FUNCTION statements to create web client functions and procedures to send SOAP or HTTP requests to a web server.

The following list outlines the requirements and recommendations for creating or altering web client functions and procedures. You can specify the following information when creating or altering a web client function or procedure:

- The URL clause, which requires an absolute URL specifying the web service endpoint. (Required)
- The TYPE clause to specify whether the request is HTTP or SOAP over HTTP. (Recommended)
- Ports that are accessible to the client application. (Optional)
- The HEADER clause to specify HTTP request headers. (Optional)
- The SOAPHEADER clause to specify SOAP header criteria within the SOAP request envelope. (Optional. For SOAP requests only)
- The namespace URI. (For SOAP requests only)

In this section:

Web Client URL Clause [page 654]
You must specify the location of the web service endpoint to make it accessible to your web client function or procedure. The URL clause of the CREATE PROCEDURE and CREATE FUNCTION statements provides the web service URL that you want to access.

Web Service Request Types [page 655]
You can specify the type of client requests to send to the web server when creating a web client function or procedure. The TYPE clause of the CREATE PROCEDURE and CREATE FUNCTION statements formats requests before sending them to the web server.

Web Client Ports [page 657]
It is sometimes necessary to indicate which ports to use when opening a server connection through a firewall. You can use the CLIENTPORT clause of the CREATE PROCEDURE and CREATE FUNCTION statements to designate port numbers on which the client application communicates using TCP/IP.

HTTP Request Header Management [page 657]
HTTP request headers can be added, changed, or removed with the HEADER clause of the CREATE PROCEDURE and CREATE FUNCTION statements.

SOAP Request Header Management [page 659]
A SOAP request header is an XML fragment within a SOAP envelope.

SOAP Namespace URI Requirement [page 663]
The namespace URI specifies the XML namespace used to compose the SOAP request envelope for the given SOAP operation. The domain component from a URL clause is used when the namespace URI is not defined.

Web Client SQL Statements [page 665]
SQL statements for creating procedures and functions are used to define the web service access mechanism.

1.20.2.3.1.1 Web Client URL Clause

You must specify the location of the web service endpoint to make it accessible to your web client function or procedure. The URL clause of the CREATE PROCEDURE and CREATE FUNCTION statements provides the web service URL that you want to access.

Specifying an HTTP Service URL

Specifying an HTTP scheme within the URL clause configures the procedure or function for non-secure communication using an HTTP protocol.

The following statement illustrates how to create a procedure that sends requests to a web service named SampleHTMLService that resides in a database named dbname hosted by an HTTP web server located at localhost on port 8082:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'http://localhost:8082/dbname/SampleHTMLService'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

The database name is only required if the HTTP server hosts more than one database. You can substitute localhost with the host name or the IP address of the HTTP server.

Specifying an HTTPS Service URL

Specifying an HTTPS scheme within the URL clause configures the procedure or function for secure communication over Transport Layer Security (TLS).

Your web client application must have access to an RSA server certificate or the certificate that signed the server certificate to submit a secure HTTPS request. The certificate is required for the client procedure to authenticate the server to prevent man-in-the-middle exploits.

Use the CERTIFICATE clause of the CREATE PROCEDURE and CREATE FUNCTION statements to authenticate the server and establish a secure data channel. You can either place the certificate in a file and provide the file name, or provide the entire certificate as a string value; you cannot do both.

The following statement demonstrates how to create a procedure that sends requests to a web service named SecureHTMLService that resides in a database named dbname in an HTTPS server located at localhost on the port 8082:

```
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
          17\\Samples\\Certificates\\rsaroot.crt'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

The CERTIFICATE clause in this example indicates that the RSA server certificate is located in the *C:\Users\Public\Documents\SQL Anywhere 17\Samples*`\Certificates\rsaroot.crt` file.

> **Note**
>
> Specifying HTTPS_FIPS forces the system to use the FIPS libraries. If HTTPS_FIPS is specified, but no FIPS libraries are present, non-FIPS libraries are used instead.

Specifying a Proxy Server URL

Some requests must be sent through a proxy server. Use the PROXY clause of the CREATE PROCEDURE and CREATE FUNCTION statements to specify the proxy server URL and redirect requests to that URL. The proxy server forwards the request to the final destination, obtains the response, and forwards the response back to SQL Anywhere.

Related Information

How to Start an HTTP Web Server [page 606]
Web Client SQL Statements [page 665]

1.20.2.3.1.2 Web Service Request Types

You can specify the type of client requests to send to the web server when creating a web client function or procedure. The TYPE clause of the CREATE PROCEDURE and CREATE FUNCTION statements formats requests before sending them to the web server.

Specifying an HTTP Request Format

Web client functions and procedures send HTTP requests when the specified format in the TYPE clause begins with an HTTP prefix.

For example, execute the following SQL statement in the web client database to create an HTTP procedure named PostOperation that sends HTTP requests to the specified URL:

```
CREATE PROCEDURE PostOperation(a INTEGER, b CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:POST';
```

In this example, requests are formatted as HTTP:POST requests, which would produce a request similar to the following:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/17.0.11.1293
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
```

Specifying a SOAP Request Format

Web client functions and procedures send HTTP requests when the specified format in the TYPE clause begins with a SOAP prefix.

For example, execute the following statement in the web client database to create a SOAP procedure named SoapOperation that sends SOAP requests to the specified URL:

```
CREATE PROCEDURE SoapOperation(intVariable INTEGER, charVariable CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleSoapService'
    TYPE 'SOAP:DOC';
```

In this example, a SOAP:DOC request is sent to the URL when you call this procedure, which would produce a request similar to the following:

```
POST /dbname/SampleSoapService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/17.0.11.1293
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>
```

The procedure name appears in the <m:SoapOperation> tag within the body. The two parameters to the procedure, intVariable and charVariable, become <m:intVariable> and <m:charVariable>, respectively.

By default, the stored procedure name is used as the SOAP operation name when building a SOAP request. Parameter names appear in SOAP envelope tagnames. You must reference these names correctly when defining a SOAP stored procedure since the server expects these names in the SOAP request. The SET clause can be used to specify an alternate SOAP operation name for the given procedure. For all but the simplest cases (for example, a SOAP RPC call returning a single string value), use function definitions rather than procedures. A SOAP function returns the full SOAP response envelope which can be parsed using OPENXML.

Related Information

Web Client SQL Statements [page 665]
Variables Supplied to Web Services [page 665]
HTTP and SOAP Request Structures [page 680]
OPENXML Operator

1.20.2.3.1.3 Web Client Ports

It is sometimes necessary to indicate which ports to use when opening a server connection through a firewall. You can use the CLIENTPORT clause of the CREATE PROCEDURE and CREATE FUNCTION statements to designate port numbers on which the client application communicates using TCP/IP.

Do not use this feature unless your firewall restricts access to a particular range of ports.

For example, execute the following SQL statement in the web client database to create a procedure named SomeOperation that sends requests to the specified URL using one of the ports in the range 5050-5060, or port 5070:

```
CREATE PROCEDURE SomeOperation()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    CLIENTPORT '5050-5060,5070';
```

Specify a range of port numbers when required. Only one connection is maintained at a time when you specify a single port number; the client application attempts to access all specified port numbers until it finds one to bind to. After closing the connection, a timeout period of several minutes is initiated so that no new connection can be made to the same server and port.

This feature is similar to setting the ClientPort network protocol option.

Related Information

Web Client SQL Statements [page 665]
ClientPort (CPORT) Protocol Option (Client Side Only)

1.20.2.3.1.4 HTTP Request Header Management

HTTP request headers can be added, changed, or removed with the HEADER clause of the CREATE PROCEDURE and CREATE FUNCTION statements.

You suppress an HTTP request header by referencing the name. You add or change an HTTP request header value by placing a colon after the header name following by the value. Header value specifications are optional.

For example, execute the following SQL statement in the web client database to create a procedure named SomeOperation2 that sends requests to the specified URL that puts restrictions on HTTP request headers:

```
CREATE PROCEDURE SomeOperation2()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:GET'
    HEADER 'SOAPAction\nDate\nFrom:\nCustomAlias:John Doe';
```

In this example, the Date header, which is automatically generated, is suppressed. The From header is included but is not assigned a value. A new header named CustomAlias is included in the HTTP request and is assigned the value of John Doe. The GET request looks similar to the following:

```
GET /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/17.0.11.1293
Accept-Charset: windows-1252, UTF-8, *
From:
Host: localhost:8082
Connection: close
CustomAlias: John Doe
```

Folding of long header values is supported, provided that one or more white spaces immediately follow the \n.

The following example illustrates long header value support:

```
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';
```

The POST request looks similar to the following:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/17.0.11.1293
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
```

> **Note**
>
> You must set the SOAPAction HTTP request header to the given SOAP service URI as specified in the WSDL when creating a SOAP function or procedure.

Automatically Generated HTTP Request Headers

Modifying automatically generated headers can have unexpected results. The following HTTP request headers should not be modified without precaution:

HTTP Header	Description
Accept-Charset	Always automatically generated. Changing or deleting this header may result in unexpected data conversion errors.
ASA-Id	Always automatically generated. This header ensures that the client application does not connect to itself to prevent deadlock.
Authorization	Automatically generated when URL contains credentials. Changing or deleting this header may result in failure of the request. Only BASIC authorization is supported. User and password information should only be included when connecting via HTTPS.
Connection	Connection: close, is always automatically generated. Client applications do not support persistent connections. The connection could hang if changed.
Host	Always automatically generated. HTTP/1.1 servers are required to respond with 400 Bad Request if an HTTP/1.1 client does not provide a Host header.
Transfer-Encoding	Automatically generated when posting a request in chunk mode. Removing this header or deleting the chunked value will result in failure when the client is using CHUNK mode.
Content-Length	Automatically generated when posting a request and not in chunk mode. This header is required to tell the server the content length of the body. If the content length is wrong the connection may hang or data loss could occur.

Related Information

Web Client SQL Statements [page 665]

1.20.2.3.1.5 SOAP Request Header Management

A SOAP request header is an XML fragment within a SOAP envelope.

While the SOAP operation and its parameters can be thought of as an RPC (Remote Procedure Call), a SOAP request header can be used to transfer meta information within a specific request or response. SOAP request headers transport application metadata such as authorization or session criteria.

The value of a SOAPHEADER clause must be a valid XML fragment that conforms to a SOAP request header entry. Multiple SOAP request header entries can be specified. The stored procedure or function automatically injects the SOAP request header entries within a SOAP header element (SOAP-ENV:Header). SOAPHEADER

values specify SOAP headers that can be declared as a static constant, or dynamically set using the parameter substitution mechanism. The following is a fragment from a sample SOAP request. It contains two XML headers called Authentication and Session respectively.

```
<?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>
```

Processing SOAP response headers (returned by the SOAP call) differs for functions and procedures. When using a function, which is the most flexible and recommended approach, the entire SOAP response envelope is received. The response envelope can then be processed using the OPENXML operator to extract SOAP header and SOAP body data. When using a procedure, SOAP response headers can only be extracted through the use of a substitution parameter that maps to an IN or INOUT variable. A SOAP procedure allows for a maximum of one IN or INOUT parameter.

A web service function must parse the response SOAP envelope to obtain the header entries.

Example

The following examples illustrate how to create SOAP procedures and functions that send parameters and SOAP headers. Wrapper procedures are used to populate the web service procedure calls and process the responses. The soapAddItemProc procedure illustrates the use of a SOAP web service procedure, the soapAddItemFunc function illustrates the use of a SOAP web service function, and the httpAddItemFunc function illustrates how a SOAP payload may be passed to an HTTP web service procedure.

The following example illustrates a SOAP client procedure that uses substitution parameters to send SOAP headers. A single INOUT parameter is used to receive SOAP headers. A wrapper stored procedure addItemProcWrapper that calls soapAddItemProc demonstrates how to send and receive soap headers including parameters.

```
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' );
```

The following example illustrates a SOAP client function that uses substitution parameters to send SOAP headers. An entire SOAP response envelope is returned. SOAP headers can be parsed using the OPENXML operator. A wrapper function addItemFuncWrapper that calls soapAddItemFunc demonstrates how to send and receive soap headers including parameters. It also shows how to process the response using the OPENXML operator.

```
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:sap-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' );
```

The following example demonstrates how an HTTP:POST can be used as a transport for an entire SOAP payload. Rather than creating a webservice client SOAP procedure, this approach creates a webservice HTTP procedure that transports the SOAP payload. A wrapper procedure addItemHttpWrapper calls httpAddItemFunc to demonstrate the use of the POST function. It shows how to send and receive soap headers including parameters and how to accept the response.

```
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' );
```

Limitations

- Server side SOAP services cannot currently define input and output SOAP header requirements. Therefore SOAP header metadata is not available in the WSDL output of a DISH service. A SOAP client toolkit cannot automatically generate SOAP header interfaces for a SOAP service endpoint.
- Soap header faults are not supported.

Related Information

How to Access Client-supplied SOAP Request Headers [page 628]

1.20.2.3.1.6 SOAP Namespace URI Requirement

The namespace URI specifies the XML namespace used to compose the SOAP request envelope for the given SOAP operation. The domain component from a URL clause is used when the namespace URI is not defined.

The server-side SOAP processor uses this URI to understand the names of the various entities in the message body of the request. The NAMESPACE clause of the CREATE PROCEDURE and CREATE FUNCTION statements specifies the namespace URI.

You may be required to specify a namespace URI before procedure calls succeed. This information is usually explained the public web server documentation, but you can obtain the required namespace URI from the WSDL available from the web server. You can generate a WSDL by accessing the DISH service if you are trying to communicate with a SQL Anywhere HTTP web server.

Generally, the NAMESPACE can be copied from the targetNamespace attribute specified at the beginning of the WSDL document within the wsdl:definition element. Be careful when including any trailing '/', as they are significant. Secondly, check for a soapAction attribute for the given SOAP operation. It should correspond to the SOAPAction HTTP header that would be generated as explained in the following paragraphs.

The NAMESPACE clause fulfills two functions. It specifies the namespace for the body of the SOAP envelope, and, if the procedure has TYPE 'SOAP:DOC' specified, it is used as the domain component of the SOAPAction HTTP header.

The following example illustrates the use of the NAMESPACE clause:

```
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/'
```

Execute the following SQL statement in Interactive SQL:

```
SELECT an_operation('a_value');
```

The statement generates a SOAP request similar to the following output:

```
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>
```

The namespace for the prefix 'm' is set to http://wsdl.domain.com/ and the SOAPAction HTTP header specifies a fully qualified URL for the SOAP operation.

The trailing slash is not a requirement for correct operation of SQL Anywhere but it can cause a response failure that is difficult to diagnose. The SOAPAction HTTP header is correctly generated regardless of the trailing slash.

When a NAMESPACE is not specified, the domain component from the URL clause is used as the namespace for the SOAP body, and if the procedure is of TYPE 'SOAP:DOC', it is used to generate the HTTP SOAPAction HTTP header. If in the above example the NAMESPACE clause is omitted, then http://wsdl.domain.com is used as the namespace. The subtle difference is that a trailing slash '/' is not present. Every other aspect of the SOAP request, including the SOAPAction HTTP header would be identical to the above example.

The NAMESPACE clause is used to specify the namespace for the SOAP body described for the SOAP:DOC case above. However, the SOAPAction HTTP header is generated with an empty value: SOAPAction: ""

When using the SOAP:DOC request type, the namespace is also used to compose the SOAPAction HTTP header.

Related Information

Web Service Request Types [page 655]
Web Client SQL Statements [page 665]
How to Create DISH Services [page 615]

1.20.2.3.1.7 Web Client SQL Statements

SQL statements for creating procedures and functions are used to define the web service access mechanism.

The following SQL statements are available to assist with web client development:

Web Client Related SQL Statements	Description
CREATE FUNCTION	Creates a web client function that makes an HTTP or SOAP over HTTP request.
ALTER FUNCTION	Modifies a function.
CREATE PROCEDURE	Creates a web client procedure that makes HTTP or SOAP requests to an HTTP server.
ALTER PROCEDURE	Modifies a procedure.

Related Information

CREATE FUNCTION Statement [Web Service]
ALTER FUNCTION Statement
CREATE PROCEDURE Statement [Web Service]
ALTER PROCEDURE Statement

1.20.2.3.2 Variables Supplied to Web Services

Variables can be supplied to a web service in various ways depending on the web service type.

Web client applications can supply variables to general HTTP web services using any of the following approaches:

- The suffix of the URL
- The body of an HTTP request

Variables can be supplied to the SOAP service type by including them as part of a standard SOAP envelope.

In this section:

Variables Supplied in the URLs to Web Services [page 666]
The HTTP web server can manage variables supplied in the URL by web browsers.

Variables Supplied in the Body of HTTP Requests [page 666]
You can supply variables in the body of an HTTP request by specifying HTTP:POST in the TYPE clause in a web client function or procedure.

Variables Supplied in SOAP Envelopes [page 667]
You can supply variables in a SOAP envelope using the SET SOAP option of a web client function or procedure to set a SOAP operation.

Related Information

Web Service Types [page 610]

1.20.2.3.2.1 Variables Supplied in the URLs to Web Services

The HTTP web server can manage variables supplied in the URL by web browsers.

These variables can be expressed in any of the following conventions:

- Appending them to the end of the URL while dividing each parameter value with a slash (/), such as in the following example:

```
http://localhost/database-name/param1/param2/param3
```

- Defining them explicitly in a URL parameter list, such as in the following example:

```
http://localhost/database-name/?arg1=param1&arg2=param2&arg3=param3
```

- A combination of appending them to the URL and defining them in a parameter list, such as in the following example:

```
http://localhost/database-name/param4/param5?
arg1=param1&arg2=param2&arg3=param3
```

The web server interpretation of the URL depends on how the web service URL clause is specified.

Related Information

How to Access Client-supplied HTTP Variables and Headers [page 624]
How to Create and Customize a Root Web Service [page 617]
How to Browse a SQL Anywhere HTTP Web Server [page 643]
CREATE SERVICE Statement [HTTP Web Service]

1.20.2.3.2.2 Variables Supplied in the Body of HTTP Requests

You can supply variables in the body of an HTTP request by specifying HTTP:POST in the TYPE clause in a web client function or procedure.

By default TYPE HTTP:POST uses application/x-www-form-urlencoded mime type. All parameters are urlencoded and passed within the body of the request. Optionally, if a media type is provided, the request Content-Type header is automatically adjusted to the provided media type and a single parameter value is uploaded within the body of the request.

Example

The following example assumes that a web service named XMLService exists on a localhost web server. Set up a SQL Anywhere client database, connect to it through Interactive SQL, and execute the following SQL statement:

```
CREATE PROCEDURE SendXMLContent(xmlcode LONG VARCHAR)
    URL 'http://localhost/XMLService'
    TYPE 'HTTP:POST:text/xml';
```

The statement creates a procedure that allows you to send a variable in the body of an HTTP request in text/xml format.

Execute the following SQL statement in Interactive SQL to send an HTTP request to the XMLService web service:

```
CALL SendXMLContent('<title>Hello World!</title>');
```

The procedure call assigns a value to the xmlcode parameter and sends it to web service.

Related Information

How to Access Client-supplied HTTP Variables and Headers [page 624]
CREATE PROCEDURE Statement [Web Service]
CREATE FUNCTION Statement [Web Service]

1.20.2.3.2.3 Variables Supplied in SOAP Envelopes

You can supply variables in a SOAP envelope using the SET SOAP option of a web client function or procedure to set a SOAP operation.

The following code illustrates how to set a SOAP operation in a web client function:

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC';
```

In this example, the addItems is the SOAP operation that contains the **amount** and **item** values, which are passed as parameters to the soapAddItemFunc function.

You can send a request by running the following sample script:

```
SELECT soapAddItemFunc(5, 'shirt');
```

A call to the soapAddItemFunc function call generates a SOAP envelope that looks similar to the following:

```
<?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>
```

As an alternative to the previous approach, you can create your own SOAP payload and send it to the server in an HTTP wrapper.

Variables to SOAP services must be included as part of a standard SOAP request. Values supplied using other methods are ignored.

The following code illustrates how to create an HTTP wrapper procedure that builds a customized SOAP envelope:

```
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;
```

The following code illustrates the web client function used to send the request:

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    TYPE 'HTTP:POST:text/xml'
    HEADER 'SOAPAction: "http://localhost:8082/addItems"';
```

You can send a request by running the following sample script:

```
CALL addItemHttpWrapper( 7, 'socks' );
```

1.20.2.3.3 Variables Accessed from Result Sets

Web service client calls can be made with stored functions or procedures.

If made from a function, the return type must be of a character data type, such as CHAR, VARCHAR, or LONG VARCHAR. The body of the HTTP response is the returned value. No header information is included. Additional information about the request, including the HTTP status information, is returned by procedures. So, procedures are preferred when access to additional information is desired.

SOAP Procedures

The response from a SOAP function is an XML document that contains the SOAP response.

SOAP responses are structured XML documents, so SQL Anywhere, by default, attempts to exploit this information and construct a more useful result set. Each of the top-level tags within the returned response document is extracted and used as a column name. The contents below each of these tags in the subtree is used as the row value for that column.

For example, SQL Anywhere would construct the shown data set given the following SOAP response:

```
<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 this example, the response document is delimited by the <ElizaResponse> tags that appear within the <SOAP-ENV:Body> tags.

Result sets have as many columns as there are top-level tags. This result set only has one column because there is only one top-level tag in the SOAP response. This single top-level tag, Eliza, becomes the name of the column.

XML Processing Facilities

Information within XML result sets, including SOAP responses, can be accessed using the OPENXML procedure.

The following example uses the OPENXML procedure to extract portions of a SOAP response. This example uses a web service to expose the contents of the SYSWEBSERVICE table as a SOAP service:

```
CREATE SERVICE get_webservices
    TYPE 'SOAP'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT * FROM SYSWEBSERVICE;
```

The following web client function, which must be created in a second SQL Anywhere database, issues a call to this web service. The return value of this function is the entire SOAP response document. The response is in the .NET DataSet format because DNET is the default SOAP service format.

```
CREATE FUNCTION get_webservices()
    RETURNS LONG VARCHAR
    URL 'HTTP://localhost/get_webservices'
    TYPE 'SOAP:DOC';
```

The following statement illustrates how you can use the OPENXML procedure to extract two columns of the result set. The service_name and secure_required columns indicate which SOAP services are secure and where HTTPS is required.

```
SELECT *
FROM OPENXML( get_webservices(), '//row' )
WITH ("Name"    CHAR(128) 'service_name',
      "Secure?" CHAR(1)   'secure_required' );
```

This statement works by selecting the decedents of the **row** node. The WITH clause constructs the result set based on the two elements of interest. Assuming only the get_webservices web service exists, this function returns the following result set:

Name	Secure?
get_webservices	N

In this section:

Result Set Retrieval from a Web Service [page 671]
Web service procedures of type HTTP return all the information about a response in a three-column result set. This result set includes the response status, header information, and body.

SOAP Data Types [page 672]
By default, the XML encoding of parameter input is string and the result set output for SOAP service formats contains no information that specifically describes the data type of the columns in the result set. For all formats, parameter data types are string.

SOAP Structured Data Types [page 678]
The XML data type is supported for use as return values and parameters within web service functions and procedures.

Related Information

XML in the Database

1.20.2.3.3.1 Result Set Retrieval from a Web Service

Web service procedures of type HTTP return all the information about a response in a three-column result set. This result set includes the response status, header information, and body.

The first column is named Attribute and is of data type LONG VARCHAR. The second column is named Value and is also of data type LONG VARCHAR. The third column is named Instance and is of data type INTEGER.

The result set has one row for each of the response header fields, a row for the HTTP status line (Status attribute), and a row for the response body (Body attribute).

Web procedures that specify multiple headers with the same name in the HEADERS clause send all the headers to the web server.

The following example represents a result set for a web service procedure that sets multiple Set-Cookie header fields:

Attribute	Value	Instance
Status	HTTP /1.0 200 OK	1
Body	<!DOCTYPE HTML ... ><HTML> ... </HTML>	1
Content-Type	text/html	1
Server	GWS/2.1	1
Content-Length	2234	1
Date	Mon, 18 Oct 2004, 16:00:00 GMT	1
Set-cookie	choice=chocolate; ...	1
Set-cookie	account=services...	2

Example

Create the following web service stored procedure to use as an example.

```
CREATE OR REPLACE PROCEDURE SAPWebPage()
URL 'http://www.sap.com/index.html'
TYPE 'HTTP';
```

If there is a proxy server between the database server and the web server you are trying to connect to, then you might have to add a PROXY clause to the procedure definition. The following is an example for a proxy server named `proxy`:

```
PROXY 'http://proxy:8080'
```

Execute the following SELECT query to obtain the response from the web service as a result set:

```
SELECT * FROM SAPWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR, Instance INT);
```

Because the web service procedure does not describe the shape of the result set, the WITH clause must define a temporary view.

The results of a query can be stored in a table. Execute the following SQL statement to create a table to contain the values of the result set:

```
CREATE TABLE StoredResults(
     Attribute LONG VARCHAR,
     Value     LONG VARCHAR,
     Instance  INT
);
```

Insert the result set into the StoredResults table:

```
INSERT INTO StoredResults
    SELECT * FROM SAPWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR, Instance INT);
```

Add clauses to modify your SELECT statement. For example, if you want only a specific row of the result set, then you can add a WHERE clause to limit the results of the SELECT to only one row.

```
SELECT * FROM SAPWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR, Instance INT)
    WHERE Attribute = 'Status';
```

This SELECT statement retrieves only the status information from the result set. It can be used to verify that the call was successful.

Related Information

Web Client SQL Statements [page 665]

1.20.2.3.3.2 SOAP Data Types

By default, the XML encoding of parameter input is string and the result set output for SOAP service formats contains no information that specifically describes the data type of the columns in the result set. For all formats, parameter data types are string.

For the DNET format, within the schema section of the response, all columns are typed as string. CONCRETE and XML formats contain no data type information in the response. This default behavior can be manipulated using the DATATYPE clause.

SQL Anywhere enables data typing using the DATATYPE clause. Data type information can be included in the XML encoding of parameter input and result set output or responses for all SOAP service formats. This simplifies parameter passing from SOAP toolkits by not requiring client code to explicitly convert parameters

to Strings. For example, an integer can be passed as an int. XML encoded data types enable a SOAP toolkit to parse and cast the data to the appropriate type.

When using string data types exclusively, the application needs to implicitly know the data type for every column within the result set. This is not necessary when data typing is requested of the web server. To control whether data type information is included, the DATATYPE clause can be used when the web service is defined.

Here is an example of a web service definition that enlists data typing for the result set response.

```
CREATE SERVICE "SASoapTest/EmployeeList"
    TYPE 'SOAP'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA
    DATATYPE OUT
    AS SELECT * FROM Employees;
```

In this example, data type information is requested for result set responses only since this service does not have parameters.

Data typing is applicable to all web services defined as type 'SOAP'.

Data Typing of Input Parameters

Data typing of input parameters is supported by simply exposing the parameter data types as their true data types in the WSDL generated by the DISH service.

A typical string parameter definition (or a non-typed parameter) would look like the following:

```
<s:element minOccurs="0" maxOccurs="1" name="a_varchar" nillable="true"
type="s:string" />
```

The String parameter may be nillable, that is, it may or may not occur.

For a typed parameter such as an integer, the parameter must occur and is not nillable. The following is an example.

```
<s:element minOccurs="1" maxOccurs="1" name="an_int" nillable="false"
type="s:int" />
```

Data Typing of Output Parameters

All web services of type 'SOAP' may expose data type information within the response data. The data types are exposed as attributes within the rowset column element.

The following is an example of a typed SimpleDataSet response from a SOAP FORMAT 'CONCRETE' web service.

```
<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">.55555555555555582</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>
```

The following is an example of a response from a SOAP FORMAT 'XML' web service returning the XML data as a string. The interior rowset consists of encoded XML and is presented here in its decoded form for legibility.

```
<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">.55555555555555582</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>
```

In addition to the data type information, the namespace for the elements and the XML schema provides all the information necessary for post processing by an XML parser. When no data type information exists in the result set (DATATYPE OFF or IN) then the xsi:type and the XML schema namespace declarations are omitted.

An example of a SOAP FORMAT 'DNET' web service returning a typed SimpleDataSet follows:

```
<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:schema>
    <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>.555555555555555582</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_outResult>
  <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>
```

Mapping SQL Anywhere Types to XML Schema Types

SQL Anywhere Type	XML Schema Type	XML Example
CHAR	string	Hello World
VARCHAR	string	Hello World
LONG VARCHAR	string	Hello World
TEXT	string	Hello World
NCHAR	string	Hello World
NVARCHAR	string	Hello World
LONG NVARCHAR	string	Hello World
NTEXT	string	Hello World

SQL Anywhere Type	XML Schema Type	XML Example
UNIQUEIDENTIFIER	string	12345678-1234-5678-9012-123456789012
UNIQUEIDENTIFIERSTR	string	12345678-1234-5678-9012-123456789012
XML	This is user defined. A parameter is assumed to be valid XML representing a complex type (for example, base64Binary, SOAP array, struct).	<inputHexBinary xsi:type="xsd:hexBinary"> 414243 </inputHexBinary> (interpreted as 'ABC')
BIGINT	long	-9223372036854775807
UNSIGNED BIGINT	unsignedLong	18446744073709551615
BIT	boolean	1
VARBIT	string	11111111
LONG VARBIT	string	00000000000000001000000000000000
DECIMAL	decimal	-1.756000
DOUBLE	double	.555555555555555582
FLOAT	float	12.3456792831420898
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==

When one or more parameters are of type NCHAR, NVARCHAR, LONG NVARCHAR, or NTEXT then the response output is in UTF8. If the client database uses the UTF-8 character encoding, there is no change in behavior (since NCHAR and CHAR data types are the same). However, if the database does not use the UTF-8 character encoding, then all parameters that are not an NCHAR data type are converted to UTF8. The value of the XML declaration encoding and Content-Type HTTP header will correspond to the character encoding used.

Mapping XML Schema Types to Java Types

XML Schema Type	Java Data Type
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
xsd:NOTATION	javax.xml.namespace.QName

1.20.2.3.3.3 SOAP Structured Data Types

The XML data type is supported for use as return values and parameters within web service functions and procedures.

XML Return Values

The database server as a web service client may interface to a web service using a function or a procedure.

A string representation within a result set may suffice for simple return data types. The use of a stored procedure may be warranted in this case.

The use of web service functions are a better choice when returning complex data such as arrays or structures. For function declarations, the RETURN clause can specify an XML data type. The returned XML can be parsed using OPENXML to extract the elements of interest.

A return of XML data such as dateTime is rendered within the result set verbatim. For example, if a TIMESTAMP column was included within a result set, it would be formatted as an XML dateTime string (2006-12-25T12:00:00.000-05:00) not as a string (2006-12-25 12:00:00.000).

XML Parameter Values

The XML data type is supported for use as a parameter within web service functions and procedures. For simple types, the parameter element is automatically constructed when generating the SOAP request body. However, for XML parameter types, this cannot be done since the XML representation of the element may require attributes that provide additional data. Therefore, when generating the XML for a parameter whose data type is XML, the root element name must correspond to the parameter name.

```
<inputHexBinary xsi:type="xsd:hexBinary">414243</inputHexBinary>
```

The XML type demonstrates how to send a parameter as a hexBinary XML type. The SOAP endpoint expects that the parameter name (or in XML terms, the root element name) is "inputHexBinary".

Cookbook Constants

Knowledge of how SQL Anywhere references namespaces is required to construct complex structures and arrays. The prefixes listed here correspond to the namespace declarations generated for a SOAP request envelope.

XML Prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance

XML Prefix	Namespace
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
m	namespace as defined in the NAMESPACE clause

1.20.2.3.4 Substitution Parameters Used for Clause Values

Declared parameters to a stored procedure or function are automatically substituted for placeholders within a clause definition each time the stored procedure or function is run.

Substitution parameters allow the creation of general web service procedures that dynamically configure clauses at run time. Any substrings that contain an exclamation mark '!' followed by the name of one of the declared parameters is replaced by that parameter's value. In this way one or more parameter values may be substituted to derive one or more clause values at runtime.

Parameter substitution requires adherence to the following rules:

- All parameters used for substitution must be alphanumeric. Underscores are not allowed.
- A substitution parameter must be followed immediately by a non-alphanumeric character or termination. For example, !sizeXL is not substituted with the value of a parameter named size because X is alphanumeric.
- A substitution parameter that is not matched to a parameter name is ignored.
- An exclamation mark (!) can be escaped with another exclamation mark.

For example, the following procedure illustrates the use of parameter substitution. URL and HTTP header definitions must be passed as parameters.

```
CREATE PROCEDURE test(uid CHAR(128), pwd CHAR(128), headers LONG VARCHAR)
    URL 'http://!uid:!pwd@localhost/myservice'
    HEADER '!headers';
```

You can then use the following statement to call the **test** procedure and initiate an HTTP request:

```
CALL test('dba', 'sql', 'NewHeader1:value1\nNewHeader2:value2');
```

Different values can be used each time this procedure is called.

Encryption Certificate Example

You can use parameter substitution to pass encryption certificates from a file and pass them to a stored procedure or stored function.

The following example illustrates how to pass a certificate as a substitution string:

```
CREATE PROCEDURE secure(cert LONG VARCHAR)
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Root';
```

The certificate is read from a file and passed to **secure** in the following call.

```
CALL secure( xp_read_file('C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt') );
```

This example is for illustration only. The certificate can be read directly from a file using the *file=* keyword for the CERTIFICATE clause.

No Matching Parameter Name Example

Placeholders with no matching parameter name are automatically deleted.

For example, the parameter size would not be substituted for the placeholder in the following procedure:

```
CREATE PROCEDURE orderitem (size CHAR(18))
    URL 'HTTP://localhost/salesserver/order?size=!sizeXL'
    TYPE 'SOAP:RPC';
```

In this example, !sizeXL is always deleted because it is a placeholder for which there is no matching parameter.

Parameters can be used to replace placeholders within the body of the stored function or stored procedure at the time the function or procedure is called. If placeholders for a particular variable do not exist, the parameter and its value are passed as part of the request. Parameters and values used for substitution in this manner are not passed as part of the request.

1.20.2.4 HTTP and SOAP Request Structures

All parameters to a function or procedure, unless used during parameter substitution, are passed as part of the web service request. The format in which they are passed depends on the type of the web service request.

Parameter values that are not of character or binary data types are converted to a string representation before being added to the request. This process is equivalent to casting the value to a character type. The conversion is done in accordance with the data type formatting option settings at the time the function or procedure is invoked. In particular, the conversion can be affected by such options as precision, scale, and timestamp_format.

HTTP Request Structures

Parameters for type HTTP:GET are URL encoded and placed within the URL. Parameter names are used verbatim as the name for HTTP variables. For example, the following procedure declares two parameters:

```
CREATE PROCEDURE test(a INTEGER, b CHAR(128))
    URL 'HTTP://localhost/myservice'
    TYPE 'HTTP:GET';
```

If this procedure is invoked with the two values 123 and 'xyz', then the URL used for the request is equivalent to that shown below:

```
HTTP://localhost/myservice?a=123&b=xyz
```

If the type is HTTP:POST, the parameters and their values are URL encoded and placed within the body of the request. After the headers, the following text appears in the body of the HTTP request for the two parameter and values:

```
a=123&b=xyz
```

SOAP Request Structures

Parameters passed to SOAP requests are bundled as part of the request body, as required by the SOAP specification:

```
<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>
```

Related Information

Substitution Parameters Used for Clause Values [page 679]

1.20.2.5 How to Log Web Client Requests

Web service client information, including HTTP requests and transport data, can be logged to the web service client log file.

The web service client log file can be specified with the -zoc server option or by using the sa_server_option system procedure.

```
CALL sa_server_option( 'WebClientLogFile', 'clientinfo.txt' );
```

Logging is enabled automatically when you specify the -zoc server option. You can enable and disable logging to this file using the sa_server_option system procedure.

```
CALL sa_server_option( 'WebClientLogging', 'ON' );
```

Related Information

-zoc Database Server Option
sa_server_option System Procedure

1.20.3 Web Service Error Code Reference

The HTTP server generates standard web service errors when requests fail. These errors are assigned numbers consistent with protocol standards.

The following are some typical errors that you may encounter:

Number	Name	SOAP Fault	Description
301	Moved permanently	Server	The requested page has been permanently moved. The server automatically redirects the request to the new location.
304	Not Modified	Server	The server has decided, based on information in the request, that the requested data has not been modified since the last request and so it does not need to be sent again.
307	Temporary Redirect	Server	The requested page has been moved, but this change may not be permanent. The server automatically redirects the request to the new location.
400	Bad Request	Client.BadRequest	The HTTP request is incomplete or malformed.
401	Authorization Required	Client.Authorization	Authorization is required to use the service, but a valid user name and password were not supplied.
403	Forbidden	Client.Forbidden	You do not have permission to access the database.
404	Not Found	Client.NotFound	The named database is not running on the server, or the named web service does not exist.
408	Request Timeout	Server.Timeout.	The maximum connection idle time was exceeded while receiving the request.

Number	Name	SOAP Fault	Description
410	Gone	Server	The requested resource is not available anymore.
411	HTTP Length Required	Client.LengthRequired	The server requires that the client include a Content-Length specification in the request. This typically occurs when uploading data to the server.
413	Entity Too Large	Server	The request exceeds the maximum permitted size.
414	URI Too Large	Server	The length of the URI exceeds the maximum allowed length.
500	Internal Server Error	Server	An internal error occurred. The request could not be processed.
501	Not Implemented	Server	The HTTP request method is not GET, HEAD, or POST.
502	Bad Gateway	Server	The document requested resides on a third-party server and the server received an error from the third-party server.
503	Service Unavailable	Server	The number of connections exceeds the allowed maximum.

Faults are returned to the client as SOAP faults as defined by the following the SOAP version 1.1 standards when a SOAP service fails:

- When an error in the application handling the request generates a SQLCODE, a SOAP Fault is returned with a faultcode of Client, possibly with a sub-category, such as Procedure. The faultstring element within the SOAP Fault is set to a detailed explanation of the error and a detail element contains the numeric SQLCODE value.
- In the event of a transport protocol error, the faultcode is set to either Client or Server, depending on the error, faultstring is set to the HTTP transport message, such as `404 Not Found`, and the detail element contains the numeric HTTP error value.
- SOAP Fault messages generated due to application errors that return a SQLCODE value are returned with an HTTP status of `200 OK`.

The appropriate HTTP error is returned in a generated HTML document if the client cannot be identified as a SOAP client.

1.20.4 HTTP Web Service Examples

Several sample implementations of web services are located in the *%SQLANYSAMP17%*`\SQLAnywhere\HTTP` directory.

For more information about the samples, see *%SQLANYSAMP17%*`\SQLAnywhere\HTTP\readme.txt`.

In this section:

Tutorial: Create a Web Server and Access it from a Web Client [page 684]
This tutorial illustrates how to create a web server and then send requests to it from a web client database server.

Tutorial: Using a Database Server to Access a SOAP/DISH Service [page 690]
This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

Tutorial: Using Microsoft Visual C# to Access a SOAP/DISH Web Service [page 700]
This tutorial illustrates how to create a Microsoft Visual C# client application to access SOAP/DISH services on a database server acting as a web server.

Tutorial: Using JAX-WS to Access a SOAP/DISH Web Service [page 708]
This tutorial illustrates how to create a Java API for XML Web Services (JAX-WS) client application to access SOAP/DISH services on a database server acting as a web server.

1.20.4.1 Tutorial: Create a Web Server and Access it from a Web Client

This tutorial illustrates how to create a web server and then send requests to it from a web client database server.

Prerequisites

You need the following:

- Familiarity with XML
- Familiarity with MIME (Multipurpose Internet Mail Extensions) types
- Basic knowledge of the web service features of the software

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Context

This tutorial shows you how to:

- Create and start a new web services database. The database server will act as the web server.
- Create a web service.
- Set up a procedure that returns the information contained in an HTTP request.
- Create and start a new web client database. The database server will act as the web client.
- Send an HTTP:POST request from the web client to the web server.
- Send an HTTP response from the web server to the web client.

1. Lesson 1: Setting Up a Web Server to Receive Requests and Send Responses [page 685]
   Set up a web server running a web service.
2. Lesson 2: Sending Requests from a Web Client and Receiving Responses [page 687]
   Set up a database client to send requests to a web server by using the POST method and to receive the web server's responses.

Related Information

The Database Server as an HTTP Web Server [page 603]

1.20.4.1.1 Lesson 1: Setting Up a Web Server to Receive Requests and Send Responses

Set up a web server running a web service.

Prerequisites

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Create a database that can be used to contain web service definitions.

   ```
   dbinit -dba DBA,passwd echo
   ```

2. Start a network database server using this database. This server will act as a web server.

   ```
   dbsrv17 -xs http(port=8082) -n echo echo.db
   ```

The HTTP web server is set to listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

3. Connect to the database server with Interactive SQL.

```
dbisql -c "UID=DBA;PWD=passwd;SERVER=echo"
```

4. Create a new web service to accept incoming requests.

```
CREATE SERVICE EchoService
TYPE 'RAW'
AUTHORIZATION OFF
SECURE OFF
USER DBA
AS CALL Echo();
```

This statement creates a new service named EchoService that calls a stored procedure named Echo when a web client sends a request to the service. It generates an HTTP response body without any formatting (RAW) for the web client. If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

5. Create the Echo procedure to handle incoming requests.

```
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
```

This procedure formats the Content-Type header and the body of the response that is sent to the web client.

Results

A web server is set up to receive requests and send responses.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Create a Web Server and Access it from a Web Client [page 684]

Next task: Lesson 2: Sending Requests from a Web Client and Receiving Responses [page 687]

Related Information

The Database Server as an HTTP Web Server [page 603]
CREATE SERVICE Statement [HTTP Web Service]
CREATE PROCEDURE Statement [Web Service]
CREATE FUNCTION Statement [Web Service]
HTTP_VARIABLE Function [Web Service]

1.20.4.1.2 Lesson 2: Sending Requests from a Web Client and Receiving Responses

Set up a database client to send requests to a web server by using the POST method and to receive the web server's responses.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This lesson contains several references to localhost. If you are not running the web client on the same computer as the web server, then use the host name or IP address of the web server from the previous lesson instead of localhost.

Procedure

1. Create a database that can be used to contain web client procedures.

   ```
   dbinit -dba DBA,passwd echo_client
   ```

2. Start the database on a network database server. This database server acts as a web client.

   ```
   dbsrv17 echo_client.db
   ```

3. Connect to the database server from Interactive SQL.

   ```
   dbisql -c "UID=DBA;PWD=passwd;SERVER=echo_client"
   ```

4. Create a new stored procedure to send requests to a web service.

```
CREATE OR REPLACE PROCEDURE SendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
TYPE 'HTTP:POST:!mimeType';
```

   The SendWithMimeType procedure has three parameters. The *value* parameter represents the body of the request that should be sent to the web service. The urlSpec parameter indicates the URL to use to connect to the web service. The mimeType indicates which MIME type to use for the HTTP:POST.

5. Send a request to the web server and obtain the response.

```
CALL SendWithMimeType('<hello>this is xml</hello>',
    'text/xml',
    'http://localhost:8082/EchoService'
);
```

   The http://localhost:8082/EchoService string indicates that the web server runs on localhost and listens on port 8082. The desired web service is named EchoService.

6. Try a different MIME type and observe the response.

```
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'
);
```

Results

A web client is set up to send HTTP requests to a web server by using the POST method and to receive the web server's response.

Example

The following is an example of a result set displayed by Interactive SQL:

Attribute	Value	Instance
Status	HTTP/1.1 200 OK	1
Body	<hello>this is xml</hello>	1
Server	SQLAnywhere/17.0.11.1234	1
Expires	Tue, 09 Oct 2012 21:06:01 GMT	1
Date	Tue, 09 Oct 2012 21:06:01 GMT	1

Attribute	Value	Instance
Content-Type	text/xml; charset=windows-1252	1
Connection	close	1

The following text is representative of the HTTP packet that is sent to the web server:

```
POST /EchoService HTTP/1.0
ASA-Id: 46758096650a44088c77237cc8719d5c
User-Agent: SQLAnywhere/17.0.11.1234
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>
```

The following text is the response from the web server:

```
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/17.0.11.1234
<hello>this is xml</hello>
```

Task overview: Tutorial: Create a Web Server and Access it from a Web Client [page 684]

Previous task: Lesson 1: Setting Up a Web Server to Receive Requests and Send Responses [page 685]

Related Information

CREATE PROCEDURE Statement [Web Service]
CREATE FUNCTION Statement [Web Service]

1.20.4.2 Tutorial: Using a Database Server to Access a SOAP/DISH Service

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

Prerequisites

You need the following:

- Familiarity with SOAP
- Basic knowledge of the web service features of the software

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE
- SERVER OPERATOR

Context

This tutorial shows you how to:

- Create and start a new web services database. The database server will act as the web server.
- Create a SOAP web service.
- Set up a procedure that converts a client-supplied Fahrenheit value to a Celsius value.
- Create and start a new web client database. The database server will act as the web client.
- Send a SOAP request from the web client to the web server.
- Send a SOAP response from the web server to the web client.

1. Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses [page 691]
   Set up a new database server and create a SOAP service to handle incoming SOAP requests.
2. Lesson 2: Setting Up a Web Client to Send SOAP Requests and Receive SOAP Responses [page 694]
   Set up a web client that sends SOAP requests and receives SOAP responses.
3. Lesson 3: Sending a SOAP Request and Receiving a SOAP Response [page 697]
   Call the wrapper procedure created in the previous lesson, which sends a SOAP request to the web server that you created earlier.

Related Information

The Database Server as an HTTP Web Server [page 603]

1.20.4.2.1 Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses

Set up a new database server and create a SOAP service to handle incoming SOAP requests.

Prerequisites

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The web server anticipates SOAP requests that provide a Fahrenheit temperature value that is converted to the equivalent Celsius degrees.

Procedure

1. Create a database that can be used to contain web service definitions.

   ```
   dbinit -dba DBA,passwd ftc
   ```

2. Start a database server using this database. This server will act as a web server.

   ```
   dbsrv17 -xs http(port=8082) -n ftc ftc.db
   ```

   The HTTP web server is set to listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

3. Connect to the database server with Interactive SQL.

   ```
   dbisql -c "UID=DBA;PWD=passwd;SERVER=ftc"
   ```

4. Create a new DISH service to accept incoming requests.

   ```
   CREATE SERVICE soap_endpoint
       TYPE 'DISH'
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA;
   ```

   This statement creates a new DISH service named soap_endpoint that handles incoming SOAP service requests. If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

5. Create a new SOAP service to handle Fahrenheit to Celsius conversions.

   ```
   CREATE SERVICE FtoCService
       TYPE 'SOAP'
   ```

```
    FORMAT 'XML'
    AUTHORIZATION OFF
    USER DBA
    AS CALL FToCConverter( :fahrenheit );
```

This statement creates a new SOAP service named FtoCService that generates XML-formatted strings as output. It calls a stored procedure named FToCConverter when a web client sends a SOAP request to the service. If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

6. Create the FToCConverter procedure to handle incoming SOAP requests. This procedure performs the necessary calculations to convert a client-supplied Fahrenheit temperature value to the equivalent Celsius temperature value.

```
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 header_loop;
    SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) AS answer;
END;
```

The NEXT_SOAP_HEADER function is used in a LOOP structure to iterate through all the header names in a SOAP request, and exits the loop when the NEXT_SOAP_HEADER function returns NULL.

> **Note**
>
> This function does not necessarily iterate through the headers in the order that they appear in the SOAP request.

The SOAP_HEADER function returns the header value or NULL when the header name does not exist. The FToCConverter procedure searches for a header named Authentication and extracts the header structure, including the @namespace and mustUnderstand attributes. The @namespace header attribute is a special attribute used to access the namespace (xmlns) of the given header entry.

The following is an XML string representation of a possible Authentication header structure, where the @namespace attribute has a value of "SecretAgent", and mustUnderstand has a value of 1:

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
    <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
    </userName>
</Authentication>
```

The OPENXML system procedure in the SELECT statement parses the XML header using the XPath string "/*:Authentication/*:userName" to extract the **alias** attribute value and the contents of the **first** and **last** tags. The result set is processed using a cursor to fetch the three column values.

At this point, you have all the information of interest that was passed to the web service. You have the temperature in Fahrenheit degrees and you have some additional attributes that were passed to the web service in a SOAP header. You could look up the name and alias that were provided to see if the person is authorized to use the web service. However, this exercise is not shown in the example.

The SET statement is used to build a SOAP response in XML format to send to the client. The following is an XML string representation of a possible SOAP response. It is based on the above Authentication header structure example.

```
<Authentication xmlns="SecretAgent" alias="99" mustUnderstand="1">
  <first>Susan</first>
  <last>Hilton</last>
</Authentication>
```

The SA_SET_SOAP_HEADER system procedure is used to set the SOAP response header that is sent to the client.

The final SELECT statement is used to convert the supplied Fahrenheit value to a Celsius value. This information is relayed back to the client.

Results

You now have a running web server that provides a service for converting temperatures from degrees Fahrenheit to degrees Celsius. This service processes a SOAP header from the client and sends a SOAP response back to the client.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using a Database Server to Access a SOAP/DISH Service [page 690]

Next task: Lesson 2: Setting Up a Web Client to Send SOAP Requests and Receive SOAP Responses [page 694]

Related Information

How to Create DISH Services [page 615]
SOAP Namespace URI Requirement [page 663]
CREATE SERVICE Statement [SOAP Web Service]
CREATE PROCEDURE Statement [Web Service]
CREATE FUNCTION Statement [Web Service]
SOAP_HEADER Function [SOAP]
NEXT_SOAP_HEADER Function [SOAP]
OPENXML Operator

1.20.4.2.2 Lesson 2: Setting Up a Web Client to Send SOAP Requests and Receive SOAP Responses

Set up a web client that sends SOAP requests and receives SOAP responses.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This lesson contains several references to localhost. Use the host name or IP address of the web server from lesson 1 instead of localhost if you are not running the web client on the same computer as the web server.

Procedure

1. Run the following command to create a web client database:

   ```
   dbinit -dba DBA,passwd ftc_client
   ```

2. Start the database client using the following command:

   ```
   dbsrv17 ftc_client.db
   ```

3. Connect to the database in Interactive SQL using the following command:

   ```
   dbisql -c "UID=DBA;PWD=passwd;SERVER=ftc_client"
   ```

4. Create a new stored procedure to send SOAP requests to a DISH service.

   Execute the following SQL statement in Interactive SQL:

   ```
   CREATE OR REPLACE PROCEDURE FtoC( fahrenheit FLOAT,
       INOUT inoutheader LONG VARCHAR,
       IN inheader LONG VARCHAR )
     URL 'http://localhost:8082/soap_endpoint'
     SET 'SOAP(OP=FtoCService)'
     TYPE 'SOAP:DOC'
     SOAPHEADER '!inoutheader!inheader';
   ```

   The http://localhost:8082/soap_endpoint string in the URL clause indicates that the web server runs on localhost and listens on port 8082. The desired DISH web service is named soap_endpoint, which serves as a SOAP endpoint.

   The SET clause specifies the name of the SOAP operation or service FtoCService that is to be called.

   The default format used when making a web service request is 'SOAP:RPC'. The format chosen in this example is 'SOAP:DOC', which is similar to 'SOAP:RPC' but allows for a richer set of data types. SOAP requests are always sent as XML documents. The mechanism for sending SOAP requests is 'HTTP:POST'.

   The substitution variables (inoutheader, inheader) in a web service client procedure like FtoC must be alpha-numeric. If the web service client is declared as a function, all its parameters are IN mode only (they cannot be assigned by the called function). Therefore, OPENXML or other string functions would have to be used to extract the SOAP response header information.

5. Create a wrapper procedure that builds two special SOAP request header entries, passes them to the FtoC procedure, and processes server responses.

   Execute the following SQL statements in Interactive SQL:

   ```
   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;
```

The first SET statement creates the XML representation of a SOAP header entry to inform the web server of user credentials:

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

The second SET statement creates the XML representation of a SOAP header entry to track the client session ID:

```
<Session xmlns="SomeSession">123456789</Session>
```

6. The OPEN statement causes the FtoC procedure to be called which sends a SOAP request to the web server and then processes the response from the web server. The response includes a header which is returned in inoutheader.

Results

At this point, you now have a client that can send SOAP requests to the web server and receive SOAP responses from the web server.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using a Database Server to Access a SOAP/DISH Service [page 690]

Previous task: Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses [page 691]

Next task: Lesson 3: Sending a SOAP Request and Receiving a SOAP Response [page 697]

Related Information

CREATE PROCEDURE Statement [Web Service]
CREATE FUNCTION Statement [Web Service]

1.20.4.2.3 Lesson 3: Sending a SOAP Request and Receiving a SOAP Response

Call the wrapper procedure created in the previous lesson, which sends a SOAP request to the web server that you created earlier.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Connect to the client database in Interactive SQL if it is not already open from lesson two.

   ```
   dbisql -c "UID=DBA;PWD=passwd;SERVER=ftc_client"
   ```

2. Enable logging of SOAP requests and responses.

   Execute the following SQL statements in Interactive SQL:

   ```
   CALL sa_server_option('WebClientLogFile', 'soap.txt');
   CALL sa_server_option('WebClientLogging', 'ON');
   ```

   These calls allow you to examine the content of the SOAP request and response. The requests and responses are logged to a file called `soap.txt`.

3. Call the wrapper procedure to send a SOAP request and receive the SOAP response.

   Execute the following SQL statement in Interactive SQL:

   ```
   CALL FahrenheitToCelsius(212);
   ```

   This call passes a Fahrenheit value of 212 to the FahrenheitToCelsius procedure, which passes the value along with two customized SOAP headers to the FToC procedure. Both client-side procedures are created in the previous lesson.

Results

The FToC web service procedure sends the Fahrenheit value and the SOAP headers to the web server. The SOAP request contains the following.

```
<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>
```

The FtoC procedure then receives the response from the web server which includes a result set based on the Fahrenheit value. The SOAP response contains the following.

```
<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'>
        &lt;tns:rowset xmlns:tns=&quot;http://localhost:8082/ftc&quot;&gt;&#x0A;
        &lt;tns:row&gt;&#x0A;
        &lt;tns:answer&gt;100
        &lt;/tns:answer&gt;&#x0A;
        &lt;/tns:row&gt;&#x0A;
        &lt;/tns:rowset&gt;&#x0A;
      </tns:FtoCServiceResult>
      <tns:sqlcode>0</tns:sqlcode>
    </tns:FtoCServiceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The content of <SOAP-ENV:Header> is returned in inoutheader.

If you examine the SOAP response, you see that the result set was encoded in the response by the FToCService web service. The result set is decoded and returned to the FahrenheitToCelsius procedure. The result set looks like the following when a Fahrenheit value of 212 is passed to the web server:

```
<tns:rowset xmlns:tns="http://localhost:8082/ftc">
  <tns:row>
```

```
    <tns:answer>100
    </tns:answer>
  </tns:row>
</tns:rowset>
```

The SELECT statement in the FahrenheitToCelsius procedure uses the OPENXML operator to parse the SOAP response, extracting the Celsius value defined by the tns:answer structure.

The following result set is generated in Interactive SQL:

```
Fahrenheit   Celsius
        212       100
```

Example

Here is another sample call to the SOAP web service that will convert a temperature in Fahrenheit degrees to Celsius degrees.

```
CALL FahrenheitToCelsius(32);
```

The following result set is generated in Interactive SQL:

```
Fahrenheit   Celsius
         32         0
```

Task overview: Tutorial: Using a Database Server to Access a SOAP/DISH Service [page 690]

Previous task: Lesson 2: Setting Up a Web Client to Send SOAP Requests and Receive SOAP Responses [page 694]

Related Information

sa_server_option System Procedure

1.20.4.3 Tutorial: Using Microsoft Visual C# to Access a SOAP/DISH Web Service

This tutorial illustrates how to create a Microsoft Visual C# client application to access SOAP/DISH services on a database server acting as a web server.

Prerequisites

You need the following:

- Familiarity with SOAP
- Familiarity with .NET framework
- Basic knowledge of the web service features of the software

The following software is required:

- Microsoft Visual Studio

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Context

This tutorial shows you how to:

- Create and start a new web services database. The database server will act as the web server.
- Create a SOAP web service.
- Set up a procedure that returns the information contained in a SOAP request.
- Create a DISH web service that provides WSDL documents and acts as a proxy.
- Set up Microsoft Visual C# on the client computer and import a WSDL document from the web server.
- Create a Java client application to retrieve information from the SOAP service using the WSDL document information.

1. Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses [page 701]
   Set up a web server running SOAP and DISH web services that handles Microsoft Visual C# client application requests.
2. Lesson 2: Creating a Microsoft Visual C# Application to Communicate with the Web Server [page 703]
   Create a Microsoft Visual C# application to communicate with the web server.

Related Information

The Database Server as an HTTP Web Server [page 603]

1.20.4.3.1 Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses

Set up a web server running SOAP and DISH web services that handles Microsoft Visual C# client application requests.

Prerequisites

A recent version of Microsoft Visual Studio is required to complete this lesson.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Start the sample database using the following command:

   ```
   dbsrv17 -xs http(port=8082) "%SQLANYSAMP17%\demo.db"
   ```

   This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.
2. Connect to the database server in Interactive SQL using the following command:

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=demo"
   ```
3. Create a new SOAP service to accept incoming requests.

   Execute the following SQL statement in Interactive SQL:

   ```
   CREATE SERVICE "SASoapTest/EmployeeList"
       TYPE 'SOAP'
       DATATYPE ON
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA
       AS SELECT * FROM Employees;
   ```

   This statement creates a new SOAP web service named SASoapTest/EmployeeList that generates a SOAP type as output. It selects all columns from the Employees table and returns the result set to the client. The service name is surrounded by quotation marks because of the slash character (/) that appears in the service name. If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

DATATYPE ON indicates that explicit data type information is generated in the XML result set response and the input parameters. This option does not affect the WSDL document that is generated.

The FORMAT clause is not specified so the SOAP service format is dictated by the associated DISH service format which is declared in the next step.

4. Create a new DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

   Execute the following SQL statement in Interactive SQL:

   ```
   CREATE SERVICE SASoapTest_DNET
       TYPE 'DISH'
       GROUP SASoapTest
       FORMAT 'DNET'
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA;
   ```

   DISH web services accessed from .NET should be declared with the FORMAT 'DNET' clause. The GROUP clause identifies the SOAP services that should be handled by the DISH service. The EmployeeList service created in the previous step is part of the GROUP SASoapTest because it is declared as SASoapTest/EmployeeList. If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

5. Verify that the DISH web service is functional by accessing the associated WSDL document through a web browser.

   Open your web browser and go to `http://localhost:8082/demo/SASoapTest_DNET`.

   The DISH service automatically generates a WSDL document that appears in the browser window.

Results

You have set up a web server running SOAP and DISH web services that can handle Microsoft Visual C# client application requests.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Microsoft Visual C# to Access a SOAP/DISH Web Service [page 700]

Next task: Lesson 2: Creating a Microsoft Visual C# Application to Communicate with the Web Server [page 703]

Related Information

How to Create DISH Services [page 615]
CREATE SERVICE Statement [SOAP Web Service]

1.20.4.3.2 Lesson 2: Creating a Microsoft Visual C# Application to Communicate with the Web Server

Create a Microsoft Visual C# application to communicate with the web server.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

A recent version of Microsoft Visual Studio is required to complete this lesson.

Context

This lesson contains several references to localhost. Use the host name or IP address of the web server from the previous lesson instead of localhost if you are not running the web client on the same computer as the web server.

This example uses functions from the .NET Framework 2.0.

Procedure

1. Start Microsoft Visual Studio.
2. Create a new Microsoft Visual C# *Windows Forms Application* project.

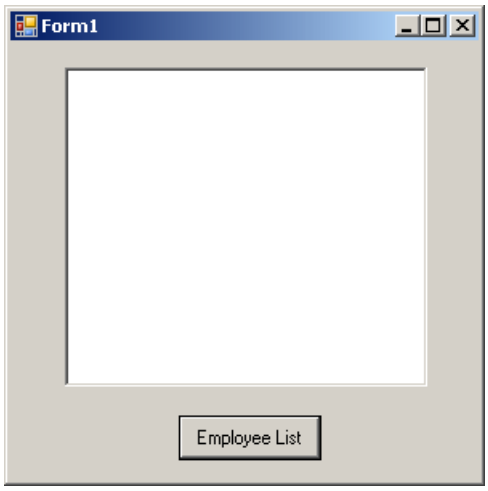
   An empty form appears.
3. Add a web reference to the project.
   a. Click *Project* > *Add Service Reference*.
   b. In the Add Service Reference window, click *Advanced*.
   c. In the Service Reference Settings window, click *Add Web Reference*.
   d. In the Add Web Reference window, type **`http://localhost:8082/`**demo**`/SASoapTest_DNET`** in the *URL* field.
   e. Click *Go* (or the green arrow).

      Microsoft Visual Studio lists the EmployeeList method available from the SASoapTest_DNET service.
   f. Click *Add Reference*.

Microsoft Visual Studio adds localhost to the project *Web References* in the *Solution Explorer* pane.

4. Populate the empty form with the desired objects for web client application.

   From the *Toolbox* pane, drag ListBox and Button objects onto the form and update the text attributes so that the form looks similar to the following diagram:



5. Write a procedure that accesses the web reference and uses the available methods.

   Double-click the Employee List button and add the following code for the button click event:

```
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. Run the application.

   Click *Debug* > *Start Debugging*.

7. Communicate with the web database server.

   Click *Employee List*.

   The ListBox object displays the EmployeeList result set as (type)name=value pairs. The following output illustrates how an entry appears in the ListBox object:

```
(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
```

   The Salary amount is converted to the client's currency format.

   Values that contain a date with no time are assigned a time of 00:00:00 or midnight (which is displayed in a format dependent on the client's locale settings).

   Values that contain null are tested using the DataTableReader.IsDBNull method.

Results

The XML response from the web server includes a formatted result set. All column data is converted to a string representation of the data. The following result set illustrates how result sets are formatted when they are sent to the client:

```
<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>
```

The DATATYPE ON clause was specified in the previous lesson to generate data type information in the XML result set response. A fragment of the response from the web server is shown below. The type information matches the data type of the database columns.

```
<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' />
```

Columns containing only the date are formatted as `yyyy-mm-dd` in the XML response.

Columns containing only the time are formatted as `hh:mm:ss.nnn-HH:MM` or `hh:mm:ss.nnn+HH:MM` in the XML response. A zone offset (-HH:MM or +HH:MM) is suffixed to the string.

Columns containing both date and time are formatted as `yyyy-mm-ddThh:mm:ss.nnn-HH:MM` or `yyyy-mm-ddThh:mm:ss.nnn+HH:MM` in the XML response. The date is separated from the time using the letter 'T'. A zone offset (-HH:MM or +HH:MM) is suffixed to the string.

Columns containing date, time, and time zone offset are formatted as `yyyy-mm-ddThh:mm:ss.nnn-HH:MM` or `yyyy-mm-ddThh:mm:ss.nnn+HH:MM` in the XML response. The date is separated from the time using the letter 'T'. A zone offset (-HH:MM or +HH:MM) is suffixed to the string.

Some examples for a server running in the Pacific time zone follow:

```
<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>
```

The format in which these dates and times are displayed in a .NET application depends on the client's time zone and locale settings. The following is an example for a client in the Eastern time zone with a United States locale (assuming that today's date is January 28, 2013).

```
(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
```

There is no separate Date and Time in the TypeCode enumeration - there is only TypeCode.DateTime - which explains why all results include both a date and time.

The data type information in the XML response for these dates and times follows:

```
<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' />
```

Task overview: Tutorial: Using Microsoft Visual C# to Access a SOAP/DISH Web Service [page 700]

Previous task: Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses [page 701]

1.20.4.4 Tutorial: Using JAX-WS to Access a SOAP/DISH Web Service

This tutorial illustrates how to create a Java API for XML Web Services (JAX-WS) client application to access SOAP/DISH services on a database server acting as a web server.

Prerequisites

You need the following:

- Familiarity with SOAP
- Familiarity with Java and JAX-WS
- Basic knowledge of the web service features of the software

The following software is required:

- JDK 1.7.0 or later version
- JAX-WS 2.2.7 or later version

The following privileges are required to perform the lessons in this tutorial.

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

Context

This tutorial shows you how to:

- Create and start a new web services database. The database server will act as the web server.
- Create a SOAP web service.
- Set up a procedure that returns the information contained in a SOAP request.
- Create a DISH web service that provides WSDL documents and acts as a proxy.
- Use JAX-WS on the client computer to process a WSDL document from the web server.
- Create a Java client application to retrieve information from the SOAP service using the WSDL document information.

1. Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses [page 709]
   Set up a web server running SOAP and DISH web services that handle JAX-WS client application requests.
2. Lesson 2: Creating a Java Application to Communicate with the Web Server [page 712]
   Process the WSDL document generated from the DISH service and create a Java application to access table data based on the schema defined in the WSDL document.

Related Information

The Database Server as an HTTP Web Server [page 603]

1.20.4.4.1 Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses

Set up a web server running SOAP and DISH web services that handle JAX-WS client application requests.

Prerequisites

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This lesson sets up the web server and a simple web service that you use later in the tutorial. It can be instructional to use proxy software to observe the XML message traffic. The proxy inserts itself between your client application and the web server.

Procedure

1. Start the sample database using the following command:

   ```
   dbsrv17 -xs http(port=8082) "%SQLANYSAMP17%\demo.db"
   ```

   This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.
2. Connect to the database server with Interactive SQL using the following command:

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=demo"
   ```
3. Create a stored procedure that lists Employees table columns.

   Execute the following SQL statements in Interactive SQL:

   ```
   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;
```

These statements create a new procedure named ListEmployees that defines the structure of the result set output, and selects certain columns from the Employees table.

4. Create a new SOAP service to accept incoming requests.

   Execute the following SQL statement in Interactive SQL:

   ```
   CREATE SERVICE "WS/EmployeeList"
       TYPE 'SOAP'
       FORMAT 'CONCRETE' EXPLICIT ON
       DATATYPE ON
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA
       AS CALL ListEmployees();
   ```

   This statement creates a new SOAP web service named WS/EmployeeList that generates a SOAP type as output. It calls the ListEmployees procedure when a web client sends a request to the service. The service name is surrounded by quotation marks because of the slash character (/) that appears in the service name.

   SOAP web services accessed from JAX-WS should be declared with the FORMAT 'CONCRETE' clause. The EXPLICIT ON clause indicates that the corresponding DISH service should generate XML Schema that describes an explicit dataset object based on the result set of the ListEmployees procedure. The EXPLICIT clause only affects the generated WSDL document.

   DATATYPE ON indicates that explicit data type information is generated in the XML result set response and the input parameters. This option does not affect the WSDL document that is generated.

   If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

5. Create a new DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

   Execute the following SQL statement in Interactive SQL:

   ```
   CREATE SERVICE WSDish
       TYPE 'DISH'
       FORMAT 'CONCRETE'
       GROUP WS
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA;
   ```

   DISH web services accessed from JAX-WS should be declared with the FORMAT 'CONCRETE' clause. The GROUP clause identifies the SOAP services that should be handled by the DISH service. The EmployeeList service created in the previous step is part of the GROUP WS because it is declared as WS/EmployeeList. If you logged in with a different user ID, then the USER DBA clause must be changed to reflect your user ID.

6. Verify that the DISH web service is functional by accessing the associated WSDL document through a web browser.

   Open your web browser and go to `http://localhost:8082/demo/WSDish`.

The DISH service automatically generates a WSDL document that appears in the browser window. Examine the EmployeeListDataset object, which looks similar to the following output:

```
<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 is the explicit object generated by the FORMAT 'CONCRETE' and EXPLICIT ON clauses in the EmployeeList SOAP service. In a later lesson, the wsimport application uses this information to generate a SOAP 1.1 client interface for this service.

Results

You have set up a web server running SOAP and DISH web services that can handle JAX-WS client application requests.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using JAX-WS to Access a SOAP/DISH Web Service [page 708]

Next task: Lesson 2: Creating a Java Application to Communicate with the Web Server [page 712]

Related Information

How to Create DISH Services [page 615]
CREATE SERVICE Statement [SOAP Web Service]

1.20.4.4.2 Lesson 2: Creating a Java Application to Communicate with the Web Server

Process the WSDL document generated from the DISH service and create a Java application to access table data based on the schema defined in the WSDL document.

Prerequisites

You must have completed the previous lessons in this tutorial.

The web server that was started in the previous lesson must be running.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

At the time of writing, the version of JAX-WS used in this tutorial was 2.2.9 and it is included in JDK 1.8. The steps that follow are based on that version. To determine if JAX-WS is present in your JDK, check for the wsimport application in the JDK `bin` directory. If it is not there then go to http://jax-ws.java.net/ to download and install the latest version of JAX-WS.

This lesson contains several references to localhost. Use the host name or IP address of the web server from lesson 1 instead of localhost if you are not running the web client on the same computer as the web server.

Procedure

1. At a command prompt, create a new working directory for your Java code and generated files. Change to this new directory.
2. Generate the interface that calls the DISH web service and imports the WSDL document using the following command:

   ```
   wsimport -keep -p employeelist "http://localhost:8082/demo/WSDish"
   ```

   If this command fails with parsing errors, then replace *localhost* with 127.0.0.1 or your real IP address or host name. For example:

   ```
   wsimport -keep -p employeelist "http://127.0.0.1:8082/demo/WSDish"
   ```

The wsimport application retrieves the WSDL document from the given URL. It generates `.java` files to create an interface for it, then compiles them into `.class` files.

The *keep* option indicates that the Java source files should not be deleted after generating the class files. The generated Java source code allows you to understand the generated class files.

The *p* option indicates that the Java source and class files should be generated into the `employeelist` package.

The wsimport application creates a new subdirectory named `employeelist` in your current working directory. The following is a list of the contents of that directory:

- `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. Write a Java application that accesses table data from the database server based on the dataset object schema defined in the generated source code.

   The following is a sample Java application that does this. Save the source code as `SoapDemo.java` in the current working directory. Your current working directory must be the directory containing the `employeelist` subdirectory.

```
// SoapDemo.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 employeelist.*;
public class SoapDemo
{
  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();
    }
  }
}
```

4. Compile your Java application using the following commands:

```
set CLASSPATH=.;%CLASSPATH%
javac SoapDemo.java
```

5. Execute the application using the following command:

   ```
   java SoapDemo
   ```

6. The application sends its request to the web server. It receives an XML result set response that consists of an EmployeeListResult with a rowset containing several row entries.

   This application prints all server-provided column data to the standard system output.

   For more information about the Java methods used in this application, see the javax.xml.bind.JAXBElement class API documentation.

Results

The following is an example of the output from running SoapDemo:

```
(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
```

The TerminationDate column is only sent when its value is not NULL. The Java application is designed to detect when the TerminationDate column is not present. For this example, the last row in the Employees table was altered such that a non-NULL termination date was set.

The following is an example of a SOAP response from the web server. The SQLCODE result from executing the query is included in the response.

```
<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>
```

Column names and data types are included in each rowset.

Task overview: Tutorial: Using JAX-WS to Access a SOAP/DISH Web Service [page 708]

Previous task: Lesson 1: Setting Up a Web Server to Receive SOAP Requests and Send SOAP Responses [page 709]

1.21 Three-tier Computing and Distributed Transactions

You can use the database server as a **resource manager**, participating in distributed transactions coordinated by a transaction server.

A three-tier environment, where an application server sits between client applications and a set of resource managers, is a common distributed-transaction environment. Application servers provide transaction logic to their client applications, guaranteeing that sets of operations are executed atomically. SAP Enterprise Application Server (SAP EAServer) and some other application servers are transaction servers.

SAP EAServer and Microsoft Transaction Server both use the Microsoft Distributed Transaction Coordinator (DTC) to coordinate transactions. The database server provides support for distributed transactions controlled by the DTC service, so you can use the database server with either of these application servers, or any other product based on the DTC model. DTC uses **OLE transactions**, which in turn use the **two-phase commit** protocol to coordinate transactions involving multiple resource managers. You must have DTC installed to use distributed transactions.

When integrating the database server into a three-tier environment, most of the work needs to be done from the application server. The concepts and architecture of three-tier computing are introduced, and an overview of relevant database server features is presented. It does not describe how to configure your application server to work with the database server. For more information, see your application server documentation.

You can also use DTC directly in your applications to coordinate transactions across multiple resource managers.

In this section:

Three-tier Computing Architecture [page 717]
In three-tier computing, application logic is held in an application server, such as SAP Enterprise Application Server (SAP EAServer), which sits between the resource manager and the client applications.

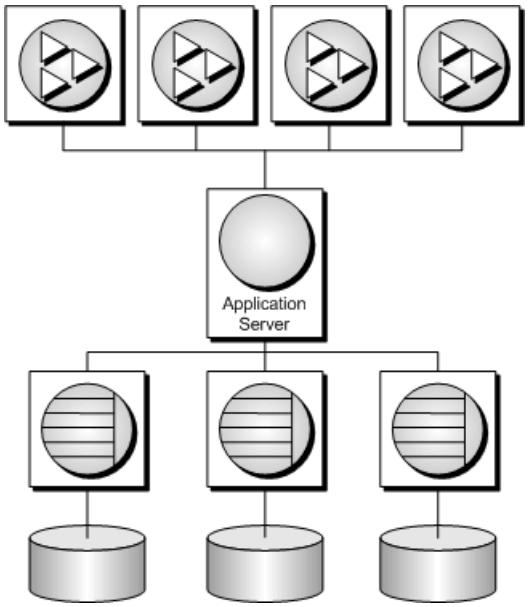
Distributed Transactions [page 719]
While the database server is enlisted in a distributed transaction, it hands transaction control over to the transaction server, and the database server ensures that it does not perform any implicit

transaction management. The following conditions are imposed automatically by the database server when it participates in distributed transactions:

1.21.1 Three-tier Computing Architecture

In three-tier computing, application logic is held in an application server, such as SAP Enterprise Application Server (SAP EAServer), which sits between the resource manager and the client applications.

In many situations, a single application server may access multiple resource managers. In the Internet case, client applications are browser-based, and the application server is generally a web server extension.



SAP EAServer stores application logic in the form of components, and makes these components available to client applications. The components may be SAP PowerBuilder components, JavaBeans, or COM components.

For more information, see your SAP EAServer documentation.

In this section:

The Vocabulary of Distributed Transactions [page 718]
Some common terms are used when discussing distributed transactions.

How Application Servers Use DTC [page 718]
SAP Enterprise Application Server (SAP EAServer) and Microsoft Transaction Server are both component servers. The application logic is held in the form of components, and made available to client applications.

Distributed Transaction Architecture [page 719]
The following diagram illustrates the architecture of distributed transactions. In this case, the resource manager proxy is either ADO.NET, OLE DB, or ODBC.

1.21.1.1 The Vocabulary of Distributed Transactions

Some common terms are used when discussing distributed transactions.

- **Resource managers** are those services that manage the data involved in the transaction.
  The database server can act as a resource manager in a distributed transaction when accessed through ADO.NET, OLE DB, or ODBC. The .NET Data Provider, OLE DB provider, and ODBC driver act as resource manager proxies on the client computer. The .NET Data Provider supports distributed transactions using DbProviderFactory and TransactionScope.
- Instead of communicating directly with the resource manager, application components can communicate with **resource dispensers**, which in turn manage connections or pools of connections to the resource managers.
  The database server supports two resource dispensers: the ODBC driver manager and OLE DB.
- When a transactional component requests a database connection (using a resource manager), the application server **enlists** each database connection that takes part in the transaction. DTC and the resource dispenser perform the enlistment process.

For additional information, see your transaction server documentation.

Two-Phase Commit

Distributed transactions are managed using two-phase commit. When the work of the transaction is complete, the transaction manager (DTC) asks all the resource managers enlisted in the transaction whether they are ready to commit the transaction. This phase is called **preparing** to commit.

If all the resource managers respond that they are prepared to commit, DTC sends a commit request to each resource manager, and responds to its client that the transaction is completed. If one or more resource managers do not respond, or respond that they cannot commit the transaction, all the work of the transaction is rolled back across all resource managers.

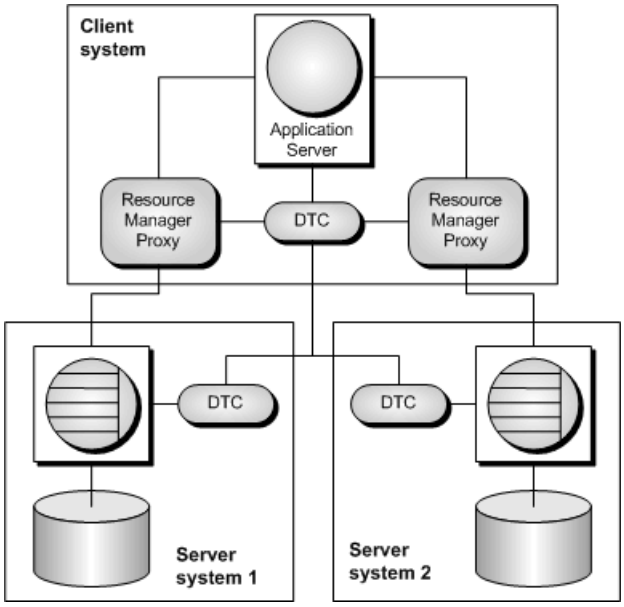
1.21.1.2 How Application Servers Use DTC

SAP Enterprise Application Server (SAP EAServer) and Microsoft Transaction Server are both component servers. The application logic is held in the form of components, and made available to client applications.

Each component has a transaction attribute that indicates how the component participates in transactions. When building the component, you must program the work of the transaction into the component (the resource manager connections, the operations on the data for which each resource manager is responsible). However, you do not need to add transaction management logic to the component. Once the transaction attribute is set, to indicate that the component needs transaction management, EAServer uses DTC to enlist the transaction and manage the two-phase commit process.

1.21.1.3 Distributed Transaction Architecture

The following diagram illustrates the architecture of distributed transactions. In this case, the resource manager proxy is either ADO.NET, OLE DB, or ODBC.



In this case, a single resource dispenser is used. The application server asks DTC to prepare a transaction. DTC and the resource dispenser enlist each connection in the transaction. Each resource manager must be in contact with both the DTC and the database, so the work can be performed and the DTC can be notified of its transaction status when required.

A Distributed Transaction Coordinator (DTC) service must be running on each computer to operate distributed transactions. You can start or stop DTC from the Microsoft Windows *Services* window; the DTC service task is named MSDTC.

For more information, see your DTC or EAServer documentation.

1.21.2 Distributed Transactions

While the database server is enlisted in a distributed transaction, it hands transaction control over to the transaction server, and the database server ensures that it does not perform any implicit transaction management. The following conditions are imposed automatically by the database server when it participates in distributed transactions:

- Autocommit is automatically turned off, if it is in use.
- Data definition statements (which commit as a side effect) are disallowed during distributed transactions.
- An explicit COMMIT or ROLLBACK issued by the application directly to the database server, instead of through the transaction coordinator, generates an error. The transaction is not aborted, however.
- A connection can participate in only a single distributed transaction at a time.

- There must be no uncommitted operations at the time the connection is enlisted in a distributed transaction.

In this section:

DTC Isolation Levels [page 720]
DTC has a set of isolation levels, which the application server specifies. These isolation levels map to the database server isolation levels as follows:

Recovery from Distributed Transactions [page 720]
If the database server faults while uncommitted operations are pending, it must either rollback or commit those operations on startup to preserve the atomic nature of the transaction.

1.21.2.1 DTC Isolation Levels

DTC has a set of isolation levels, which the application server specifies. These isolation levels map to the database server isolation levels as follows:

DTC Isolation Level	Database Server Isolation Level
ISOLATIONLEVEL_UNSPECIFIED	0
ISOLATIONLEVEL_CHAOS	0
ISOLATIONLEVEL_READUNCOMMITTED	0
ISOLATIONLEVEL_BROWSE	0
ISOLATIONLEVEL_CURSORSTABILITY	1
ISOLATIONLEVEL_READCOMMITTED	1
ISOLATIONLEVEL_REPEATABLEREAD	2
ISOLATIONLEVEL_SERIALIZABLE	3
ISOLATIONLEVEL_ISOLATED	3

1.21.2.2 Recovery from Distributed Transactions

If the database server faults while uncommitted operations are pending, it must either rollback or commit those operations on startup to preserve the atomic nature of the transaction.

If uncommitted operations from a distributed transaction are found during recovery, the database server attempts to connect to DTC and requests that it be re-enlisted in the pending or in-doubt transactions. Once the re-enlistment is complete, DTC instructs the database server to roll back or commit the outstanding operations.

If the reenlistment process fails, the database server has no way of knowing whether the in-doubt operations should be committed or rolled back, and recovery fails. If you want the database in such a state to recover, regardless of the uncertain state of the data, you can force recovery using the following database server options:

-tmf

If DTC cannot be located, the outstanding operations are rolled back and recovery continues.

-tmt

If the -tmt option is specified and re-enlistment is not achieved before the specified time, then database recovery fails. If both -tmt and -tmf are specified and re-enlistment is not achieved before the specified time, then operations are rolled back and recovery continues.

Related Information

-tmf Database Server Option
-tmt Database Server Option

1.22 Database Tools Programming

You can write applications in C or C++ that implement common database tasks such as backup and unload.

In this section:

Database Tools Interface (DBTools) [page 722]
The database server software includes administration tools and a set of utilities for managing databases. These database management utilities perform tasks such as backing up databases, creating databases, translating transaction logs to SQL, and so on.

SQL Anywhere Database Tools C API Reference [page 729]
An application program uses structures to exchange information with the DBTools API library. All of these structures are defined in `dbtools.h`, except the a_remote_sql structure, which is defined in `dbrmt.h`.

Software Component Exit Codes [page 806]
All database tools library entry points return exit codes. The database server and utilities (dbsrv17, dbbackup, dbspawn, and so on) also use these exit codes.

1.22.1 Database Tools Interface (DBTools)

The database server software includes administration tools and a set of utilities for managing databases. These database management utilities perform tasks such as backing up databases, creating databases, translating transaction logs to SQL, and so on.

Supported Platforms

All the database management utilities use a shared library called the **database tools library**. For Windows, the name of this library is `dbtool17.dll`. For UNIX and Linux, the name of this library is `libdbtool17_r.so`. For macOS, the name of this library is libdbtool17_r.dylib.

You can develop your own database management utilities or incorporate database management features into your applications by calling the database tools library.

The database tools library has functions, or entry points, for each of the database management utilities. In addition, functions must be called before use of other database tools functions and when you have finished using other database tools functions.

The `dbtools.h` Header File

The dbtools header file lists the entry points to the DBTools library and also the structures used to pass information to and from the library. The `dbtools.h` file is installed into the `SDK\Include` subdirectory under your software installation directory. Consult the `dbtools.h` file for the latest information about the entry points and structure members.

The `dbtools.h` header file includes other files such as:

sqlca.h

This is included for resolution of various macros, not for the SQLCA itself.

dllapi.h

Defines preprocessor macros for operating-system dependent and language-dependent macros.

dbtlvers.h

Defines the DB_TOOLS_VERSION_NUMBER preprocessor macro and other version specific macros.

The `sqldef.h` Header File

The `sqldef.h` header file includes error return values.

The `dbrmt.h` Header File

The `dbrmt.h` header file describes the DBRemoteSQL entry point in the DBTools library and also the structure used to pass information to and from the DBRemoteSQL entry point. The `dbrmt.h` file is installed into the `SDK\Include` subdirectory under your software installation directory. Consult the `dbrmt.h` file for the latest information about the DBRemoteSQL entry point and structure members.

In this section:

DBTools Import Libraries [page 723]
To use the DBTools functions, you must link your application against a DBTools **import library** that contains the required function definitions.

DBTools Library Initialization and Finalization [page 724]
Before using any other DBTools functions, you must call DBToolsInit. When you are finished using the DBTools library, you must call DBToolsFini.

DBTools Function Calls [page 724]
All the tools are run by first filling out a structure, and then calling a function (or **entry point**) in the DBTools library. Each entry point takes a pointer to a single structure as argument.

Callback Functions [page 725]
Several elements in DBTools structures are of type MSG_CALLBACK or SHOULD_STOP_CALLBACK. These are pointers to callback functions.

Version Numbers and Compatibility [page 727]
Each structure has a member that indicates the version number. Set the version field to the version number of the DBTools library that your application was developed against before calling any DBTools function.

Bit Fields [page 728]
Many of the DBTools structures use bit fields to hold Boolean information in a compact manner.

A DBTools Example [page 728]
You can find a sample and instructions for compiling it in the *%SQLANYSAMP17%* `\SQLAnywhere\DBTools` directory.

1.22.1.1 DBTools Import Libraries

To use the DBTools functions, you must link your application against a DBTools **import library** that contains the required function definitions.

For UNIX and Linux systems, no import library is required. Link directly against `libdbtool`*17*`.so` (non-threaded) or `libdbtool`*17*`_r.so` (threaded).

Import Libraries

Import libraries for the DBTools interface are provided for 32-bit and 64-bit Windows platforms. The `dbtlstm.lib` library can be found in the `SDK\Lib\x86` and `SDK\Lib\x64` subdirectories under your software installation directory.

1.22.1.2 DBTools Library Initialization and Finalization

Before using any other DBTools functions, you must call DBToolsInit. When you are finished using the DBTools library, you must call DBToolsFini.

The primary purpose of the DBToolsInit and DBToolsFini functions is to allow the DBTools library to load and unload the messages DLL or shared object. The messages library contains localized versions of all error messages and prompts that DBTools uses internally. If DBToolsFini is not called, the reference count of the messages library is not decremented and it will not be unloaded, so be careful to ensure there is a matched pair of DBToolsInit/DBToolsFini calls.

The following code fragment illustrates how to initialize and finalize DBTools:

```
// 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 );
```

1.22.1.3 DBTools Function Calls

All the tools are run by first filling out a structure, and then calling a function (or **entry point**) in the DBTools library. Each entry point takes a pointer to a single structure as argument.

The following example shows how to use the DBBackup function on a Windows operating system.

```
// 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 );
```

Related Information

SQL Anywhere Database Tools C API Reference [page 729]

1.22.1.4 Callback Functions

Several elements in DBTools structures are of type MSG_CALLBACK or SHOULD_STOP_CALLBACK. These are pointers to callback functions.

Uses of Callback Functions

Callback functions allow DBTools functions to return control of operation to the user's calling application. The DBTools library uses callback functions to handle messages sent to the user by the DBTools functions for four purposes:

Confirmation

Called when an action needs to be confirmed by the user. For example, if the backup directory does not exist, the tools library asks if it needs to be created.

Error message

Called to handle a message when an error occurs, such as when an operation is out of disk space.

Information message

Called for the tools to display some message to the user (such as the name of the current table being unloaded).

Status information

Called for the tools to display the status of an operation (such as the percentage done when unloading a table).

Stop processing Called to determine if processing should stop. For example, `dbunload -aot` calls this function to determine if incremental backups of the production database and applying the transaction log to the rebuilt database should continue.

Assigning a Callback Function to a Structure

You can directly assign a callback routine to the structure. The following statement is an example using a backup structure:

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

The following statement is an example using the unload structure:

```
unload_info.shouldstoprtn = (SHOULD_STOP_CALLBACK) MyStopFunction
```

MSG_CALLBACK and SHOULD_STOP_CALLBACK are defined in the `dllapi.h` header file. Tools routines can call back to the calling application with messages that should appear in the appropriate user interface, whether that be a windowing environment, standard output on a character-based system, or other user interface.

Confirmation Callback Function Example

The following example confirmation routine asks the user to answer YES or NO to a prompt and returns the user's selection:

```
extern short _callback ConfirmRtn(
        char * question )
{
    int ret = IDNO;
    if( question != NULL ) {
        ret = MessageBox( HwndParent, question,
        "Confirm", MB_ICONEXCLAMATION|MB_YESNO );
    }
    return( ret == IDYES );
}
```

Error Callback Function Example

The following is an example of an error message handling routine, which displays the error message in a window.

```
extern short _callback ErrorRtn(
        char * errorstr )
{
    if( errorstr != NULL ) {
        MessageBox( HwndParent, errorstr, "Backup Error", MB_ICONSTOP|MB_OK );
    }
    return( 0 );
}
```

Message Callback Function Example

A common implementation of a message callback function outputs the message to the screen:

```
extern short _callback MessageRtn(
        char * messagestr )
{
    if( messagestr != NULL ) {
                    OutputMessageToWindow( messagestr );
    }
    return( 0 );
}
```

Status Callback Function Example

A status callback routine is called when a tool needs to display the status of an operation (like the percentage done unloading a table). A common implementation would just output the message to the screen:

```
extern short _callback StatusRtn(
        char * statusstr )
{
    if( statusstr != NULL ) {
        OutputMessageToWindow( statusstr );
    return( 0 );
}
```

Stop Processing Callback Function Example

A Stop processing callback routine is called when a tool needs to decide if it should continue processing or stop processing. An implementation could be based on whether a Stop processing button is pressed. This example assumes there is code that handles the Stop processing button event by setting a StopProcessing global variable to non-zero.

```
extern short _callback StopProcessingRtn()
{
    return( StopProcessing );
}
```

1.22.1.5 Version Numbers and Compatibility

Each structure has a member that indicates the version number. Set the version field to the version number of the DBTools library that your application was developed against before calling any DBTools function.

The current version of the DBTools library is defined when you include the `dbtools.h` header file.

The following example assigns the current version to an instance of the a_backup_db structure:

```
backup_info.version = DB_TOOLS_VERSION_NUMBER;
```

The version number allows your application to continue working with newer versions of the DBTools library. The DBTools functions use the version number supplied by your application to allow the application to work, even if new members have been added to the DBTools structure.

When any of the DBTools structures are updated, or when a newer version of the software is released, the version number is augmented. If you use DB_TOOLS_VERSION_NUMBER and you rebuild your application with a new version of the DBTools header file, then you must deploy a new version of the DBTools library.

1.22.1.6 Bit Fields

Many of the DBTools structures use bit fields to hold Boolean information in a compact manner.

For example, the backup structure includes the following bit fields:

```
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;
```

Each bit field is one bit long, indicated by the 1 to the right of the colon in the structure declaration. The specific data type used depends on the value assigned to a_bit_field, which is set at the top of `dbtools.h`, and is operating system-dependent.

You assign a value of 0 or 1 to a bit field to pass Boolean information in the structure.

1.22.1.7 A DBTools Example

You can find a sample and instructions for compiling it in the *%SQLANYSAMP17%*`\SQLAnywhere\DBTools` directory.

The sample program code is in `main.cpp`. The sample illustrates how to use the DBTools library to perform a backup of a database.

1.22.2 SQL Anywhere Database Tools C API Reference

An application program uses structures to exchange information with the DBTools API library. All of these structures are defined in `dbtools.h`, except the a_remote_sql structure, which is defined in `dbrmt.h`.

Header Files

- `dbtools.h`
- `dbrmt.h`

Database Tools Structures

Many of the structure elements correspond to command line options on the corresponding utility. For example, several structures have a member named `quiet`, which can take on values of 0 or 1. This member corresponds to the quiet operation (-q) option used by many of the utilities.

The data structures defined in this file can be used with the DBTools API for the major version of SQL Anywhere to which the file applies. Applications built using, for example, the version 16.0.0 `dbtools.h` file cannot access `dbtool12.dll` or `dbtool17.dll`.

Within a major version, any changes to the structures are made in such a way that applications built with earlier or later versions of `dbtools.h` for the same major version continue to work. Applications built with earlier versions cannot access new fields, so DBTools provides default values which give the same behavior as in earlier versions. Generally, this means new fields added in point releases appear at the end of a structure.

The behavior for applications built with later versions within the same major release depends on the value provided in the version field of the structure. If the version number provided corresponds to an earlier version, the application can call an earlier version of the DBTools DLL just as if the application was built using that version of `dbtools.h`. If the version number provided is the current version, attempting to use an earlier version of the DBTools DLL results in an error.

See `dbtlvers.h` for version number definitions.

In this section:

DBBackup(const a_backup_db *) Method [page 732]
Backs up a database.

DBChangeLogName(a_change_log *) Method [page 733]
Changes the name of the transaction log file.

DBCreate(a_create_db *) Method [page 734]
Creates a database.

DBCreatedVersion(a_db_version_info *) Method [page 734]
Determines the version of SQL Anywhere that was used to create a database file, without attempting to start the database.

DBErase(const an_erase_db *) Method [page 735]

Erases a database file and/or transaction log file.

DBInfo(a_db_info *) Method [page 736]
Returns information about a database file.

DBInfoDump(a_db_info *) Method [page 737]
Returns information about a database file.

DBInfoFree(a_db_info *) Method [page 737]
Frees resources after the DBInfoDump function is called.

DBLicense(const a_dblic_info *) Method [page 738]
Modifies or reports the licensing information of the database server.

DBLogFileInfo(const a_log_file_info *) Method [page 739]
Returns the log file and mirror log file paths of a non-running database file.

DBRemoteSQL(a_remote_sql *) Method [page 739]
Accesses the SQL Remote Message Agent.

DBSynchronizeLog(const a_sync_db *) Method [page 740]
Synchronize a database with a MobiLink server.

DBToolsFini(const a_dbtools_info *) Method [page 741]
Decrements a reference counter and frees resources when an application is finished with the DBTools library.

DBToolsInit(const a_dbtools_info *) Method [page 741]
Prepares the DBTools library for use.

DBToolsVersion(void) Method [page 742]
Returns the version number of the DBTools library.

DBTranslateLog(const a_translate_log *) Method [page 743]
Translates a transaction log file to SQL.

DBTruncateLog(const a_truncate_log *) Method [page 743]
Truncates a transaction log file.

DBUnload(an_unload_db *) Method [page 744]
Unloads a database.

DBUpgrade(const an_upgrade_db *) Method [page 745]
Upgrades a database file.

DBValidate(const a_validate_db *) Method [page 746]
Validates all or part of a database.

Autotune Enumeration [page 746]
Used in the a_backup_db structure to control auto tuning of writers.

Checkpoint Enumeration [page 747]
Used in the a_backup_db structure to control copying of the checkpoint log.

History Enumeration [page 748]
Used in the a_backup_db structure to control enabling of backup history.

InMemory Enumeration [page 748]
The in-memory mode switch which should be added to the StartLine connection parameter.

LiveValidation Enumeration [page 749]
The type of live validation being performed, as used by the a_validate_db structure.

Padding Enumeration [page 750]
Blank padding enumeration specifies the blank_pad setting in a_create_db.

Unit Enumeration [page 750]
Used in the a_create_db structure, to specify the value of db_size_unit.

Unload Enumeration [page 751]
The type of unload being performed, as used by the an_unload_db structure.

UserList Enumeration [page 752]
The type of a user list, as used by an a_translate_log structure.

Validation Enumeration [page 752]
The type of validation being performed, as used by the a_validate_db structure.

Verbosity Enumeration [page 753]
Verbosity enumeration specifies the volume of output.

Version Enumeration [page 754]
Used in the a_db_version_info structure, to indicate the version of SQL Anywhere that initially created the database.

a_backup_db Structure [page 754]
Holds the information needed to perform backup tasks using the DBTools library.

a_change_log Structure [page 757]
Holds the information needed to perform dblog tasks using the DBTools library.

a_create_db Structure [page 760]
Holds the information needed to create a database using the DBTools library.

a_db_info Structure [page 763]
Holds the information needed to return DBInfo information using the DBTools library.

a_db_version_info Structure [page 766]
Holds information regarding which version of SQL Anywhere was used to create the database.

a_dblic_info Structure [page 767]
Holds information containing licensing information.

a_dbtools_info Structure [page 769]
DBTools information callback used to initialize and finalize the DBTools library calls.

a_log_file_info Structure [page 769]
Used to obtain the log file and mirror log file information of a non-running database.

a_name Structure [page 770]
Specifies a variable list of names.

a_remote_sql Structure [page 771]
Holds information needed for the dbremote utility using the DBTools library.

a_sync_db Structure [page 778]
Holds information needed for the dbmlsync utility using the DBTools library.

a_syncpub Structure [page 786]
Holds information needed for the dbmlsync utility.

a_sysinfo Structure [page 787]
Holds information needed for dbinfo and dbunload utilities using the DBTools library.

a_table_info Structure [page 788]

Holds information about a table needed as part of the a_db_info structure.

a_translate_log Structure [page 789]
Holds information needed for transaction log translation using the DBTools library.

a_truncate_log Structure [page 792]
Holds information needed for transaction log truncation using the DBTools library.

a_validate_db Structure [page 793]
Holds information needed for database validation using the DBTools library.

an_erase_db Structure [page 795]
Holds information needed to erase a database using the DBTools library.

an_unload_db Structure [page 796]
Holds information needed to unload a database using the DBTools library or extract a remote database for SQL Remote.

an_upgrade_db Structure [page 803]
Holds information needed to upgrade a database using the DBTools library.

DBT_USE_DEFAULT_MIN_PWD_LEN Variable [page 805]
DBT_USE_DEFAULT_MIN_PWD_LEN indicates that the default minimum password length is to be used (that is, no specific length is being specified) for the new database.

1.22.2.1 DBBackup(const a_backup_db *) Method

Backs up a database.

Syntax

```
_crtn short _entry DBBackup (const a_backup_db * pdb)
```

Parameters

pdb Pointer to a properly initialized a_backup_db structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbbackup utility.

The DBBackup function manages all client-side database backup tasks.

To perform a server-side backup, use the BACKUP DATABASE statement.

Related Information

a_backup_db Structure [page 754]

1.22.2.2 DBChangeLogName(a_change_log *) Method

Changes the name of the transaction log file.

Syntax

```
_crtn short _entry DBChangeLogName (a_change_log * pcl)
```

Parameters

pcl Pointer to a properly initialized a_change_log structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dblog utility.

The -t option of the Transaction Log utility (dblog) changes the name of the transaction log. DBChangeLogName provides a programmatic interface to this function.

Related Information

a_change_log Structure [page 757]

1.22.2.3 DBCreate(a_create_db *) Method

Creates a database.

Syntax

```
_crtn short _entry DBCreate (a_create_db * pcdb)
```

Parameters

pcdb Pointer to a properly initialized a_create_db structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbinit utility.

Related Information

a_create_db Structure [page 760]

1.22.2.4 DBCreatedVersion(a_db_version_info *) Method

Determines the version of SQL Anywhere that was used to create a database file, without attempting to start the database.

Syntax

```
_crtn short _entry DBCreatedVersion (a_db_version_info * pdvi)
```

Parameters

pdvi Pointer to a properly initialized a_db_version_info structure.

Returns

Return code, as listed in software component exit codes.

Remarks

Currently, this function only differentiates between databases built with version 9 or earlier and those built with version 10 or later.

Version information is not set if a failing code is returned.

Related Information

a_db_version_info Structure [page 766]
Version Enumeration [page 754]

1.22.2.5 DBErase(const an_erase_db *) Method

Erases a database file and/or transaction log file.

Syntax

```
_crtn short _entry DBErase (const an_erase_db * pedb)
```

Parameters

pedb Pointer to a properly initialized an_erase_db structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dberase utility.

Related Information

an_erase_db Structure [page 795]

1.22.2.6 DBInfo(a_db_info *) Method

Returns information about a database file.

Syntax

```
_crtn short _entry DBInfo (a_db_info * pdbi)
```

Parameters

pdbi Pointer to a properly initialized a_db_info structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbinfo utility.

Related Information

a_db_info Structure [page 763]
DBInfoDump(a_db_info *) Method [page 737]
DBInfoFree(a_db_info *) Method [page 737]

1.22.2.7 DBInfoDump(a_db_info *) Method

Returns information about a database file.

Syntax

```
_crtn short _entry DBInfoDump (a_db_info * pdbi)
```

Parameters

pdbi Pointer to a properly initialized a_db_info structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbinfo utility when the -u option is used.

Related Information

a_db_info Structure [page 763]
DBInfo(a_db_info *) Method [page 736]
DBInfoFree(a_db_info *) Method [page 737]

1.22.2.8 DBInfoFree(a_db_info *) Method

Frees resources after the DBInfoDump function is called.

Syntax

```
_crtn short _entry DBInfoFree (a_db_info * pdbi)
```

Parameters

pdbi Pointer to a properly initialized a_db_info structure.

Returns

Return code, as listed in software component exit codes.

Related Information

a_db_info Structure [page 763]
DBInfo(a_db_info *) Method [page 736]
DBInfoDump(a_db_info *) Method [page 737]

1.22.2.9 DBLicense(const a_dblic_info *) Method

Modifies or reports the licensing information of the database server.

Syntax

```
_crtn short _entry DBLicense (const a_dblic_info * pdi)
```

Parameters

pdi Pointer to a properly initialized a_dblic_info structure.

Returns

Return code, as listed in software component exit codes.

Related Information

a_dblic_info Structure [page 767]

1.22.2.10 DBLogFileInfo(const a_log_file_info *) Method

Returns the log file and mirror log file paths of a non-running database file.

Syntax

```
_crtn short _entry DBLogFileInfo (const a_log_file_info * plfi)
```

Parameters

plfi Pointer to a properly initialized a_log_file_info structure.

Returns

Return code, as listed in software component exit codes.

Remarks

Note that this function will only work for databases that have been created with SQL Anywhere 10.0.0 and up.

Related Information

a_log_file_info Structure [page 769]

1.22.2.11 DBRemoteSQL(a_remote_sql *) Method

Accesses the SQL Remote Message Agent.

Syntax

```
_crtn short _entry DBRemoteSQL (a_remote_sql * prs)
```

Parameters

prs Pointer to a properly initialized a_remote_sql structure.

Returns

Return code, as listed in Software component exit codes.

Related Information

a_remote_sql Structure [page 771]

1.22.2.12 DBSynchronizeLog(const a_sync_db *) Method

Synchronize a database with a MobiLink server.

Syntax

```
_crtn short _entry DBSynchronizeLog (const a_sync_db * psdb)
```

Parameters

psdb Pointer to a properly initialized a_sync_db structure.

Returns

Return code, as listed in software component exit codes.

Related Information

a_sync_db Structure [page 778]

1.22.2.13 DBToolsFini(const a_dbtools_info *) Method

Decrements a reference counter and frees resources when an application is finished with the DBTools library.

Syntax

```
_crtn short _entry DBToolsFini (const a_dbtools_info * pdi)
```

Parameters

pdi Pointer to a properly initialized a_dbtools_info structure.

Returns

Return code, as listed in software component exit codes.

Remarks

The DBToolsFini function must be called at the end of any application that uses the DBTools interface. Failure to do so can lead to lost memory resources.

Related Information

a_dbtools_info Structure [page 769]
DBToolsInit(const a_dbtools_info *) Method [page 741]

1.22.2.14 DBToolsInit(const a_dbtools_info *) Method

Prepares the DBTools library for use.

Syntax

```
_crtn short _entry DBToolsInit (const a_dbtools_info * pdi)
```

Parameters

pdi Pointer to a properly initialized a_dbtools_info structure.

Returns

Return code, as listed in Software component exit codes.

Remarks

The primary purpose of the DBToolsInit function is to load the SQL Anywhere messages library. The messages library contains localized versions of error messages and prompts used by the functions in the DBTools library.

The DBToolsInit function must be called at the start of any application that uses the DBTools interface, before any other DBTools functions.

Related Information

a_dbtools_info Structure [page 769]
DBToolsFini(const a_dbtools_info *) Method [page 741]

1.22.2.15 DBToolsVersion(void) Method

Returns the version number of the DBTools library.

Syntax

```
_crtn short _entry DBToolsVersion (void )
```

Remarks

Use the DBToolsVersion function to check that the DBTools library is not older than one against which your application is developed. While applications can run against newer versions of DBTools, they cannot run against older versions.

1.22.2.16 DBTranslateLog(const a_translate_log *) Method

Translates a transaction log file to SQL.

Syntax

```
_crtn short _entry DBTranslateLog (const a_translate_log * ptl)
```

Parameters

ptl Pointer to a properly initialized a_translate_log structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbtran utility.

Related Information

a_translate_log Structure [page 789]

1.22.2.17 DBTruncateLog(const a_truncate_log *) Method

Truncates a transaction log file.

Syntax

```
_crtn short _entry DBTruncateLog (const a_truncate_log * ptl)
```

Parameters

ptl Pointer to a properly initialized a_truncate_log structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbbackup utility.

Related Information

a_truncate_log Structure [page 792]

1.22.2.18 DBUnload(an_unload_db *) Method

Unloads a database.

Syntax

```
_crtn short _entry DBUnload (an_unload_db * pudb)
```

Parameters

pudb Pointer to a properly initialized an_unload_db structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbunload and dbxtract utilities.

Related Information

an_unload_db Structure [page 796]

1.22.2.19 DBUpgrade(const an_upgrade_db *) Method

Upgrades a database file.

Syntax

```
_crtn short _entry DBUpgrade (const an_upgrade_db * pudb)
```

Parameters

pudb Pointer to a properly initialized an_upgrade_db structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbupgrad utility.

Related Information

an_upgrade_db Structure [page 803]

1.22.2.20 DBValidate(const a_validate_db *) Method

Validates all or part of a database.

Syntax

```
_crtn short _entry DBValidate (const a_validate_db * pvdb)
```

Parameters

pvdb Pointer to a properly initialized a_validate_db structure.

Returns

Return code, as listed in software component exit codes.

Remarks

This function is used by the dbvalid utility.

Caution: Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, spurious errors may be reported indicating some form of database corruption even though no corruption actually exists.

Related Information

a_validate_db Structure [page 793]

1.22.2.21 Autotune Enumeration

Used in the a_backup_db structure to control auto tuning of writers.

Syntax

```
enum Autotune
```

Members

Member Name	Description
BACKUP_AUTO_TUNE_UNSPECIFIED	Use to leave AUTO TUNE WRITERS clause unspecified.
BACKUP_AUTO_TUNE_ON	Use to generate AUTO TUNE WRITERS ON clause.
BACKUP_AUTO_TUNE_OFF	Use to generate AUTO TUNE WRITERS OFF clause.

Related Information

a_backup_db Structure [page 754]
DBBackup(const a_backup_db *) Method [page 732]

1.22.2.22 Checkpoint Enumeration

Used in the a_backup_db structure to control copying of the checkpoint log.

Syntax

```
enum Checkpoint
```

Members

Member Name	Description
BACKUP_CHKPT_LOG_COPY	Use to generate WITH CHECKPOINT LOG COPY clause.
BACKUP_CHKPT_LOG_NOCOPY	Use to generate WITH CHECKPOINT LOG NOCOPY clause.
BACKUP_CHKPT_LOG_RECOVER	Use to generate WITH CHECKPOINT LOG RECOVER clause.
BACKUP_CHKPT_LOG_AUTO	Use to generate WITH CHECKPOINT LOG AUTO clause.
BACKUP_CHKPT_LOG_DEFAULT	Use to omit WITH CHECKPOINT clause.

Related Information

a_backup_db Structure [page 754]
DBBackup(const a_backup_db *) Method [page 732]

1.22.2.23 History Enumeration

Used in the a_backup_db structure to control enabling of backup history.

Syntax

```
enum History
```

Members

Member Name	Description
BACKUP_HISTORY_UNSPECIFIED	Use to leave HISTORY clause unspecified.
BACKUP_HISTORY_ON	Use to generate HISTORY ON clause.
BACKUP_HISTORY_OFF	Use to generate HISTORY OFF clause.

Related Information

a_backup_db Structure [page 754]
DBBackup(const a_backup_db *) Method [page 732]

1.22.2.24 InMemory Enumeration

The in-memory mode switch which should be added to the StartLine connection parameter.

Syntax

```
enum InMemory
```

Members

Member Name	Description
IM_NONE	Do not modify the StartLine connection parameter.

Member Name	Description
IM_V	Append "-im v" (in-memory validation mode) to the StartLine connection parameter. The dbvalid command line tool uses IM_V by default.
IM_C	Append "-im c" (in-memory checkpoint-only mode) to the StartLine connection parameter.
IM_NW	Append "-im nw" (in-memory never-write mode) to the StartLine connection parameter.

Related Information

a_validate_db Structure [page 793]
DBValidate(const a_validate_db *) Method [page 746]

1.22.2.25 LiveValidation Enumeration

The type of live validation being performed, as used by the a_validate_db structure.

Syntax

```
enum LiveValidation
```

Members

Member Name	Description
VALIDATE_NO_LIVE_OPTION	Validate as normal.
VALIDATE_WITH_DATA_LOCK	Obtain data locks on required tables.
VALIDATE_WITH_SNAPSHOT	Validate using Snapshots.

Related Information

a_validate_db Structure [page 793]
DBValidate(const a_validate_db *) Method [page 746]

1.22.2.26 Padding Enumeration

Blank padding enumeration specifies the blank_pad setting in a_create_db.

Syntax

```
enum Padding
```

Members

Member Name	Description
NO_BLANK_PADDING	Does not use blank padding.
BLANK_PADDING	Uses blank padding.

Related Information

a_create_db Structure [page 760]
DBCreate(a_create_db *) Method [page 734]

1.22.2.27 Unit Enumeration

Used in the a_create_db structure, to specify the value of db_size_unit.

Syntax

```
enum Unit
```

Members

Member Name	Description
DBSP_UNIT_NONE	Units not specified.
DBSP_UNIT_PAGES	Size is specified in pages.
DBSP_UNIT_BYTES	Size is specified in bytes.
DBSP_UNIT_KILOBYTES	Size is specified in kilobytes.

Member Name	Description
DBSP_UNIT_MEGABYTES	Size is specified in megabytes.
DBSP_UNIT_GIGABYTES	Size is specified in gigabytes.
DBSP_UNIT_TERABYTES	Size is specified in terabytes.

Related Information

a_create_db Structure [page 760]
DBCreate(a_create_db *) Method [page 734]

1.22.2.28 Unload Enumeration

The type of unload being performed, as used by the an_unload_db structure.

Syntax

```
enum Unload
```

Members

Member Name	Description
UNLOAD_ALL	Unload both data and schema.
UNLOAD_DATA_ONLY	Unload data. Do not unload schema. Equivalent to dbunload -d option.
UNLOAD_NO_DATA	No data. Unload schema only. Equivalent to dbunload -n option.
UNLOAD_NO_DATA_FULL_SCRIPT	No data. Include LOAD/INPUT statements in reload script. Equivalent to dbunload -nl option.
UNLOAD_NO_DATA_NAME_ORDER	No data. Objects will be output ordered by name.

Related Information

an_unload_db Structure [page 796]
DBUnload(an_unload_db *) Method [page 744]

1.22.2.29 UserList Enumeration

The type of a user list, as used by an a_translate_log structure.

Syntax

```
enum UserList
```

Members

Member Name	Description
DBTRAN_INCLUDE_ALL	Include operations from all users.
DBTRAN_INCLUDE_SOME	Include operations only from the users listed in the supplied user list.
DBTRAN_EXCLUDE_SOME	Exclude operations from the users listed in the supplied user list.

Related Information

a_translate_log Structure [page 789]
DBTranslateLog(const a_translate_log *) Method [page 743]

1.22.2.30 Validation Enumeration

The type of validation being performed, as used by the a_validate_db structure.

Syntax

```
enum Validation
```

Members

Member Name	Description
VALIDATE_NORMAL	Validate with the default check only.
VALIDATE_DATA	(obsolete)

Member Name	Description
VALIDATE_INDEX	(obsolete)
VALIDATE_EXPRESS	Validate with express check. Equivalent to dbvalid -fx option.
VALIDATE_FULL	(obsolete)
VALIDATE_CHECKSUM	Validate database checksums. Equivalent to dbvalid -s option.
VALIDATE_DATABASE	Validate database. Equivalent to dbvalid -d option.
VALIDATE_COMPLETE	Perform all possible validation activities.

Related Information

a_validate_db Structure [page 793]
DBValidate(const a_validate_db *) Method [page 746]

1.22.2.31 Verbosity Enumeration

Verbosity enumeration specifies the volume of output.

Syntax

```
enum Verbosity
```

Members

Member Name	Description
VB_QUIET	No output.
VB_NORMAL	Normal amount of output.
VB_VERBOSE	Verbose output, useful for debugging.

Related Information

a_create_db Structure [page 760]
DBCreate(a_create_db *) Method [page 734]

1.22.2.32 Version Enumeration

Used in the a_db_version_info structure, to indicate the version of SQL Anywhere that initially created the database.

Syntax

```
enum Version
```

Members

Member Name	Description	Value
VERSION_UNKNOWN	Unable to determine the version of SQL Anywhere that created the database.	0
VERSION_PRE_10	Database was created using SQL Anywhere version 9 or earlier.	9
VERSION_10	Database was created using SQL Anywhere version 10.	10
VERSION_11	Database was created using SQL Anywhere version 11.	11
VERSION_12	Database was created using SQL Anywhere version 12.	12
VERSION_16	Database was created using SQL Anywhere version 16.	16
VERSION_17	Database was created using SQL Anywhere version 17.	17

Related Information

a_db_version_info Structure [page 766]
DBCreatedVersion(a_db_version_info *) Method [page 734]

1.22.2.33 a_backup_db Structure

Holds the information needed to perform backup tasks using the DBTools library.

Syntax

```
typedef struct a_backup_db
```

Members

All members of a_backup_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public MSG_CALLBACK	confirmrtn	Address of a confirmation request callback routine or NULL.
public MSG_CALLBACK	statusrtn	Address of a status message callback routine or NULL.
public const char *	output_dir	Path to the output directory for backups, for example: "c:\backup".
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public const char *	hotlog_filename	File name for the live backup file. Set by dbbackup -l option.
public a_sql_uint32	page_blocksize	Number of pages in data blocks. If set to 0, then the default is 128. Set by dbbackup -b option.

Modifier and Type	Variable	Description
public char	chkpt_log_type	Control copying of checkpoint log. Must be one of BACKUP_CHKPT_LOG_COPY, BACKUP_CHKPT_LOG_NOCOPY, BACKUP_CHKPT_LOG_RECOVER, BACKUP_CHKPT_LOG_AUTO, or BACKUP_CHKPT_LOG_DEFAULT. Set by dbbackup -k option.
public char	backup_interrupted	Indicates that the operation was interrupted when non-zero.
public a_bit_field	backup_database	Back up the database file. Set TRUE by dbbackup -d option.
public a_bit_field	backup_logfile	Back up the transaction log file. Set TRUE by dbbackup -t option.
public a_bit_field	no_confirm	Operate without confirmation. Set TRUE by dbbackup -y option.
public a_bit_field	quiet	Operate without printing messages. Set TRUE by dbbackup -q option.
public a_bit_field	rename_log	Rename the transaction log. Set TRUE by dbbackup -r option.
public a_bit_field	truncate_log	Delete the transaction log. Set TRUE by dbbackup -x option.
public a_bit_field	rename_local_log	Rename the local backup of the transaction log. Set TRUE by dbbackup -n option.
public a_bit_field	server_backup	Perform backup on server using BACKUP DATABASE. Set TRUE by dbbackup -s option.
public a_bit_field	progress_messages	Display progress messages. Set TRUE by dbbackup -p option.
public a_bit_field	wait_before_start	Wait for open transactions to close before starting backup. Set TRUE by dbbackup -wb option.
public a_bit_field	wait_after_end	Wait for open transactions to close before finishing backup. Set TRUE by dbbackup -wa option.

Modifier and Type	Variable	Description
public const char *	backup_comment	Comment used for the WITH COMMENT clause.
public char	auto_tune_writers	Enable/disable auto tune writers. Must be one of BACKUP_AUTO_TUNE_UNSPECIFIED, BACKUP_AUTO_TUNE_ON, or BACKUP_AUTO_TUNE_OFF. Use to generate AUTO TUNE WRITERS OFF clause. Set by dbbackup -aw[-] option
public char	backup_history	Backup history. Must be one of BACKUP_HISTORY_UNSPECIFIED, BACKUP_HISTORY_ON, or BACKUP_HISTORY_OFF. Set by dbbackup -h[-] option

Related Information

Checkpoint Enumeration [page 747]
DBBackup(const a_backup_db *) Method [page 732]

1.22.2.34 a_change_log Structure

Holds the information needed to perform dblog tasks using the DBTools library.

Syntax

```
typedef struct a_change_log
```

Members

All members of a_change_log, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).

Modifier and Type	Variable	Description
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public const char *	dbname	Database file name.
public const char *	logname	Transaction log file name, or NULL if there is no log.
public const char *	mirrorname	The new name of the transaction log mirror file. Equivalent to dblog -m option.
public const char *	zap_current_offset	Change the current offset to the specified value. This is for use only in resetting a transaction log after an unload and reload to match dbremote or dbmlsync settings. Equivalent to dblog -x option.
public const char *	zap_starting_offset	Change the starting offset to the specified value. This is for use only in resetting a transaction log after an unload and reload to match dbremote or dbmlsync settings. Equivalent to dblog -z option.
public const char *	zap_timeline	Change the timeline to the specified value. This is to use in resetting transaction log after unload and reload to match dbremote or dbmlsync settings
public const char *	encryption_key	The encryption key for the database file. Equivalent to dblog -ek or -ep option.
public unsigned short	generation_number	The new generation number. Reserved, use zero.
public a_bit_field	query_only	If 1, just display the name of the transaction log. If 0, permit changing of the log name.
public a_bit_field	quiet	Operate without printing messages. Set TRUE by dblog -q option.
public a_bit_field	change_mirrorname	Set TRUE to permit changing of the mirror log name. Set TRUE by dblog -m, -n, or -r option.

Modifier and Type	Variable	Description
public a_bit_field	change_logname	Set TRUE to permit changing of the transaction log name. Set TRUE by dblog -n or -t option.
public a_bit_field	ignore_ltm_trunc	Reserved, use FALSE.
public a_bit_field	ignore_remote_trunc	For SQL Remote. Resets the offset kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed. Set TRUE by dblog -ir option.
public a_bit_field	set_generation_number	Reserved. Use FALSE.
public a_bit_field	ignore_dbsync_trunc	When using dbmlsync, resets the offset kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed. Set TRUE by dblog -is option.
public a_sql_uint64	starting_offset	Set by DBChangeLogName to the starting offset. This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.
public a_sql_uint64	current_relative_offset	If dbname is a database file, set by DBChangeLogName to the current relative offset. This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.
public a_sql_uint64	end_offset	If dbname is a transaction log file, set by DBChangeLogName to the end offset. This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.

Related Information

DBChangeLogName(a_change_log *) Method [page 733]

1.22.2.35 a_create_db Structure

Holds the information needed to create a database using the DBTools library.

Syntax

```
typedef struct a_create_db
```

Members

All members of a_create_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public const char *	dbname	Database file name.
public const char *	logname	New transaction log name. Set NULL is equivalent to dbinit -n option; otherwise must be set.
public const char *	startline	The command line used to start the database server. For example: "c:\SQLAny17\bin32\dbsrv17.exe". If NULL, the default START parameter is "dbeng17 -gp <page_size> -c 10M" for SQL Anywhere where page_size is specified below. Note that "-c 10M" is appended if page_size >= 2048.
public const char *	default_collation	The collation for the database. Equivalent to dbinit -z option.
public const char *	nchar_collation	The NCHAR COLLATION for the database when not NULL. Equivalent to dbinit -zn option.
public const char *	encoding	The character set encoding. Equivalent to dbinit -ze option.
public const char *	mirrorname	Transaction log mirror name. Equivalent to dbinit -m option.

Modifier and Type	Variable	Description
public const char *	data_store_type	Reserved. Use NULL.
public const char *	encryption_key	The encryption key for the database file. Used with encrypt, it generates the KEY clause. Equivalent to dbinit -ek option.
public const char *	encryption_algorithm	The encryption algorithm (AES, AES256, AES_FIPS, or AES256_FIPS). Used with encrypt and encryption_key, it generates the ALGORITHM clause. Equivalent to dbinit -ea option.
public void *	iq_params	Reserved. Use NULL.
public int	db_size_unit	Used with db_size, must be one of DBSP_UNIT_NONE, DBSP_UNIT_PAGES, DBSP_UNIT_BYTES, DBSP_UNIT_KILOBYTES, DBSP_UNIT_MEGABYTES, DBSP_UNIT_GIGABYTES, or DBSP_UNIT_TERABYTES. When not DBSP_UNIT_NONE, it generates the corresponding keyword (for example, DATABASE SIZE 10 MB is generated when db_size is 10 and db_size_unit is DBSP_UNIT_MEGABYTES). See Database size unit enumeration.
public unsigned int	db_size	When not 0, generates the DATABASE SIZE clause. Equivalent to dbinit -dbs option.
public unsigned short	page_size	The page size of the database. Equivalent to dbinit -p option.
public char	verbose	See Verbosity enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).
public char	accent_sensitivity	One of 'y', 'n', or 'f' (yes, no, French). Generates one of the ACCENT RESPECT, ACCENT IGNORE or ACCENT FRENCH clauses.
public char *	dba_uid	When not NULL, generates the DBA USER xxx clause. Equivalent to dbinit -dba option.
public char *	dba_pwd	When not NULL, generates the DBA PASSWORD xxx clause. Equivalent to dbinit -dba option.

Modifier and Type	Variable	Description
public a_bit_field	blank_pad	Must be one of NO_BLANK_PADDING or BLANK_PADDING. Treat blanks as significant in string comparisons and hold index information to reflect this. See Blank padding enumeration. Equivalent to dbinit -b option.
public a_bit_field	respect_case	Make string comparisons case sensitive and hold index information to reflect this. Set TRUE by dbinit -c option.
public a_bit_field	encrypt	Set TRUE to generate the ENCRYPTED ON clause or, when encrypted_tables is also set, the ENCRYPTED TABLES ON clause. Set TRUE by dbinit -e? options.
public a_bit_field	avoid_view_collisions	Set TRUE to omit the generation of Watcom SQL compatibility views SYS.SYSCOLUMNS and SYS.SYSINDEXES. Set TRUE by dbinit -k option.
public a_bit_field	jconnect	Set TRUE to include system procedures needed for jConnect. Set FALSE by dbinit -i option.
public a_bit_field	checksum	Set to TRUE for ON or FALSE for OFF. Generates one of CHECKSUM ON or CHECKSUM OFF clauses. Set TRUE by dbinit -s option.
public a_bit_field	encrypted_tables	Set TRUE to encrypt tables. Used with encrypt, it generates the ENCRYPTED TABLE ON clause instead of the ENCRYPTED ON clause. Set TRUE by dbinit -et option.
public a_bit_field	case_sensitivity_use_default	Set TRUE to use the default case sensitivity for the locale. This only affects UCA. If set TRUE then we do not add the CASE RESPECT clause to the CREATE DATABASE statement.

Modifier and Type	Variable	Description
public a_bit_field	sys_proc_definer	Set TRUE to retain the SQL SECURITY Model for version 12.0.1 or earlier system stored procedures. Set TRUE by dbinit -pd option.
public unsigned short	min_pwd_len	The minimum length for the new passwords in the database. This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.

Related Information

Padding Enumeration [page 750]
Unit Enumeration [page 750]
Verbosity Enumeration [page 753]
DBCreate(a_create_db *) Method [page 734]

1.22.2.36 a_db_info Structure

Holds the information needed to return DBInfo information using the DBTools library.

Syntax

```
typedef struct a_db_info
```

Members

All members of a_db_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.

Modifier and Type	Variable	Description
public MSG_CALLBACK	statusrtn	Address of a status message callback routine or NULL.
public a_table_info *	totals	Pointer to a_table_info structure.
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public char *	dbnamebuffer	Pointer to the database file name buffer.
public char *	lognamebuffer	Pointer to the transaction log file name buffer.
public char *	mirrornamebuffer	Pointer to the mirror file name buffer.
public char *	charcollationspecbuffer	Pointer to the char collation string buffer.
public char *	charencodingbuffer	Pointer to the char encoding string buffer.
public char *	ncharcollationspecbuffer	Pointer to the nchar collation string buffer.
public char *	ncharencodingbuffer	Pointer to the nchar encoding string buffer.
public unsigned short	dbbufsize	Size of dbnamebuffer (for example, _MAX_PATH).
public unsigned short	logbufsize	Size of lognamebuffer (for example, _MAX_PATH).
public unsigned short	mirrorbufsize	Size of mirrornamebuffer (for example, _MAX_PATH).
public unsigned short	charcollationspecbufsize	Size of charcollationspecbuffer (at least 256+1).
public unsigned short	charencodingbufsize	Size of charencodingbuffer (at least 50+1).
public unsigned short	ncharcollationspecbufsize	Size of ncharcollationspecbuffer (at least 256+1).

Modifier and Type	Variable	Description
public unsigned short	ncharencodingbufsize	Size of ncharencodingbuffer (at least 50+1).
public a_sysinfo	sysinfo	Inline a_sysinfo structure.
public a_sql_uint32	file_size	Size of database file (in pages).
public a_sql_uint32	free_pages	Number of free pages.
public a_sql_uint32	bit_map_pages	Number of bitmap pages in the database.
public a_sql_uint32	other_pages	Number of pages that are not table pages, index pages, free pages, or bitmap pages.
public a_bit_field	quiet	Set TRUE to operate without confirming messages. Set TRUE by dbinfo -q option.
public a_bit_field	page_usage	Set TRUE to report page usage statistics, otherwise FALSE. Set TRUE by dbinfo -u option.
public a_bit_field	checksum	If set TRUE, global checksums are enabled (a checksum on every database page).
public a_bit_field	encrypted_tables	If set TRUE, encrypted tables are supported.
public a_bit_field	pitr_alternate_timelines	If set TRUE, Point in Time Recovery with Alternate Timelines is supported This field was added in DB_TOOLS_VERSION_IQSP09 but is borrowing an available bit from the previous version of the structure.
public char *	current_timeline_guid	These fields were added in DB_TOOLS_VERSION_IQSP09 (16001) GUID of current timeline Leave NULL if not wanted
public size_t	current_timeline_guid_buffer_size	Size of current_timeline_guid buffer.
public char *	current_timeline_utc_creation_time	Buffer to store current timeline's UTC creation time as a string Leave NULL if not wanted.
public size_t	current_timeline_utc_creation_time_buffer_size	Size of current_timeline_utc_creation_time_buffer.
public char *	previous_timeline_guid	GUID of previous timeline Leave NULL if not wanted.
public size_t	previous_timeline_guid_buffer_size	Size of current_timeline_guid buffer.

Modifier and Type	Variable	Description
public char *	previous_timeline_utc_creation_time	Buffer to store previous timeline's UTC creation time as a string Leave NULL if not wanted.
public size_t	previous_timeline_utc_creation_time_buffer_size	Size of previous_timeline_utc_creation_time_buffer.
public a_sql_uint64	previous_timeline_branch_log_offset	Log offset within the previous timeline at which the current timeline branched.
public char *	current_translog_guid	GUID of current transaction log Leave NULL if not wanted.
public size_t	current_translog_guid_buffer_size	Size of current_translog_guid buffer.
public char *	previous_translog_guid	GUID of previous transaction log Leave NULL if not wanted.
public size_t	previous_translog_guid_buffer_size	Size of previous_translog_guid buffer.

Related Information

a_table_info Structure [page 788]
a_sysinfo Structure [page 787]
DBInfo(a_db_info *) Method [page 736]
DBInfoDump(a_db_info *) Method [page 737]
DBInfoFree(a_db_info *) Method [page 737]

1.22.2.37 a_db_version_info Structure

Holds information regarding which version of SQL Anywhere was used to create the database.

Syntax

```
typedef struct a_db_version_info
```

Members

All members of a_db_version_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public const char *	filename	Name of the database file to check.
public char	created_version	Set to one of VERSION_UNKNOWN, VERSION_PRE_10, etc. indicating the server version that created the database file.

Related Information

DBCreatedVersion(a_db_version_info *) Method [page 734]
Version Enumeration [page 754]

1.22.2.38 a_dblic_info Structure

Holds information containing licensing information.

Syntax

```
typedef struct a_dblic_info
```

Members

All members of a_dblic_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.

Modifier and Type	Variable	Description
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public char *	exename	Name of the server executable or license file.
public char *	username	User name for licensing.
public char *	compname	Company name for licensing.
public a_sql_int32	nodecount	Number of nodes licensed.
public a_sql_int32	conncount	Maximum number of connections licensed. To set, use 1000000L for default.
public a_license_type	type	See lictype.h for values. One of LICENSE_TYPE_PERSEAT, LICENSE_TYPE_CONCURRENT, or LICENSE_TYPE_PERCPU.
public char *	installkey	Reserved; set NULL. Set by dblic -k option.
public a_bit_field	quiet	Set TRUE to operate without printing messages. Set TRUE by dblic -q option.
public a_bit_field	query_only	Set TRUE to just display the license information. Set FALSE to permit changing the information.

Remarks

You must use this information only in a manner consistent with your license agreement.

Related Information

DBLicense(const a_dblic_info *) Method [page 738]

1.22.2.39 a_dbtools_info Structure

DBTools information callback used to initialize and finalize the DBTools library calls.

Syntax

```
typedef struct a_dbtools_info
```

Members

All members of a_dbtools_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.

Related Information

DBToolsInit(const a_dbtools_info *) Method [page 741]
DBToolsFini(const a_dbtools_info *) Method [page 741]

1.22.2.40 a_log_file_info Structure

Used to obtain the log file and mirror log file information of a non-running database.

Syntax

```
typedef struct a_log_file_info
```

Members

All members of a_log_file_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public const char *	dbname	Database file name.
public const char *	encryption_key	The encryption key for the database file.
public char *	logname	Buffer for transaction log file name, or NULL.
public size_t	logname_size	Size of buffer for transaction log file name, or zero.
public char *	mirrorname	Buffer for mirror log file name, or NULL.
public size_t	mirrorname_size	Size of buffer for mirror log file name, or zero.
public void *	reserved	Reserved for internal use and must set to NULL.

Related Information

DBLogFileInfo(const a_log_file_info *) Method [page 739]

1.22.2.41 a_name Structure

Specifies a variable list of names.

Syntax

```
typedef struct a_name
```

Members

All members of a_name, including inherited members.

Variables

Modifier and Type	Variable	Description
public struct a_name *	next	Pointer to the next name in the list or NULL.
public char	name	One or more bytes comprising the name.

1.22.2.42 a_remote_sql Structure

Holds information needed for the dbremote utility using the DBTools library.

Syntax

```
typedef struct a_remote_sql
```

Members

All members of a_remote_sql, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	confirmrtn	Address of a confirmation request call-back routine or NULL.
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.

Modifier and Type	Variable	Description
public MSG_QUEUE_CALLBACK	msgqueuertn	Function called by DBRemoteSQL when it wants to sleep. The parameter specifies the sleep period in milliseconds. The function should return the following, as defined in dllapi.h. • MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. This is usually the value you should return. • MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible.
public char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbeng17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbeng17.exe".
public char *	transaction_logs	Should identify the directory with offline transaction logs (DBRemoteSQL only). Corresponds to the transaction_logs_directory argument of dbremote.
public a_bit_field	receive	When set TRUE, messages are received. If receive and send are both FALSE then both are assumed TRUE. It is recommended to set receive and send FALSE. Corresponds to the dbremote -r option.

Modifier and Type	Variable	Description
public a_bit_field	send	When set TRUE, messages are sent. If receive and send are both FALSE then both are assumed TRUE. It is recommended to set receive and send FALSE. Corresponds to the dbremote -s option.
public a_bit_field	verbose	When set, extra information is produced. Corresponds to the dbremote -v option.
public a_bit_field	deleted	Normally set TRUE. When not set, messages are not deleted after they are applied. Corresponds to dbremote -p option.
public a_bit_field	apply	Normally set TRUE. When not set, messages are scanned but not applied. Corresponds to dbremote -a option.
public a_bit_field	batch	When set TRUE, force exit after applying message and scanning log (this is the same as at least one user having 'always' send time). When cleared, allow run mode to be determined by remote users send times.
public a_bit_field	more	This should be set to TRUE.
public a_bit_field	triggers	This should usually be cleared (FALSE) in most cases. When set TRUE, trigger actions are replicated. Care should be exercised.
public a_bit_field	debug	When set TRUE, debug output is included.
public a_bit_field	rename_log	When set TRUE, logs are renamed and restarted (DBRemoteSQL only).
public a_bit_field	latest_backup	When set TRUE, only logs that are backed up are processed. Don't send operations from a live log. Corresponds to the dbremote -u option.
public a_bit_field	scan_log	Reserved for internal use and must set to FALSE.
public a_bit_field	link_debug	When set TRUE, debugging will be turned on for links.

Modifier and Type	Variable	Description
public a_bit_field	full_q_scan	Reserved for internal use and must set to FALSE.
public a_bit_field	no_user_interaction	When set TRUE, no user interaction is requested.
public a_bit_field	unused	Reserved for internal use and must set to FALSE.
public a_sql_uint32	max_length	Set to the maximum length (in bytes) a message can have. This affects sending and receiving. The recommended value is 50000. Corresponds to the dbremote -l option.
public a_sql_uint32	memory	Set to the maximum size (in bytes) of memory buffers to use while building messages to send. The recommended value is at least 2 * 1024 * 1024. Corresponds to the dbremote -m option.
public a_sql_uint32	frequency	Reserved for internal use and must set to 0.
public a_sql_uint32	threads	Set the number of worker threads that should be used to apply messages. This value must not exceed 50. Corresponds to the dbremote -w option.
public a_sql_uint32	operations	This value is used when applying messages. Commits are ignored until DBRemoteSQL has at least this number of operations(inserts, deletes, updates) that are uncommitted. Corresponds to the dbremote -g option.
public char *	queueparms	Reserved for internal use and must set to NULL.
public char *	locale	Reserved for internal use and must set to NULL.
public a_sql_uint32	receive_delay	Set this to the time (in seconds) to wait between polls for new incoming messages. The recommended value is 60. Corresponds to the dbremote -rd option.

Modifier and Type	Variable	Description
public a_sql_uint32	patience_retry	Set this to the number of polls for incoming messages that DBRemoteSQL should wait before assuming that a message it is expecting is lost. For example, if patience_retry is 3 then DBRemoteSQL tries up to three times to receive the missing message. Afterward, it sends a resend request. The recommended value is 1. Corresponds to the dbremote -rp option.
public MSG_CALLBACK	logrtn	Pointer to a function that prints the given message to a log file. These messages do not need to be seen by the user.
public a_bit_field	use_hex_offsets	When set TRUE, log offsets are shown in hexadecimal notation; otherwise decimal notation is used.
public a_bit_field	use_relative_offsets	When set TRUE, log offsets are displayed as relative to the start of the current log file. When set FALSE, log offsets from the beginning of time are displayed.
public a_bit_field	debug_page_offsets	Reserved for internal use and must set to FALSE.
public a_sql_uint32	debug_dump_size	Reserved for internal use and must set to 0.
public a_sql_uint32	send_delay	Set the time (in seconds) between scans of the log file for new operations to send. Set to zero to allow DBRemoteSQL to choose a good value based on user send times. Corresponds to the dbremote -sd option.
public a_sql_uint32	resend_urgency	Set the time (in seconds) that DBRemoteSQL waits after seeing that a user needs a rescan before performing a full scan of the log. Set to zero to allow DBRemoteSQL to choose a good value based on user send times and other information it has collected. Corresponds to the dbremote -ru option.
public char *	include_scan_range	Reserved for internal use and must set to NULL.

Modifier and Type	Variable	Description
public SET_WINDOW_TITLE_CALL-BACK	set_window_title_rtn	Pointer to a function that resets the title of the window (Windows only). The title could be "database_name (receiving, scanning, or sending) - default_window_title".
public char *	default_window_title	A pointer to the default window title string.
public MSG_CALLBACK	progress_msg_rtn	Pointer to a function that displays a progress message.
public SET_PROGRESS_CALLBACK	progress_index_rtn	Pointer to a function that updates the state of the progress bar. This function takes two unsigned integer arguments index and max. On the first call, the values are the minimum and maximum values (for example, 0, 100). On subsequent calls, the first argument is the current index value (for example, between 0 and 100) and the second argument is always 0.
public char **	argv	Pointer to a parsed command line (a vector of pointers to strings). If not NULL, then DBRemoteSQL will call a message routine to display each command line argument except those prefixed with -c, -cq, or -ek.
public a_sql_uint32	log_size	DBRemoteSQL renames and restarts the online transaction log when the size of the online transaction log is greater than this value. Corresponds to the dbremote -x option.
public char *	encryption_key	Pointer to an encryption key. Corresponds to the dbremote -ek option.
public const char *	log_file_name	Pointer to the name of the DBRemoteSQL output log to which the message callbacks print their output. If send is TRUE, the error log is sent to the consolidated (unless this pointer is NULL).
public a_bit_field	truncate_remote_output_file	When set TRUE, the remote output file is truncated rather than appended to. Corresponds to the dbremote -rt option.

Modifier and Type	Variable	Description
public char *	remote_output_file_name	Pointer to the name of the DBRemoteSQL remote output file. Corresponds to the dbremote -ro or -rt option.
public MSG_CALLBACK	warningrtn	Pointer to a function that displays the given warning message. If NULL, the errorrtn function is called instead.
public char *	mirror_logs	Pointer to the name of the directory containing offline mirror transaction logs. Corresponds to the dbremote -ml option.

Remarks

The dbremote utility sets the following defaults before processing any command-line options:

- version = DB_TOOLS_VERSION_NUMBER
- argv = (argument vector passed to application)
- 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 = (set from command line)
- truncate_remote_output_file = FALSE
- remote_output_file_name = NULL
- no_user_interaction = TRUE (if user interface is not available)
- errorrtn = (address of an appropriate routine)
- msgrtn = (address of an appropriate routine)
- confirmrtn = (address of an appropriate routine)

- msgqueuertn = (address of an appropriate routine)
- logrtn = (address of an appropriate routine)
- warningrtn = (address of an appropriate routine)
- set_window_title_rtn = (address of an appropriate routine)
- progress_msg_rtn = (address of an appropriate routine)
- progress_index_rtn = (address of an appropriate routine)

Related Information

DBRemoteSQL(a_remote_sql *) Method [page 739]

1.22.2.43 a_sync_db Structure

Holds information needed for the dbmlsync utility using the DBTools library.

Syntax

```
typedef struct a_sync_db
```

Members

All members of a_sync_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public MSG_CALLBACK	confirmrtn	Address of a confirmation request callback routine or NULL.
public MSG_CALLBACK	logrtn	Address of a logging callback routine to write messages only to a log file or NULL.
public SET_WINDOW_TITLE_CALLBACK	set_window_title_rtn	Function to call to change the title of the dbmlsync window (Windows only).

Modifier and Type	Variable	Description
public MSG_QUEUE_CALLBACK	msgqueuertn	Function called by DBMLSync when it wants to sleep. The parameter specifies the sleep period in milliseconds. The function should return the following, as defined in dllapi.h. • MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. This is usually the value you should return. • MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible. • MSGQ_SYNC_REQUESTED indicates that the routine slept for less than the requested number of milliseconds and that the next synchronization should begin immediately if a synchronization is not currently in progress.
public MSG_CALLBACK	progress_msg_rtn	Function called to change the text in the status window, above the progress bar.
public SET_PROGRESS_CALLBACK	progress_index_rtn	Function called to update the state of the progress bar.
public USAGE_CALLBACK	usage_rtn	Reserved; use NULL.
public STATUS_CALLBACK	status_rtn	Reserved; use NULL.
public MSG_CALLBACK	warningrtn	Function called to display warning messages.
public a_syncpub *	upload_defs	Linked list of publications/subscriptions to synchronize.
public a_syncpub *	last_upload_def	Reserved; use NULL.
public const char *	offline_dir	Transaction logs directory. Last item specified on dbmlsync command line.
public const char *	include_scan_range	Reserved; use NULL.
public const char *	raw_file	Reserved; use NULL.
public const char *	log_file_name	Database server message log file name. Equivalent to dbmlsync -o or -ot option.

Modifier and Type	Variable	Description
public const char *	apply_dnld_file	Name of download file to apply. Equivalent to dbmlsync -ba option or NULL if option not specified.
public const char *	create_dnld_file	Name of download file to create. Equivalent to dbmlsync -bc option or NULL if option not specified.
public const char *	dnld_file_extra	Specify extra string to include in download file. Equivalent to dbmlsync -be option.
public const char *	encrypted_stream_opts	Reserved; use NULL.
public char **	argv	The argv array for this run, the last element of the array must be NULL.
public char **	ce_argv	Reserved; use NULL.
public char **	ce_reproc_argv	Reserved; use NULL.
public char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public char *	extended_options	Extended options in the form "keyword=value;...". Equivalent to dbmlsync -e option.
public char *	default_window_title	Name of the program to display in the window caption (for example, DBMLSync).
public char *	mlpassword	The MobiLink password or NULL, if the option is not specified. Equivalent to the dbmlsync -mp option.

Modifier and Type	Variable	Description
public char *	new_mlpassword	The new MobiLink password or NULL, if the option is not specified. Equivalent to the dbmlsync -mn option.
public char *	encryption_key	The encryption key for the database file. Equivalent to the dbmlsync -ek option.
public char *	user_name	The MobiLink user to synchronize (deprecated). Equivalent to the dbmlsync -u option.
public char *	sync_params	User authentication parameters. Equivalent to the dbmlsync -ap option.
public char *	preload_dlls	Reserved; use NULL.
public char *	sync_profile	Synchronization profile to execute. Equivalent to the dbmlsync -sp option.
public char *	sync_opt	Reserved; use NULL.
public a_sql_uint32	no_offline_logscan	Set TRUE to disable offline logscan (cannot use with -x). Equivalent to the dbmlsync -do option.
public a_sql_uint32	debug_dump_size	Reserved; use 0.
public a_sql_uint32	dl_insert_width	Reserved; use 0.
public a_sql_uint32	log_size	Size in bytes of log file when renaming and restarting the transaction log. Specify 0 for unspecified size. Equivalent to the dbmlsync -x option.
public a_sql_uint32	hovering_frequency	Set the logscan polling period in seconds. Usually 60. Equivalent to the dbmlsync -pp option.
public a_sql_uint32	est_upld_row_cnt	Set the estimated upload row count (for optimization). Equivalent to the dbmlsync -urc option.
public a_sql_uint32	dnld_read_size	Set the download read size. Equivalent to the dbmlsync -drs option.
public a_sql_uint32	dnld_fail_len	Reserved; use 0.
public a_sql_uint32	upld_fail_len	Reserved; use 0.

Modifier and Type	Variable	Description
public a_sql_uint32	server_port	Set communication port when running in server mode. Equivalent to the dbmlsync -po option.
public a_sql_uint32	dlg_info_msg	Reserved; use 0.
public a_sql_uint32	min_cache	Minimum size for cache. Equivalent to the dbmlsync -cl option.
public char	min_cache_suffix	Suffix for minimum cache size ('B' for bytes, 'P' for percentage, or 0 if not specified.
public a_sql_uint32	max_cache	Maximum size for cache. Equivalent to the dbmlsync -cm option.
public char	max_cache_suffix	Suffix for maximum cache size ('B' for bytes, 'P' for percentage, or 0 if not specified.
public a_sql_uint32	init_cache	Initial size for cache. Equivalent to the dbmlsync -ci option.
public char	init_cache_suffix	Suffix for initial cache size ('B' for bytes, 'P' for percentage, or 0 if not specified.
public a_sql_int32	background_retry	Number of times to retry an interrupted background synchronization. Equivalent to the dbmlsync -bkr option.
public a_bit_field	ping	Set TRUE to ping MobiLink server. Equivalent to the dbmlsync -pi option.
public a_bit_field	dnld_gen_num	Set TRUE to update generation number when download file is applied. Equivalent to the dbmlsync -bg option.
public a_bit_field	background_sync	Set TRUE to do a background synchronization. Equivalent to the dbmlsync -bk option.
public a_bit_field	kill_other_connections	Set TRUE to drop connections with locks on tables being synchronized. Equivalent to the dbmlsync -d option.
public a_bit_field	continue_download	Set TRUE to continue a previously failed download. Equivalent to the dbmlsync -dc option.
public a_bit_field	download_only	Set TRUE to perform download-only synchronization. Equivalent to the dbmlsync -ds option.

Modifier and Type	Variable	Description
public a_bit_field	ignore_hook_errors	Set TRUE to ignore errors that occur in hook functions. Equivalent to the dbmlsync -eh option.
public a_bit_field	ignore_scheduling	Set TRUE to ignore scheduling. Equivalent to the dbmlsync -is option.
public a_bit_field	autoclose	Set TRUE to close window on completion. Equivalent to the dbmlsync -qc option.
public a_bit_field	changing_pwd	Set TRUE when setting a new MobiLink password. See new_mlpassword field. Equivalent to the dbmlsync -mn option.
public a_bit_field	ignore_hovering	Set TRUE to disable logscan polling. Equivalent to the dbmlsync -p option.
public a_bit_field	persist_connection	Set TRUE to persist the MobiLink connection between synchronizations. Set FALSE to close the MobiLink connection between synchronizations. Equivalent to the dbmlsync -pc{+|-} option.
public a_bit_field	allow_schema_change	Set TRUE to check for schema changes between synchronizations. Equivalent to the dbmlsync -sc option.
public a_bit_field	retry_remote_behind	Set TRUE to resend upload using remote offset on progress mismatch. when remote offset is less than consolidated offset. Equivalent to the dbmlsync -r or -rb option.
public a_bit_field	server_mode	Set TRUE to run in server mode. Equivalent to the dbmlsync -sm option.
public a_bit_field	trans_upload	Set TRUE to upload each database transaction separately. Equivalent to the dbmlsync -tu option.
public a_bit_field	retry_remote_ahead	Set TRUE to resend upload using remote offset on progress mismatch when remote offset is greater than consolidated offset. Equivalent to the dbmlsync -ra option.

Modifier and Type	Variable	Description
public a_bit_field	upload_only	Set TRUE to perform upload-only synchronization. Equivalent to the dbmlsync -uo option.
public a_bit_field	verbose_minimum	Set TRUE to set verbosity at a minimum. Equivalent to the dbmlsync -v option.
public a_bit_field	hide_conn_str	Set FALSE to show connect string, TRUE to hide the connect string. Equivalent to the dbmlsync -vc option.
public a_bit_field	hide_ml_pwd	Set FALSE to show MobiLink password, TRUE to hide the MobiLink password. Equivalent to the dbmlsync -vp option.
public a_bit_field	verbose_row_cnts	Set TRUE to show upload/download row counts. Equivalent to the dbmlsync -vn option.
public a_bit_field	verbose_option_info	Set TRUE to show command line and extended options. Equivalent to the dbmlsync -vo option.
public a_bit_field	verbose_row_data	Set TRUE to show upload/download row values. Equivalent to the dbmlsync -vr option.
public a_bit_field	verbose_hook	Set TRUE to show hook script information. Equivalent to the dbmlsync -vs option.
public a_bit_field	verbose_upload	Set TRUE to show upload stream information. Equivalent to the dbmlsync -vu option.
public a_bit_field	verbose_msgid	Set TRUE to show message IDs. Equivalent to the dbmlsync -vi option.
public a_bit_field	rename_log	Set TRUE to rename and restart the transaction log. See log_size field. Equivalent to the dbmlsync -x option.
public a_bit_field	keep_partial_download	Set TRUE when restarting failed downloads should be allowed Equivalent to the dbmlsync -kpd option.
public a_bit_field	allow_outside_connect	Reserved; use 0.
public a_bit_field	cache_verbosity	Reserved; use 0.

Modifier and Type	Variable	Description
public a_bit_field	connectparms_allocated	Reserved; use 0.
public a_bit_field	debug	Reserved; use 0.
public a_bit_field	debug_dump_char	Reserved; use 0.
public a_bit_field	debug_dump_hex	Reserved; use 0.
public a_bit_field	debug_page_offsets	Reserved; use 0.
public a_bit_field	dl_use_put	Reserved; use 0.
public a_bit_field	entered_dialog	Reserved; use 0.
public a_bit_field	ignore_debug_interrupt	Reserved; use 0.
public a_bit_field	lite_blob_handling	Reserved; use 0.
public a_bit_field	no_schema_cache	Reserved; use 0.
public a_bit_field	no_stream_compress	Reserved; use 0.
public a_bit_field	output_to_file	Reserved; use 0.
public a_bit_field	output_to_mobile_link	Reserved; use 1.
public a_bit_field	prompt_again	Reserved; use 0.
public a_bit_field	prompt_for_encrypt_key	Reserved; use 0.
public a_bit_field	strictly_ignore_trigger_ops	Reserved; use 0.
public a_bit_field	use_fixed_cache	Reserved; use 0.
public a_bit_field	use_hex_offsets	Reserved; use 0.
public a_bit_field	use_relative_offsets	Reserved; use 0.
public a_bit_field	used_dialog_allocation	Reserved; use 0.
public a_bit_field	verbose	Reserved; use 0.
public a_bit_field	verbose_download	Reserved; use 0.
public a_bit_field	verbose_download_data	Reserved; use 0.
public a_bit_field	verbose_protocol	Reserved; use 0.
public a_bit_field	verbose_server	Reserved; use 0.
public a_bit_field	verbose_upload_data	Reserved; use 0.
public a_bit_field	protocol_add_cli_bit_to_cli_max	Reserved; use 0.
public a_bit_field	protocol_add_cli_bit_to_cli_both	Reserved; use 0.
public a_bit_field	protocol_add_serv_bit_to_cli_max	Reserved; use 0.
public a_bit_field	protocol_add_serv_bit_to_cli_both	Reserved; use 0.
public a_bit_field	protocol_add_serv_bit_to_serv_max	Reserved; use 0.
public a_bit_field	protocol_add_serv_bit_to_serv_both	Reserved; use 0.
public a_bit_field	strictly_free_memory	Reserved; use 0.
public a_bit_field	reserved	Reserved; use 0.

Remarks

Some members correspond to features accessible from the dbmlsync command line utility. Unused members should be assigned the value 0, FALSE, or NULL, depending on data type.

Related Information

DBSynchronizeLog(const a_sync_db *) Method [page 740]

1.22.2.44 a_syncpub Structure

Holds information needed for the dbmlsync utility.

Syntax

```
typedef struct a_syncpub
```

Members

All members of a_syncpub, including inherited members.

Variables

Modifier and Type	Variable	Description
public struct a_syncpub *	next	Pointer to the next node in the list, NULL for the last node.
public char *	pub_name	Publication name(s) separated by commas (deprecated). This is the same string that would follow the dbmlsync -n option. Only 1 of pub_name and subscription may be non-NULL.
public char *	subscription	Subscription name(s) separated by commas. This is the same string the would follow the dbmlsync -s option. Only 1 of pub_name and subscription may be non-NULL.

Modifier and Type	Variable	Description
public char *	ext_opt	Extended options in the form "keyword=value;...". These are the same options the would follow the dbmlsync -eu option.

Related Information

a_sync_db Structure [page 778]
DBSynchronizeLog(const a_sync_db *) Method [page 740]

1.22.2.45 a_sysinfo Structure

Holds information needed for dbinfo and dbunload utilities using the DBTools library.

Syntax

```
typedef struct a_sysinfo
```

Members

All members of a_sysinfo, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	page_size	The page size for the database.
public char	default_collation	The collation sequence for the database.
public a_bit_field	valid_data	1 to indicate that the other bit fields are valid.
public a_bit_field	blank_padding	1 if blank padding is used in this database, 0 otherwise.
public a_bit_field	case_sensitivity	1 if the database is case sensitive, 0 otherwise.
public a_bit_field	encryption	1 if the database is encrypted, 0 otherwise.

Related Information

a_db_info Structure [page 763]
DBInfo(a_db_info *) Method [page 736]
DBInfoDump(a_db_info *) Method [page 737]
DBInfoFree(a_db_info *) Method [page 737]

1.22.2.46 a_table_info Structure

Holds information about a table needed as part of the a_db_info structure.

Syntax

```
typedef struct a_table_info
```

Members

All members of a_table_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public struct a_table_info *	next	Next table in the list.
public char *	table_name	Name of the table.
public a_sql_uint32	table_id	ID number for this table.
public a_sql_uint32	table_pages	Number of table pages.
public a_sql_uint32	index_pages	Number of index pages.
public a_sql_uint32	table_used	Number of bytes used in table pages.
public a_sql_uint32	index_used	Number of bytes used in index pages.
public a_sql_uint32	table_used_pct	Table space utilization as a percentage.
public a_sql_uint32	index_used_pct	Index space utilization as a percentage.

Related Information

a_db_info Structure [page 763]
DBInfo(a_db_info *) Method [page 736]
DBInfoDump(a_db_info *) Method [page 737]
DBInfoFree(a_db_info *) Method [page 737]

1.22.2.47 a_translate_log Structure

Holds information needed for transaction log translation using the DBTools library.

Syntax

```
typedef struct a_translate_log
```

Members

All members of a_translate_log, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public MSG_CALLBACK	confirmrtn	Address of a confirmation request callback routine or NULL.
public MSG_CALLBACK	statusrtn	Address of a status message callback routine or NULL.
public MSG_CALLBACK	logrtn	Address of a logging callback routine to write messages only to a log file or NULL.
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".

Modifier and Type	Variable	Description
public const char *	logname	Name of the transaction log file. If NULL, there is no log.
public const char *	sqlname	Name of the SQL output file. If NULL, then the name is based on the transaction log file name. Equivalent to dbtran -n option.
public const char *	encryption_key	The encryption key for the database file. Equivalent to dbtran -ek option.
public const char *	logs_dir	Transaction logs directory. Equivalent to dbtran -m option. The sqlname pointer must be set and connectparms must be NULL.
public const char *	include_source_sets	Reserved, use NULL.
public const char *	include_destination_sets	Reserved, use NULL.
public const char *	include_scan_range	Reserved, use NULL.
public const char *	repserver_users	Reserved, use NULL.
public const char *	include_tables	Reserved, use NULL.
public const char *	include_publications	Reserved, use NULL.
public const char *	queueparms	Reserved, use NULL.
public const char *	match_pos	Reserved, use NULL.
public const char *	display_timelines_json	Used for the -htj option.
public const char *	display_timelines_text	Used for the -htt option.
public const char *	force_timeline_guid	Used for the -hft option.
public p_name	userlist	A linked list of user names. Equivalent to dbtran -u user1,... or -x user1,... Select or omit transactions for listed users.
public a_sql_uint32	since_time	Output from most recent checkpoint before time. The number of minutes since January 1, 0001. Equivalent to dbtran -j option.
public a_sql_uint32	debug_dump_size	Reserved, use 0.
public a_sql_uint32	recovery_ops	Reserved, use 0.
public a_sql_uint32	recovery_bytes	Reserved, use 0.

Modifier and Type	Variable	Description
public char	userlisttype	Set to DBTRAN_INCLUDE_ALL unless you want to include or exclude a list of users. DBTRAN_INCLUDE_SOME for -u, or DBTRAN_EXCLUDE_SOME for -x.
public a_bit_field	quiet	Set to TRUE to operate without printing messages. Set TRUE by dbtran -q option.
public a_bit_field	remove_rollback	Set to FALSE if you want to include roll-back transactions in output. Set FALSE by dbtran -a option.
public a_bit_field	ansi_sql	Set TRUE to produce ANSI standard SQL transactions. Set TRUE by dbtran -s option.
public a_bit_field	since_checkpoint	Set TRUE for output from most recent checkpoint. Set TRUE by dbtran -f option.
public a_bit_field	replace	Set TRUE to replace the SQL file without a confirmation. Set TRUE by dbtran -y option.
public a_bit_field	include_trigger_trans	Set TRUE to include trigger-generated transactions. Set TRUE by dbtran -t, -g and -sr options.
public a_bit_field	comment_trigger_trans	Set TRUE to include trigger-generated transactions as comments. Set TRUE by dbtran -z option.
public a_bit_field	leave_output_on_error	Set TRUE to leave the generated SQL file if log error detected. Set TRUE by dbtran -k option.
public a_bit_field	debug	Reserved; set to FALSE.
public a_bit_field	debug_sql_remote	Reserved; set to FALSE.
public a_bit_field	debug_dump_hex	Reserved; set to FALSE.
public a_bit_field	debug_dump_char	Reserved; set to FALSE.
public a_bit_field	debug_page_offsets	Reserved; set to FALSE.
public a_bit_field	omit_comments	Reserved; set to FALSE.
public a_bit_field	use_hex_offsets	Reserved; set to FALSE.
public a_bit_field	use_relative_offsets	Reserved; set to FALSE.

Modifier and Type	Variable	Description
public a_bit_field	include_audit	Reserved; set to FALSE.
public a_bit_field	chronological_order	Reserved; set to FALSE.
public a_bit_field	force_recovery	Reserved; set to FALSE.
public a_bit_field	include_subsets	Reserved; set to FALSE.
public a_bit_field	force_chaining	Reserved; set to FALSE.
public a_bit_field	generate_reciprocals	Reserved; set to FALSE.
public a_bit_field	match_mode	Reserved; set to FALSE.
public a_bit_field	show_undo	Reserved; set to FALSE.
public a_bit_field	extra_audit	Reserved; set to FALSE.

Related Information

a_name Structure [page 770]
UserList Enumeration [page 752]
DBTranslateLog(const a_translate_log *) Method [page 743]

1.22.2.48 a_truncate_log Structure

Holds information needed for transaction log truncation using the DBTools library.

Syntax

```
typedef struct a_truncate_log
```

Members

All members of a_truncate_log, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.

Modifier and Type	Variable	Description
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public char	truncate_interrupted	Truncate was interrupted if non-zero.
public a_bit_field	quiet	Set TRUE to operate without printing messages. Set TRUE by dbbackup -q option.
public a_bit_field	server_backup	Set TRUE to indicate backup on server using BACKUP DATABASE. Set TRUE by dbbackup -s option when dbbackup -x option is specified.

Related Information

DBTruncateLog(const a_truncate_log *) Method [page 743]

1.22.2.49 a_validate_db Structure

Holds information needed for database validation using the DBTools library.

Syntax

```
typedef struct a_validate_db
```

Members

All members of a_validate_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public MSG_CALLBACK	statusrtn	Address of a status message callback routine or NULL.
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public p_name	tables	Pointer to a linked list of table names or index names (when the index field is set TRUE). This is set by the dbvalid object-name-list argument.
public char	type	The type of validation to perform. One of VALIDATE_NORMAL, VALIDATE_EXPRESS, VALIDATE_CHECKSUM, etc. See Validation enumeration.
public a_bit_field	quiet	Set TRUE to operate without printing messages. Set TRUE by dbvalid -q option.

Modifier and Type	Variable	Description
public a_bit_field	index	Set TRUE to validate indexes. The tables field points to a list of indexes. Set TRUE by dbvalid -i option. Set FALSE by dbvalid -t option.
public int	inmem_mode	Controls the modification of the StartLine connection parameter to choose the in-memory mode of auto-started database servers. Set to one of the IM_ enumeration values. This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.
public char	live_type	The type of live validation to perform. One of VALIDATE_NO_LIVE_OPTION, VALIDATE_WITH_DATA_LOCK, and VALIDATE_WITH_SNAPSHOT. See Live Validation enumeration. This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.

Related Information

a_name Structure [page 770]
DBValidate(const a_validate_db *) Method [page 746]
Validation Enumeration [page 752]

1.22.2.50 an_erase_db Structure

Holds information needed to erase a database using the DBTools library.

Syntax

```
typedef struct an_erase_db
```

Members

All members of an_erase_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	confirmrtn	Address of a confirmation request callback routine or NULL.
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public const char *	dbname	Database file name.
public const char *	encryption_key	The encryption key for the database file. Equivalent to dberase -ek or -ep options.
public a_bit_field	quiet	Operate without printing messages (1), or print messages (0). Set TRUE by dberase -q option.
public a_bit_field	erase	Erase without confirmation (1) or with confirmation (0). Set TRUE by dberase -y option.

Related Information

DBErase(const an_erase_db *) Method [page 735]

1.22.2.51 an_unload_db Structure

Holds information needed to unload a database using the DBTools library or extract a remote database for SQL Remote.

Syntax

```
typedef struct an_unload_db
```

Members

All members of an_unload_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public MSG_CALLBACK	statusrtn	Address of a status message callback routine or NULL.
public MSG_CALLBACK	confirmrtn	Address of a confirmation request call-back routine or NULL.
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public const char *	temp_dir	Directory for unloading data files.
public const char *	reload_filename	Name to use for the reload SQL script file (for example, reload.sql). Set by dbunload -r option.
public const char *	ms_filename	Reserved; use NULL.
public const char *	remote_dir	Like temp_dir but for internal unloads on server side.
public const char *	subscriber_username	The subscriber name to be used by dbxtract. NULL otherwise.
public const char *	site_name	The site name to be used by dbxtract. NULL otherwise.
public const char *	template_name	The template name to be used by dbxtract. NULL otherwise.
public const char *	encryption_key	The encryption key for the database file. Set by dbunload/dbxtract -ek or -ep option.

Modifier and Type	Variable	Description
public const char *	encryption_algorithm	The encryption algorithm which may be "simple", "aes", "aes256", "aes_fips", "aes256_fips", or NULL for none. Set by dbunload/dbxtract -ea option.
public const char *	locale	Reserved; use NULL.
public const char *	startline	Reserved; use NULL.
public const char *	startline_old	Reserved; use NULL.
public char *	reload_connectparms	Connection parameters such as user ID, password, and database for the reload database. Set by dbunload/dbxtract -ac option.
public char *	reload_db_filename	Name of the new database file to create and reload. Set by dbunload/dbxtract -an option.
public char *	reload_db_logname	Filename of the new database transaction log or NULL. Set by dbxtract -al option.
public p_name	table_list	Selective table list. Set by dbunload -e and -t options.
public a_sysinfo	sysinfo	Reserved; use NULL.
public long	notemp_size	Reserved; use 0.
public int	ms_reserve	Reserved; use 0.
public int	ms_size	Reserved; use 0.
public unsigned short	isolation_level	The isolation level at which to operate. Set by dbxtract -l option.
public unsigned short	reload_page_size	The reloaded database page size. Set by dbunload -ap option.
public char	unload_type	Set Unload enumeration (UNLOAD_ALL and so on). Set by dbunload/dbxtract -d, -k, -n options.
public char	verbose	See Verbosity enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).
public char	escape_char	The escape character (normally, "\"). Used when escape_char_present is TRUE. Set TRUE by dbunload/dbxtract -p option.

Modifier and Type	Variable	Description
public char	unload_interrupted	Reserved; set to 0.
public a_bit_field	unordered	Set TRUE for unordered data. Indexes will not be used to unload data. Set by dbunload/dbxtract -u option.
public a_bit_field	no_confirm	Set TRUE to replace an existing SQL script file without confirmation. Set by dbunload/dbxtract -y option.
public a_bit_field	use_internal_unload	Set TRUE to Perform an internal unload. Set TRUE by dbunload/dbxtract -i? option. Set FALSE by dbunload/dbxtract -x? option.
public a_bit_field	refresh_mat_view	Set TRUE to generate statements to refresh text indexes and valid materialized views. Set TRUE by dbunload/dbxtract -g option.
public a_bit_field	table_list_provided	Set TRUE to indicate that a list of tables has been provided. See table_list field. Set TRUE by dbunload -e, -t, or -tl options.
public a_bit_field	exclude_tables	Set FALSE to indicate that the list contains tables to be included. Set TRUE to indicate that the list contains tables to be excluded. Set TRUE by dbunload -e option.
public a_bit_field	preserve_ids	Set TRUE to preserve user IDs for SQL Remote databases. This is the normal setting. Set FALSE by dbunload -m option.
public a_bit_field	replace_db	Set TRUE to replace the database. Set TRUE by dbunload -ar option.
public a_bit_field	escape_char_present	Set TRUE to indicate that the escape character in escape_char is defined. Set TRUE by dbunload/dbxtract -p option.

Modifier and Type	Variable	Description
public a_bit_field	use_internal_reload	Set TRUE to perform an internal reload. This is the normal setting. Set TRUE by dbunload/dbxtract -ii and -xi option. Set FALSE by dbunload/dbxtract -ix and -xx option.
public a_bit_field	recompute	Set TRUE to redo computed columns. Set TRUE by dbunload -dc option.
public a_bit_field	make_auxiliary	Set TRUE to make auxiliary catalog (for use with diagnostic tracing). Set TRUE by dbunload -k option.
public a_bit_field	profiling_uses_single_dbspace	Set TRUE to collapse to a single dbspace file (for use with diagnostic tracing). Set TRUE by dbunload -kd option.
public a_bit_field	encrypted_tables	Set TRUE to enable encrypted tables in new database (with -an or -ar). Set TRUE by dbunload/dbxtract -et option.
public a_bit_field	remove_encrypted_tables	Set TRUE to remove encryption from encrypted tables. Set TRUE by dbunload/dbxtract -er option.
public a_bit_field	extract	Set TRUE if performing a remote database extraction. Set FALSE by dbunload. Set TRUE by dbxtract.
public a_bit_field	start_subscriptions	Set TRUE to start subscriptions. This is the default for dbxtract. Set FALSE by dbxtract -b option.
public a_bit_field	exclude_foreign_keys	Set TRUE to exclude foreign keys. Set TRUE by dbxtract -xf option.
public a_bit_field	exclude_procedures	Set TRUE to exclude stored procedures. Set TRUE by dbxtract -xp option.
public a_bit_field	exclude_triggers	Set TRUE to exclude triggers. Set TRUE by dbxtract -xt option.
public a_bit_field	exclude_views	Set TRUE to exclude views. Set TRUE by dbxtract -xv option.

Modifier and Type	Variable	Description
public a_bit_field	isolation_set	Set TRUE to indicate that isolation_level has been set for all extraction operations. Set TRUE by dbxtract -l option.
public a_bit_field	include_where_subscribe	Set TRUE to extract fully qualified publications. Set TRUE by dbxtract -f option.
public a_bit_field	exclude_hooks	Set TRUE to exclude procedure hooks. Set TRUE by dbxtract -xh option.
public a_bit_field	compress_output	Set TRUE to compress table data files. Set TRUE by dbunload -cp option.
public a_bit_field	display_create	Set TRUE to display database creation command (sql or dbinit). Set TRUE by dbunload -cm sql or -cm dbinit option.
public a_bit_field	display_create_dbinit	Set TRUE to display dbinit database creation command. Set TRUE by dbunload -cm dbinit option.
public a_bit_field	preserve_identity_values	Set TRUE to preserve identity values for AUTOINCREMENT columns. Set TRUE by dbunload -l option.
public a_bit_field	no_reload_status	Set TRUE to suppress reload status messages for tables and indexes. Set TRUE by dbunload -qr option.
public a_bit_field	startline_name	Reserved; set FALSE.
public a_bit_field	debug	Reserved; set FALSE.
public a_bit_field	schema_reload	Reserved; set FALSE.
public a_bit_field	genscript	Reserved; set FALSE.
public a_bit_field	runscript	Reserved; set FALSE.
public a_bit_field	suppress_statistics	Set TRUE to suppress inclusion of column statistics. Set TRUE by dbunload -ss option.
public a_bit_field	dbdiff	Reserved; set FALSE.

Modifier and Type	Variable	Description
public a_bit_field	unload_password_hashes	Set TRUE to output real password hashes during unload Set TRUE by dbunload -up option or in presence of -ac, -ar, -an options Can never be TRUE if -no or -k options are specified.
public a_bit_field	user_list_provided	Reserved; set FALSE.
public a_bit_field	table_list_patterns	Reserved; set FALSE.
public p_name	user_list	Reserved; set NULL.
public const char *	null_string	Reserved; set NULL.
public const char *	force_data_format	Reserved; set NULL.
public p_name	table_list_data	Reserved; set NULL.
public a_bit_field	table_list_data_provided	Reserved; set FALSE.
public a_bit_field	online_rebuild	Set TRUE to perform online rebuild Set TRUE by dbunload -ao This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.
public a_bit_field	online_rebuild_from_backup	Set TRUE to perform online rebuild from a backup of production db Set TRUE by dbunload -aob This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.
public SHOULD_STOP_CALLBACK	shouldstoprtn	Address of a callback routine or NULL. The routine should return TRUE if processing should stop now, FALSE otherwise This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.
public a_sql_uint32	online_rebuild_max_apply_sec	Set to 0 to only apply incremental log once during online rebuild. Set to a positive number of seconds to loop doing an increment backup and apply the log until the time for both the incremental backup and apply is less than this number of seconds. Set by dbunload -aot This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.

Modifier and Type	Variable	Description
public const char *	temporary_directory	Directory for online rebuild temporary files (including a full backup of the original database). If NULL the default temporary directory will be used. Set by dbunload -dt This feature is present in DB_TOOLS_VERSION_17_0_0 and later versions.

Remarks

Those fields used by the dbxtract SQL Remote Extraction utility are indicated.

Related Information

Verbosity Enumeration [page 753]
DBUnload(an_unload_db *) Method [page 744]

1.22.2.52 an_upgrade_db Structure

Holds information needed to upgrade a database using the DBTools library.

Syntax

```
typedef struct an_upgrade_db
```

Members

All members of an_upgrade_db, including inherited members.

Variables

Modifier and Type	Variable	Description
public unsigned short	version	DBTools version number (DB_TOOLS_VERSION_NUMBER).
public MSG_CALLBACK	errorrtn	Address of an error message callback routine or NULL.

Modifier and Type	Variable	Description
public MSG_CALLBACK	msgrtn	Address of an information message callback routine or NULL.
public MSG_CALLBACK	statusrtn	Address of a status message callback routine or NULL.
public const char *	connectparms	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny17\bin32\dbsrv17.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny17\bin32\dbsrv17.exe".
public a_bit_field	quiet	Set TRUE to operate without printing messages. Set TRUE by dbupgrad -q option.
public a_bit_field	jconnect	Set TRUE to upgrade the database to include jConnect procedures. Set FALSE by dbupgrad -i option.
public a_bit_field	restart	Set TRUE to restart the database after the upgrade. Set FALSE by the dbupgrad -nrs option.

Modifier and Type	Variable	Description
public unsigned short	sys_proc_definer	Assign 0 to upgrade the database to have the pre-16.0 SQL SECURITY model for legacy system stored procedures when upgrading from pre-16.0 releases. When upgrading from a version 16.0 or later database retain the current SQL SECURITY model (same as not specifying -pd). Assign 1 to upgrade the database to have the pre-16.0 SQL SECURITY model for legacy system stored procedures (same as -pd y) Assign 2 to upgrade the database to have the pre-16.0 SQL SECURITY model for legacy system stored procedures (same as -pd n).

Related Information

DBUpgrade(const an_upgrade_db *) Method [page 745]

1.22.2.53 DBT_USE_DEFAULT_MIN_PWD_LEN Variable

DBT_USE_DEFAULT_MIN_PWD_LEN indicates that the default minimum password length is to be used (that is, no specific length is being specified) for the new database.

Syntax

```
#define DBT_USE_DEFAULT_MIN_PWD_LEN
```

Remarks

Use this value to set the min_pwd_len field.

1.22.3 Software Component Exit Codes

All database tools library entry points return exit codes. The database server and utilities (dbsrv17, dbbackup, dbspawn, and so on) also use these exit codes.

Code	Status	Explanation
0	EXIT_OKAY	Success
1	EXIT_FAIL	General failure
2	EXIT_BAD_DATA	Invalid file format
3	EXIT_FILE_ERROR	File not found, unable to open
4	EXIT_OUT_OF_MEMORY	Out of memory
5	EXIT_BREAK	Terminated by the user
6	EXIT_COMMUNICATIONS_FAIL	Failed communications
7	EXIT_MISSING_DATABASE	Missing a required database name
8	EXIT_PROTOCOL_MISMATCH	Client/server protocol mismatch
9	EXIT_UNABLE_TO_CONNECT	Unable to connect to the database server
10	EXIT_ENGINE_NOT_RUNNING	Database server not running
11	EXIT_SERVER_NOT_FOUND	Database server not found
12	EXIT_BAD_ENCRYPT_KEY	Missing or bad encryption key
13	EXIT_DB_VER_NEWER	Server must be upgraded to run database
14	EXIT_FILE_INVALID_DB	File is not a database
15	EXIT_LOG_FILE_ERROR	Log file was missing or other error
16	EXIT_FILE_IN_USE	File in use
17	EXIT_FATAL_ERROR	Fatal error occurred
18	EXIT_MISSING_LICENSE_FILE	Missing server license file
19	EXIT_BACKGROUND_SYNC_ABORTED	Background synchronization aborted to allow higher priority operations proceed
20	EXIT_FILE_ACCESS_DENIED	Database cannot be started because access is denied
21	EXIT_SERVER_NAME_IN_USE	Another server with the same name is currently running.
255	EXIT_USAGE	Invalid parameters on the command line

These exit codes are defined in the *%SQLANY17%*`\sdk\include\sqldef.h` file.

1.23 Database and Application Deployment

When you have completed a database application, you must deploy the application to your end users.

Depending on the way in which your application uses SQL Anywhere (as an embedded database, in a client/server fashion, and so on) you may have to deploy components of the SQL Anywhere software along with your application. You may also have to deploy configuration information, such as data source names, that enable your application to communicate with SQL Anywhere.

> **Note**
>
> Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

The following deployment steps are examined:

- Determining required files based on the choice of application platform and architecture.
- Configuring client applications.

Much of the section deals with individual files and where they must be placed. However, the recommended way of deploying SQL Anywhere components is to use the *Deployment Wizard* or to use a silent install.

In this section:

Types of Deployment [page 808]
The files required for your deployment depend on the type of deployment you choose.

Ways to Distribute Files [page 809]
There are two ways to deploy SQL Anywhere: using the SQL Anywhere installer or developing your own installer.

Installation Directories and File Names [page 810]
For a deployed application to work properly, the database server and client applications must each be able to locate the files they need. The deployed files should be located relative to each other in the same fashion as your SQL Anywhere installation.

The Deployment Wizard for Windows [page 813]
The *Deployment Wizard* is the preferred tool for creating deployments of SQL Anywhere for Windows.

Silent Installs on Microsoft Windows [page 817]
Silent installs run without user input and with no indication to the user that an install is occurring. On Microsoft Windows operating systems, you can call the SQL Anywhere installer from your own install program in such a way that the SQL Anywhere install is silent.

The Deployment Wizard for UNIX/Linux [page 822]
The Deployment Wizard is the preferred tool for creating deployments of SQL Anywhere for UNIX and Linux.

Silent Installs on UNIX/Linux [page 823]
Silent installs run without user input and with no indication to the user that an install is occurring. On UNIX or Linux operating systems, you can call the SQL Anywhere installer from your own install program in such a way that the SQL Anywhere install is silent.

Requirements for Deploying Client Applications [page 826]

To deploy a client application that runs against a database server, you must provide each end user with your client software, the client database interface components, and appropriate database connection information.

Administration Tool Deployment [page 857]
Subject to your license agreement, you can deploy a set of administration tools including Interactive SQL and *SQL Central*.

Documentation Deployment [page 880]
There are three options available for documentation.

Database Server Deployment [page 881]
To run a database server, you must install the correct set of files.

DLL Registration on Windows [page 888]
Some DLL files require registration if deployed for use with SQL Anywhere.

External Environment Support Deployment [page 889]
A number of components are required to support external calls in a database application.

Encryption Deployment [page 892]
SQL Anywhere supports AES encryption of databases and database tables and cryptographic functions, as well as RSA encryption for secure client login using a password and network traffic (TLS, HTTPS) between the client and the database server.

LDAP Deployment [page 894]
SQL Anywhere supports deployment of Lightweight Directory Access Protocol (LDAP) user authentication, allowing client applications to send user ID and password information to the database server for authentication by an LDAP server.

Embedded Database Application Deployment [page 894]
An embedded database application is one in which the application and the database both reside on the same computer.

1.23.1 Types of Deployment

The files required for your deployment depend on the type of deployment you choose.

Here are some possible deployment models:

Client deployment

You may deploy only the client portions of SQL Anywhere to your end users, so that they can connect to a centrally located network database server.

Network server deployment

You may deploy network servers to offices, and then deploy clients to each of the users within those offices.

Embedded database deployment

You may deploy an application that runs with the personal database server. In this case, both client and personal server must be installed on the end-user's computer.

SQL Remote deployment

Deploying a SQL Remote application is an extension of the embedded database deployment model.

MobiLink deployment

The deployment of MobiLink servers is described in the MobiLink documentation.

Administration tools deployment

You may deploy Interactive SQL, *SQL Central* and other management tools.

Related Information

MobiLink Deployment

1.23.2 Ways to Distribute Files

There are two ways to deploy SQL Anywhere: using the SQL Anywhere installer or developing your own installer.

Use the SQL Anywhere installer

You can make the installer available to your end users. By selecting the proper option, each end user is guaranteed to receive the files they need.

This is the simplest solution for many deployment cases. In this case, you must still provide your end users with a method for connecting to the database server (such as an ODBC data source).

Develop your own installer

There may be reasons for you to develop your own software installer that includes SQL Anywhere files. This is a more complicated option, and most of the material here addresses the needs of those who are developing their own installer.

If SQL Anywhere has already been installed for the server type and operating system required by the client application architecture, the required files can be found in the appropriately named subdirectory, located in the SQL Anywhere installation directory. For example, the `bin32` subdirectory of your software installation directory contains the files required to run the server for 32-bit Windows operating systems.

Whichever option you choose, you must not violate the terms of your license agreement.

Related Information

The Deployment Wizard for Windows [page 813]
The Deployment Wizard for UNIX/Linux [page 822]
Silent Installs on Microsoft Windows [page 817]
Silent Installs on UNIX/Linux [page 823]

1.23.3 Installation Directories and File Names

For a deployed application to work properly, the database server and client applications must each be able to locate the files they need. The deployed files should be located relative to each other in the same fashion as your SQL Anywhere installation.

In practice, most executable files belong in a single directory on Windows. For example, on Windows both client and database server executable files are installed in the same directory, which is the `bin32` or `bin64` subdirectory of the SQL Anywhere installation directory.

In this section:

UNIX/Linux Deployment Issues [page 810]
UNIX and Linux deployments are different from Microsoft Windows deployments in some ways.

File Naming Conventions [page 812]
SQL Anywhere uses consistent file naming conventions to help identify and group system components.

Related Information

How SQL Anywhere Locates Files

1.23.3.1 UNIX/Linux Deployment Issues

UNIX and Linux deployments are different from Microsoft Windows deployments in some ways.

Directory structure

For UNIX and Linux installations, the default directory structure is as follows:

Directory	Contents
/opt/sqlanywhere17`/bin32` and */opt/sqlanywhere17*`/bin64`	Executable files, License files
/opt/sqlanywhere17`/lib32` and */opt/sqlanywhere17*`/lib64`	Shared objects and libraries
/opt/sqlanywhere17`/res`	String files

On AIX, the default root directory is `/usr/lpp/sqlanywhere`*17* instead of */opt/sqlanywhere17*.

On macOS, the default root directory is */Applications/SQLAnywhere17/System* instead of */opt/sqlanywhere17*.

Depending on the complexity of your deployment, you might choose to put all of the files that you require for your application into a single directory. You may find that this is a simpler deployment option, especially if a small number of files are required for your deployment. This directory could be the same directory that you use for your own application.

File suffixes

In the tables, shared objects are listed with a suffix of `.so` or `.so.1`. The version number, 1, could be higher as updates are released. For simplicity, the version number is often not listed.

For AIX, the suffix does not contain a version number so it is simply `.so`.

Symbolic links

Each shared object is installed as a symbolic link (symlink) to a file of the same name with the additional suffix `.1` (one). For example, `libdblib`*17*`.so` is a symbolic link to the file `libdblib`*17*`.so.1` in the same directory.

The version suffix `.1` could be higher as updates are released and the symbolic link must be redirected.

On macOS, create a jnilib symbolic link for any dylib that you want to load directly from your Java client application.

Threaded and non-threaded applications

Most shared objects are provided in two forms, one of which has the additional characters `_r` before the file suffix. For example, in addition to `libdblib`*17*`.so.1`, there is a file named `libdblib`*17*`_r.so.1`. In this case, threaded applications must be linked to the shared object whose name has the `_r` suffix, while non-threaded applications must be linked to the shared object whose name does not have the `_r` suffix. Occasionally, there is a third form of shared object with `_n` before the file suffix. This is a version of the shared object that is used with non-threaded applications.

Character set conversion

To use database server character set conversion, include the following files:

- `libdbicu`*17*`.so.1`
- `libdbicu`*17*`_r.so.1`
- `libdbicudt`*17*`.so.1`
- `sqlany.cvf`

Environment variables

On UNIX and Linux, environment variables must be set for the system to locate SQL Anywhere applications and libraries. Use the appropriate configuration script for your shell, either `sa_config.sh` or `sa_config.csh` (located in the directories */opt/sqlanywhere17*`/bin32` and */opt/sqlanywhere17*`/bin64`) as a template for setting the required environment variables. Some of the environment variables set by these files include PATH, LD_LIBRARY_PATH, and SQLANY17.

Related Information

How SQL Anywhere Locates Files

1.23.3.2 File Naming Conventions

SQL Anywhere uses consistent file naming conventions to help identify and group system components.

These conventions include:

Version number

The SQL Anywhere version number is indicated in the file name of the main server components (executable files, dynamic link libraries, shared objects, license files, and so on).

For example, the file *dbsrv17*`.exe` is a version *17* executable for Windows.

Language

The language used in a language resource library is indicated by a two-letter code within its file name. The two characters before the version number indicate the language used in the library. For example, `dblgen`*17*`.dll` is the message resource library for the English language. These two-letter codes are specified by ISO standard 639-1.

For more information about language labels, see the Language Selection utility (dblang) documentation.

There are several localized versions of SQL Anywhere.

Identifying other file types

The following table identifies the platform and function of SQL Anywhere files according to their file extension. SQL Anywhere follows standard file extension conventions where possible.

File Extension	Platform	File Type
`.bat`, `.cmd`	Windows	Batch command files
`.chm, .chw`	Windows	Help system file
`.dll`	Windows	Dynamic Link Library
`.exe`	Windows	Executable file
`.ini`	All	Initialization file
`.lic`	All	License file
`.lib`	Varies by development tool	Static runtime libraries for the creation of Embedded SQL executables
`.res`	Linux, Unix, macOS	Language resource file for non-Windows environments
`.so`	Linux, Unix	Shared object or shared library file (the equivalent of a Windows DLL)
`.bundle`, `.dylib`	macOS	Shared object file (the equivalent of a Windows DLL)

Database File Names

SQL Anywhere databases are composed of two elements:

Database file

This is used to store information in an organized format. By default, this file uses a `.db` file extension. There may also be additional dbspace files. These files could have any file extension including none.

Transaction log file

This is used to record all changes made to data stored in the database file. By default, this file uses a `.log` file extension, and is generated by SQL Anywhere if no such file exists and a transaction log file is specified to be used. A transaction log mirror has the default extension of `.mlg`.

These files are updated, maintained and managed by the SQL Anywhere relational database management system.

Related Information

Localized Versions of the Software
Deployment Software Localization on Microsoft Windows
Language Selection Utility (dblang)

1.23.4 The *Deployment Wizard* for Windows

The *Deployment Wizard* is the preferred tool for creating deployments of SQL Anywhere for Windows.

The *Deployment Wizard* can be used to create a new install image or to upgrade an existing install image. The *Deployment Wizard* creates installer files that include some or all the following components:

- Client interfaces, such as ODBC
- SQL Anywhere server, including remote data access, database tools, and encryption
- UltraLite relational database
- MobiLink server, client, and encryption
- Administration tools such as Interactive SQL and *SQL Central*

You can use the *Deployment Wizard* to create a Microsoft Windows Installer Package file or a Microsoft Windows Installer Merge Module file:

Microsoft Windows Installer Package file

A file containing the instructions and data required to install an application. An Installer Package file has the extension `.msi`.

Microsoft Windows Installer Merge Module file

A simplified type of Microsoft Installer Package file that includes all files, resources, registry entries, and setup logic to install a shared component. A merge module has the extension `.msm`.

A merge module cannot be installed alone because it lacks some vital database tables that are present in an installer package file. Merge modules contain additional tables that are unique to themselves. To install

the information delivered by a merge module with an application, the module must first be merged into the application's Installer Package (`.msi`) file. A merge module consists of the following parts:

- A merge module database containing the installation properties and setup logic being delivered by the merge module.
- A merge module Summary Information Stream describing the module.
- A `MergeModule.CAB` cabinet file stored as a stream inside the merge module. This cabinet file contains all the files required by the components delivered by the merge module. Every file delivered by the merge module must be stored inside of a cabinet file that is embedded as a stream in the merge module's structured storage. In a standard merge module, the name of this cabinet is always `MergeModule.CAB`.

The *Deployment Wizard* allows you to select subsets of the components included in SQL Anywhere. Each component has dependencies on other components so the files that are selected by the wizard may include files from other categories.

To determine what files are included in each selectable component, create an MSI installer image and select all the components. An MSI log file is created that specifies what files are included in every component. This text file can be examined with a text editor. There are headings such as *Feature: SERVER32_TOOLS* and *Feature: CLIENT64_TOOLS* that closely correspond to the *Deployment Wizard* components. Looking at the file gives you an idea of what is included in each group.

In this section:

Running the Deployment Wizard on Windows [page 814]
Use the *Deployment Wizard* to create database and application deployments for Windows.

Windows msiexec Installation Uutility [page 816]
The Microsoft msiexec utility installs or uninstalls deployments on Windows.

Related Information

Silent Installs on Microsoft Windows [page 817]

1.23.4.1 Running the *Deployment Wizard* on Windows

Use the *Deployment Wizard* to create database and application deployments for Windows.

Prerequisites

Redistribution of files is subject to your license agreement. You must acknowledge that you are properly licensed to redistribute SQL Anywhere files. Check your license agreement before proceeding.

Context

Database and application deployment

Procedure

1. To run the *Deployment Wizard*, click *Start* > *Programs* > *SQL Anywhere 17* > *Administration Tools* > *Deploy to Windows*.
2. Follow the instructions in the wizard.
3. Decide whether to create a new install or an upgrade install.

Results

The *Destination Path* that you selected will contain the deployment package (for example, `sqlany17.msi`) and an MSI log file (`sqlany17.msi.log`).

Example

You can also run the *Deployment Wizard* from the `Deployment` subdirectory of your SQL Anywhere installation. For example:

```
%SQLANY17%\Deployment\DeploymentWizard.exe
```

Next Steps

Once you have created a deployment package, test it by installing it.

The product code (ProductCode) for the install image can be found in the Property table using Microsoft's Orca tool. The product code is preserved when creating an upgrade install image. The product code can be used when uninstalling the software.

Related Information

Windows msiexec Installation Uutility [page 816]

1.23.4.2 Windows msiexec Installation Uutility

The Microsoft msiexec utility installs or uninstalls deployments on Windows.

Syntax

```
msiexec [ options ] [ SQLANYDIR=path ]
```

Option	Description
/log `log-filename`	This parameter tells the Microsoft Windows Installer to log operations to a file (for example, `sqlany17.log`). The installer log file is useful for detecting problems.
/package `package-name`	This parameter tells the Microsoft Windows Installer to install the specified package (for example, `sqlany17.msi`).
REINSTALL=ALL	This parameter tells the Microsoft Windows Installer to reinstall all features of an upgrade install over an existing install.
REINSTALLMODE=vomus	This parameter tells the Microsoft Windows Installer to install an upgrade install over an existing install. See the Microsoft documentation for an explanation of the reinstall option codes. The recommended option codes are *vomus*.
/qn	This parameter tells the Microsoft Windows Installer to operate in the background with no user interaction.
/uninstall `package-name` | `product-code`	This parameter tells the Microsoft Windows Installer to uninstall the product associated with the specified MSI file or product code. The `product-code` is created when the *Deployment Wizard* is run to create the deployment package.
SQLANYDIR= `path`	The value of this parameter sets the default path to where the software is to be installed. This option is useful for silent installs. It is ignored for */uninstall*.

Remarks

The /qn option can be used to perform the operation without user messages. On Windows 7 and later versions of Windows, administrator privilege is required to install or uninstall software. When the /qn option is used, the automatic request for administrator privilege is suppressed.

Example

The following command installs a deployment file.

```
msiexec /package sqlany17.msi
```

The following command performs a silent install of the deployment file to the specified folder and logs the results to a file. Caution should be exercised with the use of the /qn option since the computer system may shut down and restart without warning.

```
msiexec /qn /package sqlany17.msi /log sqlany17.log SQLANYDIR=c:\sa17
```

The following command performs a silent uninstall using the specified package and logs the results to a file.

```
msiexec /qn /uninstall sqlany17.msi /log sqlany17.log
```

The following command performs a silent uninstall using the specified product code.

```
msiexec.exe /qn /uninstall {7C99D24E-AE1B-4770-9015-65B805950E3D}
```

The following command reinstalls a deployment using an upgrade install and creates a log file in the process.

```
msiexec /package sqlany17sp2.msi REINSTALL=ALL REINSTALLMODE=vomus /log
upgrade.log
```

Related Information

Running the Deployment Wizard on Windows [page 814]

1.23.5 Silent Installs on Microsoft Windows

Silent installs run without user input and with no indication to the user that an install is occurring. On Microsoft Windows operating systems, you can call the SQL Anywhere installer from your own install program in such a way that the SQL Anywhere install is silent.

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit.

The common options for the SQL Anywhere install program `setup.exe` are:

/l:language_id

The language identifier is a locale number that represents the language for the install. For example, locale ID 1033 identifies U.S. English, locale ID 1031 identifies German, locale ID 1036 identifies French, locale ID 1041 identifies Japanese, and locale ID 2052 identifies Simplified Chinese.

/s

This option hides the initialization window. Use this option with /v:.

/v:

Specify parameters to MSIEXEC, the Microsoft Windows Installer tool.

/qn

This is an MSIEXEC option (no user interaction).

The following command line example assumes that the install image directory is in the `pkg\SQLAnywhere` directory on the disk in drive `d:`.

```
d:\pkg\sqlanywhere\setup.exe /l:1033 /s "/v: /qn
    REGKEY=NEEDA-REAL0-KEY12-34567-89012 INSTALLDIR=c:\sa17
DIR_SAMPLES=c:\sa17\Samples"
```

> **Note**
>
> The `setup.exe` in the command above is the one located in the same directory as the SQLANY32.msi and SQLANY64.msi files. The setup.exe in the parent directory of those files does NOT support silent installs.
>
> Caution should be exercised with the use of the MSIEXEC /qn option since the computer system may shut down and restart without warning.
>
> When you use an option like /qn (no user interface), warnings and error message dialog prompts will not appear during the course of the installation. A review of the installation log will be required to know if anything unexpected happened during the install process.

A software update is delivered as a PackageForTheWeb (PFTW), an InstallShield packaged application embedded in a self-contained, self-extracting file. A silent install of a PackageForTheWeb self-extracting executable has similar syntax. The following is an example command line.

```
SA17_Windows_1704_2034_EBF.exe /s /a /l:1033 /s "/v: /qn"
```

The first /s option silences the self-extracting executable and the /a option indicates to the self-extracting executable that everything that follows is to be passed to the `setup.exe` file that is extracted and run. The remaining options (/l, /s, /v:, /qn) are described above. Properties like registration key and installation directory do not need to be specified because they are determined from the existing install.

The following properties apply to the SQL Anywhere installer:

INSTALLDIR

The value of this parameter is the path to where the software is installed.

DIR_SAMPLES

The value of this parameter is the path to where the sample programs are installed.

DIR_SQLANY_MONITOR

The value of this parameter is the path to where the SQL Anywhere Monitor database (`samonitor.db`) is installed.

USERNAME

The value of this parameter is the user name to record for this installation (for example, USERNAME=\"John Smith\").

COMPANYNAME

The value of this parameter is the company name to record for this installation (for example, COMPANYNAME=\"Smith Holdings\").

REGKEY

The value of this parameter must be a valid software registration key.

REGKEY_ADD_1

The value of this parameter must be a valid software registration key for an add-on feature such as FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

REGKEY_ADD_2

The value of this parameter must be a valid software registration key for an add-on feature such as FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

REGKEY_ADD_3

The value of this parameter must be a valid software registration key for an add-on feature such as FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

The following example shows how to specify the SQL Anywhere install properties:

```
d:\pkg\sqlanywhere\setup.exe /s "/v: /qn
USERNAME=\"John Smith\"
COMPANYNAME=\"Smith Holdings\"
REGKEY=NEEDA-REAL0-KEY12-34567-89012
REGKEY_ADD_1=<an add-on software registration key>
REGKEY_ADD_2=<another add-on software registration key>
INSTALLDIR=c:\sa17
DIR_SAMPLES=c:\sa17\Samples
DIR_SQLANY_MONITOR=c:\sa17\Monitor"
```

Although the above text is shown over several lines for reasons of length, it would be specified as a single line of text. Note the use of the backslash character to escape the interior quotation marks.

The following properties apply to the SQL Anywhere Monitor installer (located in `d:\pkg\Monitor\setup.exe`):

INSTALLDIR

The value of this parameter is the path to where the software is installed.

DIR_SQLANY_MONITOR

The value of this parameter is the path to where the SQL Anywhere Monitor database (`samonitor.db`) is installed.

REGKEY

The value of this parameter must be a valid software installation key.

REGKEY_ADD_1

The value of this parameter must be a valid software installation key for an add-on feature such as FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

REGKEY_ADD_2

The value of this parameter must be a valid software installation key for an add-on feature such as FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

The following example shows how to specify the SQL Anywhere Monitor install properties:

```
d:\software\monitor\setup.exe /s "/v: /qn
REGKEY=<SQL Anywhere Monitor registration key>
REGKEY_ADD_1=<an add-on software registration key>
REGKEY_ADD_2=<another add-on software registration key>
INSTALLDIR=c:\sa17"
```

In addition to setting the property values described above, the following SQL Anywhere components or features may be selected from the command line:

Feature	Property	x86 Default	x64 Default
Administration Tools (32-bit)	AT32	1	0
Administration Tools (64-bit)	AT64	0	1
FIPS-certified Encryption	FIPS *		
High Availability	HA *		
In-Memory Mode	IM *		
MobiLink (64-bit)	ML64	0	1
MobiLink (32-bit)	ML32	1	0
Read-only scale-out	SON *		
Relay Server (64-bit)	RS64	0	1
SACI_SDK	SACI	0	0
Samples	SAMPLES	1	1
SQL Anywhere client (32-bit)	CLIENT32	1	1
SQL Anywhere client (64-bit)	CLIENT64	0	1
SQL Anywhere server (32-bit)	SERVER32	1	0
SQL Anywhere server (64-bit)	SERVER64	0	1
SQL Anywhere Monitor (32-bit)	SM32	1	0
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

* These features require an additional appropriate software registration key.

Set the property value to 1 to select the feature or to 0 to omit the feature. For example, to omit 32-bit MobiLink, set ML32=0. To select the 32-bit Administration Tools, set AT32=1. These properties are used to override the default selection.

You don't have to specify the selection state of a feature if the default is applicable. For example, to override the default selection on a 64-bit computer by installing the 32-bit MobiLink feature and not installing the Samples, use the following command line:

```
setup.exe "/v ml32=1 samples=0"
```

Property names are case insensitive.

Your registration key may limit the available features. The command-line options cannot be used to override those limitations. For example, if your registration key does not allow the install of FIPS Encryption, then using FIPS=1 on the command line will not select the feature.

To generate an MSI log add the following to the command line after the `/v:`.

```
/l*v! logfile
```

In this example, `logfile` is the full path and file name of the message log file. The path must already exist. This option generates an extremely verbose log and significantly lengthens the time required to execute the install.

In addition to a silent install, it is also possible to perform a silent uninstall. The following is an example of a command line that would do this.

```
msiexec.exe /qn /uninstall {08787C4E-7210-4FA8-8D09-B393E76CA1A8} /l*v %temp%
\sauninstall.log
```

In the above example, you call the Microsoft Windows Installer tool directly.

/qn

This parameter tells the Microsoft Windows Installer to operate in the background with no user interaction.

/uninstall <product-code>

This parameter tells the Microsoft Windows Installer to uninstall the product associated with the specified product code. The code shown above is for the SQL Anywhere software.

/l*v %temp%\sauninstall.log

This parameter tells the Microsoft Windows Installer to create a log file in the specified location.

The product codes for SQL Anywhere version 17.0.11 are:

{08787C4E-7210-4FA8-8D09-B393E76CA1A8}

SQL Anywhere software

{8AF8FFC2-8C0E-40FE-BFD7-872E65EB235C}

SQL Anywhere client-only software

{CA1B1A3C-9CD6-4BF7-BE9A-0BD4C2601DB9}

SQL Anywhere Monitor

Related Information

Command-Line Options

1.23.6 The Deployment Wizard for UNIX/Linux

The Deployment Wizard is the preferred tool for creating deployments of SQL Anywhere for UNIX and Linux.

The deployment wizard is used to select some or all the following components:

- Client interfaces, such as ODBC
- SQL Anywhere server, including remote data access, database tools, and encryption
- UltraLite relational database
- MobiLink server, client, and encryption
- Administration tools such as Interactive SQL and *SQL Central*

The deployment wizard allows you to select subsets of the components included in SQL Anywhere. Each component has dependencies on other components so the files that are selected by the wizard may include files from other categories.

In this section:

Running the Deployment Wizard on UNIX/Linux [page 822]
Use the *Deployment Wizard* to create database and application deployments for UNIX and Linux.

1.23.6.1 Running the Deployment Wizard on UNIX/Linux

Use the *Deployment Wizard* to create database and application deployments for UNIX and Linux.

Prerequisites

Redistribution of files is subject to your license agreement. You must acknowledge that you are properly licensed to redistribute SQL Anywhere files. Check your license agreement before proceeding.

Context

Database and application deployment

Procedure

1. Set up your environment by sourcing the `sa_config.sh` / `sa_config.csh` file. For example:

   ```
   . /opt/sqlanywhere17/bin64/sa_config.sh
   ```

2. Start the deployment wizard.

   ```
   $SQLANY17/deployment/deployment_wizard.sh
   ```

3. Follow the prompts.You are prompted for the location of the install. Accept the default location if it is correct.
4. Select the components you wish to deploy.
5. Save the list of files to a text file (for example, `deployment.txt`). Use this file to create a TAR file.
6. Terminate the deployment wizard.
7. Use the text file to build a TAR file. For example:

   ```
   cd $SQLANY17
   tar czf deployment.tar -T deployment.txt
   ```

Results

You have created a deployment package in the form of a TAR archive file.

Next Steps

Once you have created a deployment package, test it by installing it.

1.23.7 Silent Installs on UNIX/Linux

Silent installs run without user input and with no indication to the user that an install is occurring. On UNIX or Linux operating systems, you can call the SQL Anywhere installer from your own install program in such a way that the SQL Anywhere install is silent.

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit.

The options for the SQL Anywhere install program *setup* are:

Option	Description
-company `company` (-c `company`)	specify the company name
-help (-h)	display this help screen
-l_accept_the_license_agreement	accept the license agreement
-install `packages`	where `packages` is a comma-separated list of packages
-install_icons `dir` (-ii `dir`)	install the icons, specifying the SQL Anywhere directory
-list_packages	display a listing of available packages
-name `username` (-n `username`)	specify the user name
-nogui	use the shell script only, do not try to invoke the gui
-regkey `key` (-k `key`)	use `key` as the registration key
-seat-model `model` (-m `model`)	specify the seat model (accepted for backward compatibility but ignored)
-seats `num` (-s `num`)	specify the number of seats or processors
-silent (-ss)	silent install
-sqlany-dir `dir` (-d `dir`)	set SQL Anywhere install directory
-type {CREATE|MODIFY|UPGRADE}	run the install as the specified type

These names of options are case sensitive so they must be specified as shown (for example, *-SILENT* is not accepted). The values for options are not case sensitive (for example, *modify* can be used instead of *MODIFY*).

The following example lists the available packages for the specified registration key.

```
setup -list_packages -k NEEDA-REAL0-KEY12-34567-89012
```

The following SQL Anywhere packages may be selected.

Feature	Package	x86 Default	x64 Default
Administration Tools (32-bit)	admintools32	Yes	No
Administration Tools (64-bit)	admintools64	No	Yes
FIPS-certified Strong Encryption	fips *		
High Availability	high_avail *		
In-Memory Mode	in_memory *		
MobiLink (64-bit)	ml64	No	Yes
MobiLink Client (32-bit)	mobilink_sqlany32	Yes	No
Read-only scale-out	scaleoutnodes *		
Relay Server (64-bit)	relayserver64	No	Yes
Samples	samples	Yes	Yes
SQL Anywhere client (32-bit)	sqlany_client32	Yes	Yes

Feature	Package	x86 Default	x64 Default
SQL Anywhere client (64-bit)	sqlany_client64	No	Yes
SQL Anywhere server (32-bit)	sqlany32	Yes	No
SQL Anywhere server (64-bit)	sqlany64	No	Yes
SQL Anywhere Monitor (32-bit)	samon32	Yes	No
SQL Anywhere Monitor (64-bit)	samon64	No	Yes
SQL Remote (32-bit)	dbremote32	Yes	No
SQL Remote (64-bit)	dbremote64	No	Yes
UltraLite Engine (64-bit)	ultralite64	Yes	Yes

* These features require an additional appropriate software registration key.

If you do not specify what packages to install then the default packages for your platform (indicated by 'Yes') are installed, limited only by your license registration key. In other words, 32-bit packages are installed on x86 platforms and 64-bit packages along with the 32-bit client package are installed on x64 platforms. The following example shows how to install the default set of packages.

```
setup -k NEEDA-REAL0-KEY12-34567-89012 -ss \
    -I_accept_the_license_agreement -name "Sally Smith" -company "SAP AG"
```

If you do specify what packages to install (using *-install*) then you must specify all packages that you wish to install. The following example shows how to specify packages.

```
setup -k NEEDA-REAL0-KEY12-34567-89012 -install sqlany64,sqlany_client64 -ss \
    -I_accept_the_license_agreement -name "Joe Jones" -company "SAP AG"
```

Package names are case insensitive.

Your registration key may limit the available features. The command-line options cannot be used to override those limitations. For example, if your registration key does not allow the installation of FIPS Encryption, then including *fips* in the *install* list will not select the feature.

If you use the *silent* option then you must have read the license agreement and you must specify the *I_accept_the_license_agreement* option on the command line. You must also specify the user name and company name using the *name* and *company* options.

If you omit the *silent* option then are prompted for settings, in which case the other options are used to set new defaults for each of the prompts.

The *seat_model* option is accepted for backward compatibility but is ignored - the seat model is defined by the license key.

To uninstall the software installed by *setup*, use the *uninstall* script that is placed in the root of your SQL Anywhere installation.

```
$SQLANY17/uninstall
```

In the above example, you uninstall only those packages that were installed by *setup*.

1.23.8 Requirements for Deploying Client Applications

To deploy a client application that runs against a database server, you must provide each end user with your client software, the client database interface components, and appropriate database connection information.

In particular, you must provide each end user with the following items:

Client application

The application software itself is independent of the database software, and so is not described here.

Client database interface files

The client application requires the files for the client database interface it uses (.NET, ADO, OLE DB, ODBC, JDBC, Embedded SQL, Perl, PHP, Python, Ruby, C/C++ API, Open Client). Except for TDS clients that use jConnect or Open Client, a client application also requires one of the encryption support libraries.

Connection information

Each client application needs database connection information.

The interface files and connection information required varies with the interface your application is using. Each interface is described separately.

The simplest way to deploy clients is to use the *Deployment Wizard*.

In this section:

.NET Client Deployment [page 827]
ADO.NET applications that use the SQL Anywhere .NET Data Provider require a number of provider components.

OLE DB and ADO Client Deployment [page 830]
OLE DB and ADO applications that use the SQL Anywhere OLE DB provider require a number of provider components.

OLE DB Provider Customization [page 831]
You can customize the OLE DB provider to have your own version of the provider.

ODBC Client Deployment [page 836]
Applications that use the SQL Anywhere ODBC driver require a number of components.

Embedded SQL Client Deployment [page 845]
Applications that use Embedded SQL require a number of components.

JDBC Client Deployment [page 847]
Applications that use JDBC require a number of components.

PHP Client Deployment [page 849]
Applications that use the SQL Anywhere PHP extension require a number of components.

Open Client Application Deployment [page 857]
To deploy Open Client applications, each client computer needs the SAP Open Client product.

Related Information

The Deployment Wizard for Windows [page 813]
The Deployment Wizard for UNIX/Linux [page 822]
Encryption Deployment [page 892]

1.23.8.1 .NET Client Deployment

ADO.NET applications that use the SQL Anywhere .NET Data Provider require a number of provider components.

The simplest way to determine what SQL Anywhere .NET Data Provider components to deploy is to use the *Deployment Wizard*.

When deploying SQL Anywhere .NET Data Provider components, complete the installation on the client computer using the following information.

- Ensure Microsoft Visual Studio is not running.
- Use the SetupVSPackage tool to install the SQL Anywhere .NET Data Provider assemblies. SetupVSPackage requires Administrator privilege for Windows 7 and later systems. If you plan to run SetupVSPackage at a Command Prompt, ensure the Command Prompt has Administrator privilege. SetupVSPackage updates the Global Assembly Cache (GAC), the Windows Microsoft.NET `machine.config` files, and Microsoft Visual Studio integration.

  > **Note**
  >
  > Installing the .NET Data Provider affects Microsoft Visual Studio integration for all versions of the provider, including earlier major versions such as 16.0 and 12.0. Only one version of the .NET Data Provider can be used with Microsoft Visual Studio integration.

- For .NET 2.0/3.x only, run *%SQLANY17%*`\Assembly\v3.5\SetupVSPackage.exe /install`. The Global Assembly Cache (GAC) is updated to contain version 3.5 of the .NET provider assemblies. The Windows Microsoft.NET version 2 `machine.config` file is updated to point the provider invariant name Sap.Data.SQLAnywhere at `Sap.Data.SQLAnywhere.v3.5`.
- For all other versions of .NET, run *%SQLANY17%*`\Assembly\v4.5\SetupVSPackage.exe /install`. By default, this tool will register the version 4.5 provider with Entity Framework 5 support. You can use the `/version` option to specify the version (short form:`/v`). Use `/v 4.5` to install the version 4.5 provider with Entity Framework 5 support. Use `/v EF6` to install the version 4.5 provider with Entity Framework 6 support. The Global Assembly Cache (GAC) will contain all versions of the .NET provider assemblies. The Windows Microsoft.NET version 4 `machine.config` file is updated to point the provider invariant name Sap.Data.SQLAnywhere at the selected provider.
- By default, SetupVSPackage will use SQL Anywhere registry settings to locate the .NET assemblies. You can specify the location of the SQL Anywhere installation by using the `salocation` option.

  ```
  %SQLANY17%\Assembly\v4.5\SetupVSPackage.exe /install /salocation %SQLANY17%
  ```

  The short form for `salocation` is `sal`.

If Microsoft SQL Server 2008 or later is installed on the system, SetupVSPackage also installs two mapping files called `MSSqlToSA.xml` and `SAToMSSql10.xml` to the Microsoft SQL Server `DTS\MappingFiles` folder.

SetupVSPackage also installs `SSDLToSA17.tt` to the Microsoft Visual Studio 2010 or later directory. It is used for generating database schema DDL for Entity Data Models. The user should set the DDL Generation property to this file when generating database schema DDL for Entity Data Models.

To create your own installation certain files must be deployed to your end users. There are several message files, each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).

Each .NET client computer must have the following:

A working .NET installation

Microsoft .NET assemblies and instructions for their redistribution are available from Microsoft Corporation. They are not described in detail here.

SQL Anywhere .NET Data Provider for .NET Framework 2.0/3.0/3.5

The SQL Anywhere installation places the assemblies for this provider in the `Assembly\V3.5` subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system binaries directory of your SQL Anywhere installation directory (for example, `bin32` and `bin64`). The following files are required.

```
Sap.Data.SQLAnywhere.v3.5.dll
policy.17.0.Sap.Data.SQLAnywhere.v3.5.dll
dblg[LL]17.dll
dbicu17.dll
dbicudt17.dll
dbrsa17.dll
```

SQL Anywhere .NET Data Provider for .NET Framework 4.x with Entity Framework 5

The SQL Anywhere installation places the Windows assemblies for the .NET Framework version 4.x with Entity Framework 5 support in the `Assembly\.5` subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system binaries directory of your SQL Anywhere installation directory (for example, `bin32` and `bin64`). The following files are required.

```
Sap.Data.SQLAnywhere.v4.5.dll
policy.17.0.Sap.Data.SQLAnywhere.v4.5.dll
dblg[LL]17.dll
dbicu17.dll
dbicudt17.dll
dbrsa17.dll
```

SQL Anywhere .NET Data Provider for .NET Framework 4.x with Entity Framework 6

The SQL Anywhere installation places the Windows assemblies for the .NET Framework version 4.x with Entity Framework 6 support in the `Assembly\V4.5` subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system binaries directory of your SQL Anywhere installation directory (for example, `bin32` and `bin64`). The following files are required.

```
Sap.Data.SQLAnywhere.EF6.dll
policy.17.0.Sap.Data.SQLAnywhere.EF6.dll
dblg[LL]17.dll
dbicu17.dll
dbicudt17.dll
dbrsa17.dll
```

The policy files can be used to override the provider version that the application was built with. The policy file is updated whenever an update to the provider is released.

ClickOnce deployment allows you to publish .NET applications to a web server or network file share for easy installation.

In this section:

ClickOnce and .NET Data Provider Unmanaged Code DLLs [page 829]
ClickOnce deployment allows you to publish .NET applications to a web server or network file share for easy installation. Microsoft Visual Studio supports the publishing and updating of applications deployed using the ClickOnce technology.

SQL Anywhere .NET Data Provider Removal [page 830]
To uninstall the current version of the SQL Anywhere .NET Data Provider, there are some steps you must perform.

Related Information

The Deployment Wizard for Windows [page 813]

1.23.8.1.1 ClickOnce and .NET Data Provider Unmanaged Code DLLs

ClickOnce deployment allows you to publish .NET applications to a web server or network file share for easy installation. Microsoft Visual Studio supports the publishing and updating of applications deployed using the ClickOnce technology.

To assist in the deployment of the unmanaged code portions of the SQL Anywhere .NET Data Provider, a sample deployment application is provided that illustrates how to add DLLs as resources to the .NET application that you are deploying. Source code for this sample is located in the *%SQLANYSAMP17%* `\SQLAnywhere\ADO.NET\DeployUtility` folder.

Build.cmd

This batch file shows you how to add the SQL Anywhere .NET Data Provider unmanaged DLLs as resources to your application executable. In particular, it shows you how to add the 32-bit DLLs, `dblgen`*17*`.dll`, `dbicu`*17*`.dll`, `dbicudt`*17*`.dll`, and `dbrsa`*17*`.dll`, as resources to a .NET application. This technique can be extended to include the 64-bit versions of the DLLs.

Program.cs

This sample program shows you how to access the DLLs as resources attached to your application and write them to a directory so that they are available to the SQL Anywhere .NET Data Provider. The technique shown can be embellished to handle both 32-bit and 64-bit DLLs.

1.23.8.1.2 SQL Anywhere .NET Data Provider Removal

To uninstall the current version of the SQL Anywhere .NET Data Provider, there are some steps you must perform.

If you choose to uninstall the SQL Anywhere .NET Data Provider assemblies, use the following information.

> **Note**
>
> Uninstalling the .NET Data Provider affects Microsoft Visual Studio integration for all versions of the provider, including earlier major versions such as 16.0 and 12.0. Only one version of the .NET Data Provider can be used with Microsoft Visual Studio integration.

- Ensure Microsoft Visual Studio is not running.
- Use the SetupVSPackage tool to uninstall the SQL Anywhere .NET Data Provider assemblies. SetupVSPackage requires Administrator privilege for Windows 7 and later systems. If you plan to run SetupVSPackage at a Command Prompt, ensure the Command Prompt has Administrator privilege.
- For .NET 2.0/3.x only, run *%SQLANY17%*`\Assembly\v3.5\SetupVSPackage.exe /uninstall`.
- For other versions, run *%SQLANY17%*`\Assembly\v4.5\SetupVSPackage.exe /uninstall`.

The short form for */uninstall* is /u.

To uninstall all builds of the 17.0.11 version of the SQL Anywhere .NET Data Provider, you must do the following.

- Ensure Microsoft Visual Studio is not running.
- Use the SetupVSPackage tool to uninstall all builds of the SQL Anywhere .NET Data Provider assemblies. SetupVSPackage requires Administrator privilege for Windows 7 and later systems. If you plan to run SetupVSPackage at a Command Prompt, ensure the Command Prompt has Administrator privilege.
- For .NET 2.0/3.x only, run *%SQLANY17%*`\Assembly\v3.5\SetupVSPackage.exe /uninstallall`.
- For other versions, run *%SQLANY17%*`\Assembly\v4.5\SetupVSPackage.exe /uninstallall`.

The short form for `/uninstallall` is `/ua`.

1.23.8.2 OLE DB and ADO Client Deployment

OLE DB and ADO applications that use the SQL Anywhere OLE DB provider require a number of provider components.

The simplest way to deploy OLE DB client libraries is to use the *Deployment Wizard*. To create your own installation, required files must be deployed to end users.

Each client system must have the following OLE DB components:

A working OLE DB installation

OLE DB files and instructions for their redistribution are available from Microsoft Corporation. For Windows clients, use Microsoft MDAC 2.7 or later.

The OLE DB provider

The following table shows the files needed for the OLE DB provider. For Windows, there are both 32-bit and 64-bit versions of these files. For some Windows platforms, there are two provider DLLs. The second DLL (`dboledba`*17*`.dll`) is an assist DLL used to provide schema support.

Description	Windows
OLE DB provider library	`dboledb`*17*`.dll`
OLE DB schema assist library	`dboledba`*17*`.dll`
Connect window	`dbcon`*17*`.dll`
Character set translation	`dbicu`*17*`.dll`
Character set translation data	`dbicudt`*17*`.dll`
Language-resource library	`dblg[LL]`*17*`.dll`
Encryption support	`dbrsa`*17*`.dll`
Elevated operations agent	`dbelevate`*17*`.exe` (Windows 7 or later only)

The table above shows a file with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).

OLE DB providers require many registry entries. You can create these entries by self-registering the `dboledb`*17*`.dll` and `dboledba`*17*`.dll` DLLs using the regsvr32 utility.

For Windows 7 or later, you must include the SQL Anywhere elevated operations agent which supports the privilege elevation required when DLLs are registered or unregistered. This file is only required as part of the OLE DB provider install or uninstall procedure.

1.23.8.3 OLE DB Provider Customization

You can customize the OLE DB provider to have your own version of the provider.

When installing the OLE DB provider, the Microsoft Windows Registry is modified using the self-registration capability built into the provider. Normally, the regsvr32 tool is used to register the OLE DB provider.

In a typical OLE DB connection string, one of the components is the Provider attribute. To indicate that the OLE DB provider is to be used, you specify the name of the provider. Here is a Microsoft Visual Basic example:

```
connectString = "Provider=SAOLEDB;DSN=SQL Anywhere 17 Demo;PWD="+myPwd
```

With ADO and/or OLE DB, there are other ways to reference the provider by name. Here is a C++ example in which you specify not only the provider name but also the version to use.

```
hr = db.Open(_T("SAOLEDB.17"), &dbinit);
```

The provider name is looked up in the registry. If you were to examine the registry on your computer system, you would find an entry in HKEY_CLASSES_ROOT for SAOLEDB.

```
[HKEY_CLASSES_ROOT\SAOLEDB]
@="SQL Anywhere OLE DB provider"
```

It has two subkeys that contain a class identifier (Clsid) and current version (CurVer) for the provider. Here is an example.

```
[HKEY_CLASSES_ROOT\SAOLEDB\Clsid]
```

```
@="{41dfe9f7-db91-11d2-8c43-006008d26a6f}"
[HKEY_CLASSES_ROOT\SAOLEDB\CurVer]
@="SAOLEDB.17"
```

There are several more similar entries. They are used to identify a specific instance of an OLE DB provider. If you look up the Clsid in the registry under HKEY_CLASSES_ROOT\CLSID and examine the subkeys, you see that one of the entries identifies the location of the provider DLL.

```
[HKEY_CLASSES_ROOT\CLSID\
{41dfe9f3-db91-11d2-8c43-006008d26a6f}\
InprocServer32]
@="c:\\sa17\\bin64\\dboledb17.dll"
"ThreadingModel"="Both"
```

However, there are risks with this model. If you uninstall SQL Anywhere from your system, the OLE DB provider registry entries would be removed from your registry and then the provider DLL would be removed from your hard drive. Any applications that depend on the provider would no longer work.

Similarly, when multiple applications use the same OLE DB provider, each installation of the same provider overwrites the common registry settings. So the version of the provider that you intended your application to work with would be overwritten by another version (newer or older) of the provider. For this reason, the OLE DB provider is customizable.

In this section:

Customizing the OLE DB Provider [page 832]
Create a unique OLE DB provider that you can deploy with your application by generating a unique set of GUIDs, naming the provider, and choosing unique DLL names.

1.23.8.3.1 Customizing the OLE DB Provider

Create a unique OLE DB provider that you can deploy with your application by generating a unique set of GUIDs, naming the provider, and choosing unique DLL names.

Procedure

1. Copy the following into a text editor and save it as a `.reg` file:

```
REGEDIT4
; Special registry entries for a private OLE DB provider.
[HKEY_CLASSES_ROOT\myoledb17]
@="Custom SQL Anywhere OLE DB provider 17"
[HKEY_CLASSES_ROOT\myoledb17\Clsid] @="{GUID1}"
; Data1 of the following GUID must be 3 greater than the
; previous, for example, 41dfe9f3 + 3 => 41dfe9ee.
[HKEY_CLASSES_ROOT\myoledba17]
@="Custom SQL Anywhere OLE DB provider 17"
[HKEY_CLASSES_ROOT\myoledba17\Clsid] @="{GUID4}"
; Current version (or version independent prog ID)
; entries (what you get when you have "SQLAny"
; instead of "SQLAny.17")
[HKEY_CLASSES_ROOT\SQLAny]
```

```
@="SQL Anywhere OLE DB provider"
[HKEY_CLASSES_ROOT\SQLAny\Clsid]
@="{GUID1}"
[HKEY_CLASSES_ROOT\SQLAny\CurVer]
@="SQLAny.17"
[HKEY_CLASSES_ROOT\SQLAnyEnum]
@="SQL Anywhere OLE DB provider Enumerator"
[HKEY_CLASSES_ROOT\SQLAnyEnum\Clsid]
@="{GUID2}" [HKEY_CLASSES_ROOT\SQLAnyEnum\CurVer]
@="SQLAnyEnum.17" [HKEY_CLASSES_ROOT\SQLAnyErrorLookup]
@="SQL Anywhere OLE DB provider Extended Error Support"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\Clsid]
@="{GUID3}"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\CurVer]
@="SQLAnyErrorLookup.17"
[HKEY_CLASSES_ROOT\SQLAnyA]
@="SQL Anywhere OLE DB provider Assist"
[HKEY_CLASSES_ROOT\SQLAnyA\Clsid]
@="{GUID4}"
[HKEY_CLASSES_ROOT\SQLAnyA\CurVer]
@="SQLAnyA.17"
; Standard entries (Provider=SQLAny.17)
[HKEY_CLASSES_ROOT\SQLAny.17]
@="SAP SQL Anywhere OLE DB provider 17"
[HKEY_CLASSES_ROOT\SQLAny.17\Clsid]
@="{GUID1}"
[HKEY_CLASSES_ROOT\SQLAnyEnum.17]
@="SAP SQL Anywhere OLE DB provider Enumerator 17"
[HKEY_CLASSES_ROOT\SQLAnyEnum.17\Clsid]
@="{GUID2}"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.17]
@="SAP SQL Anywhere OLE DB provider Extended Error Support 17"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.17\Clsid]
@="{GUID3}"
[HKEY_CLASSES_ROOT\SQLAnyA.17]
@="SAP SQL Anywhere OLE DB provider Assist 17"
[HKEY_CLASSES_ROOT\SQLAnyA.17\Clsid]
@="{GUID4}"
; SQLAny (Provider=SQLAny.17)
[HKEY_CLASSES_ROOT\CLSID\{GUID1}]
@="SQLAny.17"
"OLEDB_SERVICES"=dword:ffffffff
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors]
@="Extended Error Service"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors\{GUID3}]
@="SAP SQL Anywhere OLE DB provider Error Lookup"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\InprocServer32]
@="d:\\mypath\\bin32\\myoledb17.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\OLE DB Provider]
@="SAP SQL Anywhere OLE DB provider 17"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ProgID]
@="SQLAny.17"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\VersionIndependentProgID]
@="SQLAny"
; SQLAnyErrorLookup
[HKEY_CLASSES_ROOT\CLSID\{GUID3}]
@="SAP SQL Anywhere OLE DB provider Error Lookup 17.0"
@="SQLAnyErrorLookup.17"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\InprocServer32]
@="d:\\mypath\\bin32\\myoledb17.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\ProgID]
@="SQLAnyErrorLookup.17"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\VersionIndependentProgID]
@="SQLAnyErrorLookup"
; SQLAnyEnum [HKEY_CLASSES_ROOT\CLSID\{GUID2}]
@="SQLAnyEnum.17"
```

```
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\InprocServer32]
@="d:\\mypath\\bin32\\myoledb17.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\OLE DB Enumerator]
@="SAP SQL Anywhere OLE DB provider Enumerator"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\ProgID]
@="SQLAnyEnum.17"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\VersionIndependentProgID]
@="SQLAnyEnum"
; SQLAnyA [HKEY_CLASSES_ROOT\CLSID\{GUID4}]
@="SQLAnyA.17"
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\InprocServer32]
@="d:\\mypath\\bin32\\myoledba17.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\ProgID]
@="SQLAnyA.17"
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\VersionIndependentProgID]
@="SQLAnyA"
```

Registry value names are case sensitive.

2. Use the Microsoft Visual Studio uuidgen utility to create 4 sequential UUIDs (GUIDs).

   For example, run the following command at a command prompt:

   ```
   uuidgen -n4 -s -x >oledbguids.txt
   ```

   The 4 UUIDs or GUIDs are assigned in the following sequence:

   a. The Provider class ID (GUID1 below).
   b. The Enum class ID (GUID2 below).
   c. The ErrorLookup class ID (GUID3 below).
   d. The Provider Assist class ID (GUID4 below).

   It is important that they are sequential, which is what the -x option in the command line does.

   Each GUID should appear similar to the following:

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

   > **Note**
   >
   > The first part of the GUID (for example, 41dfe9f3) that is incrementing.

3. Use the search/replace capability of an editor to change the GUID1, GUID2, GUID3, and GUID4 in the text to the corresponding GUID (for example, GUID1 would be replaced by 41dfe9f3-db92-11d2-8c43-006008d26a6f if that was the GUID generated for you by uuidgen).
4. Choose your Provider name that is used in your application in connection strings, and so on (for example, Provider=SQLAny).

The following names are reserved by SQL Anywhere and cannot be used as a Provider name:

Version 10 or Later	Version 9 or Earlier
SAOLEDB	ASAProv
SAErrorLookup	ASAErrorLookup
SAEnum	ASAEnum
SAOLEDBA	ASAProvA

5. Use the search/replace capability of an editor to change all the occurrences of the string SQLAny to the provider name that you have chosen. This includes all those places where SQLAny may be a substring of a longer string (for example, SQLAnyEnum).

   Suppose you chose Acme for your provider name. The names that will appear in the HKEY_CLASSES_ROOT registry hive are shown in the following table along with the SQL Anywhere names (for comparison).

SQL Anywhere	Your Custom Provider
SAOLEDB	Acme
SAErrorLookup	AcmeErrorLookup
SAEnum	AcmeEnum
SAOLEDBA	AcmeA

6. Make copies of the SQL Anywhere OLE DB provider DLLs (`dboledb`*17*`.dll` and `dboledba`*17*`.dll`) under different names.

   ```
   copy dboledb17.dll myoledb17.dll
   copy dboledba17.dll myoledba17.dll
   ```

   A special registry key is created by the script that is based on the DLL name that you choose. It is important that the name be different from the standard DLL names (such as `dboledb`*17*`.dll` or `dboledba`*17*`.dll`). If you name the provider DLL `myoledb`*17* then the provider will look up a registry entry in HKEY_CLASSES_ROOT with that same name. The same is true of the provider schema assist DLL. If you name the DLL `myoledba`*17* then the provider will look up a registry entry in HKEY_CLASSES_ROOT with that same name. It is important that the name you choose is unique and is unlikely to be chosen by anyone else. Here are some examples.

DLL Name(s) Chosen	Corresponding HKEY_CLASSES_ROOT\Name
`myoledb`*17*`.dll`	HKEY_CLASSES_ROOT\myoledb*17*
`myoledba`*17*`.dll`	HKEY_CLASSES_ROOT\myoledba*17*
`acmeOledb.dll`	HKEY_CLASSES_ROOT\acmeOledb
`acmeOledba.dll`	HKEY_CLASSES_ROOT\acmeOledba
`SAcustom.dll`	HKEY_CLASSES_ROOT\SAcustom
`SAcustomA.dll`	HKEY_CLASSES_ROOT\SAcustomA

7. Use the search/replace capability of an editor to change all the occurrences of `myoledb`*17* and `myoledba`*17* in the registry script to the two DLL names you have chosen.

8. Use the search/replace capability of an editor to change all the occurrences of `d:\\mypath\\bin32\\` in the registry script to the installed location for the DLLs. Be sure to use a pair of backslashes to represent a single backslash. This step must be customized at the time of your application install.
9. Save the registry script to disk and run it.
10. Give your new provider a try. Do not forget to change your ADO / OLE DB application to use the new provider name.

Results

Your custom OLE DB provider has been configured.

Next Steps

Deploy the custom OLE DB provider with your application.

1.23.8.4 ODBC Client Deployment

Applications that use the SQL Anywhere ODBC driver require a number of components.

The simplest way to deploy ODBC clients is to use the *Deployment Wizard*.

Each ODBC client computer must have the following:

ODBC Driver Manager

Microsoft provides an ODBC Driver Manager for Windows operating systems. SQL Anywhere includes an ODBC Driver Manager for Linux, Unix, and macOS. ODBC applications can run without a driver manager but, on platforms for which an ODBC driver manager is available, this is not recommended.

Connection information

The client application must have access to the information needed to connect to the server. This information is typically included in an ODBC data source.

ODBC driver

The SQL Anywhere ODBC driver must be installed.

In this section:

ODBC Driver Required Files [page 837]
The SQL Anywhere ODBC driver requires a number of components.

ODBC Driver Configuration [page 840]
In addition to copying the ODBC driver files onto disk, your installation program must also make a set of registry entries to install the ODBC driver properly.

Database Connection Information for ODBC Applications [page 841]

ODBC client connection information is generally deployed as an ODBC data source.

Related Information

The Deployment Wizard for Windows [page 813]
The Deployment Wizard for UNIX/Linux [page 822]

1.23.8.4.1 ODBC Driver Required Files

The SQL Anywhere ODBC driver requires a number of components.

The following table shows the files needed for a working SQL Anywhere ODBC driver. For most operating-system platforms, there are both 32-bit and 64-bit versions of these files.

The multithreaded version of the ODBC driver for Linux, Unix, and macOS platforms is indicated by "MT".

Platform	Required Files

Windows	`dbodbc17.dll`
	`dbcon17.dll`
	`dbicu17.dll`
	`dbicudt17.dll`
	`dblg[LL]17.dll`
	`dbrsa17.dll`
	`dbelevate17.exe`

Linux, Solaris, HP-UX	`libdbodbc17.so.1`
	`libdbodbc17_n.so.1`
	`libdbodm17.so.1`
	`libdbtasks17.so.1`
	`libdbicu17.so.1`
	`libdbicudt17.so.1`
	`libdbrsa17.so.1`
	`dblg[LL]17.res`

Platform	Required Files
Linux, Solaris, HP-UX MT	libdbodbc*17*.so.1 libdbodbc*17*_r.so.1 libdbodm*17*.so.1 libdbtasks*17*_r.so.1 libdbicu*17*_r.so.1 libdbicudt*17*.so.1 libdbrsa*17*_r.so.1 dblg[LL]*17*.res
AIX	libdbodbc*17*.so libdbodbc*17*_n.so libdbodm*17*.so libdbtasks*17*.so libdbicu*17*.so libdbicudt*17*.so libdbrsa*17*.so dblg[LL]*17*.res
AIX MT	libdbodbc*17*.so libdbodbc*17*_r.so libdbodm*17*.so libdbtasks*17*_r.so libdbicu*17*_r.so libdbicudt*17*.so libdbrsa*17*_r.so dblg[LL]*17*.res

Platform	Required Files
macOS	dbodbc*17*.bundle libdbodbc*17*.dylib libdbodbc*17*_n.dylib libdbodm*17*.dylib libdbtasks*17*.dylib libdbicu*17*.dylib libdbicudt*17*.dylib libdbrsa*17*.dylib dblg[LL]*17*.res
macOS MT	dbodbc*17*_r.bundle libdbodbc*17*.dylib libdbodbc*17*_r.dylib libdbodm*17*.dylib libdbtasks*17*_r.dylib libdbicu*17*_r.dylib libdbicudt*17*.dylib libdbrsa*17*_r.dylib dblg[LL]*17*.res

Notes

- For Linux and Solaris platforms, create a link to the `.so.1` files. The link name matches the file name with the ".1" version suffix removed.
- There are multithreaded (MT) versions of the ODBC driver for Linux, Unix, and macOS platforms. The file names contain the "_r" suffix. Deploy these files if your application requires them.
- For Windows, a driver manager is included with the operating system. For Linux, Unix, and macOS, SQL Anywhere provides a driver manager. The file name begins with `libdbodm`*17*.
- For Windows 7 or later, you must include the SQL Anywhere elevated operations agent (`dbelevate`*17*`.exe`) which supports the privilege elevation required to register or unregister the ODBC driver. This file is only required as part of the ODBC driver install or uninstall procedure.
- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).

- For Windows, the *ODBC Configuration for SQL Anywhere* and *Connect to SQL Anywhere* window support code (`dbcon17.dll`) is needed if your end users create their own data sources, if they enter user IDs and passwords when connecting to the database, or if the Connect window is displayed for any other purpose.

1.23.8.4.2 ODBC Driver Configuration

In addition to copying the ODBC driver files onto disk, your installation program must also make a set of registry entries to install the ODBC driver properly.

Windows

The SQL Anywhere installer makes changes to the Windows Registry to identify and configure the ODBC driver. If you are building an installation program for your end users, you must make the same registry settings.

The simplest way to do this is to use the self-registering capability of the ODBC driver. You use the regsvr32 utility on Windows. For 64-bit versions of Windows, you can register both the 64-bit and 32-bit versions of the ODBC driver. By using the self-registering feature of the ODBC driver, you are ensured that the proper registry entries are created.

Give custom names to the 32-bit and 64-bit versions of the SQL Anywhere ODBC drivers that you are installing. This facilitates the installation and registration of multiple independent copies of the SQL Anywhere ODBC driver using regsvr32, and prevents your registry settings from being overwritten if another application install registers the SQL Anywhere ODBC driver.

To customize the names of the 32-bit and 64-bit versions of the ODBC driver, open a command prompt and rename the driver file as follows where `custom-name` is a meaningful string such as your company name:

```
ren dbodbc17.dll "dbodbc17custom-name.dll"
```

Do this for both the 32-bit and 64-bit versions. Preserve the `dbodbc17` prefix when renaming the files (this is required).

Register the ODBC driver using a command like the following:

```
regsvr32 "dbodbc17custom-name.dll"
```

Do this for both the 32-bit and 64-bit versions.

You can use the regedit utility to inspect the registry entries created by the ODBC driver.

This copy of the SQL Anywhere ODBC driver is identified to the system by a set of registry values in the following registry key:

```
HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\SQL Anywhere 17 - custom-name
```

Sample values for 64-bit Windows are shown below:

Name	Type	Data
Driver	REG_SZ	`install-dir\bin64\dbodbc`*`17`*`custom-name.dll`
Setup	REG_SZ	`install-dir\bin64\dbodbc`*`17`*`custom-name.dll`

There is also a registry entry in the following key:

```
HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\ODBC Drivers
```

The entry is as follows:

Name	Type	Data
SQL Anywhere 17 - `custom-name`	REG_SZ	Installed

64-bit Windows

For 64-bit Windows, the 32-bit ODBC driver registry entries ("*SQL Anywhere 17* - `custom-name`" and "ODBC Drivers") are located under the following key:

```
HKEY_LOCAL_MACHINE\Software\Wow6432Node\ODBC\ODBCINST.INI
```

To view these entries, you must be using a 64-bit version of regedit. If you cannot locate Wow6432Node on 64-bit Windows, then you are most-likely using the 32-bit version of regedit.

Third Party ODBC Drivers

If you are using a third-party ODBC driver on an operating system other than Windows, consult the documentation for that driver on how to configure the ODBC driver.

1.23.8.4.3 Database Connection Information for ODBC Applications

ODBC client connection information is generally deployed as an ODBC data source.

You can deploy an ODBC data source in one of the following ways:

Programmatically

Add a data source description to your end-user's registry or ODBC initialization files. The dbdsn utility is useful for this purpose.

Manually

Provide your end users with instructions, so that they can create an appropriate data source on their own computer.

On Microsoft Windows platforms, you can create a data source manually using the dbdsn utility. Alternatively, you can create a data source using the ODBC Data Source Administrator, from the User DSN tab or the System DSN tab. The SQL Anywhere ODBC driver displays the configuration window for entering settings. Data source settings include the location of the database file, the name of the database server, and any startup parameters and other options. Context sensitive help is accessed by clicking the *Help* button. Help text is provided locally using `sacshelp17.chm`. If this file is not present, then help is provided over the Internet from DocCommentXchange using the system's default Internet browser.

On UNIX and Linux platforms, you can create a data source manually using the dbdsn utility. Data source settings include the location of the database file, the name of the database server, any startup parameters, and other options.

The information you must know for either approach is presented below.

Types of Data Source (Microsoft Windows)

There are three kinds of data sources: User Data Sources, System Data Sources, and File Data Sources.

1. User Data Source definitions are stored in the part of the registry containing settings for the specific user currently logged on to the system.
2. System Data Sources, however, are available to all users and to Microsoft Windows services, which run regardless of whether a user is logged onto the system or not. Given a correctly configured System Data Source named MyApp, any user can use that ODBC data source by providing DSN=MyApp in the ODBC connection string.
3. File Data Sources are not held in the registry, but are stored on disk. A connection string must provide a FileDSN connection parameter to use a File Data Source. The default location of File Data Sources is specified by the *HKEY_CURRENT_USER\Software\ODBC\odbc.ini\ODBC File DSN\DefaultDSNDir* registry entry or by the *HKEY_LOCAL_MACHINE\Software\ODBC\odbc.ini\ODBC File DSN\DefaultDSNDir* registry entry (if the former is not defined). The path can be included in the FileDSN connection parameter to help locate the File Data Source when it is located elsewhere.

Data Source Registry Entries (Microsoft Windows)

User and System Data Source definitions are stored in the Microsoft Windows Registry. The simplest way to ensure the correct creation of registry entries for data source definitions is to use the dbdsn utility to create them.

For System Data Sources on 64-bit Microsoft Windows, there are two sets of registry entries - one for 64-bit applications and one for 32-bit applications. To avoid problems, both sets should be created. Use the 64-bit version of dbdsn to create the 64-bit registry key and use the 32-bit version of dbdsn to create the 32-bit registry key.

Otherwise, you must create a set of registry values in a particular registry key.

For User Data Sources, the registry key is as follows:

```
HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\user-data-source-name
```

For System Data Sources on 32-bit Microsoft Windows, the registry key is as follows:

```
HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI\system-data-source-name
```

For System Data Sources on 64-bit Microsoft Windows, there are two registry keys - one for 64-bit applications and one for 32-bit applications. The 64-bit registry key is as follows:

```
HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI\system-data-source-name
```

The 32-bit registry key is as follows

```
HKEY_LOCAL_MACHINE\Software\Wow6432Node\ODBC\ODBC.INI\system-data-source-name
```

The key contains a set of registry values, each of which corresponds to a connection parameter (except for the Driver string). The Driver string is automatically added to the registry by the Microsoft ODBC Driver Manager when a Data Source is created using the Microsoft ODBC Data Source Administrator or the dbdsn utility. For example, the SQL Anywhere 17 Demo key corresponding to the SQL Anywhere 17 Demo System Data Source Name (DSN) contains the following settings for 32-bit Microsoft Windows:

Value Name	Value Type	Value Data
AutoStop	String	YES
DatabaseFile	String	*C:\Users\Public\Documents\SQL Anywhere 17\Samples**demo.db*
Description	String	SQL Anywhere 17 Sample Database
Driver	String	*C:\Program Files\SQL Anywhere 17*\bin32\dbodbc*17*.dll
ServerName	String	demo 17
StartLine	String	*C:\Program Files\SQL Anywhere 17*\bin32*dbeng17*.exe
UID	String	DBA

> **Note**
>
> Include the ServerName parameter in connection strings for deployed applications. This ensures that the application connects to the correct server if a computer is running multiple SQL Anywhere database servers and can help prevent timing-dependent connection failures.

In these entries, for 64-bit Microsoft Windows, `bin32` would be replaced by `bin64`.

On 64-bit Microsoft Windows, there is only one registry entry for User Data Sources that is shared by both 64-bit and 32-bit applications.

ODBC Data Sources List (Microsoft Windows)

In addition, you must add the data source name to the list of data sources in the registry.

For User Data Sources, you use the following key:

```
HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\ODBC Data Sources
```

For System Data Sources on 32-bit Microsoft Windows, use the following key:

```
HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI\ODBC Data Sources
```

For System Data Sources on 64-bit Microsoft Windows, there are two registry keys - one for 64-bit applications and one for 32-bit applications. The 64-bit registry key is as follows::

```
HKEY_LOCAL_MACHINE\Software\Wow6432Node\ODBC\ODBC.INI\ODBC Data Sources
```

The 32-bit registry key is as follows::

```
HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI\ODBC Data Sources
```

The value associates each data source with an ODBC driver. The value name is the data source name, and the value data is the ODBC driver name. For example, the System Data Source installed by SQL Anywhere is named SQL Anywhere 17 Demo, and has the following value:

Value Name	Value Type	Value Data
SQL Anywhere 17 Demo	String	*SQL Anywhere 17*

> ⚠ Caution
>
> ODBC Data Source Name settings are easily viewed. User Data Source configurations can contain sensitive database settings such as a user's ID and password. These settings are stored in the registry in plain text, and can be viewed using the Microsoft Windows Registry editors `regedit.exe` or `regedt32.exe`, which are provided by Microsoft with the operating system. You can choose to encrypt passwords, or require users to enter them when connecting.

Required and Optional Connection Parameters

You can identify the data source name in an ODBC connection string in this manner,

```
DSN=ADataSourceName
```

On Microsoft Windows, when a DSN parameter is provided in the connection string, the User Data Source definitions in the Microsoft Windows Registry are searched, followed by System Data Sources. File Data Sources are searched only when the FileDSN connection parameter is provided in the ODBC connection string.

The following table illustrates the implications to the user and the application developer when a data source exists and is included in the application's connection string as a DSN or FileDSN parameter.

When the data source...	The connection string must also identify...	The user must supply...
Contains the ODBC driver name and location; the name of the database file/server; startup parameters; and the user ID and password.	No additional information	No additional information.
Contains the ODBC driver name and location; the name of the database file/server; startup parameters.	No additional information	User ID and password if not provided in the ODBC data source.
Contains only the name and location of the ODBC driver.	The name of the database file (DBF=) and/or the database server (SERVER=). Optionally, it may contain other connection parameters such as Userid (UID=) and Password (PWD=).	User ID and password if not provided in the ODBC data source or ODBC connection string.
Does not exist	The name of the ODBC driver to be used (Driver=) and the database name (DBN=), the database file (DBF=), and/or the database server (SERVER=). Optionally, it may contain other connection parameters such as Userid (UID=) and Password (PWD=).	User ID and password if not provided in the ODBC connection string.

Related Information

Database Connections

1.23.8.5 Embedded SQL Client Deployment

Applications that use Embedded SQL require a number of components.

The simplest way to deploy Embedded SQL clients is to use the *Deployment Wizard*.

Deploying Embedded SQL clients involves the following:

Installed files

Each client computer must have the files required for an Embedded SQL client application.

Connection information

The client application must have access to the information needed to connect to the server. This information may be included in an ODBC data source.

In this section:

Files Required for Embedded SQL Clients [page 846]
Embedded SQL applications require a number of components.

Database Connection Information for Embedded SQL Applications [page 847]

Embedded SQL connection information can be deployed in a number of ways.

Related Information

The Deployment Wizard for Windows [page 813]
The Deployment Wizard for UNIX/Linux [page 822]

1.23.8.5.1 Files Required for Embedded SQL Clients

Embedded SQL applications require a number of components.

The following table shows which files are needed for Embedded SQL clients.

Description	Microsoft Windows	Linux/UNIX	macOS
Interface library	`dblib17.dll`	`libdblib17_r.so`	`libdblib17_r.dylib`
Standard encryption support	`dbrsa17.dll`, `sapcrypto.dll`	`libdbrsa17_r.so`, `libsapcrypto.so`	`libdbrsa17_r.dylib`, `libsapcrypto.dylib`

Separately licensed encryption support	`dbfips17.dll`
	`sapcryptofips.dll`
	`slcryptokernel.dll`
	`slcryptokernel.dll.sha256`
	`msvcr90.dll` (the 64-bit version of this file is called `msvcr100.dll`)
		`libdbfips17_r.so`
		`libsapcryptofips.so`
		`libslcryptokernel.so`
		`libslcryptokernel.so.sha256`
			N/A

Thread support library	N/A	`libdbtasks17_r.so`	`libdbtasks17_r.dylib`
Language resource library	`dblg[LL]17.dll`	`dblg[LL]17.res`	`dblg[LL]17.res`
Connect window	`dbcon17.dll`	N/A	N/A

Notes

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).
- For non-multithreaded applications on Linux/Unix, use `libdblib17.so` and `libdbtasks17.so`.
- For non-multithreaded applications on macOS, use `libdblib17.dylib` and `libdbtasks17.dylib`.

- If the client application uses encryption then the appropriate encryption support should also be included.
- If the client application uses an ODBC data source to hold the connection parameters, your end user must have a working ODBC installation. Instructions for deploying ODBC are included in the Microsoft ODBC SDK.
- For Windows, the *ODBC Configuration for SQL Anywhere* and *Connect to SQL Anywhere* window support code (`dbcon17.dll`) is required if your end users will create their own data sources, if they will enter user IDs and passwords when connecting to the database, or if they require the *Connect* window for any other purpose.

Related Information

ODBC Client Deployment [page 836]

1.23.8.5.2 Database Connection Information for Embedded SQL Applications

Embedded SQL connection information can be deployed in a number of ways.

You can use any of the following methods:

Manual

Provide your end users with instructions for creating an appropriate data source on their computer.

File

Distribute a file that contains connection information in a format that your application can read.

ODBC data source

You can use an ODBC data source to hold connection information.

1.23.8.6 JDBC Client Deployment

Applications that use JDBC require a number of components.

You must install a Java Runtime Environment (JRE) to use JDBC. Version 1.8.0 (JRE 8) or later is recommended.

In addition to a JRE, each JDBC client requires the SQL Anywhere JDBC driver or the jConnect driver.

SQL Anywhere JDBC Driver

To deploy the SQL Anywhere JDBC driver, you must deploy `sajdbc4.jar` which is located in the SQL Anywhere installation `java` folder. This file must be listed in the application's classpath.

In addition, you must deploy a number of JDBC driver support files. The following table shows the required files for various platforms. These files should be placed in a single directory. The SQL Anywhere installation places them all in a binaries subdirectory of your SQL Anywhere installation directory (for example, `bin32`, `bin64`, `lib32`, or `lib64`). The choice of 32-bit or 64-bit files depends on the bitness of your installed Java VM.

The JDBC driver support files for all platforms are multithreaded.

Platform	Required Files

Microsoft Windows	`dbjdbc`*17*`.dll`
	`dbicu`*17*`.dll`
	`dbicudt`*17*`.dll`
	`dblg[LL]`*17*`.dll`
	`dbrsa`*17*`.dll`

Linux, Solaris, HP-UX	`libdbjdbc`*17*`.so.1`
	`libdbtasks`*17*`_r.so.1`
	`libdbicu`*17*`_r.so.1`
	`libdbicudt`*17*`.so.1`
	`libdbrsa`*17*`_r.so.1`
	`dblg[LL]`*17*`.res`

AIX	`libdbjdbc`*17*`.so`
	`libdbtasks`*17*`_r.so`
	`libdbicu`*17*`_r.so`
	`libdbicudt`*17*`.so`
	`libdbrsa`*17*`_r.so`
	`dblg[LL]`*17*`.res`

macOS	`libdbjdbc`*17*`.dylib`
	`libdbtasks`*17*`_r.dylib`
	`libdbicu`*17*`_r.dylib`
	`libdbicudt`*17*`.dylib`
	`libdbrsa`*17*`_r.dylib`
	`dblg[LL]`*17*`.res`

- For Linux and Solaris platforms, create a link to the `.so.1` files. The link name should match the file name with the ".1" version suffix removed.
- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different

languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).

jConnect JDBC Driver

To deploy the jConnect JDBC driver, you must deploy the following files:

- The jConnect driver files.
- When you use a TDS client (either Open Client or jConnect based), you have the option of sending the connection password in clear text or in encrypted form. The latter is done by performing a TDS encrypted password handshake. The handshake involves using private/public key encryption. The support for generating the RSA private/public key pair and for decrypting the encrypted password is included in a special library. The library file must be locatable by the SQL Anywhere server in its system path. For Microsoft Windows, this file is called `dbrsakp17.dll`. There are both 64-bit and 32-bit versions of the DLL. In UNIX and Linux environments, the file is a shared library called `libdbrsakp17.so`. On macOS, the file is a shared library called `libdbrsakp17.dylib`. The file is not necessary if you do not use this feature.

JDBC Database Connection URL

Your Java application needs a URL to connect to the database. This URL specifies the driver, the computer to use, and the port on which the database server is listening.

Related Information

jConnect Driver Connection Strings [page 229]
Downloading jConnect

1.23.8.7 PHP Client Deployment

Applications that use the SQL Anywhere PHP extension require a number of components.

To deploy the PHP extension, you must install the following components on the target platform:

- The PHP binaries for your platform which are available from The SAP SQL Anywhere PHP Module. For Microsoft Windows platforms, the thread-safe version of PHP must be used with the PHP extension.
- A web server such as Apache HTTP Server to run PHP scripts within a web server.
  The database server can be run on the same computer as the web server, or on a different computer.
- Supporting shared objects or libraries.

The following table summarizes the files required for PHP clients.

Description	Microsoft Windows	Linux/UNIX	macOS
PHP installation (third-party)	`php.exe`	`php`	`php`
PHP extension	`php-x.y.0_sqlanywhere.dll`	`php-x.y.0_sqlanywhere_r.so`	Build from source code
Language resource library	`dblg[LL]17.dll`	`dblg[LL]17.res`	`dblg[LL]17.res`
Connect window	`dbcon17.dll`	N/A	N/A
SQL Anywhere C API runtime	`dbcapi.dll`	`libdbcapi_r.so`	`libdbcapi_r.dylib`
DBLIB (threaded)	`dblib17.dll`	`libdblib17_r.so`	`libdblib17_r.dylib`
Thread support library	N/A	`libdbtasks17_r.so`	`libdbtasks17_r.dylib`
ICU library	`dbicu17.dll`	`libdbicu17_r.so` [2]	`libdbicu17_r.dylib` [2]
ICU Data library	`dbicudt17.dll`	`libdbicudt17.so` [2]	`libdbicudt17.dylib` [2]
Standard encryption support	`dbrsa17.dll` `sapcrypto.dll`	`libdbrsa17_r.so` `libsapcrypto.so`	`libdbrsa17_r.dylib` `libsapcrypto.dylib`

Separately licensed encryption support	`dbfips17.dll`
	`sapcryptofips.dll`
	`slcryptokernel.dll`
	`slcryptokernel.dll.sha256`
	`msvcr90.dll` (the 64-bit version of this file is called `msvcr100.dll`)
		`libdbfips17_r.so`
		`libsapcryptofips.so`
		`libslcryptokernel.so`
		`libslcryptokernel.so.sha256`
			N/A

Notes

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).
- If the client application uses encryption then the appropriate encryption support should also be included.
- If the client application uses an ODBC data source to hold the connection parameters, your end user must have a working ODBC installation. Instructions for deploying ODBC are included in the Microsoft ODBC SDK.
- For Microsoft Windows, the *ODBC Configuration for SQL Anywhere* and *Connect to SQL Anywhere* window support code (`dbcon17.dll`) is required if your end users will create their own data sources, if they will enter user IDs and passwords when connecting to the database, or if they require the *Connect* window for any other purpose.

- For more information about installing the PHP extension, see the topics that follow.

In this section:

PHP Extensions [page 851]
Some PHP applications may require a thread-safe version of the PHP extension. Other PHP applications may require a non-thread-safe version of the PHP extension.

Installing the PHP Extension on Windows [page 852]
Copy the extension DLL to your PHP installation to use the SQL Anywhere PHP extension on Windows. Optionally, add an entry to your PHP initialization file to load the extension automatically, to avoid loading it manually in each script.

Installing the PHP Extension on Linux [page 853]
Copy the extension shared object to your PHP installation to use the SQL Anywhere PHP extension on Linux. Optionally, add an entry to your PHP initialization file to load the extension automatically, to avoid loading it manually in each script.

The PHP Extension on UNIX/Linux [page 855]
To use the SQL Anywhere PHP extension on other versions of UNIX and Linux, you must build the PHP extension from the source code which is installed in the `sdk/php` subdirectory of your SQL Anywhere installation.

Configuration of the SQL Anywhere PHP Extension [page 855]
The behavior of the SQL Anywhere PHP extension can be controlled by setting values in the PHP initialization file, `php.ini`.

Related Information

ODBC Client Deployment [page 836]
Getting Started with SAP SQL Anywhere and PHP
The SAP SQL Anywhere PHP Module
PHP

1.23.8.7.1 PHP Extensions

Some PHP applications may require a thread-safe version of the PHP extension. Other PHP applications may require a non-thread-safe version of the PHP extension.

For Windows, a thread-safe version of the PHP extension must be used with a thread-safe version of PHP. The thread-safe extension file name follows this pattern where x.y represents the PHP version.

```
php-x.y.0_sqlanywhere.dll
```

For Windows, a non-thread-safe version of PHP can be used with multiprocessed but not multithreaded web servers such as Apache 1.x. The non-thread-safe PHP extension file name follows this pattern where x.y represents the PHP version.

```
php-x.y.0_sqlanywhere_nts.dll
```

For Linux, there are also thread-safe and non-thread-safe versions of the PHP extension.

If you are using Apache 2.x, use the thread-safe extension. The thread-safe extension file name follows this pattern where x.y represents the PHP version.

```
php-x.y.0_sqlanywhere_r.so
```

If you are using the CGI version of PHP or if you are using Apache 1.x, use the non-thread-safe extension. The non-thread-safe extension file name follows this pattern where x.y represents the PHP version.

```
php-x.y.0_sqlanywhere.so
```

Related Information

The SAP SQL Anywhere PHP Module

1.23.8.7.2 Installing the PHP Extension on Windows

Copy the extension DLL to your PHP installation to use the SQL Anywhere PHP extension on Windows. Optionally, add an entry to your PHP initialization file to load the extension automatically, to avoid loading it manually in each script.

Prerequisites

You must have PHP 5.4 or later installed.

Procedure

1. Download prebuilt versions of the PHP extension for Windows from The SAP SQL Anywhere PHP Module.
2. Locate the `php.ini` file for your PHP installation, and open it in a text editor.
3. Locate the line that specifies the location of the extension_dir entry.
4. If the entry does not exist, create the extension_dir entry and point it to an isolated directory for better system security.
5. Copy the `php-x.y.0_sqlanywhere.dll` file to the directory specified by the extension_dir entry in the `php.ini` file.

   The string `x.y` corresponds to the PHP version that you have installed.

   Your choice of DLL depends on the version of PHP that is installed, and whether it is a 32-bit or a 64-bit version. There is also a version that is not thread-safe (`php-x.y.0_sqlanywhere_nts.dll`).

If your version of PHP is more recent than any of the prebuilt extensions, you must build the extension. Source code for the PHP extension is installed in the `sdk\php` subdirectory of your software installation.

6. Add a line similar to the following to the Dynamic Extensions section of the `php.ini` file to load the PHP extension automatically.

   ```
   extension=php-x.y.0_sqlanywhere.dll
   ```

   Without this step, the PHP extension needs to be loaded manually every time a script requires it.
7. Save and close `php.ini`.
8. The 32-bit version of the PHP extension requires the `Bin32` directory to be in your path. The 64-bit version of the PHP extension requires the `Bin64` directory to be in your path.

Results

The PHP extension is supported on your Windows environment and ready for use.

Related Information

Configuration of the SQL Anywhere PHP Extension [page 855]
Creating and Running PHP Test Pages [page 471]

1.23.8.7.3 Installing the PHP Extension on Linux

Copy the extension shared object to your PHP installation to use the SQL Anywhere PHP extension on Linux. Optionally, add an entry to your PHP initialization file to load the extension automatically, to avoid loading it manually in each script.

Prerequisites

You must have PHP 5.4 or later installed.

Procedure

1. Download prebuilt versions of the PHP extension for Linux from The SAP SQL Anywhere PHP Module.
2. Locate the `php.ini` file for your PHP installation, and open it in a text editor.
3. Locate the line that specifies the location of the extension_dir entry.

4. If the entry does not exist, create the extension_dir entry and point it to an isolated directory for better system security.
5. Copy the `php-x.y.0_sqlanywhere_r.so` file to the directory specified by the extension_dir entry in the `php.ini` file.

   The string `x.y` is the PHP version number corresponding to the version that you have installed. The `_r` suffix indicates the thread-safe version of the PHP extension.

   Your choice of shared object depends on the version of PHP that is installed, and whether it is a 32-bit or a 64-bit version. There is also a version that is not thread-safe (`php-x.y.0_sqlanywhere.so`).

   If your version of PHP is more recent than any of the prebuilt extensions, you must build the extension. Source code for the PHP extension is installed in the `sdk/php` subdirectory of your software installation.
6. Add a line similar to the following to the Dynamic Extensions section of the `php.ini` file to load the PHP extension automatically.

   ```
   extension=php-x.y.0_sqlanywhere_r.so
   ```

   Without this step, the PHP extension needs to be loaded manually every time a script requires it.
7. Save and close `php.ini`.
8. Verify that your PHP execution environment is set up for SQL Anywhere.

   Depending on which shell you are using, you must edit the configuration script for your web server's environment and add the appropriate command to source the SQL Anywhere configuration script from the SQL Anywhere installation directory.

Option	Action
sh, ksh, or bash	`. /bin32/sa_config.sh`
csh or tcsh	`source /bin32/sa_config.csh`

   The 32-bit version of the PHP extension requires the `bin32` directory to be in your path. The 64-bit version of the PHP extension requires the `bin64` directory to be in your path.

   The configuration file in which this line should be inserted is different for different web servers and on different Linux distributions. Here are some examples for the Apache server on the indicated distributions:

   RedHat/Fedora/CentOS

   /etc/sysconfig/httpd

   Debian/Ubuntu

   /etc/apache2/envvars

> **Note**
>
> The web server must be restarted after editing its environment configuration.

Results

The PHP extension is supported on your Linux environment and ready for use.

Related Information

PHP Extensions [page 851]
Configuration of the SQL Anywhere PHP Extension [page 855]
Creating and Running PHP Test Pages [page 471]

1.23.8.7.4 The PHP Extension on UNIX/Linux

To use the SQL Anywhere PHP extension on other versions of UNIX and Linux, you must build the PHP extension from the source code which is installed in the `sdk/php` subdirectory of your SQL Anywhere installation.

> **Note**
>
> On macOS 10.11 or a later version, set the SQLANY_API_DLL environment variable to the full path for `libdbcapi_r.dylib`.

Related Information

Building the PHP Extension on UNIX/Linux [page 477]

1.23.8.7.5 Configuration of the SQL Anywhere PHP Extension

The behavior of the SQL Anywhere PHP extension can be controlled by setting values in the PHP initialization file, `php.ini`.

The following entries are supported:

extension

Causes PHP to load the SQL Anywhere PHP extension automatically each time PHP starts. Adding this entry to your PHP initialization file is optional, but if you don't add it, each script you write must start with a few lines of code that ensure that this extension is loaded. The following entry is used for Windows platforms.

```
extension=php-x.y.0_sqlanywhere.dll
```

On Linux platforms, use one of the following entries. The second entry is thread safe.

```
extension=php-x.y.0_sqlanywhere.so
```

```
extension=php-x.y.0_sqlanywhere_r.so
```

In these entries, `x.y` identifies the PHP version that you are using.

If the SQL Anywhere PHP extension is not always automatically loaded when PHP starts, you must prefix each script you write with the following lines of code. This code ensures that the SQL Anywhere PHP extension is loaded.

```
# Ensure that the SQL Anywhere PHP extension is loaded
if( !extension_loaded('sqlanywhere') ) {
    # Find out which version of PHP is running
    $version = explode('.',phpversion());
    $xy = $version[0].'.'.$version[1].'.0';
    $extension_name = 'php-'.$xy.'_sqlanywhere';
    if( strtoupper(substr(PHP_OS, 0, 3) == 'WIN' )) {
        $extension_ext = '.dll';
    } else {
        $extension_ext = '.so';
    }
    dl( $extension_name.$extension_ext );
}
```

allow_persistent

Allows persistent connections when set to On. It does not allow them when set to Off. The default value is On.

```
sqlanywhere.allow_persistent=On
```

max_persistent

Sets the maximum number of persistent connections. The default value is -1, which means no limit.

```
sqlanywhere.max_persistent=-1
```

max_connections

Sets the maximum number of connections that can be opened at once through the SQL Anywhere PHP extension. The default value is -1, which means no limit.

```
sqlanywhere.max_connections=-1
```

auto_commit

Specifies whether the database server performs a commit operation automatically. The commit is performed immediately following the execution of each statement when set to On. When set to Off, transactions should be ended manually with either the sasql_commit or sasql_rollback functions, as appropriate. The default value is On.

```
sqlanywhere.auto_commit=On
```

row_counts

Returns the exact number of rows affected by an operation when set to On or an estimate when set to Off. The default value is Off.

```
sqlanywhere.row_counts=Off
```

verbose_errors

Returns verbose errors and warnings when set to On. Otherwise, you must call the sasql_error or sasql_errorcode functions to get further error information. The default value is On.

```
sqlanywhere.verbose_errors=On
```

Related Information

sasql_set_option [page 509]

1.23.8.8 Open Client Application Deployment

To deploy Open Client applications, each client computer needs the SAP Open Client product.

When you use a TDS client (either Open Client or jConnect based), you have the option of sending the connection password in clear text or in encrypted form. The latter is done by performing a TDS encrypted password handshake. The handshake involves using private/public key encryption. The support for generating the RSA private/public key pair and for decrypting the encrypted password is included in a special library. The library file must be locatable by the SQL Anywhere server in its system path. For Windows, this file is called `dbrsakp`*17*`.dll`. There are both 64-bit and 32-bit versions of the DLL. On Linux and Unix environments, the file is a shared library called `libdbrsakp`*17*`.so`. On macOS, the file is a shared library called `libdbrsakp`*17*`_r.dylib`. The file is not necessary if you do not use this feature.

Connection information for Open Client clients is held in the interfaces file.

1.23.9 Administration Tool Deployment

Subject to your license agreement, you can deploy a set of administration tools including Interactive SQL and *SQL Central*.

The simplest way to deploy the administration tools is to use the *Deployment Wizard*.

Initialization files can simplify the deployment of the administration tools. Each of the launcher executables for the administration tools (*SQL Central* and Interactive SQL) can have a corresponding `.ini` file. This eliminates the need for registry entries and a fixed directory structure for the location of the JAR files. These `ini` files are located in the same directory and with the same file name as the executable file.

dbisql.ini

This is the name of the Interactive SQL initialization file.

scjview.ini

This is the name of the *SQL Central* initialization file.

The initialization file will contain the details on how to load the database administration tool. For example, the initialization file can contain the following lines:

JRE_DIRECTORY=path

This is the location of the required JRE. The JRE_DIRECTORY specification is required.

VM_ARGUMENTS=any-required-VM-arguments

VM arguments are separated by semicolons (;). Any path values that contain blanks should be enclosed in quotation marks. VM arguments can be discovered by using the `-batch` option of the administration tool and examining the corresponding batch file that is created. For example, on Windows, launching *SQL Central* with `scjview -batch` at a command prompt generates `scjview.bat` and launching Interactive SQL with `dbisql -batch` generates `dbisql.bat`. The VM_ARGUMENTS specification is optional.

JAR_PATHS=path1 ; path2 ;...

A delimited list of directories which contain the JAR files for the program. They are separated by semicolons (;). TheJAR_PATHS specification is optional.

ADDITIONAL_CLASSPATH=path1 ; path2 ;...

Classpath values are separated by semicolons (;). The ADDITIONAL_CLASSPATH specification is optional.

LIBRARY_PATHS=path1 ; path2 ;...

These are paths to the DLLs/shared objects. They are separated by semicolons (;). The LIBRARY_PATHS specification is optional.

APPLICATION_ARGUMENTS=arg1 ; arg2 ;...

These are any application arguments. They are separated by semicolons (;). Application arguments can be discovered by using the `-batch` option of the administration tool and examining the corresponding batch file that is created. For example, on Windows, launching *SQL Central* with `scjview -batch` at a command prompt generates `scjview.bat` and launching Interactive SQL with `dbisql -batch` generates `dbisql.bat`. The APPLICATION_ARGUMENTS specification is optional.

Here are the contents of a sample initialization file for *SQL Central*.

```
JRE_DIRECTORY=c:\jdk1.8.0\jre
VM_ARGUMENTS=
JAR_PATHS=c:\scj\jars
ADDITIONAL_CLASSPATH=
LIBRARY_PATHS=c:\scj\bin
APPLICATION_ARGUMENTS=-screpository=C:\Users\Public\Documents\SQL Central 17;-
installdir=c:\scj
```

This scenario assumes that a copy of the JRE is located in `c:\jdk1.8.0\jre`. As well, the *SQL Central* executable and shared libraries (DLLs) like `jsyblib`*17*`00` are stored in `c:\scj\bin`. The JAR files are stored in `c:\scj\jars`.

> **Note**
>
> When you are deploying applications, the personal database server (dbeng17) is required for creating databases using the dbinit utility. It is also required if you are creating databases from *SQL Central* on the local computer when no other database servers are running.

1. Administration Tool Deployment on Windows [page 859]

To install Interactive SQL (dbisql) and *SQL Central* (including the SQL Anywhere, MobiLink, and UltraLite plug-ins) on a Windows computer without using the Deployment Wizard, certain steps must be followed. The information here is intended for those who want to create an installer for these administration tools.

2. Administration Tool Deployment on Linux, Solaris, and macOS [page 868]
   To install Interactive SQL (dbisql) and *SQL Central* (including the SQL Anywhere and MobiLink plug-ins) on Linux, Solaris, and macOS computers, certain steps must be followed. The information here is intended for those who want to create an installer for these administration tools.
3. Administration Tools Configuration [page 875]
   You can specify which features are shown or enabled by the administration tools by using the initialization file named `OEM.ini`.
4. dbisqlc Deployment [page 878]
   If your deployed application requires a query execution or scripting tool and is running on computers with limited resources, you could deploy the dbisqlc executable instead of Interactive SQL (dbisql).
5. About the SQL AnywhereMonitor Production Edition [page 879]
   The Production Edition is intended for deployment and production use. It is installed separately, runs as a service, and includes a fully contained SQL Anywhere installation.

Related Information

The Deployment Wizard for Windows [page 813]
The Deployment Wizard for UNIX/Linux [page 822]
Supported Platforms

1.23.9.1 Administration Tool Deployment on Windows

To install Interactive SQL (dbisql) and *SQL Central* (including the SQL Anywhere, MobiLink, and UltraLite plug-ins) on a Windows computer without using the Deployment Wizard, certain steps must be followed. The information here is intended for those who want to create an installer for these administration tools.

This information applies to all supported Windows platforms. The instructions given here are specific to version 17.0.11 and cannot be applied to earlier or later versions of the software.

> **Note**
>
> Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

Before You Begin

An understanding of the Windows Registry, including the REGEDIT application, is required. The names of the registry values are case sensitive.

> ⚠ Caution
>
> Modifying the Windows Registry is dangerous and you do so at your own risk. Back up your system before modifying the registry.

In this section:

Step 1: Decide What Software to Deploy [page 860]
Depending on your requirements, the administration tools may be bundled in a number of ways.

Step 2: Copy the Required Files [page 861]
The administration tools require a specific directory structure.

Step 3: Register the Administration Tools with Windows [page 865]
The administration tools require certain registry keys to be set up. The names of the registry values are case sensitive.

Step 4: Update the System Path [page 866]
To run the administration tools, the directories with `.exe` and `.dll` files must be included in the path. You must add one of the `c:\sa`*17*`\Bin32` or `c:\sa`*17*`\Bin64` directories to the system path.

Step 5: Create Connection Profiles for SQL Central [page 866]
This step involves the configuration of *SQL Central*. If you are not installing *SQL Central*, you can skip it.

Step 6: Register the SQL Anywhere ODBC Driver [page 867]
You must install the SQL Anywhere ODBC driver before it can be used by the administration tools JDBC driver.

Parent topic: Administration Tool Deployment [page 857]

Next: Administration Tool Deployment on Linux, Solaris, and macOS [page 868]

1.23.9.1.1 Step 1: Decide What Software to Deploy

Depending on your requirements, the administration tools may be bundled in a number of ways.

You can install any combination of the following software bundles:

- Interactive SQL
- *SQL Central* with the SQL Anywhere plug-in
- *SQL Central* with the MobiLink plug-in
- *SQL Central* with the UltraLite plug-in

The following components are also required when installing any of the above software bundles:

- The SQL Anywhere ODBC Driver.
- Java SE Runtime Environment (JRE) 8. You could install the 32-bit version of the JRE, the 64-bit version of the JRE, or both depending on your target platform.
- The Java Access Bridge for Microsoft Windows Operating System (to enable accessibility features of the administration tools on Windows platforms). Enabling this feature will permit the administration tools to be used with screen reader technologies.

The steps presented here are structured so that you can install any (or all) of these bundles without conflicts.

1.23.9.1.2 Step 2: Copy the Required Files

The administration tools require a specific directory structure.

You are free to put the directory tree in any directory, on any drive. Throughout the following discussion, `c:\sa17` is used as the example installation folder. The software must be installed into a directory tree structure having the following layout:

Directory	Description
`sa17`	The root folder. While the following steps assume you are installing into `c:\sa17`, you are free to put the directory anywhere (for example, `C:\Program Files\SQLAny17`).
`sa17\Bin32`	Holds the native 32-bit Windows components used by the program, including the programs that launch the applications.
`sa17\Bin32\jre180`	The 32-bit Java Runtime Environment.
`sa17\Bin64`	Holds the native 64-bit Windows components used by the program, including the programs that launch the applications.
`sa17\Bin64\jre180`	The 64-bit Java Runtime Environment.
`sa17\Java`	Holds Java program JAR files.

The following tables list the files required for each of the administration tools and the *SQL Central* plug-ins. Make a list of the files you need, and then copy them into the directory structure outlined above.

The tables show files with the folder designation BinXX. There are 32-bit and 64-bit versions of these files, in the `Bin32` and `Bin64` folders respectively. If you are installing 32-bit and 64-bit administration tools, then you must install both sets of files in their respective folders.

The tables show files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to add the resource files for these languages.

The administration tools require JRE 1.8.0. Do not substitute a later patch version of the JRE unless you have a specific requirement to do so.

If you are deploying 32-bit administration tools, then copy the 32-bit version of the JRE files from the *%SQLANY17%*`\Bin32\jre180` directory. Copy the entire `jre180` tree, including subdirectories.

If you are deploying 64-bit administration tools, then copy the 64-bit version of the JRE files from the *%SQLANY17%*`\Bin64\jre180` directory. Copy the entire `jre180` tree, including subdirectories.

Interactive SQL

Interactive SQL (dbisql) requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbelevate17.exe
BinXX\dbicu17.dll
BinXX\dbicudt17.dll
BinXX\dbisql.com
BinXX\dbisql.exe
BinXX\dbjodbc17.dll
BinXX\dblg[LL]17.dll
BinXX\dblib17.dll
BinXX\dbodbc17.dll
BinXX\dbrsa17.dll
BinXX\jsyblib1700.dll
BinXX\jre180\...
Java\batik-all-1.9.jar
Java\isql.jar
Java\JComponents1700.jar
Java\jodbc4.jar
Java\jsyblib1700.jar
Java\ngdbc.jar
Java\saip17.jar
Java\SCEditor1700.jar
Java\xml-apis-ext.jar
Java\xmlgraphics-commons-2.2.jar
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied. The 32-bit version of Interactive SQL requires the 32-bit version of the JRE files (`Bin32\jre180`). The 64-bit version of Interactive SQL requires the 64-bit version of the JRE files (`Bin64\jre180`).

SQL Central

SQL Central (scjview) requires the following files.

```
BinXX\jsyblib1700.dll
BinXX\scjview.exe
BinXX\jre180\...
Java\jsyblib1700.jar
Java\SCEditor1700.jar
Java\sqlcentral1700.jar
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied. The 32-bit version of *SQL Central* requires the 32-bit version of the JRE files (`Bin32\jre180`). The 64-bit version of *SQL Central* requires the 64-bit version of the JRE files (`Bin64\jre180`).

SQL Central with SQL Anywhere Plug-in

The SQL Anywhere plug-in of *SQL Central* requires the personal server (*dbeng17*`.exe` and supporting files) and the following files.

```
BinXX\dbdsn.exe
```

```
BinXX\dbicu17.dll
BinXX\dbicudt17.dll
BinXX\dbjodbc17.dll
BinXX\dblg[LL]17.dll
BinXX\dblib17.dll
BinXX\dbodbc17.dll
BinXX\dbput17.dll
BinXX\dbrsa17.dll
BinXX\dbtool17.dll
Java\batik-all-1.9.jar
Java\dbdiff.jar
Java\debugger.jar
Java\diffutils-1.2.1.jar
Java\isql.jar
Java\JComponents1700.jar
Java\jodbc4.jar
Java\ngdbc.jar
Java\saip17.jar
Java\saplugin.jar
Java\salib.jar
Java\SQLAnywhere.jpr
Java\xml-apis-ext.jar
Java\xmlgraphics-commons-2.2.jar
```

SQL Central with MobiLink Plug-in

The MobiLink plug-in of *SQL Central* requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbicu17.dll
BinXX\dbicudt17.dll
BinXX\dblg[LL]17.dll
BinXX\dblib17.dll
BinXX\dbput17.dll
BinXX\dbrsa17.dll
BinXX\dbtool17.dll
BinXX\mljodbc17.dll
Java\commons-collections-4.4.2.jar
Java\commons-lang-2.6.jar
Java\commons-logging-1.2.jar
Java\isql.jar
Java\JComponents1700.jar
Java\jodbc4.jar
Java\mldesign.jar
Java\mlplugin.jar
Java\MobiLink.jpr
Java\ngdbc.jar
Java\saip17.jar
Java\salib.jar
Java\velocity-engine-core-2.0-dep.jar
java\commons-io-2.6
java\commons-lang3-3.8.1.jar
java\slf4j-1.7.25.jar
```

SQL Central with UltraLite Plug-in

The UltraLite plug-in of *SQL Central* requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbicu17.dll
BinXX\dbicudt17.dll
BinXX\dblg[LL]17.dll
BinXX\dblib17.dll
BinXX\dbput17.dll
BinXX\dbrsa17.dll
BinXX\dbtool17.dll
BinXX\mlcrsa17.dll
BinXX\ulscutil17.dll
BinXX\ulutils17.dll
Java\batik-all-1.9.jar
Java\isql.jar
Java\JComponents1700.jar
Java\jodbc4.jar
Java\ngdbc.jar
Java\saip17.jar
Java\salib.jar
Java\ulplugin.jar
Java\UltraLite.jpr
Java\xml-apis-ext.jar
Java\xmlgraphics-commons-2.2.jar
```

International Message and Context-Sensitive Help Files

All displayed text and context-sensitive help for the administration tools is translated from English into French, German, Japanese, and Simplified Chinese. The resources for each language are held in separate files. The English files contain en in the file names. French files have similar names, but use fr instead of en. German file names contain de, Japanese file names contain ja, and Chinese file names contain zh.

To install support for different languages, you have to add the message files for those other languages. There are 32-bit and 64-bit versions of these files, in the Bin32 and Bin64 folders respectively. If you are installing 32-bit and 64-bit administration tools, then you must install both sets of files in their respective folders. The translated files are as follows:

`dblgen17.dll`	English
`dblgde17.dll`	German
`dblgfr17.dll`	French
`dblgja17.dll`	Japanese
`dblgzh17.dll`	Simplified Chinese

These files are included with localized versions of SQL Anywhere.

Accessibility Components

The Java Access Bridge provides accessibility for the Java-based administration tools (you can download the Java Access Bridge for Microsoft Windows Operating System from the Java web site). Here are the install locations for the accessibility components.

32-bit administration tools

`accessibility.properties`	`Bin32\jre180\lib`
`jaccess.jar`	`Bin32\jre180\lib\ext`
`access-bridge.jar`	`Bin32\jre180\lib\ext`
`JavaAccessBridge.dll`	`Bin32\jre180\bin`
`JAWTAccessBridge.dll`	`Bin32\jre180\bin`
`WindowsAccessBridge.dll`	`%windir%\system32` (32-bit Windows) or `%windir%\SysWOW64` (64-bit Windows)

64-bit administration tools

`accessibility.properties`	`Bin64\jre180\lib`
`jaccess.jar`	`Bin64\jre180\lib\ext`
`access-bridge.jar`	`Bin64\jre180\lib\ext`
`JavaAccessBridge-64.dll`	`Bin64\jre180\bin`
`JAWTAccessBridge-64.dll`	`Bin64\jre180\bin`
`WindowsAccessBridge-64.dll`	`%windir%\system32`

Your installer must create an `accessibility.properties` file in the `BinXX\jre180\lib` folder and it must contain an *assistive_technologies* option as shown below.

```
#
# Load the Java Access Bridge class into the JVM
#
assistive_technologies=com.sun.java.accessibility.AccessBridge
#screen_magnifier_present=true
```

The SQL Anywhere installer creates this file but all lines are commented. To enable assistive technologies, the *assistive_technologies* line must be uncommented (the # must be removed).

1.23.9.1.3 Step 3: Register the Administration Tools with Windows

The administration tools require certain registry keys to be set up. The names of the registry values are case sensitive.

You must set the following registry key for the administration tools. The names of the registry values are case sensitive.

- In *HKEY_LOCAL_MACHINE\Software\SAP\SQL Anywhere\17.0*

  Location

  The fully qualified path to the root of the installation folder (*C:\Program Files\SQL Anywhere 17* for example).

You may set the following registry keys for the administration tools. The registry keys are optional and only required to be set if the OS language is different from the language that the administration tools will use. The names of the registry keys are case sensitive.

- In *HKEY_LOCAL_MACHINE\Software\SAP\SQL Anywhere\17.0*

  Language

  The two-letter code for the language used by SQL Anywhere and the administration tools. This must be one of the following: EN, DE, FR, JA, or ZH for English, German, French, Japanese, and Simplified Chinese, respectively.

On 64-bit Windows, the equivalent registry entries for 32-bit software are under *Software\Wow6432Node\SAP*.

Paths should *not* end in a backslash.

Your installer can encapsulate all this information by creating a `.reg` file and then executing it. Using the example installation folder of `c:\sa17`, the following is a sample `.reg` file:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\Software\SAP\SQL Anywhere\17.0]
"Location"="c:\\sa17"
"Language"="FR"
```

Backslashes in file paths must be escaped by another backslash in a `.reg` file.

1.23.9.1.4 Step 4: Update the System Path

To run the administration tools, the directories with `.exe` and `.dll` files must be included in the path. You must add one of the `c:\sa17\Bin32` or `c:\sa17\Bin64` directories to the system path.

On Windows, the system path is stored in the following registry key:

```
HKEY_LOCAL_MACHINE\
  SYSTEM\
    CurrentControlSet\
      Control\
        Session Manager\
          Environment\
            Path
```

1.23.9.1.5 Step 5: Create Connection Profiles for *SQL Central*

This step involves the configuration of *SQL Central*. If you are not installing *SQL Central*, you can skip it.

When *SQL Central* is installed on your system, a connection profile for SQL Anywhere 17 Demo is created in the repository file. If you do not want to create one or more connection profiles, then you can skip this step.

The following commands were used to create the SQL Anywhere 17 Demo connection profile. Use this as a model for creating your own connection profiles.

```
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Name" "SQL Anywhere 17
Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/FirstTimeStart" "false"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Description" "Suitable
Description"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/ProviderId"
"sqlanywhere1700"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Provider" "SQL Anywhere
17"
scjview -write "ConnectionProfiles/SQL
Anywhere 17 Demo/Data/ConnectionProfileSettings"
"DSN\eSQL^0020Anywhere^002016^0020Demo;UID\eDBA;PWD\e35c624d517fb"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Data/
ConnectionProfileName" "SQL Anywhere 17 Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Data/
ConnectionProfileType" "SQL Anywhere"
```

The connection profile strings and values can be extracted from the repository file. Define a connection profile using *SQL Central* and then look at the repository file for the corresponding lines.

Here is a portion of the repository file that was created using the process described above .The repository file can be found in `C:\users\public\Documents\SQL Central 17`. Some entries have been split across multiple lines for display purposes. In the file, each entry appears on a single line:

```
#  Version: 17.0.1154
#  Thu Oct 04 12:07:53 EDT 2012
#
ConnectionProfiles/SQL Anywhere 17 Demo/Name=SQL Anywhere 17 Demo
ConnectionProfiles/SQL Anywhere 17 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 17 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 17 Demo/ProviderId=sqlanywhere1700
ConnectionProfiles/SQL Anywhere 17 Demo/Provider=SQL Anywhere 17
ConnectionProfiles/SQL Anywhere 17 Demo/Data/ConnectionProfileSettings=
    DSN\eSQL^0020Anywhere^002017^0020Demo;
    UID\eDBA;
    PWD\e35c624d517fb
ConnectionProfiles/SQL Anywhere 17 Demo/Data/ConnectionProfileName=
    SQL Anywhere 17 Demo
ConnectionProfiles/SQL Anywhere 17 Demo/Data/ConnectionProfileType=
     SQL Anywhere
```

1.23.9.1.6 Step 6: Register the SQL Anywhere ODBC Driver

You must install the SQL Anywhere ODBC driver before it can be used by the administration tools JDBC driver.

Related Information

ODBC Driver Configuration [page 840]

1.23.9.2 Administration Tool Deployment on Linux, Solaris, and macOS

To install Interactive SQL (dbisql) and *SQL Central* (including the SQL Anywhere and MobiLink plug-ins) on Linux, Solaris, and macOS computers, certain steps must be followed. The information here is intended for those who want to create an installer for these administration tools.

The instructions given here are specific to version 17.0.11 and may not be applicable to earlier or later versions of the software.

Note also that the dbisqlc command line utility is supported on Linux, Solaris, macOS, HP-UX, and AIX.

> **Note**
>
> Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

Before You Begin

Before you begin, you must install SQL Anywhere on one computer as a source for program files. This is the **reference installation** for your deployment.

The general steps involved are as follows:

1. Decide which programs you want to deploy.
2. Copy the required files.
3. Set environment variables.
4. Register the *SQL Central* plug-ins.

Each of these steps is explained in detail.

In this section:

Step 1: Decide What Software to Deploy [page 869]
Depending on your requirements, the administration tools may be bundled in a number of ways.

Step 2: Copy the Required Files [page 869]
Your installer copies a subset of the files that are installed by the SQL Anywhere installer. Keep the same directory structure.

Step 3: Set Environment Variables [page 873]
To run the administration tools, several environment variables must be defined or modified. This is usually done in the `sa_config.sh` file, which is created by the SQL Anywhere installer. To use the `sa_config.sh` file, just copy it and set SQLANY17 to point to the deployment location.

Step 4: Create Connection Profiles for SQL Central [page 874]
This step involves the configuration of *SQL Central*. If you are not installing *SQL Central*, you can skip it.

Parent topic: Administration Tool Deployment [page 857]

Previous: Administration Tool Deployment on Windows [page 859]

Next: Administration Tools Configuration [page 875]

Related Information

dbisqlc Deployment [page 878]

1.23.9.2.1 Step 1: Decide What Software to Deploy

Depending on your requirements, the administration tools may be bundled in a number of ways.

You can install any combination of the following software bundles:

- Interactive SQL
- *SQL Central* with the SQL Anywhere plug-in
- *SQL Central* with the MobiLink plug-in

The following components are also required when installing any of the above software bundles:

- The SQL Anywhere ODBC Driver
- Java SE Runtime Environment 8 (JRE 8) or later. There are both 32-bit and 64-bit versions of the JRE to consider.

> **Note**
>
> To check your JRE version on macOS, to go to the *Apple* menu, and then click *System Preferences* > *Software Updates*. Click *Installed Updates* for a list of updates that have been applied.

The instructions in the next section are structured so that you can install any (or all) of these five bundles without conflicts.

1.23.9.2.2 Step 2: Copy the Required Files

Your installer copies a subset of the files that are installed by the SQL Anywhere installer. Keep the same directory structure.

Preserve the permissions on the files when you copy them from your reference SQL Anywhere installation. In general, all users and groups are allowed to read and execute all files.

The administration tools require JRE 1.8.0. Do not substitute a later patch version of the JRE unless you have a specific requirement to do so.

For Linux/Solaris, the administration tools require the either the 64-bit or 32-bit version of the JRE (depending on your target architecture). The MobiLink server requires the 64-bit version of the JRE.

For macOS, the administration tools require the 64-bit version of the JRE.

Not all platform versions of the JRE are bundled with SQL Anywhere. The UNIX platforms that are included with SQL Anywhere support Linux on x86/x64 and Solaris SPARC. Other platforms versions must be obtained from the appropriate vendor.

If the platform that you require is included with SQL Anywhere, then copy the JRE files from an installed copy of SQL Anywhere. Copy the entire tree, including subdirectories.

For example, if you are working with a Linux install, copy the entire `jre180` tree, including subdirectories.

The following tables list the files required for each of the administration tools and the *SQL Central* plug-ins. Make a list of the files you need, and then copy them into the directory structure outlined above.

The tables show files with the folder designation binXX. Depending on the platform, there are 32-bit and 64-bit versions of these files, in the `bin32` and `bin64` folders respectively. If you are installing 32-bit and 64-bit administration tools, then install both sets of files in their respective folders.

The tables show files with the folder designation libXX. Depending on the platform, there are 32-bit and 64-bit versions of these files, in the `lib32` and `lib64` folders respectively. If you are installing 32-bit and 64-bit administration tools, then install both sets of files in their respective folders.

The creation of several links is required for the administration tools and *SQL Central* plug-ins.

For Linux and Solaris, create symbolic links for all the shared objects that you deploy. Also, create a symbolic link in *$SQLANY17*`/bin64` or *$SQLANY17*`/bin32`. The symbolic link for Linux and other systems is `jre180`. Here are some examples:

```
libdblib17_r.so -> $SQLANY17/lib32/libdblib17_r.so.1
jre180 -> $SQLANY17/bin32/jre180
```

For the MobiLink plug-in on 64-bit Linux, create an additional symbolic link in *$SQLANY17*`/bin64`. The symbolic link for Linux is `jre180` for the 64-bit JRE.

```
jre180 -> $SQLANY17/bin64/jre180 (Linux)
```

For macOS, shared objects have a `.dylib` extension. Symlink (symbolic link) creation is necessary for the following dylibs:

```
libdbjodbc17.jnilib -> libdbjodbc17.dylib
libdblib17_r.jnilib -> libdblib17_r.dylib
libdbput17_r.jnilib -> libdbput17_r.dylib
libmljodbc17.jnilib -> libmljodbc17.dylib
```

The tables show files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, add the resource files for these languages.

Interactive SQL

Interactive SQL (dbisql) requires the following files.

```
binXX/dbisql
binXX/jre180/...
libXX/libdbicu17_r.so (.dylib)
libXX/libdbicudt17.so (.dylib)
```

```
libXX/libdbjodbc17.so (.dylib)
libXX/libdblib17_r.so (.dylib)
libXX/libdbodbc17_r.so (.dylib)
libXX/libdbodm17.so (.dylib)
libXX/libjsyblib1700_r.so (.dylib)
res/dblg[LL]17.res
java/batik-all-1.9.jar
java/isql.jar
java/JComponents1700.jar
java/jodbc4.jar
java/jsyblib1700.jar
java/ngdbc.jar
java/saip17.jar
java/SCEditor1700.jar
java/xml-apis-ext.jar
java/xmlgraphics-commons-2.2.jar
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied. The 32-bit version of Interactive SQL requires the 32-bit version of the JRE files (`bin32/jre180`). The 64-bit version of Interactive SQL requires the 64-bit version of the JRE files (`bin64/jre180`).

SQL Central

SQL Central (scjview) requires the following files.

```
binXX/scjview
binXX/jre180/...
libXX/libjsyblib1700_r.so (.dylib)
java/jsyblib1700.jar
java/SCEditor1700.jar
java/sqlcentral1700.jar
```

Some file paths above end with "...". This indicates that you must copy the entire tree, including subdirectories. The 32-bit version of *SQL Central* requires the 32-bit version of the JRE files (`bin32/jre180`). The 64-bit version of *SQL Central* requires the 64-bit version of the JRE files (`bin64/jre180`).

SQL Central with SQL Anywhere plug-in

The SQL Anywhere plug-in of *SQL Central* requires the personal server (*dbeng17* and supporting files) and the following files.

```
libXX/libdbicu17_r.so (.dylib)
libXX/libdbicudt17.so (.dylib)
libXX/libdbjodbc17.so (.dylib)
libXX/libdblib17_r.so (.dylib)
libXX/libdbodbc17_r.so (.dylib)
libXX/libdbodm17.so (.dylib)
libXX/libdbput17_r.so (.dylib)
libXX/libdbtasks17_r.so (.dylib)
libXX/libdbtool17_r.so (.dylib)
res/dblg[LL]17.res
java/batik-all-1.9.jar
java/dbdiff.jar
java/debugger.jar
java/diffutils-1.2.1.jar
```

```
java/isql.jar
java/JComponents1700.jar
java/jodbc4.jar
java/ngdbc.jar
java/saip17.jar
java/salib.jar
java/saplugin.jar
java/SQLAnywhere.jpr
java/xml-apis-ext.jar
java/xmlgraphics-commons-2.2.jar
```

SQL Central with MobiLink Plug-in

The MobiLink plug-in of *SQL Central* requires the following files.

```
libXX/libdbicu17_r.so (.dylib)
libXX/libdbicudt17.so (.dylib)
libXX/libdblib17_r.so (.dylib)
libXX/libdbput17_r.so (.dylib)
libXX/libdbtasks17_r.so (.dylib)
libXX/libdbtool17_r.so (.dylib)
libXX/libmljodbc17.so (.dylib)
res/dblg[LL]17.res
java/commons-collections-4.2.2.jar
java/commons-lang-2.6.jar
java/commons-logging-1.2.jar
java/isql.jar
java/JComponents1700.jar
java/jodbc4.jar
java/mldesign.jar
java/mlplugin.jar
java/MobiLink.jpr
java/ngdbc.jar
java/saip17.jar
java/salib.jar
java/velocity-engine-core-2.0-dep.jar
java/commons-lang3-3.8.1.jar
java/slf4j-1.7.25.jar
```

SQL Central with UltraLite plug-in

The UltraLite plug-in of *SQL Central* requires the following files.

```
libXX/libdbicu17_r.so (.dylib)
libXX/libdbicudt17.so (.dylib)
libXX/libdblib17_r.so (.dylib)
libXX/libdbput17_r.so (.dylib)
libXX/libdbtasks17_r.so (.dylib)
libXX/libdbtool17_r.so (.dylib)
libXX/libmlcrsa17.so (.dylib)
libXX/libulscutil17.so (.dylib)
libXX/libulutils17.so (.dylib)
res/dblg[LL]17.res
java/batik-all-1.9.jar
java/isql.jar
java/JComponents1700.jar
java/jodbc4.jar
```

```
java/ngdbc.jar
java/saip17.jar
java/salib.jar
java/ulplugin.jar
java/UltraLite.jpr
java/xml-apis-ext.jar
java/xmlgraphics-commons-2.2.jar
```

International Message and Context-Sensitive Help Files

For Linux systems only, all displayed text and context-sensitive help for the administration tools have been translated from English into German, French, Japanese, and Simplified Chinese. The resources for each language are in separate files. The English files contain en in the file names. German file names contain de, French file names contain fr, Japanese file names contain ja, and Chinese files contain zh.

To install support for different languages, you must add the message files for those other languages. The translated files are as follows:

`dblgen17.res`	English
`dblgde17_iso_1.res`, `dblgde17_utf8.res`	German (Linux only)
`dblgja17_eucjis.res`, `dblgja17_sjis.res`, `dblgja17_utf8.res`	Japanese (Linux only)
`dblgzh17_cp936.res`, `dblgzh17_eucgb.res`, `dblgzh17_utf8.res`	Simplified Chinese (Linux only)

These files are included with localized versions of SQL Anywhere.

1.23.9.2.3 Step 3: Set Environment Variables

To run the administration tools, several environment variables must be defined or modified. This is usually done in the `sa_config.sh` file, which is created by the SQL Anywhere installer. To use the `sa_config.sh` file, just copy it and set SQLANY17 to point to the deployment location.

Otherwise, to set the environment up, you must do the following:

1. Set the following environment variable:

   ```
   SQLANY17=SQL-Anywhere-install-dir
   ```

2. Set the PATH to include one of the following:

   32-bit Linux

   ```
   $SQLANY17/bin32
   $SQLANY17/bin32/jre180/bin
   ```

   64-bit Linux

   ```
   $SQLANY17/bin64
   $SQLANY17/bin32
   ```

```
$SQLANY17/bin64/jre180/bin
```

64-bit Solaris

```
$SQLANY17/bin64
$SQLANY17/bin32
$SQLANY17/bin64/jre180/bin/sparcv9
```

3. Set LD_LIBRARY_PATH to include the following folders:

   32-bit Linux

   ```
   $SQLANY17/lib32
   $SQLANY17/lib64
   $SQLANY17/bin32/jre180/lib/i386/server
   $SQLANY17/bin32/jre180/lib/i386
   ```

   64-bit Linux and Solaris

   ```
   $SQLANY17/lib64
   $SQLANY17/lib32
   $SQLANY17/bin64/jre180/lib/amd64/server
   $SQLANY17/bin64/jre180/lib/amd64
   ```

On macOS, the administration tools make use of a shell script stub launcher called `sa_java_stub_launcher.sh`, generated at install time and placed inside the `Contents/MacOS` folder of each Java application bundle. As generated, the script sources the `System/bin64/sa_config.sh` file to set up the environment and then runs the JavaApplicationStub binary, which starts the actual Java application. For deployment purposes, `sa_java_stub_launcher.sh` can be modified as required to set up the environment. The name of the script can be changed by modifying the `Info.plist` file inside the Java application bundle and changing the string value of the key CFBundleExecutable.

1.23.9.2.4 Step 4: Create Connection Profiles for *SQL Central*

This step involves the configuration of *SQL Central*. If you are not installing *SQL Central*, you can skip it.

When *SQL Central* is installed on your system, a connection profile for SQL Anywhere 17 Demo is created in the repository file. If you do not want to create one or more connection profiles, then you can skip this step.

The following commands were used to create the SQL Anywhere 17 Demo connection profile. Use this as a model for creating your own connection profiles.

```
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Name" "SQL Anywhere 17
Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/FirstTimeStart" "false"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Description" "Suitable
Description"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/ProviderId"
"sqlanywhere1700"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Provider" "SQL Anywhere
17"
scjview -write "ConnectionProfiles/SQL
Anywhere 17 Demo/Data/ConnectionProfileSettings"
"DSN\eSQL^0020Anywhere^002017^0020Demo;UID\eDBA;PWD\e35c624d517fb"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Data/
ConnectionProfileName" "SQL Anywhere 17 Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 17 Demo/Data/
ConnectionProfileType" "SQL Anywhere"
```

The connection profile strings and values can be extracted from the repository file. Define a connection profile using *SQL Central* and then look at the repository file for the corresponding lines.

Here is a portion of the repository file that was created using the process described above. Some entries have been split across multiple lines for display purposes. In the file, each entry appears on a single line:

```
#  Version: 17.0.1154
#  Thu Oct 04 12:07:53 EDT 2012
#
ConnectionProfiles/SQL Anywhere 17 Demo/Name=SQL Anywhere 17 Demo
ConnectionProfiles/SQL Anywhere 17 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 17 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 17 Demo/ProviderId=sqlanywhere1700
ConnectionProfiles/SQL Anywhere 17 Demo/Provider=SQL Anywhere 17
ConnectionProfiles/SQL Anywhere 17 Demo/Data/ConnectionProfileSettings=
    DSN\eSQL^0020Anywhere^002017^0020Demo;
    UID\eDBA;
    PWD\e35c624d517fb
ConnectionProfiles/SQL Anywhere 17 Demo/Data/ConnectionProfileName=
    SQL Anywhere 17 Demo
ConnectionProfiles/SQL Anywhere 17 Demo/Data/ConnectionProfileType=
     SQL Anywhere
```

1.23.9.3 Administration Tools Configuration

You can specify which features are shown or enabled by the administration tools by using the initialization file named `OEM.ini`.

This file must be in the same directory as the JAR files used by the administration tools (for example, *C:\Program Files\SQL Anywhere 17*`\Java`). If the file is not found, default values are used. Also, defaults are used for values that are missing from `OEM.ini`.

The default location of the dbisql preferences file `%appdata%\SAP\DBISQL 17`. The 64-bit version of this file is named `.isqlPreferences17_64`.

> **Note**
>
> When you redeploy the administration tools, the tools cannot check for SQL Anywhere software updates. The *Check For Updates* menu items and options do not appear in redeployed versions.

Here is a sample `OEM.ini` file:

```
[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
disableExecuteAll=false
# lockedPreferences is assigned a comma-separated
# list of one or more of the following option names:
#    autoCommit
#    autoRefetch
#    commitOnExit
```

```
#    disableResultsEditing
#    executeToolbarButtonSemantics
#    maximumDisplayedRows
```

Any line beginning with the # character is a comment line and is ignored. The specified option names and values are case sensitive.

The `OEM.ini` file is divided into the following subsections:

[errors]

The option set in this subsection applies to all administration tools.

If reportErrors is false, the administration tool does not present a window to the user inviting them to submit error information to Technical Support if the software crashes. Instead, the standard window appears.

[updates]

The option set in this subsection applies to all administration tools.

If checkForUpdates is false, the administration tool does not check for SQL Anywhere software updates automatically, nor does it give the user the option to do so at their discretion.

[preferences]

The option set in this subsection applies to all administration tools.

Set the *directory* option to specify the directory used by the administration tools to save user-specific configuration files. These files contain information related to the administration tools' settings and history. For example, they may contain Interactive SQL statement history, recently opened files, or saved window positions.

You must specify a fully qualified directory name (such as `c:\work\prefs`) that does not end with a path separator (backslash on Microsoft Windows; forward slash on UNIX and Linux).

On Microsoft Windows, the default setting for the user preferences directory is `%appdata%\SAP`. This setting permits multiple users on a single system to each have their own directory for the recording of preferences. Overriding this setting in the `OEM.ini` file disables this capability.

[performancedata]

The option set in this subsection applies only to Interactive SQL and *SQL Central*.

showPerformanceDataUI

If showPerformanceDataUI is false, the administration tools do not provide the option for the user to enable or disable the automatic submission of performance data to the software development team.

[dbisql]

The options set in this subsection apply only to Interactive SQL.

allowPasswordsInFavorites

If allowPasswordsInFavorites is false, Interactive SQL removes the *Save The Connection Password* checkbox from the *Add To Favorites* window. The default setting is true, which means the checkbox is present.

disableExecuteAll

If disableExecuteAll is set to true, then the *SQL* > *Execute* menu item and the F5 accelerator key are disabled in Interactive SQL. If the *Execute* toolbar button is configured for *Execute*, then it is disabled also. Therefore, you might want to set the Execute toolbar button to *Execute Selection* in Interactive SQL, and then set the executeToolbarButtonSemantics option in the `OEM.ini` file to prevent users from changing the *Execute* toolbar button.

lockedPreferences

You can lock the settings of Interactive SQL options so that users cannot change them. The option names are case sensitive. The following is an example:

```
[dbisql]
lockedPreferences=autoCommit
```

You can prevent users from changing the following Interactive SQL option settings:

autoCommit

Prevents users from customizing the *Commit After Every Statement* option.

autoRefetch

Prevents users from customizing the *Automatically Refetch Results* option.

commitOnExit

Prevents users from customizing the *Commit On Exit Or Disconnect* option.

disableResultsEditing

Prevents users from customizing the *Disable Editing* option.

executeToolbarButtonSemantics

Prevents users from customizing the behavior of the *Execute* toolbar button.

maximumDisplayedRows

Prevents users from customizing the *Maximum Number Of Rows To Display* option.

Parent topic: Administration Tool Deployment [page 857]

Previous: Administration Tool Deployment on Linux, Solaris, and macOS [page 868]

Next: dbisqlc Deployment [page 878]

Related Information

Regularly Submit Performance Data
Managing the Favorites List
Executing SQL Statements (Interactive SQL)
Executing SQL Statements (Interactive SQL)
auto_commit Option [Interactive SQL]
auto_refetch Option [Interactive SQL]
commit_on_exit Option [Interactive SQL]
isql_maximum_displayed_rows Option [Interactive SQL]

1.23.9.4 dbisqlc Deployment

If your deployed application requires a query execution or scripting tool and is running on computers with limited resources, you could deploy the dbisqlc executable instead of Interactive SQL (dbisql).

However, dbisqlc is deprecated, and no new features are being added to it. Also, dbisqlc does not contain all the features of Interactive SQL and compatibility between the two is not guaranteed.

The dbisqlc executable requires the standard Embedded SQL client-side libraries.

Parent topic: Administration Tool Deployment [page 857]

Previous: Administration Tools Configuration [page 875]

Next: About the SQL AnywhereMonitor Production Edition [page 879]

Related Information

dbisqlc Utility (Deprecated)
Interactive SQL Utility (dbisql)

1.23.9.5 About the SQL AnywhereMonitor Production Edition

The Production Edition is intended for deployment and production use. It is installed separately, runs as a service, and includes a fully contained SQL Anywhere installation.

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit.

Upgrades and updates to SQL Anywhere do not overwrite or affect the Monitor Production Edition. In contrast, upgrades and updates can affect the Monitor Developer Edition as it uses the installed SQL Anywhere on the back end.

You should install the Monitor on a computer that is different from the computer where the resources are running. This has two advantages:

- The impact on the database server, the MobiLink server, or other applications is minimized.
- Monitoring is not affected if something happens to the computer where the resources are installed.

In this section:

Installing the Monitor Production Edition (Linux) [page 879]
Install the Monitor Production Edition on Linux.

Parent topic: Administration Tool Deployment [page 857]

Previous: dbisqlc Deployment [page 878]

1.23.9.5.1 Installing the Monitor Production Edition (Linux)

Install the Monitor Production Edition on Linux.

Prerequisites

You must be the root user.

Context

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit.

The Monitor runs as a service, and includes a fully contained SQL Anywhere installation.

Procedure

1. As the root user, run `setup.tar` from the `Monitor` directory on your installation media, and follow the instructions provided.

   On Linux, by default, the Monitor Production Edition automatically starts the Monitor service.
2. Open the default URL for logging in to the Monitor: `http://localhost:4950`.

   > **Note**
   >
   > If you are logging in remotely to the Monitor, browse to http://`computer-name`:4950, where `computer-name` is the name of the computer where the Monitor is running.
3. Log in.

   When prompted, enter your user name and password for the Monitor. The default user is a Monitor administrator with the name *admin* and the password *admin*.

Results

The Monitor Production Edition is installed.

Next Steps

You can start the Monitor.

1.23.10 Documentation Deployment

There are three options available for documentation.

You can use DocCommentXchange at http://dcx.sap.com.

Documentation is also available on the SAP Help Portal at https://help.sap.com/viewer/p/SAP_SQL_Anywhere.

For Windows, you can deploy the HTMLHelp (.chm) help files. HTML-based help documentation is installed to the *%SQLANY17%*`\Documentation` directory tree. HTML-based help documentation is not available for other operating systems.

1.23.11 Database Server Deployment

To run a database server, you must install the correct set of files.

You can deploy a database server by making the SQL Anywhere installer available to your end users. By selecting the proper option, each end user is guaranteed of getting the files they need.

The simplest way to deploy a personal database server or a network database server is to use the *Deployment Wizard*.

The files required by the database server are listed in the following table. All redistribution of these files is governed by the terms of your license agreement. You must confirm whether you have the right to redistribute the database server files before doing so.

Microsoft Windows	UNIX/Linux	macOS
dbeng17`.exe`	*dbeng17*	*dbeng17*
dbeng17`.lic`	*dbeng17*`.lic`	*dbeng17*`.lic`
dbsrv17`.exe`	*dbsrv17*	*dbsrv17*
dbsrv17`.lic`	*dbsrv17*`.lic`	*dbsrv17*`.lic`
`dbserv`*17*`.dll`	`libdbserv`*17*`_r.so`, `libdbtasks`*17*`_r.so`	`libdbserv`*17*`_r.dylib`, `libdbtasks`*17*`_r.dylib`
`dbscript`*17*`.dll`	`libdbscript`*17*`_r.so`	`libdbscript`*17*`_r.dylib`
`dblg[LL]`*17*`.dll`	`dblg[LL]`*17*`.res`	`dblgen`*17*`.res`
`dbghelp.dll`	N/A	N/A
`dbctrs`*17*`.dll`	N/A	N/A
`dbextf.dll`[1]	`libdbextf.so`[1]	`libdbextf.dylib`[1]
`dbicu`*17*`.dll`[2]	`libdbicu`*17*`_r.so`[2]	`libdbicu`*17*`_r.dylib`[2]
`dbicudt`*17*`.dll`[2]	`libdbicudt`*17*`.so`[2]	`libdbicudt`*17*`.dylib`[2]
`sqlany.cvf`	`sqlany.cvf`	`sqlany.cvf`
`dbrsa`*17*`.dll`	`libdbrsa`*17*`_r.so`	`libdbrsa`*17*`_r.dylib`
`dbrsakp`*17*`.dll`[3]	`libdbrsakp`*17*`_r.so`[3]	`libdbrsakp`*17*`_r.dylib`[3]
`dbodbc`*17*`.dll`[4]	`libdbodbc`*17*`.so`[4]	`libdbodbc`*17*`.dylib`[4]
`dbjodbc`*17*`.dll`[4]	`libdbjodbc`*17*`.so`[4]	`libdbjodbc`*17*`.dylib`[4]
N/A	`libdbodbc`*17*`_n.so`[4]	`libdbodbc`*17*`_n.dylib`[4]

Microsoft Windows	UNIX/Linux	macOS
N/A	`libdbodbc17_r.so`[4]	`libdbodbc17_r.dylib`[4]
`dbjdbc17.dll`[5]	`libdbjdbc17.so`[5]	`libdbjdbc17.dylib`[5]
`java\sajdbc4.jar`[5]	`java/sajdbc4.jar`[5]	`java/sajdbc4.jar`[5]
`java\sajvm.jar`[5]	`java/sajvm.jar`[5]	`java/sajvm.jar`[5]
`dbcis17.dll`[6]	`libdbcis17.so`[6]	`libdbcis17.dylib`[6]
`libsybbr.dll`[7]	`libsybbr.so`[7]	`libsybbr.dylib`[7]
`sqlacockpit.template`[8]	`sqlacockpit.template`[8]	`sqlacockpit.template`[8]
`dbsupport.exe`[9]	`dbsupport`[9]	`dbsupport`[9]
`sapcrypto.dll`[10]	`libsapcrypto.so`[10]	libsapcrypto.dylib[10]

[1] Required only if using the deprecated version of the system extended stored procedures and functions (xp_*). The `use_old_dbextf.sql` script must be used for databases that want to use the deprecated version.

[2] Required only if the database character set is multi-byte or if the UCA collation sequence is used.

[3] Required only for encrypted TDS connections.

[4] Required only if using Java in the database with version 10 databases.

[5] Required only if using Java in the database.

[6] Required only if using remote data access. Note that the Cockpit uses remote data access to retrieve information from tenant databases.

[7] Required only for archive backups.

[8] Required only if using the SQL Anywhere Cockpit

[9] Required only if you wish to automatically submit crash reports to SAP (run the following once on the target system: `dbsupport -cc autosubmit`).

[10]Required for SAP Crypto.

Notes

- Depending on your situation, choose whether to deploy the personal database server (dbeng17) or the network database server (dbsrv17).
- You must include the separate corresponding license file (*dbeng17*`.lic` or *dbsrv17*`.lic`) when deploying a database server. The license files are located in the same directory as the server executables.
- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **`en`**, **`de`**, **`ja`**, and so on).
- The Java VM jar file (`sajvm.jar`) is required only if the database server is to use the Java in the Database functionality.

- The table does not include files needed to run utilities such as dbbackup.
- The database server -xd option is useful in deployed applications since it prevents the database server from becoming the default database server. Connections are accepted only when the server name is included in the connection string.

In this section:

Improving Robustness on Intel Storage Drivers [page 883]
Improve the robustness across power failures on systems using certain Intel storage drivers by specifying the EnableFlush parameter in the registry. The absence of this parameter can result in lost data and corrupted databases in the event of a power failure.

Formatting Event Log Messages [page 884]
Ensure that messages written by the server to the Event Log on Windows are formatted correctly by creating registry keys.

Apple macOS Considerations [page 886]
On macOS, DBLauncher use the User Defaults system to locate the SQL Anywhere installation.

Database Deployment [page 886]
You deploy a database file by installing the database file onto your end-user's disk.

Related Information

The Deployment Wizard for Windows [page 813]
The Deployment Wizard for UNIX/Linux [page 822]
Administration Tool Deployment [page 857]
-xd Database Server Option
ServerName (Server) Connection Parameter

1.23.11.1 Improving Robustness on Intel Storage Drivers

Improve the robustness across power failures on systems using certain Intel storage drivers by specifying the EnableFlush parameter in the registry. The absence of this parameter can result in lost data and corrupted databases in the event of a power failure.

Procedure

1. Check that the following registry entry exists:

   ```
   HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\iastor\Parameters\
   ```

2. If the registry entry exists, add a REG_DWORD value named EnableFlush within the key, and assign a data value of 1.

3. Check that the following registry entry exists.

   ```
   HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\iastorv\Parameters\
   ```

4. If the registry entry exists, add a REG_DWORD value named EnableFlush within the key and assign a data value of 1.

Results

The EnableFlush parameter is specified, which prevents data loss in the event of a power failure.

Example

The following example illustrates a registry file that would specify the EnableFlush parameter in both possible locations:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iastor\Parameters]
"EnableFlush"=dword:00000001
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iastorv\Parameters]
"EnableFlush"=dword:00000001
```

1.23.11.2 Formatting Event Log Messages

Ensure that messages written by the server to the Event Log on Windows are formatted correctly by creating registry keys.

Prerequisites

These steps are required when the database server software is installed as part of another product, or is deployed to other computers by the licensee.

Procedure

1. Create a registry script file called `eventlog.reg` using a text editor.

   Add the following text to this file:

   ```
   REGEDIT4
   [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQL
   ANY 17.0]
   ```

```
"EventMessageFile"="<path>\\bin32\\dblgen17.dll"
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQL
ANY 17.0 Admin]
"EventMessageFile"="<path>\\bin32\\dblgen17.dll"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQL
ANY64 17.0]
"EventMessageFile"="<path>\\bin64\\dblgen17.dll"
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQL
ANY64 17.0 Admin]
"EventMessageFile"="<path>\\bin64\\dblgen17.dll"
```

The first two `SQLANY` entries are used by the 32-bit database server software. The last two `SQLANY64` entries are used by the 64-bit database server software.

The two `Admin` entries ensure that messages written by MESSAGE...TO EVENT LOG statements to the Event Log are formatted correctly.

2. Within this script, change `<path>` to the fully qualified installed location of the database server software, such as *C:\Program Files\SQL Anywhere 17*.

   Note that the path separator characters must be doubled (\\) in the file path.

   If you don't have the English language messages file, then you can substitute another language version (DE, FR, JA, etc.).

   If you have installed only the 32-bit version of the database software, then you may omit (delete) the last two `SQLANY64` entries from the script but retain the first two `SQLANY` entries.

   If you have installed only the 64-bit version of the database server software, then you may omit (delete) the first two `SQLANY` entries from the script but retain the last two `SQLANY64` entries.

   If you have installed both 32-bit and 64-bit versions of the database server software, then keep all four entries.

3. (Optional) If you want to suppress certain types of event log entries, then add the following lines to the end of the registry script file:

   ```
   [HKEY_LOCAL_MACHINE\SOFTWARE\SAP\SQL Anywhere\17.0]
   "EventLogMask"=dword:00000003
   ```

   The EventLogMask is an integer value representing a combination of bits. The following bits are used by the database server:

   ```
   EVENTLOG_ERROR_TYPE         0x0001
   EVENTLOG_WARNING_TYPE       0x0002
   EVENTLOG_INFORMATION_TYPE   0x0004
   ```

   If the EventLogMask key is set to zero, no messages appear at all. A better setting would be 1, so that informational and warning messages do not appear, but errors do. The default setting (no entry present) is for all message types to appear. The setting used in the example is `dword:00000003` (1 + 2), which means error and warning messages are logged, but information messages are not.

   The key can be placed in the HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE hive. The local machine setting affects all users of the computer.

4. Save your registry script file to disk and, at a command prompt, run the registry script.

   ```
   eventlog.reg
   ```

You will be prompted to allow the script to make changes to the registry.

Results

Event Log messages are formatted in the desired language, and are reported according to the bit mask value assigned to the EventLogMask registry key.

Example

The following example illustrates a sample registry file that formats Event Log messages recorded by the 32-bit version of the server:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQLANY
 17.0]
"EventMessageFile"="c:\\sa17\\bin32\\dblgen17.dll"
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\SQLANY
 17.0 Admin]
"EventMessageFile"="c:\\sa17\\bin32\\dblgen17.dll"
[HKEY_LOCAL_MACHINE\SOFTWARE\SAP\SQL Anywhere\17.0]
"EventLogMask"=dword:00000003
```

In this example, the English version of the DLL, presumed to be located in `c:\sa17\bin32\dblgen17.dll`, is used to format Event Log messages. The bit mask value of the EventLogMask indicates that only errors and warnings are recorded, not informational messages.

1.23.11.3 Apple macOS Considerations

On macOS, DBLauncher use the User Defaults system to locate the SQL Anywhere installation.

The SQL Anywhere installer writes the installation location to the user defaults repository of the user performing the installation. Other users are prompted for the installation directory when DBLauncher is first used. Also, the Preferences panel can be used to change this setting.

1.23.11.4 Database Deployment

You deploy a database file by installing the database file onto your end-user's disk.

As long as the database server shuts down cleanly, you do not need to deploy a transaction log file with your database file. When your end user starts running the database, a new transaction log is created.

For SQL Remote applications, the database should be created in a properly synchronized state, in which case no transaction log is needed. You can use the Extraction utility for this purpose.

In this section:

International Considerations [page 887]
When you are deploying a database worldwide, consider the locales in which the database is to be used. Different locales may have different sort orders or text comparison rules.

Database Employment on Read-only Media [page 888]
You can distribute and run databases on read-only media, such as a CD-ROM, as long as you run them in read-only mode.

Related Information

Remote Database Extraction

1.23.11.4.1 International Considerations

When you are deploying a database worldwide, consider the locales in which the database is to be used. Different locales may have different sort orders or text comparison rules.

For example, the database that you are going to deploy may have been created using the 1252LATIN1 collation and this may not be suitable for some of the environments in which it is to be used.

Since the collation of a database cannot be changed after it has been created, you might consider creating the database during the install phase and then populating the database with the schema and data that you want it to have. The database can be created during the install either by using the dbinit utility, or by starting the database server with the utility database and issuing a CREATE DATABASE statement. Then, you can use SQL statements to create the schema and do whatever else is necessary to set up your initial database.

If you decide to use the UCA collation, you can specify additional collation tailoring options for finer control over the sorting and comparing of characters using the dbinit utility or the CREATE DATABASE statement. These options take the form of `keyword=value` pairs, assembled in parentheses, following the collation name. Using the CREATE DATABASE statement, for example, a collation tailoring can be specified using syntax like the following:

```
CHAR COLLATION 'UCA( locale=es;case=respect;accent=respect )'
```

As an alternative, you could create multiple database templates, one for each of the locales in which the database is to be used. This may be suitable if the set of locales you are deploying the database to is relatively small. You can have the installer choose which database to install.

Related Information

Collation Considerations
CREATE DATABASE Statement
Initialization Utility (dbinit)

1.23.11.4.2 Database Employment on Read-only Media

You can distribute and run databases on read-only media, such as a CD-ROM, as long as you run them in read-only mode.

To run a database in read-only mode, use the -r database server option.

If you must make changes to the database, copy the database from the CD-ROM to a location where it can be modified, such as a hard drive.

Related Information

-r Database Server Option

1.23.12 DLL Registration on Windows

Some DLL files require registration if deployed for use with SQL Anywhere.

For Windows 7 and later, you must include the elevated operations agent (`dbelevate17.exe`) which supports the privilege elevation required when DLLs are registered or unregistered.

There are many ways you can register these DLLs, including the use of an install script or the regsvr32 utility.

The following table lists the DLLs that require registration if deployed on Windows:

File	Description
`dbctrs17.dll`	The Windows Performance Monitor counters.
`dbodbc17.dll`	The SQL Anywhere ODBC driver.
`dboledb17.dll`	The SQL Anywhere OLE DB provider.
`dboledba17.dll`	The SQL Anywhere OLE DB provider schema assist module (Windows only).

Example

Run the following command to register the ODBC driver using the regsvr32 utility on Windows:

```
regsvr32 dbodbc17.dll
```

1.23.13 External Environment Support Deployment

A number of components are required to support external calls in a database application.

The following tables summarize the components that must be deployed.

ESQL/Microsoft ODBC External Calls

Component	Microsoft Windows	Linux / Unix	macOS
ESQL and Microsoft ODBC launcher	`dbexternc`*17*`.exe`	`dbexternc`*17*	`dbexternc`*17*
Bridge	`dbextenv`*17*`.dll`	`libdbextenv`*17*`_r.so`	`libdbextenv`*17*`_r.dylib`
SQL Anywhere C API runtime	`dbcapi.dll`	`libdbcapi_r.so`	`libdbcapi_r.dylib`
DBLIB	`dblib`*17*`.dll`	`libdblib`*17*`_r.so`	`libdblib`*17*`_r.dylib`
Thread support library	N/A	`libdbtasks`*17*`_r.so`	`libdbtasks`*17*`_r.dylib`

For additional files required by Embedded SQL applications, see the documentation on Embedded SQL client deployment.

For additional files required by Microsoft ODBC applications, see the documentation on Microsoft ODBC client deployment.

Java External Calls

Component	Windows	Linux / Unix	macOS
Java installation (third-party)	`java.exe`	`java`	`java`
Launcher	`sajvm.jar`	`sajvm.jar`	`sajvm.jar`
SQL Anywhere JDBC driver (server-side calls)	`dbjdbc`*17*`.dll`	`libdbjdbc`*17*`.so`	`libdbjdbc`*17*`.dylib`

For additional files required by JDBC applications, see the documentation on JDBC client deployment.

JavaScript External Calls (Deprecated)

Component	Windows	Linux/UNIX	macOS
Node.js (third-party)	`node.exe`	`node`	`node`

Component	Windows	Linux/UNIX	macOS
JavaScript support folder	%SQLANY17%\Node\sqlanywhere_jsextenv	$SQLANY17/node/sqlanywhere_jsextenv	$SQLANY17/node/sqlanywhere_jsextenv
Bridge (matching bitness of Node.js)	dbextenv*17*.dll	libdbextenv*17*_r.so	libdbextenv*17*_r.dylib
DBLIB (matching bitness of Node.js)	dblib*17*.dll	libdblib*17*_r.so	libdblib*17*_r.dylib
Thread support library	N/A	libdbtasks*17*_r.so	libdbtasks*17*_r.dylib

For additional files required by JDBC applications, see the documentation on JDBC client deployment.

.NET CLR External Calls

Component	Microsoft Windows	Linux/UNIX	macOS
.NET 3.5 or later	(from Microsoft)	N/A	N/A
SQL Anywhere .NET 3.5 or 4.5 Data Provider	(from SAP)	N/A	N/A
.NET CLR bridge	dbextclr*17*.exe or dbextclr*17*_v4.5.exe	N/A	N/A
Bridge	dbextenv*17*.dll	N/A	N/A
.NET CLR support	dbclrenv*17*.dll	N/A	N/A
DBLIB	dblib*17*.dll	N/A	N/A
Thread support library	N/A	libdbtasks*17*_r.so	libdbtasks*17*_r.dylib

Perl External Calls

Component	Microsoft Windows	Linux/UNIX	macOS
Perl installation (third-party)	perl.exe	perl	perl
Perl launcher	perlenv.pl	perlenv.pl	perlenv.pl
Bridge	dbextenv*17*.dll	libdbextenv*17*_r.so	libdbextenv*17*_r.dylib
SQL Anywhere C API runtime	dbcapi.dll	libdbcapi_r.so	libdbcapi_r.dylib
DBLIB	dblib*17*.dll	libdblib*17*_r.so	libdblib*17*_r.dylib
Thread support library	N/A	libdbtasks*17*_r.so	libdbtasks*17*_r.dylib

For additional files required by Perl applications, see the documentation on DBD::SQLAnywhere.

PHP External Calls (Deprecated)

Component	Microsoft Windows	Linux	macOS
PHP installation (third-party)	`php.exe`	`php`	`php`
PHP launcher	`phpenv.php`	`phpenv.php`	`phpenv.php`
Bridge	`dbextenv`*`17`*`.dll`	`libdbextenv`*`17`*`_r.so`	`libdbextenv`*`17`*`_r.dylib`
PHP external environment module	`php-x.y.0_sqlanywhere_extenv`*`17`*`.dll`	`php-x.y.0_sqlanywhere_extenv`*`17`*`_r.so`	Build from source code
DBLIB	`dblib`*`17`*`.dll`	`libdblib`*`17`*`_r.so`	`libdblib`*`17`*`_r.dylib`
Thread support library	N/A	`libdbtasks`*`17`*`_r.so`	`libdbtasks`*`17`*`_r.dylib`

Download prebuilt versions of the PHP extension and the external environment module from The SAP SQL Anywhere PHP Module

There are other files required for using the SQL Anywhere PHP extension. See the documentation on the SQL Anywhere PHP extension.

Related Information

Embedded SQL Client Deployment [page 845]
ODBC Client Deployment [page 836]
JDBC Client Deployment [page 847]
DBD::SQLAnywhere [page 448]
SQL Anywhere PHP Extension (Deprecated) [page 469]
The ESQL and ODBC External Environments [page 419]
The Java External Environment [page 427]
The JavaScript External Environment (Deprecated) [page 432]
The CLR External Environment [page 415]
The Perl External Environment [page 438]
The PHP External Environment (Deprecated) [page 442]

1.23.14 Encryption Deployment

SQL Anywhere supports AES encryption of databases and database tables and cryptographic functions, as well as RSA encryption for secure client login using a password and network traffic (TLS, HTTPS) between the client and the database server.

The following table summarizes the components that support encryption:

Encryption Option	Encryption Type	Included in Module	Licensable
Database encryption	AES	Microsoft Windows: • `dbrsa17.dll` • `sapcrypto.dll` UNIX/Linux: • `libdbrsa17.so` • `libdbrsa17_r.so` • `libsapcrypto.so` macOS: • `libdbrsa17.dylib` • `libdbrsa17_r.dylib` • `libsapcrypto.dylib`	included[1]
Database encryption	FIPS-certified AES	Microsoft Windows: • `dbfips17.dll` • `sapcryptofips.dll` • `slcryptokernel.dll` • `slcryptokernel.dll.sha256` • `msvcr90.dll` (the 64-bit version of this file is called `msvcr100.dll`) UNIX/Linux • `libdbfips17_r.so` • libsapcryptofips.so • libslcryptokernel.so • libslcryptoker-nel.so.sha256	separately licensed[2]

Encryption Option	Encryption Type	Included in Module	Licensable
Transport layer security (TLS), HTTPS, secure login, cryptographic functions	RSA	Microsoft Windows: • `dbrsa17.dll` • `sapcrypto.dll` UNIX/Linux: • `libdbrsa17.so` • `libdbrsa17_r.so` • `libsapcrypto.so` macOS: • `libdbrsa17.dylib` • `libdbrsa17_r.dylib` • `libsapcrypto.dylib`	included[1]
Transport layer security (TLS), HTTPS, secure login, cryptographic functions	FIPS-certified RSA	Microsoft Windows: • `dbfips17.dll` • `sapcryptofip.dll` • `slcryptokernel.dll` • `slcryptokernel.dll.sha256` • `msvcr90.dll` (the 64-bit version of this file is called `msvcr100.dll`) UNIX/Linux: • `libdbfips17_r.so` • libsapcryptofips.so • libslcryptokernel.so • libslcryptoker-nel.so.sha256	separately licensed[2]

[1] AES and RSA encryption are included with the software, and do not require a separate license, but these libraries are not FIPS-certified.

[2] FIPS-certified encryption requires a separate license. FIPS-certified AES and RSA encryption libraries are installed when your software license key includes access to FIPS-certified encryption technology.

1.23.15 LDAP Deployment

SQL Anywhere supports deployment of Lightweight Directory Access Protocol (LDAP) user authentication, allowing client applications to send user ID and password information to the database server for authentication by an LDAP server.

SQL Anywhere also supports server lookup for client connections using the LDAP protocol option. If you use this feature then include one or more of the files described in the table below.

The following table summarizes driver files required to deploy LDAP user authentication:

Operating System	32-bit	64-bit
Microsoft Windows	`dbldap`*17*`.dll`	`dbldap`*17*`.dll`
Microsoft Windows with FIPS support	`dbldapfips`*17*`.dll`	`dbldapfips`*17*`.dll`
UNIX/Linux (threaded)	`libdbldap`*17*`_r.so` (Linux only)	`libdbldap`*17*`_r.so`
UNIX/Linux (threaded with FIPS support)	`libdbldapfips`*17*`_r.so` (Linux only)	`libdbldapfips`*17*`_r.so`
UNIX/Linux (non-threaded)	`libdbldap`*17*`.so` (Linux only)	`libdbldap`*17*`.so`

LDAP user authentication is not supported on macOS.

1.23.16 Embedded Database Application Deployment

An embedded database application is one in which the application and the database both reside on the same computer.

The following are the components of an embedded database application:

Client application

This includes the SQL Anywhere client components.

Database server

The SQL Anywhere personal database server.

Database file

You must deploy a database file holding the data the application uses.

SQL Remote

If your application uses SQL Remote replication, you must deploy the SQL Remote Message Agent.

In this section:

Personal Server Deployment [page 895]
When you deploy an application that uses the personal server, you must deploy both the client application components and the database server components.

Database Utility Deployment [page 895]
If you deploy database utilities (such as dbbackup), some additional files are required.

Unload Support Deployment for Version 9 and Earlier Databases [page 896]
Your application might require the ability to convert version 9.0 or earlier databases to the current format.

SQL Remote Deployment [page 898]
Your application might require the SQL Remote Message Agent.

Related Information

Database Server Deployment [page 881]
Requirements for Deploying Client Applications [page 826]

1.23.16.1 Personal Server Deployment

When you deploy an application that uses the personal server, you must deploy both the client application components and the database server components.

The language resource library (`dblgen17.dll`) is shared between the client and the server. You need only one copy of this file.

Follow the SQL Anywhere installation behavior, and install the client and server files in the same directory.

1.23.16.2 Database Utility Deployment

If you deploy database utilities (such as dbbackup), some additional files are required.

If you deploy database utilities along with your application, then include the following support files:

Description	Windows	Linux / Unix	macOS
Database tools library	`dbtool17.dll`	`libdbtool17_r.so`, `libdbtasks17_r.so`	`libdbtool17_r.dylib`, `libdbtasks17_r.dylib`
Interface Library	`dblib17.dll`	`libdblib17_r.so`	`libdblib17_r.dylib`
Language resource library	`dblg[LL]17.dll`	`dblg[LL]17.res`	`dblgen17.res`
Connect window	`dbcon17.dll`		
Version 9 or earlier physical store library	`dboftsp.dll`	`libdboftsp_r.so`	N/A

Notes

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different

languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **ja**, and so on).

- For non-multithreaded applications on Linux and Unix, you can use `libdbtasks17.so` and `libdblib17.so`.
- For non-multithreaded applications on macOS, you can use `libdbtasks17.dylib` and `libdblib17.dylib`.
- For Windows, the *ODBC Configuration for SQL Anywhere* and *Connect to SQL Anywhere* window support code (`dbcon17.dll`) is required if your end users will create their own data sources, if they will enter user IDs and passwords when connecting to the database, or if they require the Connect window for any other purpose.
- The version 9 or earlier physical store library is required by some utilities (dblog, dbtran, dberase) to access log files created by versions of the software earlier than 10.0.0. If you are not deploying these utilities, then you do not require this library.
- If you require support for unloading version 9.0 or earlier databases, then you need the database unload utility, dbunload, as well as some other components. These components are documented elsewhere.
- The personal database server (dbeng17) is required for creating databases using the *dbinit* utility. It is also required if you are creating databases from *SQL Central* on the local computer when no other database servers are running.

Related Information

Database Server Deployment [page 881]
Unload Support Deployment for Version 9 and Earlier Databases [page 896]

1.23.16.3 Unload Support Deployment for Version 9 and Earlier Databases

Your application might require the ability to convert version 9.0 or earlier databases to the current format.

If your application requires the conversion of version 9.0 or earlier databases, then you must include the database unload utility, dbunload, together with the following additional files:

Description	Windows	Linux / Unix	macOS
Unload support for pre-10.0 databases	`dbunlspt.exe`	`dbunlspt`	`dbunlspt`
Message resource library	`dbus[LL].dll`	`dbus[LL].res`	`dbus[LL].res`
Unload script file	`optdeflt.sql`	`optdeflt.sql`	`optdeflt.sql`
Unload script file	`opttemp.sql`	`opttemp.sql`	`opttemp.sql`
Unload script file	`unloadold.sql`	`unloadold.sql`	`unloadold.sql`

The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to add the resource files for these languages. The message files are as follows.

Windows Message Files

`dbusde.dll`	German
`dbusen.dll`	English
`dbuses.dll`	Spanish
`dbusfr.dll`	French
`dbusit.dll`	Italian
`dbusja.dll`	Japanese
`dbusko.dll`	Korean
`dbuslt.dll`	Lithuanian
`dbuspl.dll`	Polish
`dbuspt.dll`	Portuguese
`dbusru.dll`	Russian
`dbustw.dll`	Traditional Chinese
`dbusuk.dll`	Ukrainian
`dbuszh.dll`	Simplified Chinese

Linux Message Files

`dbusde_iso_1.res`, `dbusde_utf8.res`, `dbusen.res`	German
`dbusen.res`	English
`dbusja_eucjis.res`, `dbusja_sjis.res`, `dbusja_utf8.res`	Japanese
`dbuszh_cp936.res`, `dbuszh_eucgb.res`, `dbuszh_utf8.res`	Chinese

These files are included with localized versions of SQL Anywhere.

In addition to these files, you also need the files required for database utility deployment.

Related Information

Database Utility Deployment [page 895]

1.23.16.4 SQL Remote Deployment

Your application might require the SQL Remote Message Agent.

If you are deploying the SQL Remote Message Agent, you must include the following files:

Description	Windows	Linux / Solaris	macOS
Message Agent	`dbremote.exe`	`dbremote`	`dbremote`
Encoding/decoding library	`dbencod`*`17`*`.dll`	`libdbencod`*`17`*`_r.so.1`	`libdbencod`*`17`*`_r.dylib`
FILE message link library [1]	`dbfile`*`17`*`.dll`	`libdbfile`*`17`*`_r.so.1`	`libdbfile`*`17`*`_r.dylib`
FTP message link library [1]	`dbftp`*`17`*`.dll`	`libdbftp`*`17`*`_r.so.1`	`libdbftp`*`17`*`_r.dylib`
HTTP message link library [1]	`dbhttp`*`17`*`.dll`	`libdbhttp`*`17`*`_r.so.1`	`libdbhttp`*`17`*`_r.dylib`
Language resource library	`dblg[LL]`*`17`*`.dll`	`dblg[LL]`*`17`*`.res`	`dblg[LL]`*`17`*`.res`
Interface Library	`dblib`*`17`*`.dll`	`libdblib`*`17`*`_r.so.1`	`libdblib`*`17`*`_r.dylib`
SMTP message link library [1]	`dbsmtp`*`17`*`.dll`	`libdbsmtp`*`17`*`_r.so.1`	`libdbsmtp`*`17`*`_r.dylib`
Database tools library	`dbtool`*`17`*`.dll`	`libdbtool`*`17`*`_r.so.1`	`libdbtool`*`17`*`_r.dylib`
Thread support library	N/A	`libdbtasks`*`17`*`_r.so.1`	`libdbtasks`*`17`*`_r.dylib`

[1] Only deploy the library for the message link you are using.

A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. To install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **`en`**, **`de`**, **`ja`**, and so on).

Follow the SQL Anywhere installation behavior, and install the SQL Remote files in the same directory as the SQL Anywhere files.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2025 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.