CHAPTER 28

Accessing Remote Data

About this chapter

Adaptive Server Anywhere can access data located on different servers, both Sybase and non-Sybase, as if the data were stored on the local server.

This chapter describes how to configure Adaptive Server Anywhere to access remote data.

Contents

Introduction

Using Adaptive Server Anywhere you can:

- ♦ Access data in relational databases such as Sybase, Oracle, and DB2.
- ♦ Access desktop data such as Excel spreadsheets, MS-Access databases, FoxPro, and text files.
- ♦ Access any other data source that supports an ODBC interface.
- ♦ Perform joins between local and remote data.
- ♦ Perform joins between tables in separate Adaptive Server Anywhere databases.
- ♦ Use Adaptive Server Anywhere features on data sources that would normally not have that ability. For instance, you could use a Java function against data stored in Oracle, or perform a subquery on spreadsheets. Adaptive Server Anywhere will compensate for features not supported by a remote data source by operating on the data after it is retrieved.
- ♦ Use Adaptive Server Anywhere to move data from one location to another using insert-select.
- ♦ Access remote servers directly using passthrough mode.
- ♦ Execute remote procedure calls to other servers.

Adaptive Server Anywhere allows access to the following external data sources:

- ♦ Adaptive Server Anywhere
- ♦ Adaptive Server Enterprise
- ♦ Oracle
- ♦ IBM DB2
- ♦ Microsoft SQL Server
- ♦ Other ODBC data sources

Platform availability
The remote data access features are supported on the Windows 95 and Windows NT platforms only.

Accessing remote data from PowerBuilder DataWindows

You can access remote data can be accessed from a PowerBuilder DataWindow by setting the DBParm Block parameter to 1 on connect.

- In the design environment, you can set the Block parameter by accessing the Transaction tab in the Database Profile Setup dialog and setting the Retrieve Blocking Factor to 1.
- In a connection string, use the following phrase:

  ```
  DBParm="Block=1"
  ```

Basic concepts

This section describes the basic concepts required to access remote data.

Remote table mappings

Adaptive Server Anywhere presents tables to a client application as if all the data in the tables were stored in the database to which the application is connected. Internally, when a query involving remote tables is executed, the storage location is determined, and the remote location is accessed so that data can be retrieved.

To have remote tables appear as local tables to the client, you create local **proxy tables** that map to the remote data.

❖ **To create a proxy table:**

1 Define the server where the remote data is located. This specifies the type of server and location of the remote server.

   For more information, see "Working with remote servers" on page 736.

2 Map the local user login information to the remote server user login information if the logins on the two servers are different.

   For more information, see "Working with external logins" on page 741.

3 Create the proxy table definition. This specifies the mapping of a local proxy table to the remote table. This includes the server where the remote table is located, and the database name, owner name, table name, and column names of the remote table.

   For more information, see "Working with proxy tables" on page 743.

Administering remote table mappings

To manage remote table mappings and remote server definitions, you can either use Sybase Central (Java Edition) or you can use a tool such as Interactive SQL and execute the SQL statements directly.

Server classes

A **server class** is assigned to each remote server. The server class specifies the access method used to interact with the server. Different types of remote servers require different access methods. The server classes provide Adaptive Server Anywhere detailed server capability information. Adaptive Server Anywhere adjusts its interaction with the remote server based on those capabilities.

There are currently two groups of server classes. The first is JDBC-based; the second is ODBC-based.

The JDBC-based server classes are:

♦ **asajdbc** for Adaptive Server Anywhere (version 6 and later)

♦ **asejdbc** for Adaptive Server Enterprise and SQL Server (version 10 and later)

The ODBC-based server classes are:

♦ **asaodbc** for Adaptive Server Anywhere (version 5.5 and later)

♦ **aseodbc** for Adaptive Server Enterprise and SQL Server (version 10 and later)

♦ **db2odbc** for IBM DB2

♦ **mssodbc** for Microsoft SQL Server

♦ **oraodbc** for Oracle servers (version 8.0 and later)

♦ **odbc** for all other ODBC data sources

For a full description of remote server classes, see "Server Classes for Remote Data Access" on page 761.

Working with remote servers

Before you can map remote objects to a local proxy table, you must define the remote server where the remote object is located. When you define a remote server, an entry is added to the **sysservers** table for the remote server. This section describes how to create, alter, and delete a remote server definition.

Creating remote servers

Use the CREATE SERVER statement to set up remote server definitions. You can execute the statements directly, or use Sybase Central (Java Edition).

For ODBC connections, each remote server corresponds to an ODBC data source. For some systems, including Adaptive Server Anywhere, each data source describes a database, so a separate remote server definition is needed for each database.

Example 1

The following statement creates an entry in the **sysservers** table for the Adaptive Server Enterprise named **ASEserver**:

```
CREATE SERVER ASEserver
CLASS 'asejdbc'
USING 'rimu:6666'
```

where:

- **ASEserver** is the name of the remote server
- **asejdbc** specifies the server is an Adaptive Server Enterprise and the connection to it is JDBC-based
- **rimu:6666** is the machine name and the TCP/IP port number where the remote server is located

Example 2

The following statement creates an entry in the sysservers table for the ODBC-based Adaptive Server Anywhere named testasa:

```
CREATE SERVER testasa
CLASS 'asaodbc'
USING 'test4'
```

where:

- **testasa** is the name by which the remote server is known within this database.
- **asaodbc** specifies that the server is an Adaptive Server Anywhere and the connection to it uses ODBC.

- **test4** is the ODBC data source name.

For a full description of the CREATE SERVER statement, see "CREATE SERVER statement" on page 413 of the book *Adaptive Server Anywhere Reference Manual*.

Creating remote servers using Sybase Central

To create a remote server using Sybase Central (Java):

1. Connect to the host database from Sybase Central (Java) using a JDBC connection.
2. In the left panel, open the Remote Server folder. Then double-click Add Remote Server on the right panel.
3. On the first page of the Wizard, enter a name to use for the remote server. This is the name you use to refer to the remote server will be from within the local database, and need not correspond to the server name the server supplies. Then click Next.
4. On the next page, select an appropriate class for the server and click Next.
5. Select a data access method (JDBC or ODBC) and supply connection information:
   - For JDBC, supply a URL in the form *machine-name:port-number*
   - For ODBC, supply a data source name.
6. Click Finish to create the remote server definition.

Notes

- The data access method (JDBC or ODBC) is the method used by Adaptive Server Anywhere to access the remote database. This is different from that used by Sybase Central to connect to your database.

Deleting remote servers

Use the DROP SERVER statement to drop a remote server from the Adaptive Server Anywhere system tables. All remote tables defined on that server must already be dropped for this statement to succeed. You can execute the statements directly, or use Sybase Central (Java Edition).

Example

The following statement drops the server named testasa:

```
DROP SERVER testasa
```

For a full description of the DROP SERVER statement, see "DROP SERVER statement" on page 457 of the book *Adaptive Server Anywhere Reference Manual*.

Deleting remote servers using Sybase Central

❖ **To delete a remote server using Sybase Central (Java):**

1 Connect to the host database from Sybase Central (Java) using a JDBC connection.

2 In the left panel, open the Remote Server folder. Then right-click the remote server on the right panel.

3 Select Delete from the popup menu.

Altering remote servers

Use the ALTER SERVER statement to modify the attributes of a server. These changes do not take effect until the next connection to the remote server. You can execute the statements directly, or use Sybase Central (Java Edition).

Example

The following statement changes the server class of the server named ASEserver to aseodbc:

```
ALTER SERVER ASEserver
CLASS 'aseodbc'
```

The Data Source Name for the server is ASEserver.

The ALTER SERVER statement can also be used to enable or disable a server's known capabilities.

For a complete description of the ALTER SERVER statement, see "ALTER SERVER statement" on page 349 of the book *Adaptive Server Anywhere Reference Manual*.

Altering remote servers using Sybase Central

❖ **To alter the properties of a remote server using Sybase Central (Java):**

1 Connect to the host database from Sybase Central (Java) using a JDBC connection.

2 In the left panel, open the Remote Server folder. Then right-click the remote server on the right panel.

3 Select Properties from the popup menu and make the changes you need in the server property sheet.

Listing the remote tables on a server

It may be helpful when you are configuring your Adaptive Server Anywhere to get a list of the remote tables available on a particular server. The **sp_remote_tables** procedure returns a list of the tables on a server.

```
sp_remote_tables servername
                 [,tablename]
                 [, owner ]
                 [, database]
```

If *tablename*, *owner*, or *database* is given, the list of tables is limited to only those that match.

For example, to get a list of all of the Microsoft Excel worksheets available from an ODBC data source named **excel**:

```
sp_remote_tables  excel
```

Or to get a list of all of the tables in the **production** database in an ASE named **asetest**, owned by 'fred':

```
sp_remote_tables  asetest, null, fred, production
```

For more information, see "sp_remote_tables system procedure" on page 758 of the book *Adaptive Server Anywhere Reference Manual*.

Listing remote server capabilities

The **sp_servercaps** procedure displays information about a remote server's capabilities. Adaptive Server Anywhere uses this capability information to determine how much of a SQL statement can be passed of to a remote server.

The system tables which contain server capabilities are not populated until after Adaptive Server Anywhere first connects to the remote server. This information comes from the SYSCAPABILITY and SYSCAPABILITYNAME system tables. The servername specifed must be the same servername used in the CREATE SERVER statement.

Issue the stored sp_servercaps as follows:

```
sp_servercaps  servername
```

For more information, see "sp_servercaps system procedure" on page 759 of the book *Adaptive Server Anywhere Reference Manual*.

Working with external logins

By default, Adaptive Server Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by creating external logins. External logins are alternate login names and passwords to be used when communicating with a remote server.

Creating external logins

The following statement allows the local user **fred** to gain access to the server **ASEserver**, using the remote login **frederick** with password **banana**.

```
CREATE EXTERNLOGIN fred
TO ASEserver
REMOTE LOGIN frederick
IDENTIFIED BY banana
```

For more information, see "CREATE EXTERNLOGIN statement" on page 395 of the book *Adaptive Server Anywhere Reference Manual*.

Creating external logins from Sybase Central

❖ **To create an external login using Sybase Central (Java):**

1 Connect to the host database from Sybase Central (Java) using a JDBC connection.

2 In the left panel, open the Remote Server folder. Then right-click the remote server on the right panel.

3 Select Properties from the popup menu. Change to the External Logins tab and make the changes you need in the property sheet.

4 Click OK to save the changes.

Dropping external logins

Use the DROP EXTERNLOGIN statement to remove external logins from the Adaptive Server Anywhere system tables.

Example

The following statement drops the external login for the local user fred created in the example above:

```
DROP EXTERNLOGIN fred TO ASEserver
```

For more information, see "DROP EXTERNLOGIN statement" on page 455 of the book *Adaptive Server Anywhere Reference Manual*.

Working with proxy tables

Location transparency of remote data is enabled by creating a local proxy table that maps to the remote object. To create a proxy table you use one of the following statements:

- If the table already exists at the remote storage location, use the CREATE EXISTING TABLE statement. This statement defines the proxy table for an existing table on the remote server.
- If the table does not exist at the remote storage location, use the CREATE TABLE statement. This statement creates a new table on the remote server, and also defines the proxy table for that table.

Specifying proxy table locations

The AT keyword is used with both CREATE TABLE and CREATE EXISTING TABLE to define the location of an existing object. This location string has 4 components that are separated by either a period or a semicolon. Semicolons allow filenames and extensions to be used in the database and owner fields.

```
... AT 'server.database.owner.tablename'
```

Server

This is the name by which the server is known in the current database, as specified in the CREATE SERVER statement. This field is mandatory for all remote data sources.

Database

The meaning of the database field depends on the data source. In some cases this field does not apply and should be left empty. The periods are still required, however.

- **Adaptive Server Enterprise** Specifies the database where the table exists. For example **master** or **pubs2**.
- **Adaptive Server Anywhere** This field does not apply; leave it empty.

  The database name for an Adaptive Server Anywhere ODBC data source should be specified when the data source name is defined in the ODBC Administrator.

  For jConnect-based connections, the database should be specified in the USING clause of the CREATE SERVER statement.

  For both ODBC and JDBC based connections to Adaptive Server Anywhere, you need a separate CREATE SERVER statement for each Adaptive Server Anywhere database being accessed.

- **Excel, Lotus Notes, Access** For these file-based data sources, the database name is the name of the file containing the table. Since file names can contain a period, a semicolon should be used as the delimiter between server, database, owner, and table.

Owner

If the database supports the concept of ownership, this field represents the owner name. This field is only required when several owners have tables with the same name.

Tablename

Tablename specifies the name of the table. In the case of an Excel spreadsheet, this is the name of the "sheet" in the workbook. If the table name is left empty, the remote table name is assumed to be the same as the local proxy table name.

Examples:

The following examples illustrate the use of location strings:

- Adaptive Server Anywhere:

  ```
  'testasa..dba.employee'
  ```

- Adaptive Server Enterprise:

  ```
  'ASEServer.pubs2.dbo.publishers'
  ```

- Excel:

  ```
  'excel;d:\pcdb\quarter3.xls;;sheet1$'
  ```

- Access:

  ```
  'access;\\server1\production\inventory.mdb;;parts'
  ```

Creating proxy tables using Sybase Central

You must use the Java version of Sybase Central to create remote tables, unless you choose to construct the SQL statements yourself. You cannot use the Windows version of Sybase Central.

❖ To create a proxy table using Sybase Central (Java):

1. Connect to the host database from Sybase Central (Java) using a JDBC connection.
2. In the left panel, open the Remote Server folder. Then open the server for which you wish to create a proxy table.
3. Double-click Add Proxy Table, and follow the instructions in the wizard:
   - On the first page, enter the remote database name. For Adaptive Server Anywhere, leave this blank.

For more information on identifying proxy tables, see "Specifying proxy table locations" on page 743.

- On the next page, enter the name and owner of the remote table, as they are specified in the remote database. Also, add a name for the proxy table. This can be different from the remote table name if you wish.

4 Click Finish to create the proxy table. You may have to refresh the display to show the proxy table.

Notes

- The proxy table is displayed under the remote server, inside the remote servers folder.
- The proxy table also appears in database tables folder. It is distinguished from other tables by a letter P on the icon.
- You can display the column properties for the proxy table by double clicking the table.

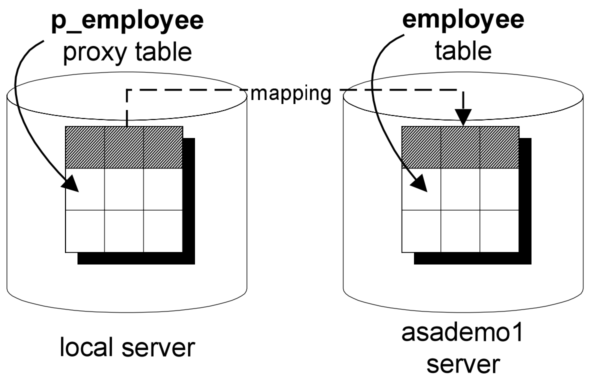
Using the CREATE EXISTING TABLE statement

The CREATE EXISTING TABLE statement creates a proxy table that maps to an existing table on the remote server. Adaptive Server Anywhere derives the column attributes and index information from the object at the remote location.

Example 1

To create a proxy table named **p_employee** on the current server to a remote table named **employee** on the server named **asademo1**, use the following syntax:

```
CREATE EXISTING TABLE p_employee
AT 'asademo1..dba.employee'
```



Example 2

The following statement maps the proxy table a1 to the Microsoft Access file mydbfile.mdb. In this example, the AT keyword uses the semicolon (;) as a delimiter. The server defined for Microsoft Access is named **access**.

```
CREATE EXISTING TABLE a1
AT'access;d:\mydbfile.mdb;;a1'
```

For a full description of the CREATE EXISTING TABLE statement, see "CREATE EXISTING TABLE statement" on page 393 of the book *Adaptive Server Anywhere Reference Manual*.

Using the CREATE TABLE statement

The CREATE TABLE statement creates a new table on the remote server, and defines the proxy table for that table when you use the AT option. You enter the CREATE TABLE statement using Adaptive Server Anywhere data types. Adaptive Server Anywhere automatically converts the data into the remote server's native types.

Example

The following statement creates a table named **members** on the remote server **asademo1**, and creates a proxy table named **employee** that maps to the remote location:

```
CREATE TABLE members
( membership_id INTEGER NOT NULL,
member_name CHAR(30) NOT NULL,
office_held CHAR( 20 ) NULL)
AT 'asademo1..dba.employee'
```

For a complete description of the CREATE TABLE statement, see "CREATE TABLE statement" on page 415 of the book *Adaptive Server Anywhere Reference Manual*.

Listing the columns on a remote table

If you are entering a CREATE EXISTING statement and you are specifying a column list, it may be helpful to get a list of the columns that are available on a remote table. The **sp_remote_columns** system procedure produces a list of the columns on a remote table and a description of those data types.

```
sp_remote_columns servername [,tablename] [, owner ] [,
database]
```

If a table name, owner, or database name is given, the list of columns is limited to only those that match.

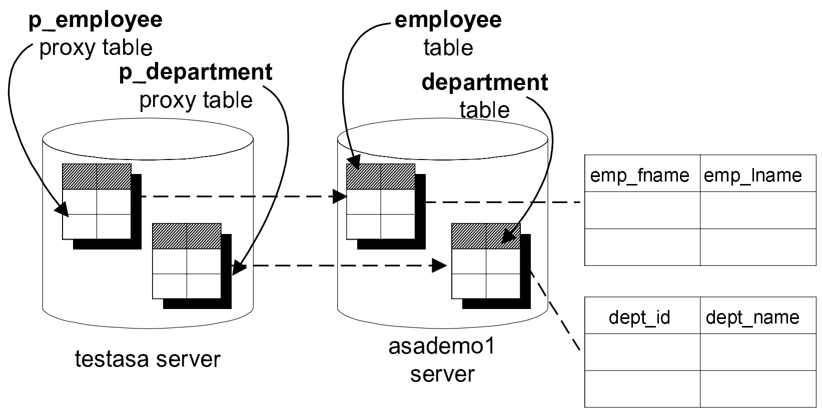
For example, to get a list of the columns in the **sysobjects** table in the production database in an Adaptive Server Enterprise server named **asetest**:

```
sp_remote_columns  asetest, sysobjects, null, production
```

For more information, see "sp_remote_columns system procedure" on page 758 of the book *Adaptive Server Anywhere Reference Manual*.

Example: a join between two remote tables

The following figure illustrates the remote Adaptive Server Anywhere tables **employee** and **department** in the sample database mapped to the local server named **testasa**.



This example shows how to:

- ♦ Define the remote **testasa** server
- ♦ Create the proxy tables **employee** and **department**
- ♦ Perform a join between the remote **employee** and **department** tables.

In real-world cases, you may use joins between tables on different Adaptive Server Anywhere databases. Here we describe a simple case using just one database, which may not be particularly useful, to illustrate the principles.

❖ To perform a join between two remote tables, using Interactive SQL:

1 Create a new database named *empty.db*.

This database holds no data. We will use it only to define the remote objects, and access the sample database from it.

2 Start a database server running both empty.db and the sample database. You can do this using the following command line, executed from the installation directory:

```
dbeng6 asademo empty
```

3 Connect to *empty.db* it from Interactive SQL using user ID **dba** and password **sql**.

4 In the new database, create a remote server named **testasa**. Its server class is **asajdbc**, and the connection information is **'ASA 6.0 Sample'**:

```
CREATE SERVER testasa
CLASS 'asaodbc'
USING 'ASA 6.0 Sample'
```

5 In this example, we use the same user ID and password on the remote database as on the local database, so no external logins are needed.

6 Define the **employee** proxy table:

```
CREATE EXISTING TABLE employee
AT 'testasa..dba.employee'
```

7 Define the **department** proxy table:

```
CREATE EXISTING TABLE department
AT 'testasa..dba.department'
```

8 Use the proxy tables in the SELECT statement to perform the join.

```
SELECT emp_fname, emp_lname, dept_name
FROM employee JOIN department
ON employee.dept_id = department.dept_id
ORDER BY emp_lname
```

Accessing multiple local databases

An Adaptive Server Anywhere server may have several local databases running at one time. By defining tables in other local Adaptive Server Anywhere databases as remote tables, you can perform cross database joins.

For example, if you are using database **db1** and you want to access data in tables in database **db2**, you need to set up proxy table definitions that point to the tables in database **db2**. For instance, on an Adaptive Server Anywhere named **testasa**, you might have three databases available, **db1**, **db2**, and **db3**.

- If using ODBC, create an ODBC data source name entry for each database you will be accessing.
- Connect to one of the databases that you will be performing joins from. For example, connect to **db1**.
- Perform a CREATE SERVER for each other local database you will be accessing. This sets up a **loopback** connection to your Adaptive Server Anywhere server

```
CREATE SERVER local_db2
CLASS 'asaodbc'
USING 'testasa_db2'

CREATE SERVER local_db3
CLASS 'asaodbc'
USING 'testasa_db3'
```

  or using JDBC:

```
CREATE SERVER local_db2
CLASS 'asajdbc'
USING 'mypc1:2638/db2'

CREATE SERVER local_db3
CLASS 'asajdbc'
USING 'mypc1:2638/db3'
```

- Create proxy table definitions using CREATE EXISTING to the tables in the other databases you want to access.

```
CREATE EXISTING TABLE employee
AT 'local_db2...employee'
```

Sending native statements to remote servers

Use the FORWARD TO statement to send one or more statements to the remote server in its native syntax. This statement can be used in two ways:

- ♦ To send a statement to a remote server
- ♦ To place Adaptive Server Anywhere into passthrough mode for sending a series of statements to a remote server

If a connection cannot be made to the specified server, the reason is contained in a message returned to the user. If a connection is made, any results are converted into a form that can be recognized by the client program.

The FORWARD TO statement can be used to verify that a server is configured correctly. If you send a statement to the remote server and Adaptive Server Anywhere does not return an error message, the remote server is configured correctly.

Example 1

The following statement verifies connectivity to the server named **ASEserver** by selecting the version string:

```
FORWARD TO ASEserver {SELECT @@version}
```

Example 2

The following statements show a passthrough session with the server named **ASEserver**:

```
FORWARD TO ASEserver
select * from titles
select * from authors
FORWARD TO
```

For a complete description of the FORWARD TO statement, see "FORWARD TO statement" on page 474 of the book *Adaptive Server Anywhere Reference Manual*.

Using remote procedure calls (RPCs)

Adaptive Server Anywhere users can issue procedure calls to remote servers that support the feature.

Sybase Adaptive Server Anywhere and Adaptive Server Enterprise, Oracle, and DB2 support this feature. Issuing a remote procedure call is similar to using a local procedure call.

❖ **To issue a remote procedure call:**

1 First define the procedure to Adaptive Server Anywhere.

  The syntax is the same as a local procedure definition except instead of using SQL statements to make up the body of the call, a location string is given defining the location where the procedure resides.

```
CREATE PROCEDURE remotewho ()
AT 'bostonase.master.dbo.sp_who'
```

2 Execute the procedure as follows:

```
call remotewho()
```

Here is an example with a parameter:

```
CREATE PROCEDURE remoteuser (IN uname char(30))
AT 'bostonase.master.dbo.sp_helpuser'

call remoteuser('joe')
```

Transaction management and remote data

Transactions provide a way to group SQL statements so that they are treated as a unit—either all work performed by the statements is committed to the database, or none of it is.

For the most part, transaction management with remote tables is the same as transaction management for local tables in Adaptive Server Anywhere, but there are some differences. They are discussed in the following section.

For a general discussion of transactions, see "Using Transactions and Locks" on page 367.

Remote transaction management overview

The method for managing transactions involving remote servers uses a **two-phase commit** protocol. Adaptive Server Anywhere implements a strategy that ensures transaction integrity for most scenarios. However, when more than one remote server is invoked in a transaction, there is still a chance that a distributed unit of work will be left in an undetermined state. Even though two-phase commit protocol is used, no recovery process is included.

The general logic for managing a user transaction is as follows:

1. Adaptive Server Anywhere prefaces work to a remote server with a BEGIN TRANSACTION notification.
2. When the transaction is ready to be committed, Adaptive Server Anywhere sends a PREPARE TRANSACTION notification to each remote server that has been part of the transaction. This ensures the that remote server is ready to commit the transaction.
3. If a PREPARE TRANSACTION request fails, all remote servers are told to roll back the current transaction.

   If all PREPARE TRANSACTION requests are successful, the server sends a COMMIT TRANSACTION request to each remote server involved with the transaction.

Any statement preceded by BEGIN TRANSACTION can begin a transaction. Other statements are sent to a remote server to be executed as a single, remote unit of work.

Restrictions on transaction management

Restrictions on transaction management are as follows:

- Savepoints are not propagated to remote servers.
- If nested BEGIN TRANSACTION and COMMIT TRANSACTION statements are included in a transaction that involves remote servers, only the outermost set of statements is processed. The innermost set, containing the BEGIN TRANSACTION and COMMIT TRANSACTION statements, is not transmitted to remote servers.

Internal operations

This section describes the underlying operations on remote servers performed by Adaptive Server Anywhere on behalf of client applications.

Query parsing

When a statement is received from a client, it is parsed. An error is raised if the statement is not a valid Adaptive Server Anywhere SQL statement.

Query normalization

The next step is called query normalization. During this step, referenced objects are verified and some data type compatibility is checked.

For example, consider the following query:

```
SELECT *
FROM t1
WHERE c1 = 10
```

The query normalization stage verifies that table **t1** with a column **c1** exists in the system tables. It also verifies that the data type of column **c1** is compatible with the value 10. If the column's data type is datetime, for example, this statement is rejected.

Query preprocessing

Query preprocessing prepares the query for optimization. It may change the representation of a statement so that the SQL statement Adaptive Server Anywhere generates for passing to a remote server will be syntactically different from the original statement.

Preprocessing performs view expansion so that a query can operate on tables referenced by the view. Expressions may be reordered and subqueries may be transformed to improve processing efficiency. For example, some subqueries may be converted into joins.

Server capabilities

The previous steps are performed on all queries, both local and remote.

The following steps depend on the type of SQL statement and the capabilities of the remote servers involved.

Each remote server defined to Adaptive Server Anywhere has a set of capabilities associated with it. These capabilities are stored in the **syscapabilities** system table. These capabilities are initialized during the first connection to a remote server. The generic server class odbc relies strictly on information returned from the ODBC driver to determine these capabilities. Other server classes such as db2odbc have more detailed knowledge of the capabilities of a remote server type and use that knowledge to supplement what is returned from the driver.

Once **syscapabilities** is initialized for a server, the capability information is retrieved only from the system table. This allows a user to alter the known capabilities of a server.

Since a remote server may not support all of the features of a given SQL statement, Adaptive Server Anywhere must break the statement into simpler components to the point that the query can be given to the remote server. SQL features not passed off to a remote server must be evaluated by Adaptive Server Anywhere itself.

For example, a query may contain an ORDER BY statement. If a remote server cannot perform ORDER BY, the statement is sent to a the remote server without it and Adaptive Server Anywhere performs the ORDER BY on the result returned, before returning the result to the user. The result is that the user can employ the full range of Adaptive Server Anywhere supported SQL without concern for the features of a particular back end.

Complete passthrough of the statement

The most efficient way to handle a statement is usually to hand as much of the original statement as possible off to the remote server involved. Adaptive Server Anywhere will attempt to pass off as much of the statement as is possible. In many cases this will be the complete statement as originally given to Adaptive Server Anywhere.

Adaptive Server Anywhere will hand off the complete statement when:

- Every table in the statement resides in the same remote server.
- The remote server is capable of processing all of the syntax in the statement.

In rare conditions, it may actually be more efficient to let Adaptive Server Anywhere do some of the work instead of passing it off. For example, Adaptive Server Anywhere may have a better sorting algorithm. In this case you may consider altering the capabilities of a remote server using the ALTER SERVER statement.

For more information see "ALTER SERVER statement" on page 349 of the book *Adaptive Server Anywhere Reference Manual*.

Partial passthrough of the statement

If a statement contains references to multiple servers, or uses SQL features not supported by a remote server, the query is decomposed into simpler parts.

Select

SELECT statements are broken down by removing portions that cannot be passed on and letting Adaptive Server Anywhere perform the feature. For example, let's say a remote server can not process the atan2() function in the following statement:

```
select a,b,c where atan2(b,10) > 3 and c = 10
```

The statement sent to the remote server would be converted to:

```
select a,b,c where c = 10
```

Locally, Adaptive Server Anywhere would apply "where atan2(b,10) > 3" to the intermediate result set.

Joins

Adaptive Server Anywhere processes joins using a nested loop algorithm. When two tables are joined, one table is selected to be the outer table. The outer table is scanned based on the WHERE conditions that apply to it. For every qualifying row found, the other table, known as the inner table is scanned to find a row that matches the join condition.

This same algorithm is used when remote tables are referenced. Since the cost of searching a remote table is usually much higher than a local table (due to network I/O), every effort is made to make the remote table the outermost table in the join.

Update and delete

If Adaptive Server Anywhere cannot pass off an UPDATE or DELETE statement entirely to a remote server, it must change the statement into a table scan containing as much of the original WHERE clause as possible, followed by positioned UPDATE or DELETE "where current of cursor" when a qualifying row is found.

For example, when the function **atan2** is not supported by a remote server:

```
UPDATE t1
SET a = atan2(b, 10)
WHERE b > 5
```

Would be converted to the following:

```
SELECT a,b
FROM t1
WHERE  b > 5
```

Each time a row is found, Adaptive Server Anywhere would calculate the new value of **a** and issue:

```
UPDATE t1
SET a = 'new value'
WHERE CURRENT OF CURSOR
```

If **a** already has a value that equals the "new value", a positioned UPDATE would not be necessary and would not be sent remotely.

In order to process an UPDATE or DELETE that requires a table scan, the remote data source must support the ability to perform a positioned UPDATE or DELETE ("where current of cursor"). Some data sources do not support this capability.

Temporary tables cannot be updated
In this release of Adaptive Server Anywhere an UPDATE or DELETE cannot be performed if an intermediate temporary table is required in Adaptive Server Anywhere. This occurs in queries with ORDER BY and some queries with subqueries.

Troubleshooting remote data access

This section provides some hints for troubleshooting remote servers.

Features not supported for remote data

The following Adaptive Server Anywhere features are not supported on remote data. Attempts to use these features will therefore run into problems:

- ALTER TABLE statement against remote tables
- Triggers defined on proxy tables will not fire
- SQL Remote
- Java data types
- Foreign keys that refer to remote tables are ignored
- The READTEXT, WRITETEXT, and TEXTPTR functions.
- Positioned UPDATE and DELETE
- UPDATE and DELETE requiring an intermediate temporary table.
- Backwards scrolling on cursors opened against remote data. Fetch statements must be NEXT or RELATIVE 1.
- If a column on a remote table has a name that is a keyword on the remote server, you cannot access data in that column. Adaptive Server Anywhere cannot know all of the remote server reserved words. You can execute a CREATE EXISTING TABLE statement, and import the definition but you cannot select that column.

Case sensitivity

The case sensitivity setting of your Adaptive Server Anywhere database should match the settings used by any remote servers accessed.

Adaptive Server Anywhere databases are created case insensitive by default. With this configuration, unpredictable results may occur when selecting from a case sensitive database. Different results will occur depending on whether ORDER BY or string comparisons are pushed off to a remote server or evaluated by the local Adaptive Server Anywhere.

Connectivity problems

Take the following steps to be sure you can connect to a remote server:

- Determine that you can connect to a remote server using a client tool such as Interactive SQL before configuring Adaptive Server Anywhere.
- Perform a simple passthrough statement to a remote server to check your connectivity and remote login configuration. For example:

  ```
  FORWARD TO testasa {select @@version}
  ```

- Turn on remote tracing for a trace of the interactions with remote servers.

  ```
  SET OPTION cis_option = 2
  ```

General problems with queries

If you are faced with some type of problem with the way Adaptive Server Anywhere is handling a query against a remote table, it is usually helpful to understand how Adaptive Server Anywhere is executing that query. You can display remote tracing as well as a description of the query execution plan:

```
SET OPTION cis_option = 6
```

Queries blocked on themselves

If you access multiple databases on a single Adaptive Server Anywhere server, you may need to increase the number of threads used by the database server on Windows NT using the `-gx` command-line switch.

You must have enough threads available to support the individual tasks that are being run by a query. Failure to provide the number of required tasks can lead to a query becoming blocked on itself.