

CHAPTER 21

Managing User IDs and Permissions

About this chapter

Each user of a database must be assigned a user ID: the name they type when connecting to the database. This chapter describes how to manage user IDs.

Contents

Topic	Page
Database permissions overview	576
Managing individual user IDs and permissions	580
Managing groups	586
Database object names and prefixes	591
Using views and procedures for extra security	593
How user permissions are assessed	596
Managing the resources connections use	597
Users and permissions in the system tables	598

Database permissions overview

Proper management of user IDs and permissions lets users of a database carry out their jobs effectively, while maintaining the security and privacy of information within the database.

You use SQL statements for assigning user IDs to new users of a database, granting and revoking permissions for database users, and finding out the current permissions of users.

Database permissions are assigned to user IDs. Throughout this chapter, the term **user** is used as a synonym for user ID. You should remember, however, that permissions are granted and revoked for each user ID.

Setting up individual user IDs

Even if there are no security concerns regarding a multi-user database, there are good reasons for setting up an individual user ID for each user. The administrative overhead is very low if a group with the appropriate permissions is set up. Groups of users are discussed in this chapter.

Among the reasons for using individual user IDs are the following:

- ◆ The log translation utility can selectively extract the changes made by individual users from a transaction log. This is very useful when troubleshooting or piecing together what happened if data is incorrect.
- ◆ The network server screen and the listing of connections in Sybase Central are both much more useful with individual user IDs, as you can tell which connections are which users.
- ◆ Row locking messages (with the BLOCKING option set to OFF) are more informative.

DBA authority overview

When a database is created, a single usable user ID is created. This first user ID is **DBA**, and the password is initially set to **SQL**. The **DBA** user ID is automatically given DBA permissions, also called DBA authority, within the database. This level of permission enables the DBA user ID to carry out any activity in the database: create tables, change table structures, create new user IDs, revoke permissions from users, and so on.

Users with DBA authority

A user with DBA authority is referred to as the **database administrator** or **database owner**. In this chapter, frequent reference is made to the database administrator, or **DBA**. This is shorthand for *any user or users with DBA authority*.

Although DBA authority may be granted or transferred to other user IDs, in this chapter it is assumed that the **DBA** user ID is the database administrator, and the abbreviation *DBA* is used to mean both the **DBA** user ID and any user ID with DBA authority.

Adding new users

The DBA has the authority to add new users to the database. As users are added, they are also granted permissions to carry out tasks on the database. Some users may need to simply look at the database information using SQL queries, others may need to add information to the database, and others may need to modify the structure of the database itself. Although some of the responsibilities of the DBA may be handed over to other user IDs, the DBA is responsible for the overall management of the database by virtue of the DBA authority.

The DBA has authority to create database objects and assign ownership of these objects to other user IDs.

RESOURCE authority overview

RESOURCE authority is the permission to create database objects, such as tables, views, stored procedures, and triggers. RESOURCE authority may be granted only by the DBA.

In order to create a trigger, a user needs ALTER permissions on the table to which the trigger applies, in addition to RESOURCE authority.

Ownership permissions overview

The creator of a database object becomes the owner of that object. Ownership of a database object carries with it permissions to carry out actions on that object. These are not assigned to users in the same way that other permissions in this chapter are assigned.

Owners

A user who creates a new object within the database is called the **owner** of that object, and automatically has permission to carry out any operation on that object. The owner of a table may modify the structure of that table, for instance, or may grant permissions to other database users to update the information within the table.

The DBA has permission to modify any component within the database, and so could delete a table created by another user, for instance. The DBA has all the permissions regarding database objects that the owners of each object have.

The DBA is also able to create database objects for other users, and in this case the owner of an object is not the user ID that executed the CREATE statement. A use for this ability is discussed in "Groups without passwords" on page 589. Despite this possibility, this chapter refers interchangeably to the owner and creator of database objects.

Table and views permissions overview

There are several distinct permissions that may be granted to user IDs concerning tables and views:

Permission	Description
ALTER	Permission to alter the structure of a table or create a trigger on a table
DELETE	Permission to delete rows from a table or view
INSERT	Permission to insert rows into a table or view
REFERENCES	Permission to create indexes on a table, and to create foreign keys that reference a table
SELECT	Permission to look at information in a table or view
UPDATE	Permission to update rows in a table or view. This may be granted on a set of columns in a table only
ALL	All the above permissions

Group permissions overview

Setting permissions individually for each user of a database can be a time-consuming and error-prone process. For most databases, permission management based on groups, rather than on individual user IDs, is a much more efficient approach.

You can assign permissions to a group in exactly the same way as to an individual user. You can then assign membership in appropriate groups to each new user of the database, and they gain a set of permissions by virtue of their group membership.

Example

For example, you may create groups for different departments in a company database (**sales**, **marketing**, and so on) and assign these groups permissions. Each salesperson is made a member of the **sales** group, and automatically gains access to the appropriate areas of the database.

Any user ID can be a member of several groups, and inherits all permissions from each of the groups.

Managing individual user IDs and permissions

This section describes how to create new users and grant permissions to them. For most databases, the bulk of permission management should be carried out using **groups**, rather than by assigning permissions to individual users one at a time. However, as a group is simply a user ID with special properties, you should read and understand this section before moving on to the discussion of managing groups.

Creating new users

A new user is added to a database by the DBA using the GRANT CONNECT statement. For example:

❖ **To add a new user to a database, with user ID M_Haneef and password welcome:**

- 1 From Interactive SQL, connect to the database as a user with DBA authority.
- 2 Issue the SQL statement:

```
GRANT CONNECT TO M_Haneef  
IDENTIFIED BY welcome
```

Only the DBA has the authority to add new users to a database.

Initial permissions
for new users


By default, new users are not assigned any permissions beyond connecting to the database and viewing the system tables. In order to access tables in the database, they need to be assigned permissions.

The DBA can set the permissions granted automatically to new users by assigning permissions to the special **PUBLIC** user group, as discussed in "Special groups" on page 589.

Creating users in
Sybase Central

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

- 1 Connect to the database.
- 2 Click the Users and Groups folder for that database.
- 3 Double-click Add User. A Wizard is displayed, which leads you through the process.

 For more information, see the Sybase Central online Help.

Changing a password

Changing a user's password

You can change your password, or that of another user if you have DBA authority, using the GRANT statement. For example, the following command changes the password for user ID **M_Haneef** to **new_password**:

```
GRANT CONNECT TO M_Haneef
IDENTIFIED BY new_password
```

Changing the DBA password

The default password for the **DBA** user ID for all databases is **SQL**. You should change this password to prevent unauthorized access to your database. The following command changes the password for user ID **DBA** to **new_password**:

```
GRANT CONNECT TO DBA
IDENTIFIED BY new_password
```

Granting DBA and RESOURCE authority

DBA and RESOURCE authority are granted in the same manner.

❖ To grant RESOURCE permissions to a user ID:

- 1 Connect to the database as a user with DBA authority.
- 2 Type and execute the SQL statement:

```
GRANT RESOURCE TO userid
```

For DBA authority, the appropriate SQL statement is:

```
GRANT DBA TO userid
```

Notes

- ◆ Only the DBA may grant DBA or RESOURCE authority to database users.
- ◆ DBA authority is very powerful, granting the ability to carry out any action on the database and access to all the information in the database. It is generally inadvisable to grant DBA authority to more than a very few people.
- ◆ You should consider giving users with DBA authority two user IDs, one with DBA authority and one without, so that they connect as DBA only when necessary.
- ◆ RESOURCE authority allows the user to create new database objects, such as tables, views, indexes, procedures, or triggers.

Granting permissions on tables and views

You can assign a set of permissions on individual tables and views. Users can be granted combinations of these permissions to define their access to a table or view.

Combinations of permissions

- ◆ The ALTER permission allows a user to alter the structure of a table or to create triggers on a table. The REFERENCES permission allows a user to create indexes on a table, and to create foreign keys. These permissions grant the authority to modify the database schema, and so will not be assigned to most users. These permissions do not apply to views.
- ◆ The DELETE, INSERT, and UPDATE permissions grant the authority to modify the data in a table or view. Of these, the UPDATE permission may be restricted to a set of columns in the table or view.
- ◆ The SELECT permission grants authority to look at data in a table or view, but does not give permission to alter it.
- ◆ ALL permission grants all the above permissions.

Example 1

All table and view permissions are granted in a very similar fashion. You can grant permission to **M_Haneef** to delete rows from the table named **sample_table** as follows:

- 1 Connect to the database as a user with DBA authority, or as the owner of **sample_table**.
- 2 Type and execute the SQL statement:

```
GRANT DELETE
ON sample_table
TO M_Haneef
```

Example 2

You can grant permission to **M_Haneef** to update the **column_1** and **column_2** columns only in the table named **sample_table** as follows:

- 1 Connect to the database as a user with DBA authority, or as the owner of **sample_table**.
- 2 Type and execute the following SQL statement:


```
GRANT UPDATE column_1, column_2
ON sample_table
TO M_Haneef
```

One limitation of table and view permissions is that they apply to all the data in a table or view (except for the UPDATE permission which may be restricted). Finer tuning of user permissions can be accomplished by creating procedures that carry out actions on tables, and then granting users the permission to execute the procedure.

Granting user permissions on tables in Sybase Central

One way to grant a user permissions on a table in Sybase Central is as follows:

- 1 Connect to the database.
- 2 Double-click the Tables folder for that database, to display the tables in the left panel.
- 3 Click the Users and Groups folder, and locate the user you want to grant permissions to.
- 4 Drag the user to the table for which you want to grant permissions.

 For more information, see the Sybase Central online Help.

Granting users the right to grant permissions

Each of the table and view permissions described in "Granting permissions on tables and views" on page 582 can be assigned WITH GRANT OPTION. This option gives the right to pass on the permission to other users. This feature is discussed in the context of groups in section "Permissions of groups" on page 588.

Example

You can grant permission to **M_Haneef** to delete rows from the table named **sample_table**, and the right to pass on this permission to other users, as follows:

- 1 Connect to the database as a user with DBA authority, or as the owner of **sample_table**:
- 2 Type and execute the SQL statement:

```
GRANT DELETE ON sample_table
TO M_Haneef
WITH GRANT OPTION
```

Granting permissions on procedures

There is only one permission that may be granted on a procedure, and that is the EXECUTE permission to execute (or CALL) the procedure.

Permission to execute stored procedures may be granted by the DBA or by the owner of the procedure (the user ID that created the procedure).

The method for granting permissions to execute a procedure is similar to that for granting permissions on tables and views, discussed in "Granting permissions on tables and views" on page 582.

Example

You can grant **M_Haneef** permission to execute a procedure named **my_procedure**, as follows:

- 1 Connect to the database as a user with DBA authority or as owner of **my_procedure** procedure.
- 2 Execute the SQL statement:

```
GRANT EXECUTE
ON my_procedure
TO M_Haneef
```

Execution permissions of procedures


Procedures execute with the permissions of their owner. Any procedure that updates information on a table will execute successfully only if the owner of the procedure has UPDATE permissions on the table.

As long as the procedure owner does have the proper permissions, the procedure will execute successfully when called by any user assigned permission to execute it, whether or not they have permissions on the underlying table. You can use procedures to allow users to carry out well-defined activities on a table, without having any general permissions on the table.

Granting user permissions on procedures in Sybase Central


One way to grant a user permissions on a procedure in Sybase Central is as follows:

- 1 Connect to the database.
- 2 Click the Users and Groups folder, and locate the user you want to grant permissions to.
- 3 Right-click the user, and select Copy from the popup menu.
- 4 Locate the procedure you want to allow the user to execute, in the Stored Procedures folder.
- 5 Click the procedure, and choose Edit>Paste from the main menu to grant permissions.

 For more information, see the Sybase Central online Help.

Execution permissions of triggers

Triggers are executed by the server in response to a user action; no permissions are required for triggers to be executed. When a trigger executes, it does so with the permissions of the creator of the table with which they are associated.

 For more information on trigger permissions, see "Trigger execution permissions" on page 235.

Revoking user permissions

Any user's permissions are a combination of those that have been granted and those that have been revoked. By revoking and granting permissions, you can manage the pattern of user permissions on a database.

The REVOKE statement is the exact converse of the GRANT statement. To disallow **M_Haneef** from executing **my_procedure**, the command is:

```
REVOKE EXECUTE
ON my_procedure
FROM M_Haneef
```

This command must be issued by the DBA or by the owner of the procedure.

Permission to delete rows from **sample_table** may be revoked by issuing the command:

```
REVOKE DELETE
ON sample_table
FROM M_Haneef
```

Managing groups

DBA, RESOURCE,
and GROUP
permissions

Once you understand how to manage permissions for individual users (as described in the previous section) working with groups is straightforward. A group is identified by a user ID, just like a single user, but this user ID is granted the permission to have **members**.

When permissions on tables, views, and procedures are granted to or revoked from a group, all members of the group inherit those changes. The DBA, RESOURCE, and GROUP permissions are not inherited: they must be assigned individually to each individual user ID requiring them.

A group is simply a user ID with special permissions. Granting permissions to a group and revoking permissions from a group are done in exactly the same manner as any other user, using the commands described in "Managing individual user IDs and permissions" on page 580.

A group can also be a member of a group. A hierarchy of groups may be constructed, each inheriting permissions from its parent group.

A user ID may be granted membership in more than one group, so the user-to-group relationship is many-to-many.

The ability to create a group without a password enables you to prevent anybody from signing on using the group user ID. This security feature is discussed in "Groups without passwords" on page 589.

Creating groups

❖ **To create a group with a name and password:**

- 1 Connect to the database as a user with DBA authority.
- 2 Create the group's user ID just as you would any other user ID, using the following SQL statement:

```
GRANT CONNECT  
TO personnel  
IDENTIFIED BY group_password
```


- 3 Give the personnel user ID the permission to have members, with the following SQL statement:

```
GRANT GROUP TO personnel
```

The GROUP permission, which gives the user ID the ability to have members, is not inherited by members of a group. If this were not the case, every user ID would automatically be a group as a consequence of its membership in the special PUBLIC group.

Creating groups in
Sybase Central❖ **To create a group in Sybase Central:**

- 1 Connect to the database.
- 2 Click the Users and Groups folder for that database.
- 3 Double-click Add Group. A Wizard is displayed, which leads you through the process.

 For more information, see the Sybase Central online Help.

Granting group membership to users

Making a user a member of a group is done with the GRANT statement. Membership in a group can be granted only by the DBA. You can grant user **M_Haneef** membership in a group **personnel** as follows:


- 1 Connect to the database as a user with DBA authority, or as the group user ID **personnel**.
- 2 Grant membership in the group to **M_Haneef** with the following SQL statement:

```
GRANT MEMBERSHIP
  IN GROUP personnel
  TO M_Haneef
```

When a user is assigned membership in a group, they inherit all the permissions on tables, views, and procedures associated with that group.

Adding users to
groups in Sybase
Central❖ **To add a user to a group in Sybase Central:**

- 1 Connect to the database.
- 2 Double-click the Users and Groups folder for that database, to open it. Groups are displayed in the left panel, and both users and groups are displayed in the right panel.
- 3 In the right panel, select the users you want to add to a group, and drag them to the group.

 For more information, see the Sybase Central online Help.

Permissions of groups

Permissions may be granted to groups in exactly the same way as to any other user ID. Permissions on tables, views, and procedures are inherited by members of the group, including other groups and their members. There are some complexities to group permissions that database administrators need to keep in mind.

Notes

The DBA, RESOURCE, and GROUP permissions are not inherited by the members of a group. Even if the **personnel** user ID is granted RESOURCE permissions, the members of **personnel** do not have RESOURCE permissions.

Ownership of database objects is associated with a single user ID and is not inherited by group members. If the user ID **personnel** creates a table, then the **personnel** user ID is the owner of that table and has the authority to make any changes to the table, as well as to grant privileges concerning the table to other users. Other user IDs who are members of **personnel** are not the owners of this table, and do not have these rights. If, however, SELECT authority is explicitly granted to the **personnel** user ID by the DBA or by the **personnel** user ID itself, all group members do have select access to the table. In other words, only granted permissions are inherited.

Referring to tables owned by groups

Groups are used for finding tables and procedures in the database. For example, the query

```
SELECT * FROM SYSGROUPS
```

will always find the table SYSGROUPS, because all users belong to the PUBLIC group, and PUBLIC belongs to the SYS group which owns the SYSGROUPS table. (The SYSGROUPS table contains a list of *group_name*, *member_name* pairs representing the group memberships in your database.)

If a table named **employees** is owned by the user ID **personnel**, and if **M_Haneef** is a member of the personnel group, then **M_Haneef** can refer to the employees table simply as **employees** in SQL statements. Users who are not members of the personnel group need to use the qualified name **personnel.employees**.

Creating a group to own the tables

It is advisable that you create a group whose only purpose is to own the tables. Do not grant any permissions to this group, but make all users members of the group. This allows everyone to access the tables without qualifying names. You can then create permission groups and grant users membership in these permission groups as warranted. For an example of this, see the section "Database object names and prefixes" on page 591.

Groups without passwords

Users connected to a group's user ID have certain permissions. This user ID can grant and revoke membership in the group. Also, this user would have ownership permissions over any tables in the database created in the name of the group's user ID.

It is possible to set up a database so that all handling of groups and their database objects is done by the DBA, rather than permitting other user IDs to make changes to group membership.

This is done by disallowing connection as the group's user ID when creating the group. To do this, the GRANT CONNECT statement is typed without a password. Thus:

```
GRANT CONNECT
TO personnel
```

creates a user ID **personnel**. This user ID can be granted group permissions, and other user IDs can be granted membership in the group, inheriting any permissions that have been given to **personnel**. However, nobody can connect to the database using the **personnel** user ID, because it has no valid password.

The user ID **personnel** can be an owner of database objects, even though no user can connect to the database using this user ID. The CREATE TABLE statement, CREATE PROCEDURE statement, and CREATE VIEW statement all allow the owner of the object to be specified as a user other than that executing the statement. This assignment of ownership can be carried out only by the DBA.

Special groups

When a database is created, two groups are also automatically created. These are SYS and PUBLIC. Neither of these groups has passwords, so it is not possible to connect to the database as either SYS or as PUBLIC. The two groups serve important functions in the database.

The SYS group

The SYS group is owner of the system tables and views for the database, which contain the full description of database structure, including all database objects and all user IDs.

✍ For a description of the system tables and views, together with a description of access to the tables, see the chapters "System Tables" on page 771 of the book *Adaptive Server Anywhere Reference Manual*, and also "System Views" on page 827 of the book *Adaptive Server Anywhere Reference Manual*.

The PUBLIC group

When a database is created, the PUBLIC group is automatically created, with CONNECT permissions to the database and SELECT permission on the system tables.

The PUBLIC group is a member of the SYS group, and has read access for some of the system tables and views, so that any user of the database can find out information about the database schema. If you wish to restrict this access, you can REVOKE PUBLIC's membership in the SYS group.

Any new user ID is automatically a member of the PUBLIC group and inherits any permissions specifically granted to that group by the DBA. You can also REVOKE membership in PUBLIC for users if you wish.

Database object names and prefixes

The name of every database object is an identifier. The rules for valid identifiers are described in "Statement elements" on page 180 of the book *Adaptive Server Anywhere Reference Manual*.

In queries and sample SQL statements throughout this guide, database objects from the sample database are generally referred to using their simple name. For example:

```
SELECT *
FROM employee
```

Tables, procedures, and views all have an owner. The owner of the tables in the sample database is the user ID **DBA**. In some circumstances, you must prefix the object name with the owner user ID, as in the following statement.

```
SELECT *
FROM "DBA".employee
```

The **employee** table reference is said to be **qualified**. In other circumstances it is sufficient to give the object name. This section describes when you need to use the owner prefix to identify tables, view and procedures, and when you do not.

When referring to a database object, a prefix is required unless:

- ◆ You are the owner of the database object.
- ◆ The database object is owned by a group ID of which you are a member.

Example

Consider the following example of a corporate database. All the tables are created by the user ID **company**. This user ID is used by the database administrator and is therefore given DBA authority.

```
GRANT CONNECT TO company
IDENTIFIED BY secret;
GRANT DBA TO company;
```

The tables in the database are created by the **company** user ID.

```
CONNECT USER company IDENTIFIED BY secret;
CREATE TABLE company.Customers ( ... );
CREATE TABLE company.Products ( ... );
CREATE TABLE company.Orders ( ... );
CREATE TABLE company.Invoices ( ... );
CREATE TABLE company.Employees ( ... );
CREATE TABLE company.Salaries ( ... );
```

Not everybody in the company should have access to all information. Consider two user IDs in the sales department, Joe and Sally, who should have access to the **Customers**, **Products** and **Orders** tables. To do this, you create a **Sales** group.

```
GRANT CONNECT TO Sally IDENTIFIED BY xxxxxx;  
GRANT CONNECT TO Joe IDENTIFIED BY xxxxxx;  
GRANT CONNECT TO Sales IDENTIFIED BY xxxxxx;  
GRANT GROUP TO Sales;  
GRANT ALL ON Customers TO Sales;  
GRANT ALL ON Orders TO Sales;  
GRANT SELECT ON Products TO Sales;  
GRANT MEMBERSHIP IN GROUP Sales TO Sally;  
GRANT MEMBERSHIP IN GROUP Sales TO Joe;
```

Now Joe and Sally have permission to use these tables, but they still have to qualify their table references because the table owner is **company**, and Sally and Joe are not members of the **company** group:

```
SELECT *  
FROM company.customers
```

To rectify the situation, make the **Sales** group a member of the **company** group.

```
GRANT GROUP TO company;  
GRANT MEMBERSHIP IN GROUP company TO Sales;
```

Now Joe and Sally, being members of the **Sales** group, are indirectly members of the **company** group, and can reference their tables without qualifiers. The following command will now work:

```
SELECT *  
FROM Customers
```

Note

Joe and Sally do not have any extra permissions because of their membership in the **company** group. The **company group** has not been explicitly granted any table permissions. (The **company** user ID has implicit permission to look at tables like **Salaries** because it created the tables and has DBA authority.) Thus, Joe and Sally still get an error executing either of these commands:

```
SELECT *  
FROM Salaries;  
  
SELECT *  
FROM company.Salaries
```


In either case, Joe and Sally do not have permission to look at the **Salaries** table.

Using views and procedures for extra security

For databases that require a high level of security, defining permissions directly on tables has limitations. Any permission granted to a user on a table applies to the whole table. There are many cases when users' permissions need to be shaped more precisely than on a table-by-table basis. For example:

- ◆ It is not desirable to give access to personal or sensitive information stored in an employee table to users who need access to other parts of the table.
- ◆ You may wish to give sales representatives update permissions on a table containing descriptions of their sales calls, but limit such permissions to their own calls.

In these cases, you can use views and stored procedures to tailor permissions to suit the needs of your organization. This section describes some of the uses of views and procedures for permission management.

 For information on how to create views, see "Working with views" on page 77.

Using views for tailored security

Views are computed tables that contain a selection of rows and columns from base tables. Views are useful for security when it is appropriate to give a user access to just one portion of a table. The portion can be defined in terms of rows or in terms of columns. For example, you may wish to disallow a group of users from seeing the salary column of an employee table, or you may wish to limit a user to see only the rows of a table that they have created.

Example

The Sales manager needs access to information in the database concerning employees in the department. However, there is no reason for the manager to have access to information about employees in other departments.

This example describes how to create a user ID for the sales manager, create views that provides the information she needs, and grants the appropriate permissions to the sales manager user ID.

- 1 Create the new user ID using the GRANT statement, from a user ID with DBA authority. Enter the following:

```
CONNECT "dba"
IDENTIFIED by sql ;
```

```
GRANT CONNECT
TO SalesManager
IDENTIFIED BY sales
```

- 2 Define a view which only looks at sales employees as follows:

```
CREATE VIEW emp_sales AS
  SELECT emp_id, emp_fname, emp_lname
  FROM "dba".employee
  WHERE dept_id = 200
```

The table should therefore be identified as **dba.employee**, with the owner of the table explicitly identified, for the **SalesManager** user ID to be able to use the view. Otherwise, when SalesManager uses the view, the SELECT statement refers to a table that user ID does not recognize.

- 3 Give **SalesManager** permission to look at the view:

```
GRANT SELECT
ON emp_sales
TO SalesManager
```

Exactly the same command is used to grant permission on a view as to grant permission on a table.

Example 2

The next example creates a view which allows the Sales Manager to look at a summary of sales orders. This view requires information from more than one table for its definition:

- 1 Create the view.

```
CREATE VIEW order_summary AS
  SELECT order_date, region, sales_rep, company_name
  FROM "dba".sales_order
  KEY JOIN "dba".customer
```

- 2 Grant permission for the Sales Manager to examine this view.

```
GRANT SELECT
ON order_summary
TO SalesManager
```

- 3 To check that the process has worked properly, connect to the **SalesManager** user ID and look at the views you have created:

```
CONNECT SalesManager
IDENTIFIED BY sales ;

SELECT *
FROM "dba".emp_sales ;

SELECT *
FROM "dba".order_summary ;
```

No permissions have been granted to the Sales Manager to look at the underlying tables. The following commands produce permission errors.

```
SELECT * FROM "dba".employee ;  
SELECT * FROM "dba".sales_order
```

**Other permissions
on views**

The previous example shows how to use views to tailor SELECT permissions. INSERT, DELETE, and UPDATE permissions can be granted on views in the same way.

↪ For information on allowing data modification on views, see "Using views" on page 78.

Using procedures for tailored security

While views restrict access on the basis of data, procedures restrict the actions a user may take. As described in "Granting permissions on procedures" on page 583, a user may have EXECUTE permission on a procedure without having any permissions on the table or tables on which the procedure acts.

Strict security

For strict security, you can disallow all access to the underlying tables, and grant permissions to users or groups of users to execute certain stored procedures. With this approach, the manner in which data in the database can be modified is strictly defined.

How user permissions are assessed

Groups do introduce complexities in the permissions of individual users. Suppose user **M_Haneef** has been granted SELECT and UPDATE permissions on a specific table individually, but is also a member of two groups. Suppose one of these groups has no access to the table at all, and one has only SELECT access. What are the permissions in effect for this user?

Adaptive Server Anywhere decides whether a user ID has permission to carry out a specific action in the following manner:

- 1 If the user ID has DBA permissions, the user ID can carry out any action in the database.
- 2 Otherwise, permission depends on the permissions assigned to the individual user. If the user ID has been granted permission to carry out the action, then the action is allowed to proceed.
- 3 If no individual settings have been made for that user, permission depends on the permissions of each of the groups of which the user is a member. If any of these groups has permission to carry out the action, the user ID has permission by virtue of membership in that group, and the action is allowed to proceed.

This approach minimizes problems associated with the order in which permissions are set.

Managing the resources connections use

Building a set of users and groups allows you to manage permissions on a database. Another aspect of database security and management is to limit the resources an individual user can use.

For example, you may wish to prevent a single connection from taking too much of the available memory or CPU resources, in order to avoid a connection from slowing down other users of the database.

Adaptive Server Anywhere provides a set of database options that the DBA can use to control resources. These options are called **resource governors**.

Setting options

You can set database options using the SET OPTION statement, which has the following syntax:

```
SET [ TEMPORARY ] OPTION
... [ userid. | PUBLIC. ]option-name = [ option-value ]
```

For reference information about options, see "Database Options" on page 127 of the book *Adaptive Server Anywhere Reference Manual*. For information on the SET OPTION statement, see "SET OPTION statement" on page 553 of the book *Adaptive Server Anywhere Reference Manual*.

Resources that can be managed

The following options can be used to manage resources:

- ◆ **JAVA_HEAP_SIZE** Sets the maximum size (in bytes) of that part of the memory that is allocated to Java applications on a per connection basis.
- ◆ **MAX_CURSOR_COUNT** Limits the number of cursors for a connection.
- ◆ **MAX_STATEMENT_COUNT** Limits the number of prepared statements for a connection.
- ◆ **BACKGROUND_PRIORITY** Limits the impact requests on the current connection have on the performance of other connections

Database option settings are not inherited through the group structure.

Users and permissions in the system tables

Information about the current users of a database and about their permissions is stored in the database system tables and system views.

🔗 For a description of each of these tables, see "System Tables" on page 771 of the book *Adaptive Server Anywhere Reference Manual*.

The system tables are owned by the special user ID **SYS**. It is not possible to connect to the **SYS** user ID.

The **DBA** has **SELECT** access to all system tables, just as to any other tables in the database. The access of other users to some of the tables is limited. For example, only the **DBA** has access to the **SYS.SYSUSERPERM** table, which contains all information about the permissions of users of the database, as well as the passwords of each user ID. However, **SYS.SYSUSERPERMS** is a view containing all information in **SYS.SYSUSERPERM** except for the password, and by default all users have **SELECT** access to this view. All permissions and group memberships set up in a new database for **SYS**, **PUBLIC**, and **DBA** can be fully modified.

The following table summarizes the system tables containing information about user IDs, groups, and permissions. All tables and views are owned by user ID **SYS**, and so their qualified names are **SYS.SYSUSERPERM** and so on.

Appropriate **SELECT** queries on these tables generates all the user ID and permission information stored in the database.

Table	Default	Contents
SYSUSERPERM	DBA only	Database-level permissions and password for each user ID
SYSGROUP	PUBLIC	One row for each member of each group
SYSTABLEPERM	PUBLIC	All permissions on table given by the GRANT commands
SYSCOLPERM	PUBLIC	All columns with UPDATE permission given by the GRANT command
SYSDDUMMY	PUBLIC	Dummy table, can be used to find the current user ID
SYSPROCPerm	PUBLIC	Each row holds one user granted permission to use one procedure

The following table summarizes the system views containing information about user IDs, groups, and permissions

Views	Default	Contents
SYSUSERAUTH	DBA only	All information in SYSUSERPERM except for user numbers
SYSUSERPERMS	PUBLIC	All information in SYSUSERPERM except for passwords
SYSUSERLIST	PUBLIC	All information in SYSUSERAUTH except for passwords
SYSGROUPS	PUBLIC	Information from SYSGROUP in a more readable format
SYSTABAUTH	PUBLIC	Information from SYSTABLEPERM in a more readable format
SYSCOLAUTH	PUBLIC	Information from SYSCOLPERM in a more readable format
SYSPROCAUTH	PUBLIC	Information from SYSROCPERM in a more readable format

In addition to these, there are tables and views that contain information about each object in the database.

