CHAPTER 30

Transact-SQL Compatibility

About this chapter

Transact-SQL is the dialect of SQL supported by Sybase Adaptive Server Enterprise.

This chapter is a guide for creating applications that are compatible with both Adaptive Server Anywhere and Adaptive Server Enterprise. It describes Adaptive Server Anywhere support for Transact-SQL language elements and statements, and for Adaptive Server Enterprise system tables, views, and procedures.

Contents

Before you begin

For brevity, in this chapter, Adaptive Server Enterprise is occasionally abbreviated to Enterprise, and Adaptive Server Anywhere is occasionally abbreviated to Anywhere.

The dialect of SQL supported by Enterprise is called Transact-SQL. In this chapter, the alternative dialect, supported by Anywhere, is called Watcom-SQL.

An overview of Transact-SQL support

Adaptive Server Anywhere supports a large subset of **Transact-SQL**, which is the dialect of SQL supported by Sybase Adaptive Server Enterprise. This chapter describes compatibility of SQL between Anywhere and Enterprise.

Goals

The goals of Transact-SQL support in Adaptive Server Anywhere are as follows:

- **Application portability** Many applications, stored procedures, and batch files can be written to be used with both Enterprise and Anywhere databases.
- **Data portability** Data can be exchanged and replicated between Anywhere and Enterprise databases with a minimum of effort.

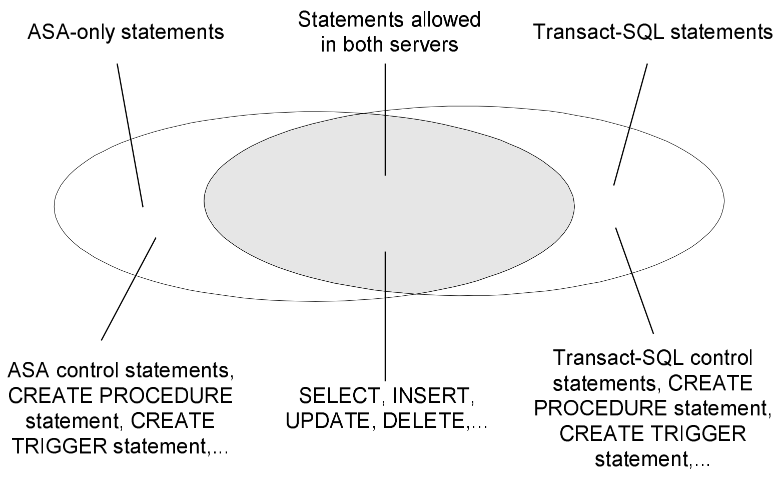
The aim is to make it possible to write applications to work with both Enterprise and Anywhere. Existing Adaptive Server Enterprise applications will generally require some changes to run on a Anywhere database.

How Transact-SQL is supported

Transact-SQL support in Anywhere takes the following form:

- Many SQL statements are compatible between Anywhere and Enterprise.
- For some statements, particularly in the procedure language used in procedures, triggers, and batches, a separate Transact-SQL statement is supported together with the syntax supported in previous versions of Adaptive Server Anywhere. For these statements, Adaptive Server Anywhere supports two **dialects** of SQL. In this chapter, we name those dialects Transact-SQL and Watcom-SQL.
- A procedure, trigger, or batch is executed in either the Transact-SQL or Watcom-SQL dialect. You must use control statements from one dialect only throughout the batch or procedure. For example, each dialect has different flow control statements.

The following diagram illustrates how the two dialects overlap.


ASA-only statements
Statements allowed in both servers
Transact-SQL statements
ASA control statements, CREATE PROCEDURE statement, CREATE TRIGGER statement,...
SELECT, INSERT, UPDATE, DELETE,...
Transact-SQL control statements, CREATE PROCEDURE statement, CREATE TRIGGER statement,...

Similarities and differences

Adaptive Server Anywhere supports a very high percentage of Transact-SQL language elements, functions, and statements for working with existing data. For example, all of the numeric functions are supported, all but one of the string functions are supported, all aggregate functions are supported, and all date and time functions are supported. As another example, Adaptive Server Anywhere supports Transact-SQL outer joins (using =* and *= operators) and extended DELETE and UPDATE statements using joins.

Further, a very high percentage of the Transact-SQL stored procedure language is supported (CREATE PROCEDURE and CREATE TRIGGER syntax, control statements, and so on), and many but not all aspects of Transact-SQL data definition language statements are supported.

There are design differences in the architectural and configuration facilities supported by each product. Device management, user management, and maintenance tasks such as backups tend to be system-specific. Even here, Transact-SQL system tables are provided in Adaptive Server Anywhere as views, where the tables that are not meaningful in Adaptive Server Anywhere have no rows. Also, a set of system procedures is provided for some of the more common administrative tasks.

This chapter looks first at some system-level issues where differences are most marked, before discussing data manipulation and data definition language aspects of the dialects where compatibility is high.

Transact-SQL only

There are SQL statements supported by Adaptive Server Anywhere that are part of one dialect, but not the other. The two dialects cannot be mixed within a procedure, trigger, or batch. For example: the following are supported in Adaptive Server Anywhere, but as part of the Transact-SQL dialect only:

- ♦ The Transact-SQL control statements IF and WHILE.

- The Transact-SQL EXECUTE statement.
- The Transact-SQL CREATE PROCEDURE and CREATE TRIGGER statements.
- The Transact-SQL BEGIN TRANSACTION statement.
- SQL Statements *not* separated by semicolons are part of a Transact-SQL procedure or batch.

Adaptive Server Anywhere only

The following statements are among those not supported in Adaptive Server Enterprise:

- The control statements CASE, LOOP, and FOR
- The Adaptive Server Anywhere versions of IF and WHILE.
- The CALL statement.
- The Adaptive Server Anywhere versions of the CREATE PROCEDURE, CREATE FUNCTION, and CREATE TRIGGER statements.
- SQL Statements separated by semicolons are not supported in Adaptive Server Enterprise.

Notes

The two dialects cannot be mixed within a procedure, trigger, or batch. That is:

- You can include Transact-SQL-only statements together with statements that are part of both dialects in a batch, procedure, or trigger.
- You can include statements not supported by Adaptive Server Enterprise together with statements that are supported by both servers in a batch, procedure, or trigger.
- You cannot include Transact-SQL-only statements together with Adaptive Server Anywhere-only statements in a batch, procedure, or trigger.

Adaptive Server architectures

Adaptive Server Enterprise and Adaptive Server Anywhere are complementary products, with architectures designed to suit their distinct purposes. Adaptive Server Anywhere is designed as a workgroup or departmental server requiring little administration, and as a personal database. Adaptive Server Enterprise is designed as an enterprise-level server for the largest databases.

This section describes architectural differences between Adaptive Server Enterprise and Adaptive Server Anywhere. It also describes the Adaptive Server Enterprise-like tools that Adaptive Server Anywhere includes for compatible database management.

Servers and databases

The relationship between servers and databases is different in Adaptive Server Enterprise and in Adaptive Server Anywhere.

In Adaptive Server Enterprise, each database exists inside a server, and each server can contain several databases. Users are granted login rights to the server, and can connect to the server. They can then use each database on that server for which they have been granted permissions. System-wide system tables, held in a master database, contain information common to all databases on the server.

No master database in Adaptive Server Anywhere

In Adaptive Server Anywhere, there is no level corresponding to the Adaptive Server Enterprise master database. Instead, each database is an independent entity, containing all of its system tables. Users are granted connection rights to a database, not to the server. When a user connects, the connection is to an individual database. There is no system-wide set of system tables maintained at a master database level. Each Adaptive Server Anywhere database server can dynamically load and unload multiple databases, and users can maintain independent connections on each, but there is no way of addressing more than one database from within a single connection.

Adaptive Server Anywhere provides tools in its Transact-SQL support and in its Open Server support to allow some tasks to be carried out in a Adaptive Server Enterprise-like manner. For example, Adaptive Server Anywhere provides an implementation of the Adaptive Server Enterprise **sp_addlogin** system procedure that carries out the nearest equivalent action: adding a user to a database.

For information about Open Server support, see "Adaptive Server Anywhere as an Open Server" on page 815.

File manipulation statements

The Transact-SQL statements CREATE DATABASE, DROP DAT

ABASE, DUMP DATABASE, and LOAD DATABASE are not supported in Adaptive Server Anywhere, although Adaptive Server Anywhere does have its own CREATE DATABASE statement that is different in syntax.

Device management

Adaptive Server Anywhere and Adaptive Server Enterprise use different models for managing devices and disk space, reflecting the different uses for the two products. While Adaptive Server Enterprise sets out a comprehensive resource management scheme using a variety of Transact-SQL statements, Adaptive Server Anywhere is designed to be able to manage its own resources automatically, and its databases are regular operating system files.

Adaptive Server Anywhere does not support Transact-SQL DISK statements, such as DISK INIT, DISK MIRROR, DISK REFIT, DISK REINIT, DISK REMIRROR, and DISK UNMIRROR.

For information on disk management, see "Working with Database Files" on page 613

Defaults and rules

Adaptive Server Anywhere does not support the Transact-SQL CREATE DEFAULT statement or CREATE RULE statement. The CREATE DOMAIN statement allows a default and a rule (called a CHECK condition) to be incorporated into the definition of a user-defined data type, and so provides similar functionality to the Transact-SQL CREATE DEFAULT and CREATE RULE statements.

In Adaptive Server Enterprise, the CREATE DEFAULT statement creates a named **default**. This default can be used as a default value for columns by binding the default to a particular column or as a default value for all columns of a user-defined data type by binding the default to the data type. A default is bound to a data type or column using the **sp_bindefault** system procedure.

The CREATE RULE statement creates a named **rule** which can be used to define the domain for columns by binding the rule to a particular column or which can be used as a rule for all columns of a user-defined data type by binding the rule to the data type. A rule is bound to a data type or column using the **sp_bindrule** system procedure.

In Adaptive Server Anywhere, a user-defined data type can have a default value and a CHECK condition associated with it, which are applied to all columns defined on that data type. The user-defined data type is created using the CREATE DATATYPE statement.

Default values and rules, or CHECK conditions, can be defined for individual columns using the CREATE TABLE statement or the ALTER TABLE statement.

For a description of the Adaptive Server Anywhere syntax for these statements, see "SQL Statements" on page 339 of the book *Adaptive Server Anywhere Reference Manual*.

System tables

In addition to its own system tables, Adaptive Server Anywhere provides a set of system views that mimic relevant parts of the Adaptive Server Enterprise system tables. These are listed and described individually in "Views for Transact-SQL Compatibility" on page 851 of the book *Adaptive Server Anywhere Reference Manual*, which describes the system catalogs of the two products. This section provides a brief overview of the differences.

The Adaptive Server Anywhere system tables are held entirely within each database, while the Adaptive Server Enterprise system tables are held partly inside each database and partly in the master database. The Adaptive Server Anywhere architecture does not include a master database.

In Adaptive Server Enterprise, the system tables are owned by the database owner, user ID **dbo**. In Adaptive Server Anywhere, the system tables are owned by the system owner, user ID **SYS**. The Adaptive Server Enterprise-compatible system views provided by Adaptive Server Anywhere are owned by a **dbo** user ID.

Administrative roles

Adaptive Server Enterprise has a more elaborate set of administrative roles than Adaptive Server Anywhere. In Adaptive Server Enterprise there is a set of distinct roles, although more than one login account on a Adaptive Server Enterprise can be granted any role, and one account can possess more than one role.

Adaptive Server Enterprise roles

In Adaptive Server Enterprise the following are distinct roles:

♦ **System Administrator** Responsible for general administrative tasks unrelated to specific applications; can access any database object.

- **System Security Officer** Responsible for security-sensitive tasks in Adaptive Server Enterprise, but has no special permissions on database objects.
- **Database Owner** Has full permissions on objects inside the database that he or she owns, can add users to a database and can grant other users the permission to create objects and execute commands within the database.
- **Data definition statements** Permissions can be granted to users for specific data definition statements, such as CREATE TABLE or CREATE VIEW, enabling the user to use those statements to create database objects.
- **Object owner** Each database object has an owner, who may grant permissions to other users to access the object. The owner of an object automatically has all permissions on the object.

In Adaptive Server Anywhere, the following database-wide permissions have administrative roles:

- The Database Administrator (DBA permissions) has, like the Adaptive Server Enterprise database owner, full permissions on objects inside the database that he or she owns and can grant other users the permission to create objects and execute commands within the database. The default database administrator is user ID **DBA**.
- The RESOURCE permission allows a user to create any kind of object within a database. This is instead of the Adaptive Server Enterprise scheme of granting permissions on individual CREATE statements.
- Adaptive Server Anywhere has object owners in the same way that Adaptive Server Enterprise does. The owner of an object automatically has all permissions on the object, including the right to grant permissions.

For seamless access to data held in both Adaptive Server Enterprise and Adaptive Server Anywhere, you should create user IDs with appropriate permissions in the database (RESOURCE in Adaptive Server Anywhere, or permission on individual CREATE statements in Adaptive Server Enterprise) and create objects from that user ID. If the same user ID is used in each environment, object names and qualifiers can be identical in the two databases, helping to ensure compatible access.

Users and groups

There are some differences between the Adaptive Server Enterprise and Adaptive Server Anywhere models of users and groups.

In Adaptive Server Enterprise, connections are made to a server, and each user requires a login ID and password to the server as well as a user ID for each database they will access on that server. Each user of a database can be a member of at most one group.

In Adaptive Server Anywhere, where connections are made to a database, there is nothing corresponding to the login ID. Instead, each user is granted a user ID and password on a database in order to use that database. Users can be members of many groups, and group hierarchies are allowed.

Both servers support user groups, so that you can grant permissions to many users at one time. However, there are differences in the specifics of groups in the two servers. For example, Adaptive Server Enterprise allows each user to be a member of only one group, while Adaptive Server Anywhere has no such restriction. You should compare the documentation on users and groups in the two products for specific information.

Both Adaptive Server Enterprise and Adaptive Server Anywhere have a **public** group, for defining default permissions. Every user is automatically a member of the **public** group.

Adaptive Server Anywhere supports the following Adaptive Server Enterprise system procedures for managing users and groups.

For the arguments to each procedure, see "Adaptive Server Enterprise system and catalog procedures" on page 767 of the book *Adaptive Server Anywhere Reference Manual*.

System procedure	Description
sp_addlogin	In Adaptive Server Enterprise, this adds a user to the server. In Adaptive Server Anywhere, this adds a user to a database.
sp_adduser	In Adaptive Server Enterprise and Adaptive Server Anywhere, this adds a user to a database. While this is a distinct task from **sp_addlogin** in Adaptive Server Enterprise, in Adaptive Server Anywhere, they are the same.
sp_addgroup	Adds a group to a database.
sp_changegroup	Adds a user to a group, or moves a user from one group to another.
sp_droplogin	In Adaptive Server Enterprise, removes a user from the server. In Adaptive Server Anywhere, removes a user from the database.
sp_dropuser	Removes a user from the database.
sp_dropgroup	Removes a group from the database.

In Adaptive Server Enterprise, login IDs are created on a server-wide basis. In Adaptive Server Anywhere, users are created for individual databases.]

Database object permissions

The Adaptive Server Enterprise and Adaptive Server Anywhere GRANT and REVOKE statements for granting permissions on individual database objects are very similar. Both allow SELECT, INSERT, DELETE, UPDATE, and REFERENCES permissions on database tables and views, and UPDATE permissions on selected columns of database tables. Both allow EXECUTE permissions to be granted on stored procedures.

For example, the following statement is valid in both Adaptive Server Enterprise and Adaptive Server Anywhere:

```
GRANT INSERT, DELETE
ON TITLES
TO MARY, SALES
```

This statement grants permission to use the INSERT and DELETE statements on the **titles** table to user **Mary** and to the **sales** group.

The WITH GRANT OPTION clause, allowing the recipient of permission to grant them in turn, is supported in both Adaptive Server Anywhere and Adaptive Server Enterprise, although Adaptive Server Anywhere does not permit WITH GRANT OPTION to be used on a GRANT EXECUTE statement.

Database-wide permissions

Adaptive Server Enterprise and Adaptive Server Anywhere use different models for database-wide user permissions. These are discussed in "Users and groups" on page 788. Adaptive Server Anywhere employs DBA permissions to allow a user full authority within a database. This permission is enjoyed by the System Administrator in Adaptive Server Enterprise, for all databases on a server. However, DBA authority on an Adaptive Server Anywhere database is different from the permissions of a Adaptive Server Enterprise Database Owner, who must use the Adaptive Server Enterprise **setuser** statement to gain permissions on objects owned by other users.

Adaptive Server Anywhere employs RESOURCE permissions to allow a user the right to create objects in a database. A closely corresponding Adaptive Server Enterprise permission is GRANT ALL used by a Database Owner.

General guidelines for writing portable SQL

When writing SQL that will be used on more than one database management system, you should be as explicit as possible in your SQL statements. Even if a given SQL statement is supported by more than one server, it may be a mistake to assume that default behavior when an option is not specified is the same on each system. The following general guidelines apply when writing compatible SQL:

- Spell out all of the available options, rather than using default behavior.
- Make the order of execution within statements explicit using parentheses, rather than assuming identical default order of precedence for operators.
- Use the Transact-SQL convention of an @ sign preceding variable names for Adaptive Server Enterprise portability.
- In procedures, triggers, and batches, declare variables and cursors immediately following a BEGIN statement. This is required by Adaptive Server Anywhere, although Adaptive Server Enterprise allows declarations to be made anywhere in a procedure, trigger, or batch.
- Avoid using reserved words from either Adaptive Server Enterprise or Adaptive Server Anywhere as identifiers in your databases.

Configuring databases for Transact-SQL compatibility

Some differences in behavior between Adaptive Server Anywhere and Adaptive Server Enterprise can be eliminated by selecting appropriate options when creating a database or, if you are working on an existing database, when rebuilding the database. Other differences can be controlled by connection level options using the SET TEMPORARY OPTION statement in Adaptive Server Anywhere or the SET statement in Adaptive Server Enterprise.

Creating a Transact-SQL-compatible database

This section describes choices that must be made when a database is created or rebuilt.

Quick start

Here are the steps you need to take. The remainder of the section describes what the options are that .

❖ **To create a Transact-SQL compatible database from Sybase Central:**

♦ One page of the Create Database wizard is named Choosing the Database Attributes. This page provides a button that sets each of the available choices to emulate Adaptive Server Enterprise..

❖ **To create a Transact-SQL compatible database using dbinit:**

♦ Enter the following command at a system prompt:

```
dbinit -b -c -k db-name.db
```

❖ **To create a Transact-SQL compatible database using the CREATE DATABASE statement:**

♦ Enter the following statement, for example in Interactive SQL:

```
CREATE DATABASE 'db-name.db'
ASE COMPATIBLE
```

In this statement, ASE COMPATIBLE meanse compatible with Adaptive Server Enterprise.

Make the database case-sensitive

By default, string comparisons in Adaptive Server Enterprise databases are case-sensitive, while those in Adaptive Server Anywhere are case insensitive.

When building a Adaptive Server Enterprise-compatible database using Adaptive Server Anywhere, you should choose the case-sensitive option.

- ♦ If you are using Sybase Central, this option is in the Create Database wizard.
- ♦ If you are using the *dbinit* command-line utility, specify the -c command-line switch.

Ignore trailing blanks in comparisons

When building an Adaptive Server Enterprise-compatible database using Adaptive Server Anywhere, you should choose the option to ignore trailing blanks in comparisons.

- ♦ If you are using Sybase Central, this option is in the Create Database wizard.
- ♦ If you are using Interactive SQL for Windows 3.x, this option is on the Create Database dialog box that appears when you select Create Database from the Database Tools window and click Create.
- ♦ If you are using the *dbinit* command line utility, specify the `-b` command line switch.

With this option chosen, the following two strings are considered equal by both Adaptive Server Enterprise and Adaptive Server Anywhere:

```
'ignore the trailing blanks   '

'ignore the trailing blanks'
```

If this option is not chosen, the two strings above are considered different by Adaptive Server Anywhere.

A side effect of this option is that strings are padded with blanks when fetched by a client application.

Remove historical system views

Older versions of Adaptive Server Anywhere employed two system views whose names conflict with the Adaptive Server Enterprise system views provided for compatibility. These views are SYSCOLUMNS and SYSINDEXES. If you are not using Watcom SQL 4.0 and you are using Open Client or JDBC interfaces, you should create your database excluding these views. You can do this with the *dbinit* `-k` command-line switch.

If you do not use this option when creating your database, the following two statements return different results:

```
SELECT * FROM SYSCOLUMNS ;

SELECT * FROM dbo.SYSCOLUMNS ;
```

❖ To drop the system views from an existing database:

1 Connect to the database as a user with DBA authority.

2 Execute the following statements:

```
DROP VIEW SYS.SYSCOLUMNS ;
```

```
DROP VIEW SYS.SYSINDEXES
```

Caution
Ensure that you do not drop the dbo.SYSCOLUMNS or dbo.SYSINDEXES system view.

Setting options for Transact-SQL compatibility

Adaptive Server Anywhere database options are set using the SET OPTION statement. Several database option settings are relevant to Transact-SQL behavior.

Set the allow_nulls_by_default option

By default, Adaptive Server Enterprise does not allow NULLs on new columns unless they are explicitly declared to allow NULLs. Adaptive Server Anywhere permits NULL in new columns by default, which is compatible with the SQL/92 ISO standard.

To make Adaptive Server Enterprise behave in a SQL/92-compatible manner, use the **sp_dboption** system procedure to set the **allow_nulls_by_default** option to true.

To make Adaptive Server Anywhere behave in a Transact-SQL-compatible manner, set the **allow_nulls_by_default** option to OFF. You can do this using the SET OPTION statement as follows:

```
SET OPTION PUBLIC.allow_nulls_by_default = 'OFF'
```

Set the quoted_identifier option

By default, the Adaptive Server Enterprise treatment of identifiers and of strings differs from the Adaptive Server Anywhere behavior, which matches the SQL/92 ISO standard.

The **quoted_identifier** option is available in both Adaptive Server Enterprise and Adaptive Server Anywhere. You should ensure that the option is set to the same value in both databases, for identifiers and strings to be treated in a compatible manner.

For SQL/92 behavior, set the **quoted_identifier** option to ON in both Adaptive Server Enterprise and Adaptive Server Anywhere.

For Transact-SQL behavior, set the **quoted_identifier** option to OFF in both Adaptive Server Enterprise and Adaptive Server Anywhere. If you choose this, you can no longer use identifiers that are the same as identifiers, enclosed in double quotes.

For more information on the **quoted_identifier** option, see "QUOTED_IDENTIFIER option" on page 170 of the book *Adaptive Server Anywhere Reference Manual*.

Set the automatic_ timestamp option to ON

Transact-SQL defines a **timestamp** column with special properties. With the **automatic_timestamp** option set to ON, the Adaptive Server Anywhere treatment of **timestamp** columns is more similar to Adaptive Server Enterprise behavior.

With the **automatic_timestamp** option set to ON in Adaptive Server Anywhere (the default setting is OFF), any new columns with the TIMESTAMP data type that do not have an explicit default value defined are given a default value of **timestamp**.

For information on **timestamp** columns, see "The special Transact-SQL timestamp column and data type" on page 797.

Set the string_rtruncation option

Both Adaptive Server Enterprise and Adaptive Server Anywhere support the string_rtruncation option, which affects whether or not error messages are reported when an INSERT or UPDATE string is truncated. You should ensure that the option is set to the same value in each database.

For more information on the STRING_RTRUNCATION option, see "STRING_RTRUNCATION option" on page 173 of the book *Adaptive Server Anywhere Reference Manual*.

For more information on database options for Transact-SQL compatibility, see "Transact-SQLcompatibility options" on page 134 of the book *Adaptive Server Anywhere Reference Manual*.

Case-sensitivity

Case sensitivity in databases refers to the following:

- **Data** The case sensitivity of the data is reflected in indexes, in the results of queries, and so on.
- **Identifiers** Identifiers include table names, column names, and so on.
- **User IDs and passwords** Case sensitivity of user IDs and passwords is treated differently to other identifiers.

Case sensitivity of data

The case-sensitivity of Adaptive Server Anywhere data in comparisons is decided when the database is created. By default, Adaptive Server Anywhere databases are case-insensitive in comparisons, although data is always held in the case in which it is entered.

Adaptive Server Enterprise's sensitivity to the case (upper or lower) of data depends on the sort order installed on the Adaptive Server Enterprise system. Case sensitivity can be changed for single-byte character sets by reconfiguring the Adaptive Server Enterprise sort order.

Case sensitivity of identifiers

Adaptive Server Anywhere does not support case-sensitive identifiers. In Adaptive Server Enterprise, the case sensitivity of identifiers follows the case sensitivity of the data.

In Adaptive Server Enterprise, user-defined data type names are case sensitive. In Adaptive Server Anywhere, they are case insensitive, with the exception of Java data types.

User IDs and passwords

In Adaptive Server Anywhere, user IDs and passwords follow the case sensitivity of the data. The default user ID and password for case sensitive databases are upper case **DBA** and **SQL**, respectively.

In Adaptive Server Enterprise, the case sensitivity of user IDs and passwords follows the case sensitivity of the server.

Ensuring compatible object names

Each database object must have a unique name within a certain **name space**. Outside this name space, duplicate names are allowed. There are some database objects that occupy different name spaces in Adaptive Server Enterprise and Adaptive Server Anywhere.

In Adaptive Server Anywhere, indexes and triggers are owned by the owner of the table on which they are created. Index and trigger names must be unique for a given owner. For example, while the tables **t1** owned by user **user1** and **t2** owned by user **user2** may have indexes of the same name, no two tables owned by a single user may have an index of the same name.

Adaptive Server Enterprise has a less restrictive name space for index names than Adaptive Server Anywhere. Index names must be unique on a given table, but any two tables may have an index of the same name. For compatible SQL, you should stay within the Adaptive Server Anywhere restriction of unique index names for a given table owner.

Adaptive Server Enterprise has a more restrictive name space on trigger names than Adaptive Server Anywhere. Trigger names must be unique in the database. For compatible SQL, you should stay within the Adaptive Server Enterprise restriction and make your trigger names unique in the database.

The special Transact-SQL timestamp column and data type

Adaptive Server Anywhere supports the Transact-SQL special **timestamp** column. The **timestamp** column is used together with the **tsequal** system function to check whether a row has been updated.

Two meanings of timestamp
Adaptive Server Anywhere has a TIMESTAMP data type, which holds accurate date and time information. This is distinct from the special Transact-SQL TIMESTAMP column and data type.

Creating a Transact-SQL timestamp column in Adaptive Server Anywhere

To create a Transact-SQL **timestamp** column, create a column that has the (Adaptive Server Anywhere) data type TIMESTAMP and has a default setting of **timestamp**. The column can have any name, although the name **timestamp** is commonly used.

For example, the following CREATE TABLE statement includes a Transact-SQL **timestamp** column:

```
CREATE TABLE table_name (
   column_1 INTEGER ,
   column_2 TIMESTAMP default timestamp
)
```

The following ALTER TABLE statement adds a Transact-SQL **timestamp** column to the **sales_order** table:

```
ALTER TABLE sales_order
ADD timestamp TIMESTAMP default timestamp
```

In Adaptive Server Enterprise a column with the name **timestamp** and no data type specified is automatically given a TIMESTAMP data type. In Adaptive Server Anywhere you must explicitly assign the data type yourself.

If you have the AUTOMATIC_TIMESTAMP database option set to ON, you do not need to set the default value: any new column created with TIMESTAMP data type and with no explicit default is given a default value of **timestamp**. The following statement sets AUTOMATIC_TIMESTAMP to ON:

```
SET OPTION PUBLIC.AUTOMATIC_TIMESTAMP='ON'
```

The data type of a timestamp column

Adaptive Server Enterprise treats a **timestamp** column as a user-defined data type that is VARBINARY(8), allowing NULL, while Adaptive Server Anywhere treats a **timestamp** column as the TIMESTAMP data type, which consists of the date and time, with fractions of a second held to six decimal places.

When fetching from the table for later updates, the variable into which the timestamp value is fetched should correspond to the way the column is described.

Timestamping an existing table

If you add a special **timestamp** column to an existing table, all existing rows have a NULL value in the **timestamp** column. To enter a timestamp value (the current timestamp) for existing rows, update all rows in the table such that the data does not change. For example, the following statement updates all rows in the **sales_order** table, without changing the values in any of the rows:

```
UPDATE sales_order
SET region = region
```

In Interactive SQL, you may need to set the TIMESTAMP_FORMAT option to see the differences in values for the rows. The following statement sets the TIMESTAMP_FORMAT option to display all six digits in the fractions of a second:

```
SET OPTION TIMESTAMP_FORMAT='YYYY-MM-DD
HH:MM:ss.SSSSSS'
```

If all six digits are not shown, some **timestamp** column values may appear to be equal: they are not.

Using tsequal for updates

With the **tsequal** system function you can tell whether a **timestamp** column has been updated or not.

For example, an application may SELECT a **timestamp** column into a variable. When an UPDATE of one of the selected rows is submitted, it can use the **tsequal** function to check that the row has not been changed. The **tsequal** function compares the timestamp value in the table with the timestamp value obtained in the SELECT. If they are the same, the row has not been changed; if they differ, the row has been changed since the SELECT was carried out.

The following is a typical UPDATE statement using the **tsequal** function:

```
UPDATE publishers
SET city = 'Springfield'
WHERE pub_id = '0736'
AND TSEQUAL(timestamp, '1995/10/25 11:08:34.173226')
```

The first argument to the **tsequal** function is the name of the special **timestamp** column; the second argument is the timestamp retrieved in the SELECT statement. In Embedded SQL, the second argument is likely to be a host variable containing a TIMESTAMP value from a recent FETCH on the column.

The special IDENTITY column

To create an IDENTITY column, use the following CREATE TABLE syntax:

```
CREATE TABLE table-name (
   ...
   column-name numeric(n,0) IDENTITY NOT NULL,
   ...
)
```

where *n* is large enough to hold the value of the maximum number of rows that may be inserted into the table.

The IDENTITY column is used to store sequential numbers, such as invoice numbers or employee numbers, which are automatically generated. The value of the IDENTITY column uniquely identifies each row in a table.

In Adaptive Server Enterprise, each table in a database can have one IDENTITY column. The data type must be numeric with scale zero, and the IDENTITY column should not allow nulls.

In Adaptive Server Anywhere, the IDENTITY column is implemented as a column default setting. Values that are not part of the sequence may be explicitly inserted into the column with an INSERT statement. Adaptive Server Enterprise does not allow INSERTs into identity columns unless the **identity_insert** option is set to *on*. In Adaptive Server Anywhere, you need to set the NOT NULL property yourself and ensure that no more than one column is an IDENTITY column. Adaptive Server Anywhere allows any numeric data type to be used as an IDENTITY column.

In Adaptive Server Anywhere the **identity** column is identical to the AUTOINCREMENT default setting for a column.

Retrieving IDENTITY Column Values with @@identity

The first time you insert a row into the table, a value of 1 is assigned to an IDENTITY column. On each subsequent insert, the value of the column is incremented by one. The value most recently inserted into an identity column is available in the **@@identity** global variable.

The value of @@identity changes each time a statement attempts to insert a row into a table.

- If the statement affects a table without an IDENTITY column, @@identity is set to 0.
- If the statement inserts multiple rows, @@identity reflects the last value inserted into the IDENTITY column.

This change is permanent. @@identity does not revert to its previous value if the statement fails or if the transaction that contains it is rolled back Also, the value for @@identity within a stored procedure or trigger does not affect the value outside the stored procedure or trigger.

Writing compatible SQL statements

This section describes some issues of compatibility between Adaptive Server Enterprise and Adaptive Server Anywhere at the SQL statement level.

Creating compatible tables

Adaptive Server Anywhere does not support named constraints or named defaults, but does support user-defined data types which allow constraint and default definitions to be encapsulated in the data type definition. It also supports explicit defaults and CHECK conditions in the CREATE TABLE statement.

Temporary tables

You can create a temporary table by preceding the table name in a CREATE TABLE statement with a pound sign (#). These temporary tables are Adaptive Server Anywhere declared temporary tables, and are available only in the current connection. For information about declared temporary tables in Adaptive Server Anywhere, see "DECLARE LOCAL TEMPORARY TABLE statement" on page 441 of the book *Adaptive Server Anywhere Reference Manual*.

By default, columns in Adaptive Server Enterprise default to NOT NULL, whereas in Adaptive Server Anywhere the default setting is NULL, to allow NULL values. This setting can be controlled using the **allow_nulls_by_default** option. You should explicitly specify NULL or NOT NULL to make your data definition statements transferable.

For information on this option, see "Setting options for Transact-SQL compatibility" on page 794.

Physical placement of a table is carried out differently in Adaptive Server Enterprise and in Adaptive Server Anywhere. The **ON** *segment-name* clause is supported in Adaptive Server Anywhere, but *segment-name* refers to an Adaptive Server Anywhere dbspace.

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

Writing compatible queries

There are two criteria for writing a query that runs on both Adaptive Server Anywhere and Adaptive Server Enterprise databases:

♦ Ensure that the data types, expressions, and search conditions in the query are compatible.

♦ Ensure that the syntax of the SELECT statement itself is compatible.

This section is concerned with compatible SELECT statement syntax, and assumes compatible data types, expressions, and search conditions. The examples assume that the QUOTED_IDENTIFIER setting is OFF: the default Adaptive Server Enterprise setting, but not the default Adaptive Server Anywhere setting.

The following subset of the Transact-SQL SELECT statement is supported in Adaptive Server Anywhere.

Syntax

```
SELECT [ ALL | DISTINCT ] select-list
      ...[ INTO #temporary-table-name ]
      ...[ FROM table-spec [ HOLDLOCK | NOHOLDLOCK ],
      ...          table-spec [ HOLDLOCK | NOHOLDLOCK ], ... ]
      ...[ WHERE search-condition ]
      ...[ GROUP BY column-name, ... ]
      ...[ HAVING search-condition ]
      ...| [ ORDER BY expression [ ASC | DESC ], ... ]         |
         | [ ORDER BY integer [ ASC | DESC ], ... ] |
```

Parameters

```
select-list:
      { table-name | alias-name = expression }…

table-spec:
            [ owner . ]table-name
      …     [ [ AS ] correlation-name ]
      …     [ ( INDEX index_name [ PREFETCH size  ][ LRU | MRU ] )]

alias-name:
            identifier | 'string' | "string"
```

For a full description of the SELECT statement, see "SELECT statement" on page 542 of the book *Adaptive Server Anywhere Reference Manual*.

The following keywords and clauses of the Transact-SQL SELECT statement syntax are not supported by Adaptive Server Anywhere:

♦ The SHARED keyword.

♦ The COMPUTE clause.

♦ The FOR BROWSE clause.

♦ The GROUP BY ALL clause.

Notes

♦ The INTO table_name clause, which creates a new table based on the SELECT statement result set, is supported only for declared temporary tables where the table name starts with a #. Declared temporary tables exist for a single connection only.

- Adaptive Server Anywhere does not support the Transact-SQL extension to the GROUP BY clause allowing references to columns and expressions that are not used for creating groups. In Adaptive Server Enterprise, this extension produces summary reports.
- The FOR READ ONLY clause and the FOR UPDATE clause are parsed, but have no effect.
- The performance parameters part of the table specification is parsed, but has no effect.
- The HOLDLOCK keyword is supported by Adaptive Server Anywhere. It makes a shared lock on a specified table or view more restrictive by holding it until the completion of a transaction (instead of releasing the shared lock as soon as the required data page is no longer needed, whether or not the transaction has been completed). For the purposes of the table for which the HOLDLOCK is specified, the query is carried out at isolation level 3.
- The HOLDLOCK option applies only to the table or view for which it is specified, and only for the duration of the transaction defined by the statement in which it is used. Setting the isolation level to 3 applies a holdlock for each select within a transaction. You cannot specify both a HOLDLOCK and NOHOLDLOCK option in a query.
- The NOHOLDLOCK keyword is recognized by Adaptive Server Anywhere, but has no effect.
- Transact-SQL uses the SELECT statement to assign values to local variables:

  ```
  SELECT @localvar = 42
  ```

  The corresponding statement in Adaptive Server Anywhere is the SET statement:

  ```
  SET localvar = 42
  ```

  However, using the Transact-SQL SELECT to assign values to variables is supported inside batches.
- The following clauses of the SELECT statement syntax are not supported by Adaptive Server Enterprise:
  - INTO host-variable-list
  - INTO variable-list.
  - Parenthesized queries.
- Adaptive Server Enterprise does not support the use of the FROM clause and the ON condition for joins. Instead, it uses join operators in the WHERE clause.

Compatibility of joins

In Transact-SQL, joins are specified in the WHERE clause, using the following syntax:

```
Start of select, update, insert, delete, or subquery
      FROM {table-list | view-list} WHERE [ NOT ]
      ...[ table-name.| view name.]column-name
         join-operator
      ...[ table-name.| view-name.]column_name
      ...[ { AND | OR } [ NOT ]
      ... [ table-name.| view-name.]column_name
            join-operator
         [ table-name.| view-name.]column-name
        ]...
      end of select, update, insert, delete, or subquery
```

The *join_operator* in the WHERE clause may be any of the comparison operators, or may be either of the following **outer-join operators**:

- ***=** Left outer join operator
- **=*** Right outer join operator.

The Transact-SQL outer-join operators are supported in Adaptive Server Anywhere as an alternative to the native SQL/92 syntax. You cannot mix dialects within a query. This rule applies also to views used by a query—an outer-join query on a view must follow the dialect used by the view-defining query.

Adaptive Server Anywhere also provides a SQL/92 syntax for joins other than outer joins, in which the joins are placed in the FROM clause rather than the WHERE clause.

For information about joins in Adaptive Server Anywhere and in SQL/92, see "FROM clause" on page 476 of the book *Adaptive Server Anywhere Reference Manual*.

For more information on Transact-SQL compatibility of joins, see "Transact-SQL outer joins" on page 155.

Transact-SQL procedure language overview

The **stored procedure language** is that part of SQL used in stored procedures, triggers, and batches.

Adaptive Server Anywhere supports a large part of the Transact-SQL stored procedure language in addition to the Watcom-SQL dialect based on SQL/92.

Transact-SQL stored procedure overview

The Adaptive Server Anywhere stored procedure language is based on the ISO/ANSI draft standard, which differs from the Transact-SQL dialect in many ways. Many of the concepts and features are similar, but the syntax is different. Adaptive Server Anywhere support for Transact-SQL takes advantage of the similar concepts by providing automatic translation between dialects. However, a procedure must be written in one of the two dialects exclusively, not in a mixture.

Adaptive Server Anywhere support for Transact-SQL stored procedures

There are several aspects to Adaptive Server Anywhere support for Transact-SQL stored procedures:

- Passing parameters
- Returning result sets
- Returning status information
- Providing default values for parameters
- Control statements
- Error handling

Transact-SQL trigger overview

Trigger compatibility requires compatibility of trigger features and of trigger syntax. This section provides an overview of the feature compatibility of Transact-SQL and Adaptive Server Anywhere triggers.

Adaptive Server Enterprise triggers are executed after the triggering statement has completed: they are **statement level**, **after** triggers. Adaptive Server Anywhere supports both **row level** triggers (which execute before or after each row has been modified) and statement level triggers (which execute after the entire statement has been executed).

Row-level triggers are not discussed here, as they are not part of the Transact-SQL compatibility features. For information on row-level triggers in Adaptive Server Anywhere, see "Using Procedures, Triggers, and Batches" on page 221.

Description of unsupported or different Transact-SQL triggers

The following list describes some features of Transact-SQL triggers that are either not supported or are different in Adaptive Server Anywhere:

- **Triggers firing other triggers** Suppose a trigger carries out an action that would, if carried out directly by a user, fire another trigger. Adaptive Server Anywhere and Adaptive Server Enterprise have slightly different behavior for this case. The default Adaptive Server Enterprise behavior is for triggers to fire other triggers up to a configurable nesting level, which has the default value of 16. The nesting level can be controlled by the Adaptive Server Enterprise option **nested triggers**. In Adaptive Server Anywhere, triggers fire other triggers without limit unless memory is exhausted.
- **Triggers firing themselves** Suppose a trigger carries out an action that would, if carried out directly by a user, fire the same trigger. Adaptive Server Anywhere and Adaptive Server Enterprise have different behavior for this case.

  In Adaptive Server Anywhere, non-Transact-SQL triggers fire themselves recursively, while Transact-SQL dialect triggers do not fire themselves recursively.

  The default Adaptive Server Enterprise behavior is that a trigger does not call itself recursively, but you can turn on the **self_recursion** option to allow triggers to call themselves recursively.
- **ROLLBACK statement in triggers** Within a trigger, Adaptive Server Enterprise permits the ROLLBACK TRANSACTION statement, which rolls back the entire transaction of which the trigger is a part. Adaptive Server Anywhere does not permit ROLLBACK (or ROLLBACK TRANSACTION) statements in triggers. A triggering action and its trigger together form an atomic statement, and Adaptive Server Anywhere does not permit ROLLBACKs within atomic statements.

Transact-SQL batch overview

In Transact-SQL, a **batch** is a set of SQL statements submitted together and executed as a group, one after the other. Batches can be stored in command files. The Interactive SQL utility in Adaptive Server Anywhere and the *isql* utility in Adaptive Server Enterprise provide similar capabilities for executing batches interactively.

The control statements used in procedures can also be used in batches. Adaptive Server Anywhere supports the use of control statements in batches and the Transact-SQL-like use of non-delimited groups of statements terminated with a GO statement to signify the end of a batch.

For batches stored in command files, Adaptive Server Anywhere supports the use of parameters in command files. Adaptive Server Enterprise does not support parameters.

For information on parameters, see "PARAMETERS statement" on page 518 of the book *Adaptive Server Anywhere Reference Manual*.

Automatic translation of stored procedures

In addition to supporting Transact-SQL alternative syntax, Adaptive Server Anywhere provides aids for translating statements between the Watcom-SQL and Transact-SQL dialects. The following functions return information about SQL statements and enable automatic translation of SQL statements:

- **SQLDialect(statement)** Returns **Watcom-SQL** or **Transact-SQL**.
- **WatcomSQL(statement)** Returns the Watcom-SQL syntax for the statement.
- **TransactSQL(statement)** Returns the Transact-SQL syntax for the statement.

These are functions, and so can be accessed, for example, using a select statement from Interactive SQL. For example, the following statement:

```
select SqlDialect('select * from employee')
```

returns the value WatcomSQL.

Using Sybase Central to translate stored procedures

Sybase Central has facilities for creating, viewing, and altering procedures and triggers.

❖ **To translate a stored procedure using Sybase Central:**

1. Connect to a database using Sybase Central, either as owner of the procedure you wish to change, or as a DBA user.
2. Double-click the Procedures folder for the database to list the stored procedures in the database.
3. Using the right mouse button, click the procedure you wish to translate, and choose the dialect you wish to translate it to from the popup menu: either Watcom-SQL or Transact-SQL.

   The procedure is displayed in the selected dialect. If the selected dialect is not the one in which the procedure is stored, it is translated to that dialect by the server. Any untranslated lines are displayed as comments.
4. Rewrite any untranslated lines as needed, and click the Execute Script button to save the translated version to the database. You can also export the text to a file for editing outside Sybase Central.

Returning result sets from Transact-SQL procedures

Adaptive Server Anywhere uses a RESULT clause to specify returned result sets. In Transact-SQL procedures, the column names or alias names of the first query are returned to the calling environment.

Example of Transact-SQL procedure

The following Transact-SQL procedure illustrates how result sets are returned from Transact-SQL stored procedures:

```
CREATE PROCEDURE showdept @deptname varchar(30)
AS
   SELECT employee.emp_lname, employee.emp_fname
   FROM department, employee
   WHERE department.dept_name = @deptname
   AND department.dept_id = employee.dept_id
```

Example of Watcom-SQL procedure

The following is the corresponding Adaptive Server Anywhere procedure:

```
CREATE PROCEDURE showdept(in deptname varchar(30))
RESULT ( lastname char(20), firstname char(20))
ON EXCEPTION RESUME
BEGIN
   SELECT employee.emp_lname, employee.emp_fname
   FROM department, employee
   WHERE department.dept_name = deptname
   AND department.dept_id = employee.dept_id
END
```

Notes

- Multiple result sets with a different number of columns or incompatible data types cannot be returned from procedures in Adaptive Server Anywhere.
- When a RESULT clause is not specified (as is the case with Transact-SQL procedures), Adaptive Server Anywhere determines the result set from the first SELECT statement in the procedure. The first SELECT statement is identified without regard for IF statements or other control statements: you cannot have a procedure return one result set under one set of conditions and an incompatible result set under other conditions.

Variables in Transact-SQL procedures

Adaptive Server Anywhere uses the SET statement to assign values to variables in a procedure. In Transact-SQL, values are assigned using the SELECT statement with an empty table-list. The following simple procedure illustrates how the Transact-SQL syntax works:

```
CREATE PROCEDURE multiply
                @mult1 int,
                @mult2 int,
                @result int output
AS
SELECT @result = @mult1 * @mult2
```

This procedure can be called as follows:

```
CREATE VARIABLE @product int ;
```

```
EXECUTE multiply 5, 6, @product OUTPUT;
```

The variable **@product** has a value of 30 after the procedure is executed.

For more information on using the SELECT statement to assign variables, see "Writing compatible queries" on page 801. For more information on using the SET statement to assign variables, see "SET statement" on page 546 of the book *Adaptive Server Anywhere Reference Manual*.

Error handling in Transact-SQL procedures

Default procedure error handling is different in the Watcom-SQL and Transact-SQL dialects. By default, Watcom-SQL dialect procedures exit when an error is encountered, returning SQLSTATE and SQLCODE values to the calling environment.

Explicit error handling can be built into Watcom-SQL stored procedures using the EXCEPTION statement, or the procedure can be instructed by the ON EXCEPTION RESUME statement to continue execution at the next statement when an error is encountered.

When an error is encountered in a Transact-SQL dialect procedure, execution continues at the following statement. The global variable **@@error** holds the error status of the most recently executed statement. You can check this variable following a statement to force return from a procedure. For example, the following statement causes an exit if an error occurs.

```
IF @@error != 0 RETURN
```

When the procedure completes execution, a return value indicates the success or failure of the procedure. This return status is an integer, and can be accessed as follows:

```
DECLARE @status INT
EXECUTE @status = proc_sample
IF @status = 0
   PRINT 'procedure succeeded'
ELSE
   PRINT 'procedure failed'
```

The following table describes the built-in procedure return values and their meanings:

Value	Meaning
0	Procedure executed without error
-1	Missing object
-2	Data type error
-3	Process was chosen as deadlock victim
-4	Permission error
-5	Syntax error
-6	Miscellaneous user error
-7	Resource error, such as out of space
-8	Non-fatal internal problem
-9	System limit was reached

Value	Meaning
-10	Fatal internal inconsistency
-11	Fatal internal inconsistency
-12	Table or index is corrupt
-13	Database is corrupt
-14	Hardware error

The RETURN statement can be used to return integers other than these, with their own user-defined meanings.

Using the RAISERROR statement in procedures

The RAISERROR statement is a Transact-SQL statement for generating user-defined errors. It has a similar function to the SIGNAL statement.

For a description of the RAISERROR statement, see "RAISERROR statement" on page 526 of the book *Adaptive Server Anywhere Reference Manual*.

By itself, the RAISERROR statement does not cause an exit from the procedure, but it can be combined with a RETURN statement or a test of the **@@error** global variable to control execution following a user-defined error.

If you set the CONTINUE_AFTER_RAISERROR database option to ON, the RAISERROR statement no longer signals an execution-ending error. Instead, the RAISERROR status code and message are stored and the most recent RAISERROR is returned when the procedure completes. If the procedure that caused the RAISERROR was called from another procedure, the RAISERROR is not returned until the outermost calling procedure terminates.

Intermediate RAISERROR statuses and codes are lost after the procedure terminates. If at return time an error occurs along with the RAISERROR then the error information is returned and the RAISERROR information is lost. The application can query intermediate RAISERROR statuses by examining **@@error** global variable at different execution points.

Transact-SQL-like error handling in the Watcom-SQL dialect

You can make a Watcom-SQL dialect procedure handle errors in a Transact-SQL-like manner by supplying the ON EXCEPTION RESUME clause to the CREATE PROCEDURE statement:

```
CREATE PROCEDURE sample_proc()
ON EXCEPTION RESUME
BEGIN
   ...
END
```

Explicit exception handling code is not executed if an ON EXCEPTION RESUME clause is present.