CHAPTER 23

Working with Database Files

About this Chapter

This chapter describes how to create and work with database and associated files.

Contents

Overview of database files

Basic database files

Many databases consist of a single database file. In this case, the database server uses three files while it is running:

- **The database file** This file holds the database information. It typically has the extension *.db*.

  For information on creating databases, see "Working with databases" on page 68.

- **The transaction log** This file holds a record of the changes made to the database file, and is needed for recovery and replication. It typically has the extension *.log*.

  For information on the transaction log, see "Backup and Data Recovery" on page 553.

- **The temporary file** This file is used by the database server to hold information needed during a session. It is not needed once the database server shuts down. The file has a server-generated name with the extension *.tmp*. It is held in the system temporary directory.

  The temporary file is created and maintained by the server. You do not need to do anything with the temporary file.

Additional files

Other files can also be used as part of a database system:

- **Additional database files** You can spread your data over several files, each on held in a separate file. These additional files are called **dbspaces**.
- **Transaction log mirror files** For additional security, you can create a mirror copy of the transaction log. This file typically has the extension *.mlg*.
- **Write files** If the database file is a read-only file (for example, if it is distributed on CD-ROM) an additional file named a write file can be used, which holds changes made to the data.
- **Compressed database files** You can compress a database file. The resulting file is read only, but can be used in conjunction with a write file.

sdkjsalksadkljdsakljdsakljsdakljadssad

Chapter goals

This chapter describes how to create, name, and delete the different kinds of files involved in a database system.

Using additional dbspaces

This section describes how to use additional database files, named **dbspaces**.

Only needed for large databases
For many databases, it is convenient to keep the database as a single file. This section is intended only for users of large databases.

When a database is initialized, it is composed of one database file. This first database file is called the **root file**. All database objects and all data are placed in the root file.

Each database file has a maximum size of two Gb, so you may wish to divide large databases among more than one file. On Windows NT drives using the NTFS file system, this limitation is removed and files can be up to one terabyte.

You create a new database file, or **dbspace**, from Sybase Central, or using the CREATE DBSPACE statement. A new dbspace may be on the same disk drive as the root file or on another disk drive. DBA authority is required to create database files.

For more information, see "CREATE DBSPACE statement" on page 389 of the book *Adaptive Server Anywhere Reference Manual*.

For each database, you can create up to twelve dbspaces, including the root file.

Placing tables in dbspaces

When created, a new dbspace has no contents. When you create a new table you can place it in a specific dbspace with an IN clause in the CREATE TABLE statement. If no IN clause is used, the table is placed in the root file.

Each table must be contained in a single dbspace. By default, indexes are placed in the same dbspace as their table, but they can be placed in a separate dbspace by supplying an IN clause.

For information on creating tables, see "Creating tables" on page 71.

Example

The following command creates a new dbspace called **library** in the file *library.db* in the same directory as the root file:

```
CREATE DBSPACE library
AS 'library.db'
```

To create a table and place it in the library dbspace, you can use the following command:

```
CREATE TABLE Library_Books (
title char(100),
author char(50),
isbn char(30)
) IN library
```

Splitting existing databases

If you wish to split existing database objects among several dbspaces, you need to unload your database and modify the command file for rebuilding the database. To do so, add IN clauses to specify the dbspace for each table you do not wish to place in the root file.

Creating a dbspace in Sybase Central

❖ **To create a dbspace in Sybase Central:**

1 Connect to the database.

2 Click the DB Spaces folder for that database.

3 Double-click Add DB Space in the right panel.

4 Enter the dbspace name and filename.

5 Click OK to create the dbspace.

Preallocating space for database files

Adaptive Server Anywhere automatically takes new disk space for database files as needed. Unless you are working with a large database with a high rate of inserts and deletes, you do not need to worry about explicitly allocating space for database files.

Rapidly changing database files could lead to excessive file fragmentation on the disk and possible performance problems. You may pre-allocate disk space for database files or for transaction logs in order to prevent this. You do this using the ALTER DBSPACE statement.

For example, the following statement adds 200 pages to the database file with dbspace name **library**. The database page size is fixed when the database is created.

```
ALTER DBSPACE library
ADD 200
```

For more information on this statement, see "ALTER DBSPACE statement" on page 345 of the book *Adaptive Server Anywhere Reference Manual*.

Running a disk defragmentation utility after pre-allocating disk space helps ensure that the database file is not fragmented over many disjoint areas of the disk drive. Performance can suffer if there is excessive fragmentation of database files.

Preallocating disk space in Sybase Central

❖ **To preallocate disk space for a dbspace in Sybase Central:**

1 Connect to the database.

2 Click the DB Spaces folder for that database.

3 Double-click the dbspace in the right panel.

4 Click Add Pages, and enter the number of database pages to preallocate.

5 Click OK.

Working with write files

If you have a database file that is read-only (for example, if you distribute a database on a CD-ROM), you can use a write file to enable changes to be made to the database.

You create a write file using the Write File utility or using the CREATE WRITEFILE statement. In this section, the examples use the command-line utilities.

For a description of the CREATE WRITEFILE statement, see "CREATE WRITEFILE statement" on page 432 of the book *Adaptive Server Anywhere Reference Manual*.

❖ **To use a write file:**

1 Create the write file for your database.

   For example, to create a write file for the sample database, you can enter the following command in the Adaptive Server Anywhere installation directory:

   ```
   dbwrite -c asademo.db
   ```

   This command creates a write file named *asademo.wrt*, with a transaction log named *asademo.wlg*.

2 Start a database server, loading the write file. By default, the server locates files with the extension *.wrt* first, so the following command in the installation directory starts the personal server running the sample database write file:

   ```
   dbeng6 asademo
   ```

   The messages on the server window indicate which file is started.

3 Connect to the database using Interactive SQL. You can use the user ID **DBA** and the password **SQL**, as the sample database is the default.

4 You can execute queries just as usual. The following query lists the contents of the **department** table.

   ```
   SELECT *
   FROM department
   ```

5 Try inserting a row. The following statement inserts a row into the department table:

   ```
   INSERT
   INTO department (dept_id, dept_name)
   VALUES (202, 'Eastern Sales')
   ```

If you committed this change, it would be written to the *asademo.wlg* transaction log, and when the database checkpoints, it is written to the *asademo.wrt* write file.

If you now query this table, the results are taken from the write file and the database file.

6 Try deleting a row. Set the WAIT_FOR_COMMIT option to avoid referential integrity complaints here:

```
SET TEMPORARY OPTION wait_for_commit = 'on' ;

DELETE
FROM department
WHERE dept_id = 100
```

If you committed this change, the deletion would be marked in the write file. No change is made to the database file.

For some purposes, it is useful to use a write file with a shared database. If you have a read-only database on a network server, you could let each user have their own write file. In this way, they could add their own information, which would be stored on their own machine, without affecting the database. This can be useful for application development also.

Deleting a write file

You can use the *dberase* utility to delete a write file and its associated transaction log.

Using the utility database

You can connect to a server using the **utility database**, a phantom database that has no physical representation—that is, there is no database file for this database, and it can contain no data.

The utility database can be loaded by specifying **utility_db** as the database name when connecting. The utility database permits only a narrow range of specialized functions. It is provided so that you can execute database file manipulation statements such as CREATE DATABASE, or ALTER WRITEFILE, without first connecting to a physical database.

Sybase Central

You cannot connect to the utility database from Sybase Central. However, as you can already create and delete a database from Sybase Central without first connecting to a database, this is not a practical limitation.

Starting a server with no database

Normally when you start a database server, you specify the database you wish to load. However, you can start a server that does not load a database.

For example, the following command from the command line starts a personal database server named **TestEng** but does not load a database.

```
dbeng6 -n TestEng
```

❖ **To load and connect to the utility database:**

1 Start a database server with the following command:

```
dbeng6.exe -n TestEng
```

2 Start Interactive SQL.

3 On the Login tab of the Connection Window, enter **DBA** as the user ID and **SQL** as the password. On the Database tab, enter **utility_db** as the database name.

4 Click OK to connect.

   Interactive SQL connects to the utility database on the personal server named **TestEng**. No real database is actually loaded.

You can now execute the database file administration statements. For example, executing the following statement after having connected to the utility database creates a database named *new.db* in the directory *C:\temp*.

```
CREATE DATABASE 'C:\\temp\\new.db'
```

For more information on the syntax of those statements, see "CREATE DATABASE statement" on page 385 of the book *Adaptive Server Anywhere Reference Manual*.

Utility database server security

There are two aspects to utility database server security:

- ♦ Who can connect to the utility database?
- ♦ Who can execute file administration statements?

These are discussed in this section.

Utility database passwords

There is a different security model for the personal server and the network server.

For the personal server, you must specify the user ID **DBA**. You can use any password; as the personal server is intended for single machine use, a security restriction is not needed.

For the network server, you must specify the user ID **DBA**, but the password is held in a file named *util_db.ini*, which is stored in the server executable directory. As this directory is on the server, you can control access to the file, and thereby control who has access to the password.

util_db.ini

The *util_db.ini* file has the following contents:

```
[UTILITY_DB]
PWD=password
```

Use of the **utility_db** security level relies on the physical security of the computer hosting the database server since the *util_db.ini* file can be easily read using a text editor.

Permission to execute file administration statements

Based on the utility database, a new level of security has been added for the ability to create and drop databases. The `-gu` database server command-line option controls who can execute the file administration statements.

There are four levels of permission for the use of file administration statements. These levels are: **all**, **none**, **dba**, and **utility_db**. The **utility_db** level permits only a person able to connect to the utility database to use the file administration statements.

-gu switch option	Effect	applies to
all	Anyone can execute file administration statements	Any database including utility database
none	No one can execute file administration statements	Any database including utility database
dba	Only dba-authority users can execute file administration statements	Any database including utility database
utility_db	Only the user who can connect to utility database can execute file administration statements	Only the utility database

For more information on the database server `-gu` command line switch, see "The database server" on page 12 of the book *Adaptive Server Anywhere Reference Manual*.

Examples

♦ To prevent the use of the file administration statements, start the database server using the **none** permission level of the `-gu` switch. The following command starts a database server and names it **TestSrv**, loads the sample database, but prevent anyone from using that server to create or delete a database.

```
dbsrv6.exe -n TestSrv -gu none asademo.db
```

With the database server named **TestSrv** started in this manner, all users are prevented from using the server to create or add a database, regardless of their resource creation rights, or whether or not they can load and connect to the utility database.

♦ To permit only the user knowing the utility database password to connect to create or delete databases, start the server at the command line with the following command.

```
dbsrv6 -n TestSrv -gu utility_db
```

Assuming the utility database password has been set during installation to **asa**, the following command starts the Interactive SQL utility as a client application, connects to the server named **TestSrv**, loads the utility database and connects the user.

```
dbisql -c
"uid=dba;pwd=asa;dbn=utility_db;eng=TestSrv"
```

Having executed the above statement successfully, the user is connected to the utility database, and is able to create or delete databases.