

[3 4](#)

Chapter 34

User and Security Management

In this chapter, you'll learn how to manage users and security in a Microsoft SQL Server 2000 environment. Along with backup and recovery planning, sizing, and space management, security management is one of the most common jobs of the DBA. It is also one of the most important tasks that the DBA must perform. If the security of the system is breached, data could be lost or corrupted.

This chapter covers a number of topics related to user and security management. You'll learn how to create and manage user logins and about the modes of authentication; in addition, you'll learn about SQL Server user IDs. A *user login* is used to authenticate access to SQL Server. The user login can be authenticated through Microsoft Windows NT or Microsoft Windows 2000 or through SQL Server. A *user ID* is used to assign a user permissions to access specific objects within the individual databases. IDs are associated with user logins and might or might not consist of the same name, as you'll see later this chapter. You'll also learn about the types of permissions that can be assigned in SQL Server and to use them. And you'll learn how to use roles to more easily manage users. Finally you will learn important feature in SQL Server 2000 called security account delegation. By the end of this chapter, have the knowledge necessary to manage user logins and security.

[3 4](#)

Creating and Administering User Logins

Let's begin our examination of user and security management by looking at user logins. In this section, we'll first look at why logins are important and at the authentication methods that can be used to logins. We'll then look at three methods for creating logins: using SQL Server Enterprise Manager, Transact-SQL (T-SQL), and using the Create Login Wizard. And finally, we'll see how to use Enterprise Manager and T-SQL to create new user accounts.

Why Create User Logins?

User logins enable you to protect your data from being intentionally and unintentionally modified by unauthorized users. With user logins, SQL Server is able to identify individual authorized users. Each login is assigned a unique name and password. Each user should be assigned his or her own SQL Server login account. Without user logins, all connections to SQL Server would use the same identifier, which means that you could not create different levels of security based on who is accessing the database.

With user logins, you can create multiple levels of security by granting different login accounts different permissions to access objects and to perform functions. You can also encrypt certain database objects, such as stored procedures and views, to hide their definitions from unauthorized users. And with user logins, you can allow only some users to insert and update information in certain database tables, and grant read-only access to the tables to the general user community.

To see how data access can be limited with user logins, let's return to the example, first presented in [Chapter 18](#), of using a view to restrict access to sensitive data. Suppose you have an *Employee* table that

contains information about employees, including each employee's name, phone number, office number, grade level, salary, bonus, and so on. To prevent certain users from accessing the confidential data in the table, you would first create a view that contains only nonsensitive information, such as employee phone numbers, and office numbers. Then, by implementing user logins, you can restrict access to the underlying table while allowing all users to access the view. However, if you did not take advantage of user logins, any user could access either the view or the table, thus defeating the purpose of using the view.

Authentication Modes

Two authentication modes are available for accessing SQL Server: Microsoft Windows Authentication and Mixed Mode Authentication. With Windows Authentication, the operating system is responsible for authenticating the user. SQL Server then uses this operating system authentication to determine which user permissions to apply. With Mixed Mode Authentication, both Windows NT/2000 and SQL Server are responsible for authenticating the user. To access SQL Server, you always have to first log on to a Windows NT/2000 account, so when choosing an authentication mode, you must decide simply whether you want to use SQL Server Authentication in addition to Windows Authentication. Let's look at each of these authentication modes in more detail. Later in this section, you'll learn how to implement these modes.

Windows Authentication

As mentioned, with Windows Authentication, SQL Server relies on Windows NT/2000 to provide the login security. When a user logs on to Windows NT/2000, the user's account identity is validated. SQL Server verifies that the user was validated by Windows NT/2000 and allows access based on that authentication. SQL Server integrates its login security process with the Windows login security process to provide these services. Network security attributes are validated through a sophisticated encryption process provided by Windows NT/2000. Because the SQL Server and Windows login security processes are integrated when this mode is used, once you are authenticated by the operating system, no further authentication methods are required for you to access SQL Server. The only password you need to in order to log on to SQL Server is your Windows NT/2000 password.

Windows Authentication is considered a better security method than Mixed Mode Authentication of the additional security features it provides. These features include secure validation and encryption of passwords, auditing, password expiration, minimum password length, and automated account lockout after a certain number of unsuccessful logon attempts.

Mixed Mode Authentication

With Mixed Mode Authentication, users can access SQL Server by using either Windows or SQL Server Authentication. When Mixed Mode Authentication is used, if a connection is made from an insecure system, SQL Server authenticates the login by verifying whether a SQL Server login account has been set up for the user requesting access. SQL Server performs this account authentication by the name and password provided by the user attempting to connect to SQL Server with login account information stored in the database. If a login account has not been set up for the user or if the user does not provide the correct name and password, SQL Server access is denied.

Windows Authentication mode is not available when you are running SQL Server on Windows 95/98, you must use SQL Server Authentication (by using Mixed Mode Authentication) on those platforms. In addition, Web applications require SQL Server Authentication (through Microsoft Internet Information

Server) because users of these applications will most likely not be within the same domain as the server and thus they can't rely on Windows security. Other applications that require database access might require SQL Server Authentication as well: some application developers prefer to use SQL Server for their applications because it simplifies the security of their applications. When applications use SQL Server security (within a trusted network), application developers do not have to provide security authentication within the application itself, which simplifies their job.

Setting Up the Authentication Mode

To set up the authentication mode, follow these steps:

1. Open the Enterprise Manager window. In the left-hand pane, right-click the name of the server hosts the database for which you want to set the authentication mode, and choose Properties from the shortcut menu to display the SQL Server Properties window. Click the Security tab, shown in Figure 34-1.

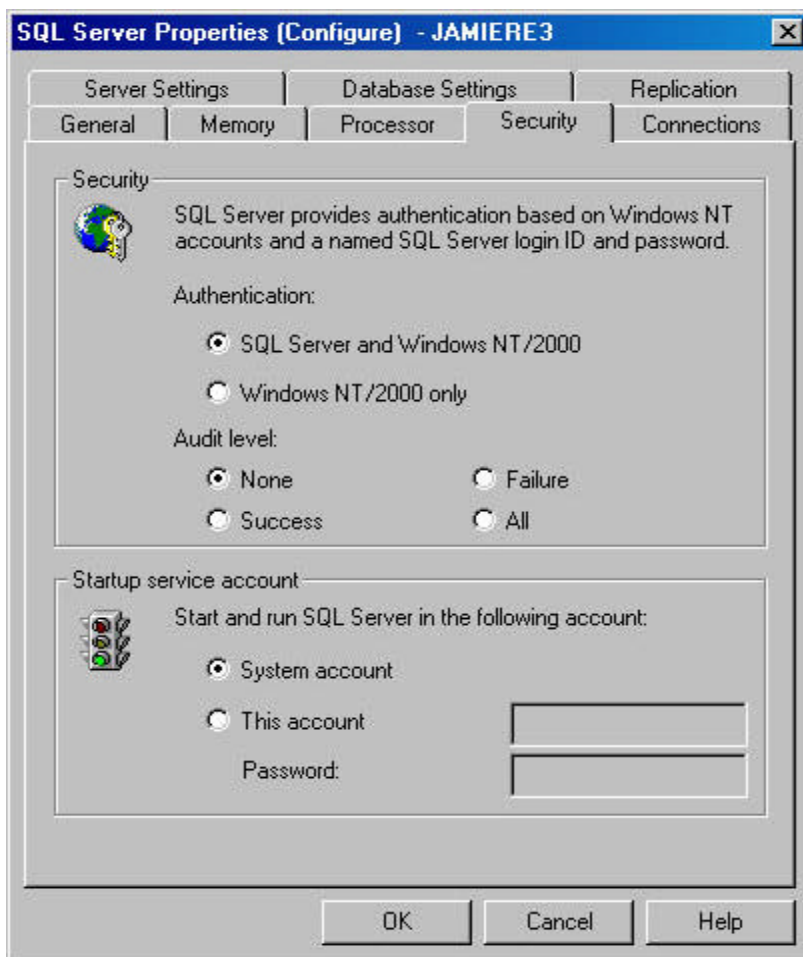


Figure 34-1. The Security tab of the SQL Server Properties window.

2. On this tab, you can choose both the security method and the startup service account. In the area, specify whether Windows NT/2000 and SQL Server (Mixed Mode) Authentication should be used or whether just Windows NT/2000 Authentication should be used. You can also specify the login audit level. This setting determines what, if any, type of login auditing is performed. The

level you choose will depend on your security requirements. The following four levels are

- **None** Performs no login auditing. This is the default setting.
- **Success** Logs all successful login attempts.
- **Failure** Logs all failed login attempts.
- **All** Logs all login attempts.

NOTE

The auditing level is a database property. As such, the same auditing level will all logins.

3. In the Startup Service Account area, specify which Windows NT account should be used when the SQL Server service is started up. You can either use the built-in local system account or specify account, such as Administrator, and a password. Click OK to accept your settings.

Logins vs. Users

In the next couple of sections, you'll learn how to create logins and users. But before you start this you need to understand what logins and users are. This section provides brief definitions of these two terms.

As we've seen, a Windows NT/2000 user account might be needed to connect to the database. SQL Authentication might also be involved. Whether you are using Windows NT/2000 Authentication or Mixed Mode Authentication, the account that you use to connect to SQL Server is known as the SQL Server *login*. In addition to the SQL Server login, each database has a set of user pseudo-accounts assigned to it. These pseudoaccounts provide aliases to SQL Server login accounts. For example, in Northwind database, you might have a user named manager that is associated with the SQL Server login, and the pubs database might have a user named manager that is associated with the SQL Server login account. By default, a SQL Server login account does not have a database user ID associated thus, it has no permissions.

Creating SQL Server Logins

You can perform most SQL Server administrative tasks by using one of several methods, and the task of creating user logins is no exception. As mentioned earlier, you can create a login in one of three ways: using Enterprise Manager, using T-SQL, or using the Create Login Wizard. In this section, you'll learn how to create SQL Server logins by using each of these methods.

Using Enterprise Manager to Create SQL Server Logins

To create SQL Server logins by using Enterprise Manager, follow these steps:

1. In the left-hand pane of the Enterprise Manager window, expand the server group, expand the server, and then expand the Security folder. Right-click Logins, and choose New Login from the shortcut menu to display the SQL Server Login Properties window, shown in Figure 34-2. On the General tab, enter a SQL Server login name in the Name text box. If you are using Windows Authentication, this name must be a valid Windows NT or Windows 2000 account name. Next specify the Windows NT or Windows 2000 domain in the Domain text box. In the Default area,

specify the default database and language that the user will use. In the Authentication area, specify whether a Windows NT or Windows 2000 account will be used or whether SQL Server Authentication will be used. If you choose SQL Server Authentication, Mixed Mode will be used.

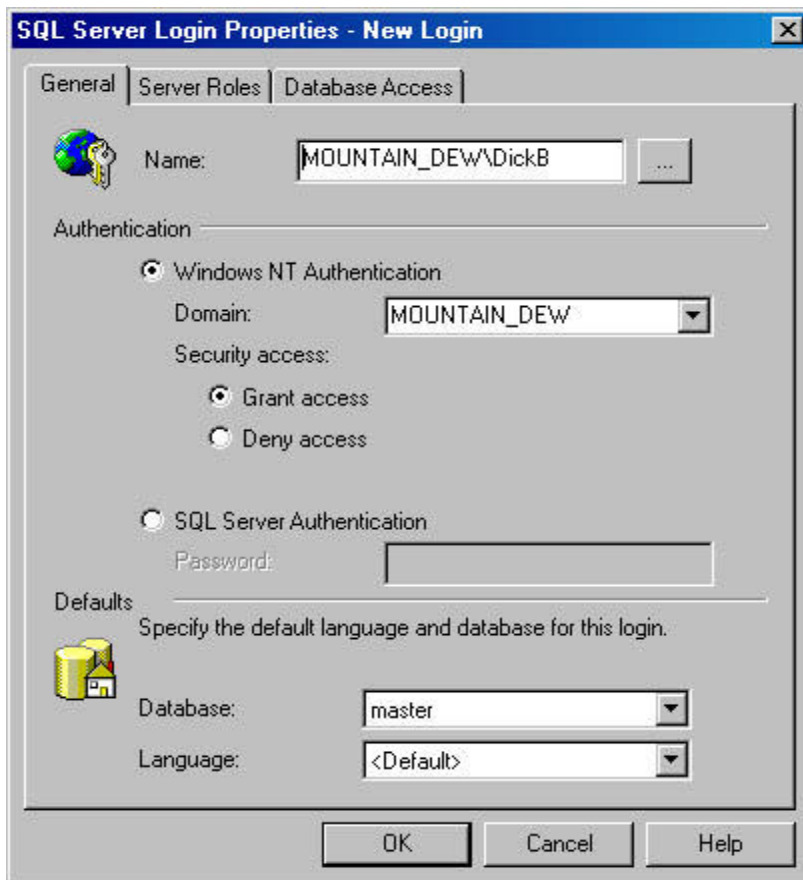


Figure 34-2. *The General tab of the SQL Server Login Properties window.*

2. Click the Server Roles tab, shown in Figure 34-3. On this tab, you can specify which server roles the new login will be able to choose by selecting roles from the list of roles available to the user. Clicking Properties allows you to view and modify the role that you selected. (Roles are explained in the section "[Administering Database Roles](#)" later in this chapter.)

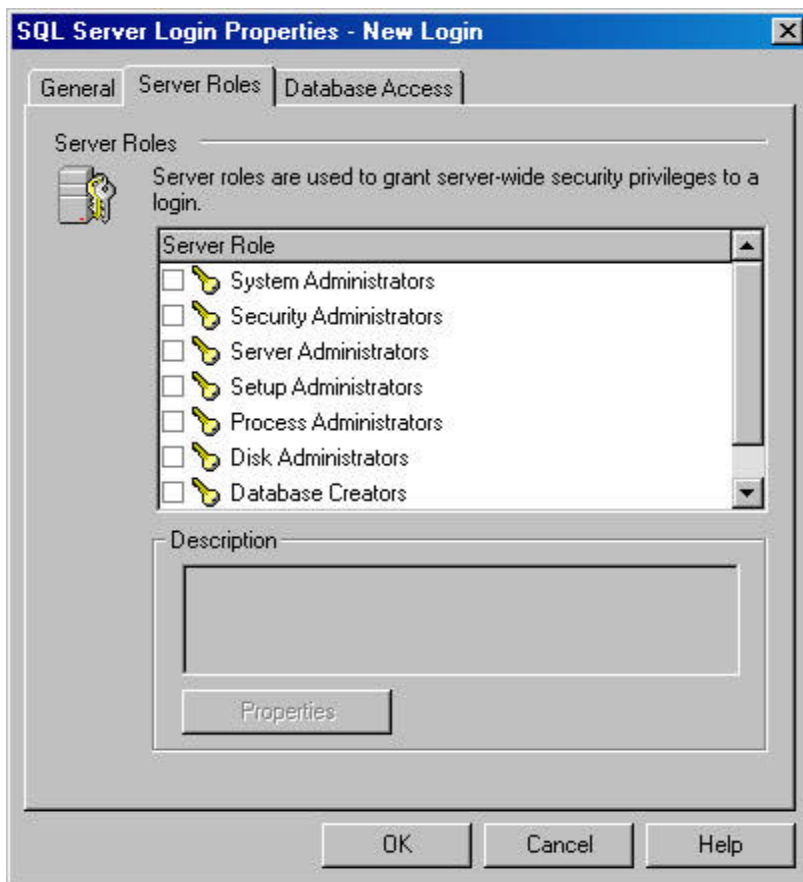


Figure 34-3. *The Server Roles tab of the SQL Server Login Properties window.*

3. Click the Database Access tab, shown in Figure 34-4. This tab lets you specify which databases user has permission to access. (Database permissions are covered in the section "[Administering Database Permissions](#)" later in this chapter.) You can select multiple databases and the roles that available to those databases. Clicking Properties allows you to view and manage the database properties.

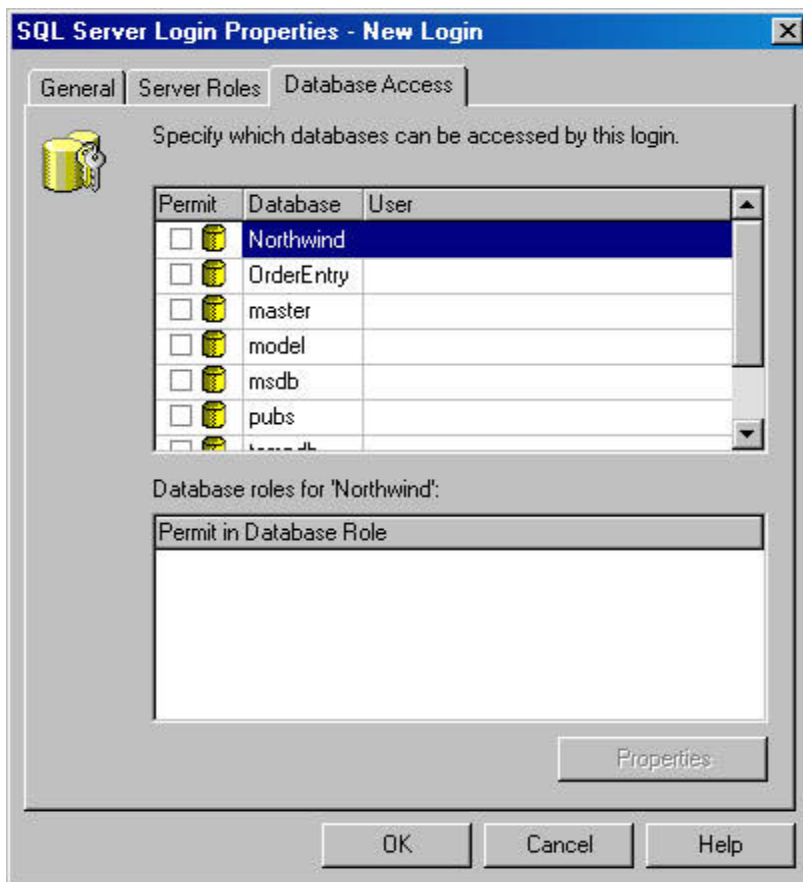


Figure 34-4. The Database Access tab of the SQL Server Login Properties window.

- When you have finished specifying options, save the login by clicking OK. To see the new login listed with the other logins, click the Logins folder in Enterprise Manager. The logins will be in the right-hand pane.

Using T-SQL to Create Logins

To create logins by using T-SQL, you use the *sp_addlogin* stored procedure or the *sp_grantlogin* stored procedure. The *sp_addlogin* stored procedure can add only a SQL Server_authenticated user to a SQL Server database. The *sp_grantlogin* stored procedure can add a Windows NT/2000_authenticated user.

The *sp_addlogin* stored procedure has the following syntax:

```
sp_addlogin [ @ loginname = ] ' login'
           [ , [ @ passwd = ] ' password' ]
           [ , [ @ defdb = ] ' database' ]
           [ , [ @ deflanguage = ] ' language' ]
           [ , [ @ sid = ] ' sid' ]
           [ , [ @ encryptopt = ] ' encryption_option' ]
```

The optional parameters are as follows:

- password** Specifies the SQL Server login password. The default is *NULL*.
- database** Specifies the default database of the login. The default is master.

- **language** Specifies the default language of the login. The default is the current SQL Server language setting.
- **sid** Specifies the security identifier, which is a unique number. If you do not specify a value, one will be generated for you. The *sid* parameter is not generally used by users, but administrators use *sid* in a number of situations. When the DBA performs troubleshooting tasks, *sid* might be needed to determine which login is being checked. The *sid* parameter is the internal identifier for the
- **encryption_option** Specifies whether the password will be encrypted in the system tables. The default value is *NULL*, which means that the password will be encrypted. Specifying *skip_encryption* means that the password will not be encrypted. If you specify the password that was encrypted already by an earlier version of SQL Server will not be again. This setting should be changed from the default only if you want to avoid encrypting the password in the system tables.

A simple example of adding a login is shown here:

```
EXEC sp_addlogin 'PatB'
```

Remember to use the EXEC keyword before the stored-procedure name.

A more complex example of adding a login is shown here:

```
sp_addlogin 'SharonR', 'mypassword', 'Northwind', 'us_english'
```

This command creates a user named SharonR with the password "mypassword." The default database is Northwind, and the default language is U.S. English. In general, you should let SQL Server create a security identifier for you instead of creating your own.

The *sp_grantlogin* stored procedure has the following syntax:

```
sp_grantlogin 'login_name'
```

An example of using the *sp_grantlogin* stored procedure is shown here:

```
EXEC sp_grantlogin 'MOUNTAIN_DEW\DickB'
```

"DickB" is the Windows NT or Windows 2000 account name. "MOUNTAIN_DEW" is the system

After you add these logins, you can view them in Enterprise Manager. To do so, click the Logins folder the left pane.

Using the Create Login Wizard

To create a SQL Server login by using the Create Login Wizard, follow these steps:

1. In Enterprise Manager, expand a server group, and click a server name. From the Tools menu, choose Wizards. In the Select Wizard dialog box that appears, expand the Database folder, click Create Login Wizard (shown in Figure 34-5), and then click OK. The Create Login Wizard welcome screen appears, as shown in Figure 34-6.

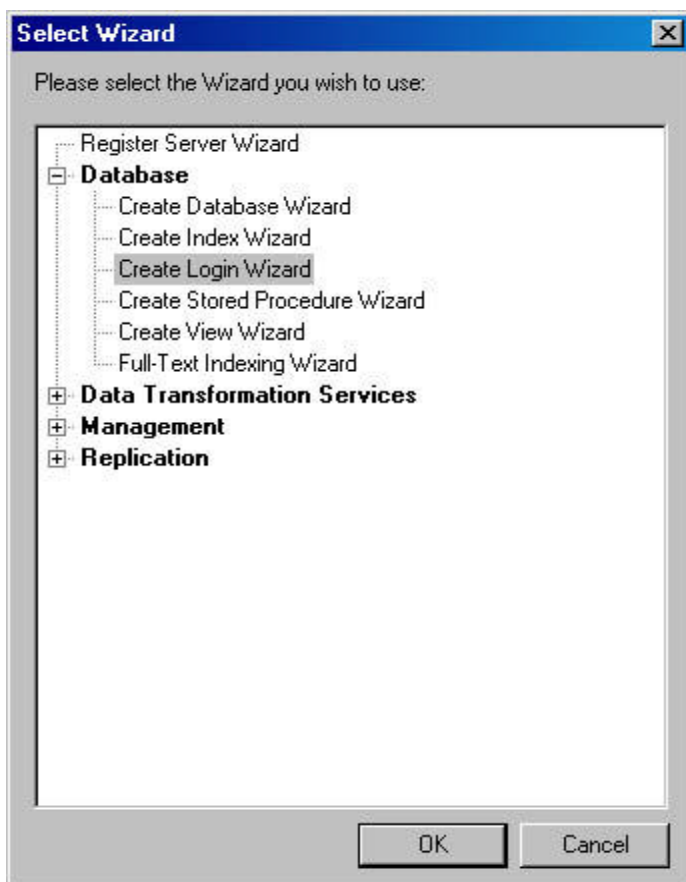


Figure 34-5. *A Select Wizard screen.*



Figure 34-6. *The Create Login Wizard welcome screen.*

2. Click Next to display the Select Authentication Mode For This Login screen, as shown in Figure

34-7. In this screen, you can specify whether Windows Authentication or SQL Server (Mixed Mode) Authentication should be used.

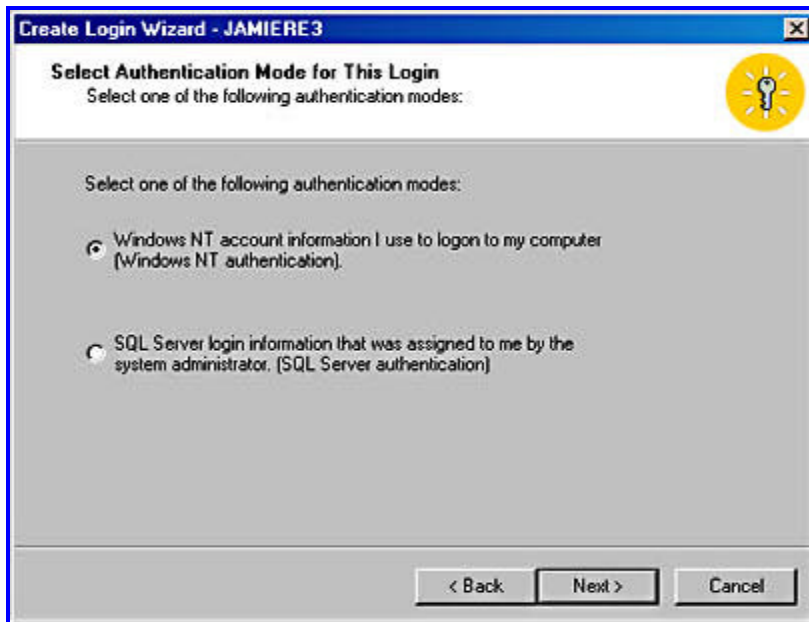


Figure 34-7. *The Select Authentication Mode For This Login screen.*

3. Click Next to display the Authentication With Windows NT screen or Authentication With SQL Server screen, depending on the authentication mode you selected in step 2. Figure 34-8 shows the latter screen. In this screen, you specify the login ID and password. If you had selected Windows NT Authentication, you would be prompted to enter a domain name and user account name for the authentication.

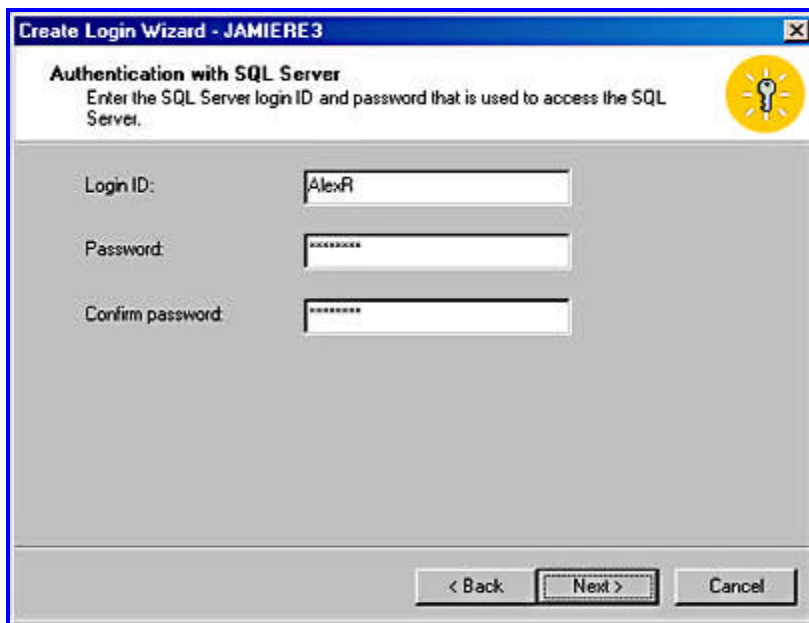


Figure 34-8. *The Authentication With SQL Server screen.*

4. Click Next to display the Grant Access To Security Roles screen, as shown in Figure 34-9. In this screen, you can select the database roles to be assigned to this login.

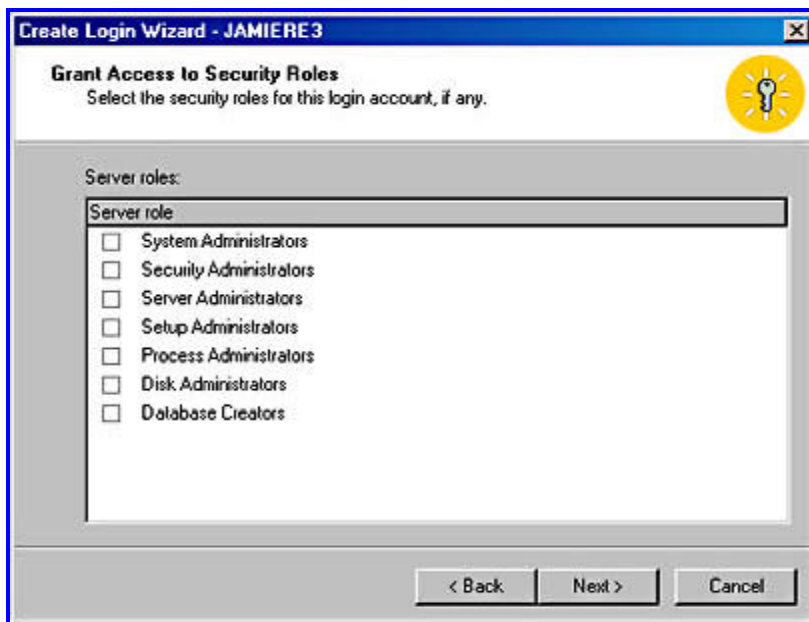


Figure 34-9. *The Grant Access To Security Roles screen.*

5. Click Next to display the Grant Access To Databases screen, shown in Figure 34-10. In this you can select the databases to which this login will have access.

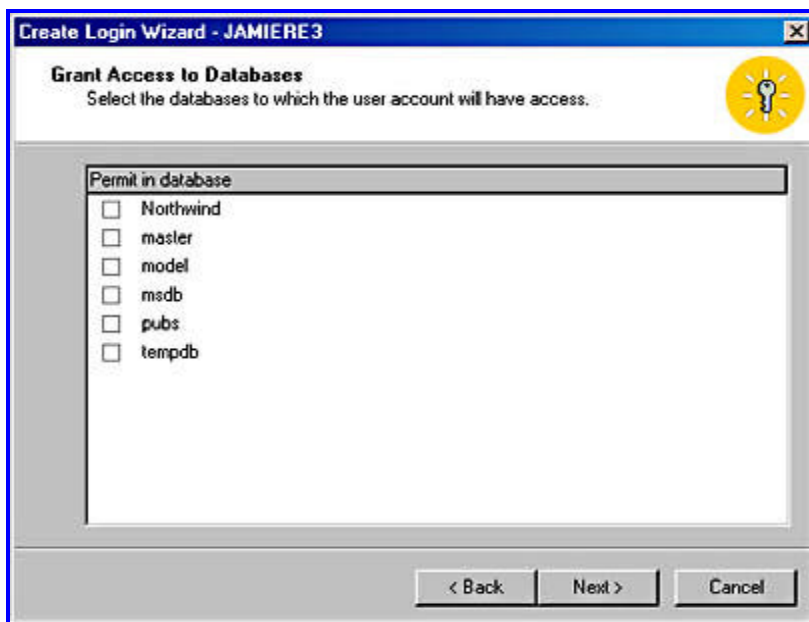


Figure 34-10. *The Grant Access To Databases screen.*

6. Click Next to display the Completing The Create Login Wizard screen, shown in Figure 34-11, where you can examine the summary information in the text box. Click Back to make any and click Finish to create the login.



Figure 34-11. *The Completing The Create Login Wizard screen.*

Creating SQL Server Users

You can create SQL Server users by using Enterprise Manager or T-SQL. (SQL Server does not include wizard to help you with this process.) In this section, you'll learn how to use both these methods to create SQL Server users. Remember, a SQL Server user is defined for a particular database and permissions assigned for that database to a specific user login. The SQL Server user ID can be thought of as a reference to the SQL Server login, but it is not required to have the same name as the login.

NOTE

To create a SQL Server user, you must already have defined the SQL Server login for that user because the user name is a reference to a SQL Server login.

Using Enterprise Manager to Create Users

Unlike SQL Server logins, which are created from within the Security folder of Enterprise Manager, SQL Server users are created from within the specific database folder in the left-hand pane of Enterprise Manager. To create users by using Enterprise Manager, follow these steps:

1. Right-click the database in which the user is to be created, point to New in the shortcut menu, and then choose Database User to display the Database User Properties window, shown in Figure 34-12. Enter a valid SQL Server login name in the Login Name drop-down list, and type a new user name in the User Name text box. Then specify the database roles that you want the new user to be a member of by selecting the appropriate check boxes in the Database Role Membership list box. As you'll see later in this chapter, by assigning permissions to these roles, you can apply the permissions to the user.
2. Click Properties to display the Database Role Properties window, as shown in Figure 34-13. In this window, you can modify a database role you have selected. This task is explained in the section "[Administering Database Roles](#)" later in this chapter.

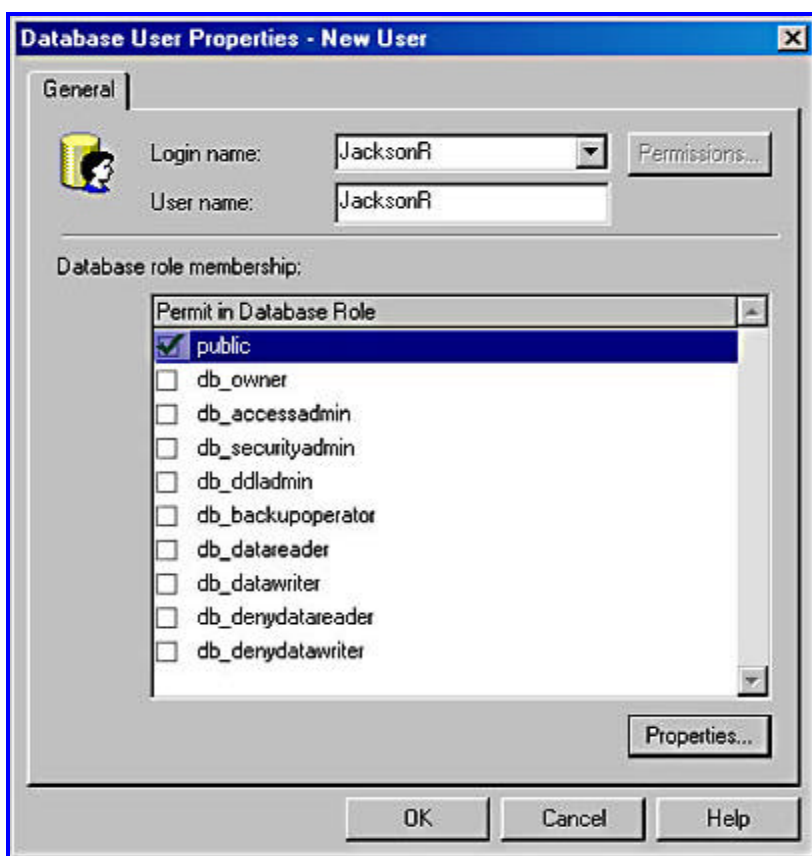


Figure 34-12. *The Database User Properties window.*

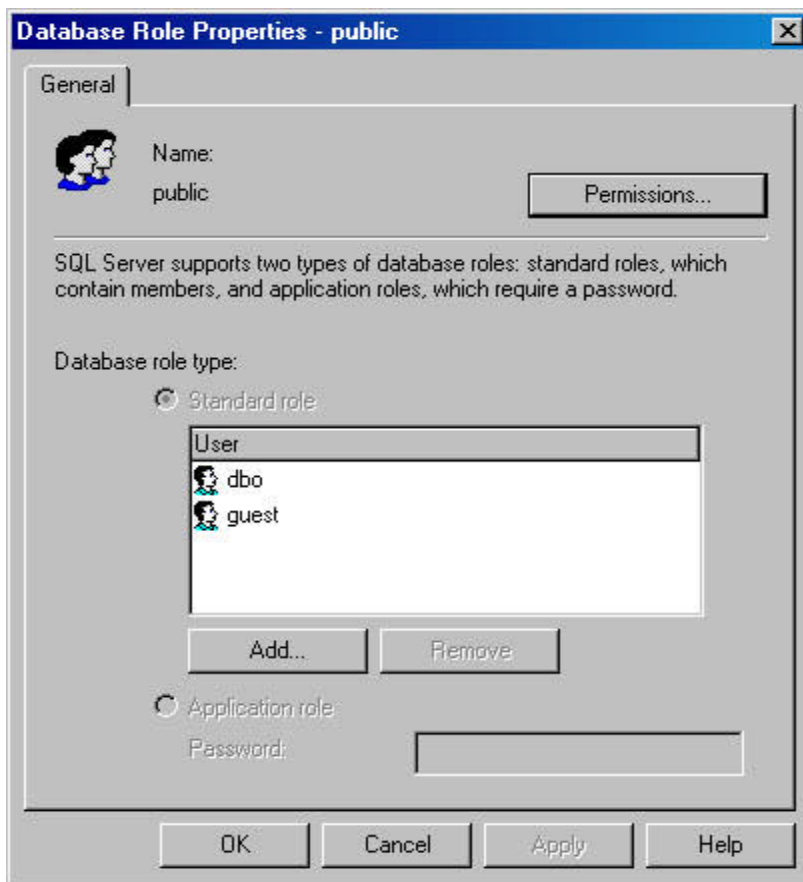


Figure 34-13. *The Database Role Properties window.*

3. When you are finished setting options, click OK twice to create the database user.

Using T-SQL to Create Users

To use T-SQL to create database users, you run the *sp_adduser* stored procedure. This stored procedure can be run from ISQL or OSQL and has the following syntax:

```
sp_adduser[ @loginame = ] 'login'
          [ , [ @name_in_db = ] 'user' ]
          [ , [ @grpname = ] 'group' ]
```

The *login* parameter is the SQL Server login account name and must be provided. The *user* variable is new user name, and *group* is the group or role that the new user will belong to. If a *user* value is not specified, it will be the same as the *login* parameter.

The following command creates a new database user with the name JackR and the Windows NT or Windows 2000 account FORT_WORTH\DB_User:

```
sp_adduser 'FORT_WORTH\DB_User', 'JackR'
```

"FORT_WORTH" is the system or domain name. "DB_User" is the Windows NT or Windows 2000 account name.

[34](#)

Administering Database Permissions

Permissions are used to control access to database objects and to specify which users can perform database actions. You can set both server and database permissions. *Server permissions* are used to DBAs to perform database administration tasks. *Database permissions* are used to allow or disallow access to database objects and statements. In this section, we'll look at the types of permissions and allocate them.

Server Permissions

As just mentioned, server permissions are assigned to DBAs and allow them to perform administrative tasks. These permissions are defined on the fixed server roles. User logins can be assigned the fixed roles, but these roles cannot be modified. (Server roles are explained in the section "[Using Fixed Server Roles](#)" later in this chapter.) Server permissions include SHUTDOWN, CREATE DATABASE, BACKUP DATABASE, and CHECKPOINT permissions. Server permissions are used only for authorizing DBAs to perform administrative tasks and do not need to be modified or granted to users.

Database Object Permissions

Database object permissions are a class of permissions that are granted to allow access to database Object permissions are necessary to access a table or view by using SQL statements such as SELECT, INSERT, UPDATE, and DELETE. An object permission is also needed to use the EXECUTE statement to run a stored procedure. You can use Enterprise Manager or T-SQL commands to assign object permissions.

Using Enterprise Manager to Assign Object Permissions

To use Enterprise Manager to grant database object permissions to a user, follow these steps:

1. Expand a server group, expand a server, expand the database you want to assign permissions for, and click the Users folder. The users are then listed in the right pane. Right-click a user name, and choose Properties from the shortcut menu to display the Database User Properties window, shown in Figure 34-14.

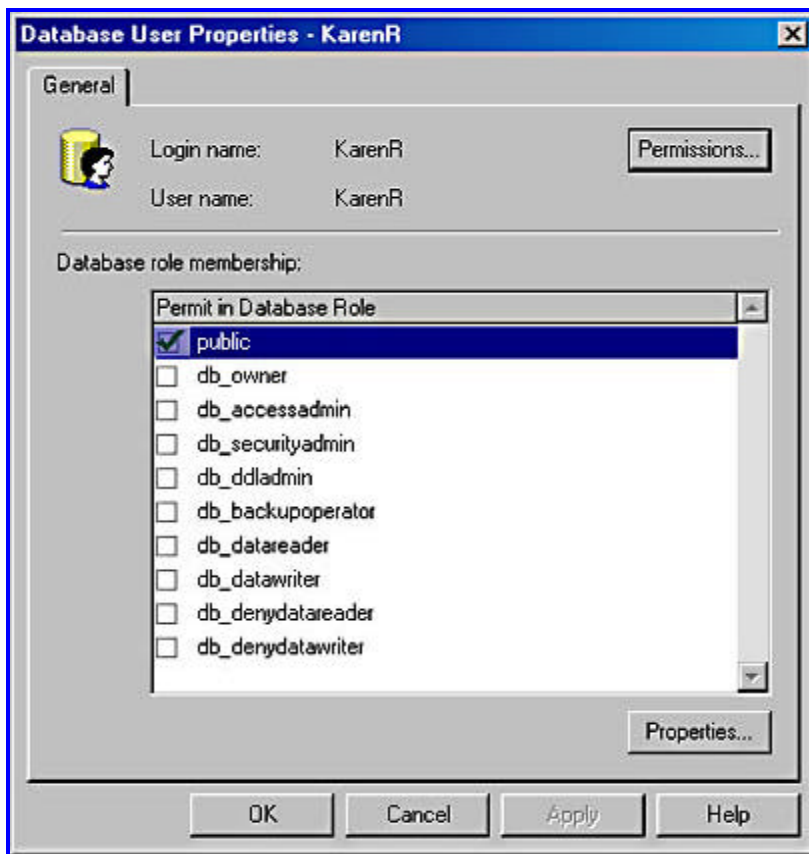


Figure 34-14. *The Database User Properties window.*

2. Click the Permissions button to display the Database User Properties window, shown in Figure 15. (To display this tab, you can also right-click the user name, point to All Tasks in the shortcut menu, and then choose Manage Permissions.) On this tab, you manage the permissions assigned this user. To assign permissions to this user to access objects within the database, select the appropriate check boxes in the SELECT, INSERT, UPDATE, DELETE, EXEC, and DRI of the list box. ("DRI" stands for "declarative referential integrity.") The objects are listed in the Object column. You can use the option buttons at the top of the window to view all objects or just those that this user already has permissions to access.

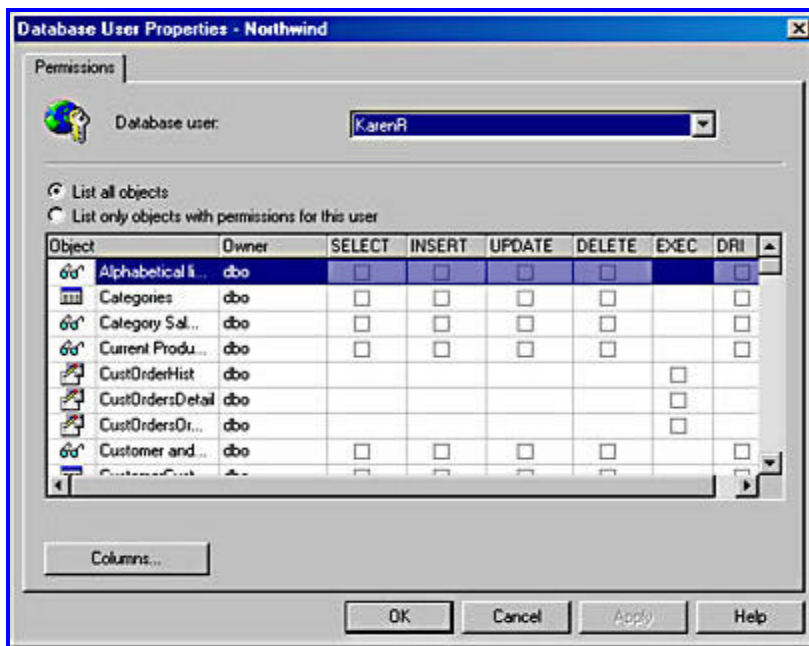


Figure 34-15. The *Permissions* tab of the *Database User Properties* window.

Using T-SQL to Assign Object Permissions

To use T-SQL to assign object permissions to a user, you run the GRANT statement. The GRANT statement has the following syntax:

```
GRANT { ALL | permission }
      [ column ON { table | view } ] |
      [ ON table(column) ] |
      [ ON view(column) ] |
      [ ON { stored_procedure | extended_procedure } ]
TO security_account
   [ WITH GRANT OPTION ]
   [ AS { group | role } ]
```

The *security_account* parameter must be one of the following account types:

- SQL Server user
- SQL Server role
- Windows NT or Windows 2000 user
- Windows NT or Windows 2000 group

Using the GRANT OPTION keyword allows the user or users specified in the statement to grant the specified permission to other users. This can be useful when you grant permissions to other DBAs. However, the GRANT option should be used with care.

The AS option specifies whose authority the GRANT statement is run under. To run the GRANT statement, a user or role must have been specifically granted authority to do so.

Here is an example of using the GRANT statement:

```
GRANT SELECT, INSERT, UPDATE
ON      Customers
TO      MaryW
WITH    GRANT OPTION
AS      Accounting
```

The *AS Accounting* option is used because the *Accounting* role has permissions to grant permissions on *Customers* table. The GRANT OPTION keyword allows MaryW to grant these permissions to other

MORE INFO

To view a list of permissions that can be specified in the GRANT statement, look up "GRANT, described (GRANT)" in the Books Online index.

Using T-SQL to Revoke Object Permissions

You can use the T-SQL REVOKE command to revoke a user's object permissions. The REVOKE statement has the following syntax:

```
REVOKE [ GRANT OPTION FOR ]
      { ALL [ PRIVILEGES ] | permission }
      [ column ON { table | view } ] |
      [ ON table(column) ] |
      [ ON view(column) ] |
      [ ON { stored_procedure | extended_procedure } ]
      { TO | FROM } security_account
      [ CASCADE ]
      [ AS { group | role } ]
```

The *security_account* parameter must be one of the following account types:

- SQL Server user
- SQL Server role
- Windows NT or Windows 2000 user
- Windows NT or Windows 2000 group

The GRANT OPTION FOR option allows you to revoke permissions you previously granted by using GRANT OPTION keyword, as well as revoke the permission. The AS option specifies whose authority the REVOKE statement is run under.

Here is an example of using the REVOKE statement:

```
REVOKE ALL
ON      Customers
FROM    MaryW
```

The REVOKE ALL statement will remove all permissions the user MaryW has on the *Customers* table.

MORE INFO

To view a list of permissions that can be specified in the REVOKE statement, look up "REVOKE, described (REVOKE)" in the Books Online index.

Database Statement Permissions

In addition to assigning database object permissions, you can assign statement permissions. Object permissions enable users to access existing objects within the database, whereas statement permissions authorize them to create database objects, including databases and tables. The statement permissions are listed here:

- **BACKUP DATABASE** Allows the user to execute the BACKUP DATABASE command
- **BACKUP LOG** Allows the user to execute the BACKUP LOG command
- **CREATE DATABASE** Allows the user to create new databases
- **CREATE DEFAULT** Allows the user to create default values that can be bound to columns
- **CREATE PROCEDURE** Allows the user to create stored procedures
- **CREATE RULE** Allows the user to create rules
- **CREATE TABLE** Allows the user to create new tables
- **CREATE VIEW** Allows the user to create new views

You can assign statement permissions by using either Enterprise Manager or T-SQL.

Using Enterprise Manager to Assign Statement Permissions

To use Enterprise Manager to grant database statement permissions to a user, follow these steps:

1. Expand a server group, expand a server, and then expand the Databases folder. Right-click the of the database you want to assign permissions for, and choose Properties from the shortcut menu display the database's Properties window, shown in Figure 34-16.

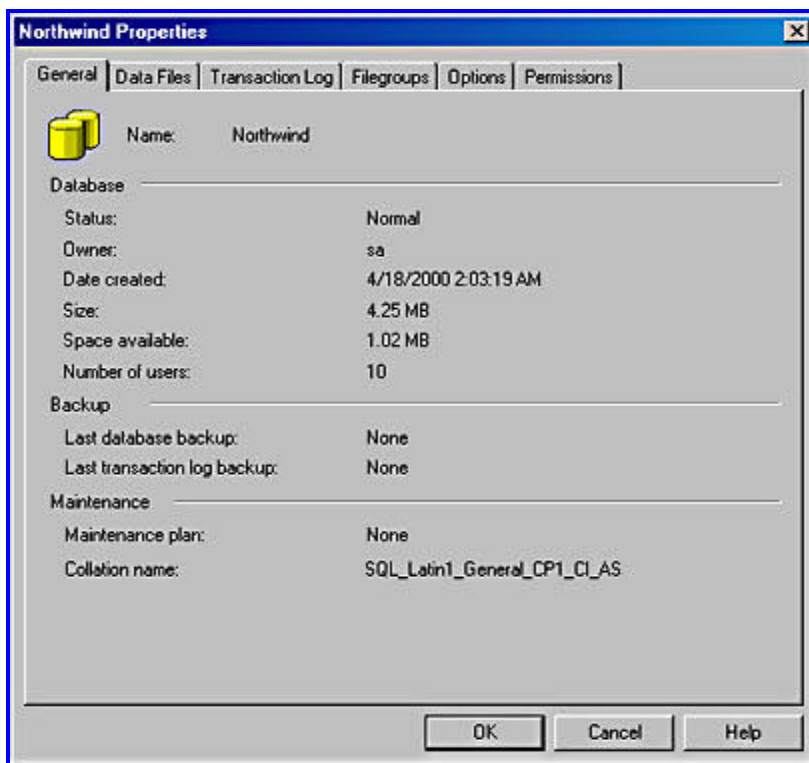


Figure 34-16. A database's Properties window.

- Click the Permissions tab, shown in Figure 34-17. Here you can assign statement permissions to users and roles that have access to this database. The columns containing check boxes define the statement permissions that can be assigned, and the User/Role column lists the users and roles that have access to this database.

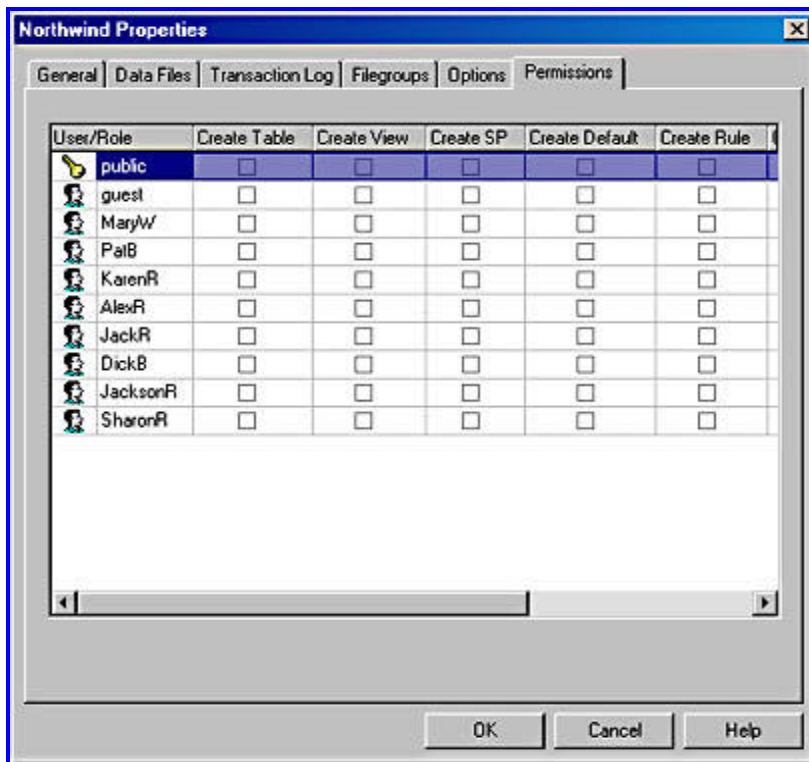


Figure 34-17. The Permissions tab of a database's Properties window.

Using T-SQL to Assign Statement Permissions

To assign statement permissions to a user by using T-SQL, you use the GRANT statement. The GRANT statement has the following syntax:

```
GRANT { ALL / statement }
TO    security_account
```

The statement permissions that can be assigned to a user are CREATE DATABASE, CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW, BACKUP DATABASE, and BACKUP LOG, as described earlier. For example, to add the CREATE DATABASE and CREATE TABLE statement permissions to the user account JackR, use the following command:

```
GRANT CREATE DATABASE, CREATE TABLE
TO    'JackR'
```

As you can see, adding statement permissions to a user account is not a complex process.

Using T-SQL to Revoke Statement Permissions

You can use the T-SQL statement **REVOKE** to remove statement permissions from a user account. The **REVOKE** statement has the following syntax:

```
REVOKE { ALL / statement }  
FROM security_account
```

For example, to remove just the **CREATE DATABASE** statement permissions from the user account **JackR**, use the following command:

```
REVOKE CREATE DATABASE  
FROM 'JackR'
```

As you can see, removing statement permissions from a user account is also not a complex process.

[3.4](#)

Administering Database Roles

You can simplify the task of managing many permissions to many users by using database roles. roles are designed to allow groups of users to receive the same database permissions without your to assign these permissions individually. Rather than assigning individual permissions to individual you can create a role that represents the permissions used by a group of users and then assign it to the group.

Typically, roles are set up for particular workgroups, job classes, or job tasks. In this manner, new users can be members of one or more database roles based on the jobs they will be performing. For example, roles might be defined for job classes such as accounts payable, accounts receivable, engineering, and human resources. When a user joins one of these departments or groups, he or she is simply assigned as member of the role created for that group. A user can be a member of one or more roles, but the user is required to be a member of any roles. In addition to being assigned as a member of a database role, a can be assigned individual permissions.

Creating and Modifying Roles

You accomplish the tasks of creating and modifying database roles by using the same tools that you use perform most tasks related to database administration: Enterprise Manager or TSQL commands. (SQL Server does not provide a wizard for these tasks.) Whichever method you use, you must accomplish the following tasks when you implement a role:

- Create the database role.
- Assign permissions to the role.
- Assign users to the role.

When viewing a role, you will be able to see both the permissions assigned to the role and the users assigned to the role.

Using Enterprise Manager to Administer Roles

To create database roles by using Enterprise Manager, follow these steps:

1. Expand a server group, expand a server, and then expand the Databases folder. Right-click the database you want to create the role in (we will use Northwind for this example), point to New in the shortcut menu, and then choose Database Role. Alternatively, you can expand the database, right-click Roles, and choose New Database Role from the shortcut menu. Either way, the Role Properties window appears, as shown in Figure 34-18.

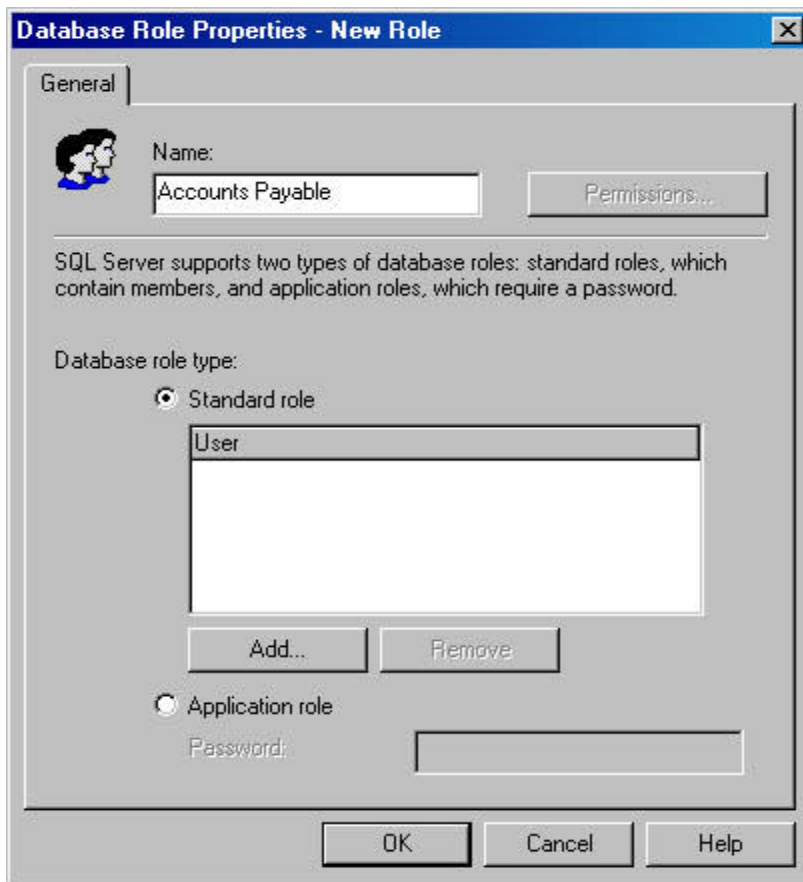


Figure 34-18. *The Database Role Properties window.*

2. Assign a descriptive name to the role by typing the name in the Name text box—choose a name will help you remember the function of the role. Figure 34-18 shows the name *Accounts Payable* chosen for a role.
3. To assign users to the role, click Add. A list of user accounts that have access to the database appears, as shown in Figure 34-19. Select the users you want to assign to this role. To cancel a selection, simply click the appropriate user name again. When you have finished modifying the membership of the role, click OK and the role will be created. You will be returned to the Manager window.



Figure 34-19. *The Add Role Members dialog box.*

4. To assign permissions to the role, first open the Database Role Properties window by expanding Roles folder, right-clicking the role name, and choosing Properties from the shortcut menu. Then click Permissions to display the Database Role Properties - Northwind window, as shown in 34-20.

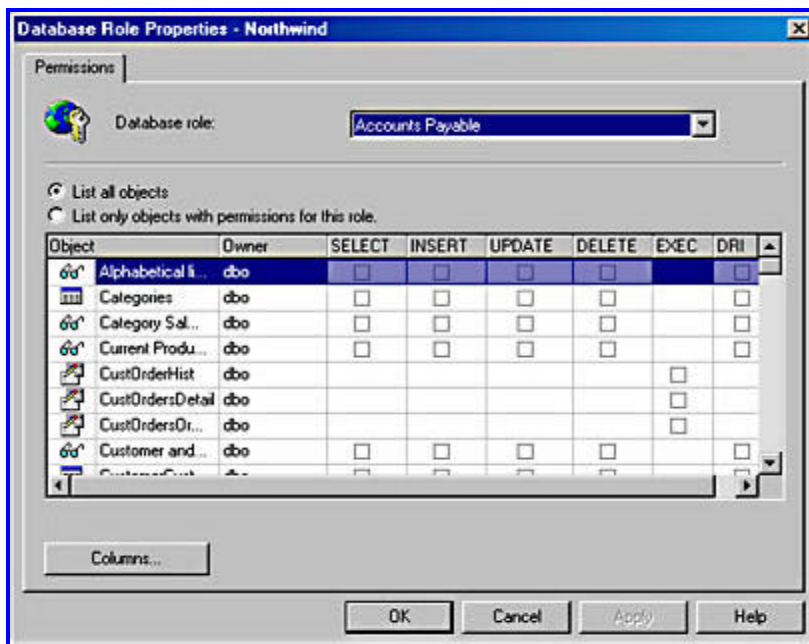


Figure 34-20. *The Database Role Properties - Northwind window.*

In this window, you can assign various permissions to this role for the objects within the containing the role. To do so, select the appropriate check boxes in the list box. The database objects are listed in the Object column. You can use the option buttons at the top of the window view all objects or just those for which this role already has permissions. Once you assign this

to a user, the user will receive all of the permissions that have been assigned to the role.

Once you create a role, you can modify it in the Database Role Properties window. To modify a role, follow the steps used to add permissions to a role. You can add and delete users and permissions in the Database Role Properties window.

Using T-SQL to Administer Roles

You can also create roles by using the *sp_addrole* stored procedure. The *sp_addrole* stored procedure the following syntax:

```
sp_addrole [ @rolename = ] 'role'  
          [ , [ @ownername = ] 'owner' ]
```

For example, to add a role named *readonly* to the Northwind database, use the following T-SQL command:

```
USE Northwind  
GO  
sp_addrole 'readonly' , 'dbo'  
GO
```

The *USE Northwind* command will select Northwind as your current database. If you do not specify a database, the role will be created in your default database.

This stored procedure will only create the role. To add permissions to the role, use the GRANT which was described earlier. To remove permissions from the role, use the REVOKE statement, also described earlier.

For example, to add the SELECT permission on the *Employees*, *Customers*, and *Orders* tables to the *readonly* role, use the following GRANT statement:

```
USE Northwind  
GO  
GRANT SELECT  
ON      Employees  
TO      readonly  
GO  
GRANT SELECT  
ON      Customers  
TO      readonly  
GO
```

To add users to the role, use the *sp_addrolemember* stored procedure. The *sp_addrolemember* stored procedure has the following syntax:

```
sp_addrolemember 'role', 'security_account'
```

The following command adds a user to the *readonly* role:

```
USE Northwind  
GO  
sp_addrolemember 'readonly' , 'Guest'
```


GO

Using Fixed Server Roles

A number of predefined roles that apply at the server level are created when SQL Server is installed. These fixed server roles are used to grant permissions to DBAs and can contain both server permissions and object and statement permissions. These roles are listed here:

- **bulkadmin** Can perform bulk inserts
- **dbcreator** Can create and alter databases
- **diskadmin** Can manage disk files
- **processadmin** Can manage SQL Server processes
- **securityadmin** Can manage logins and create database permissions
- **serveradmin** Can set any server options and can shut down the database
- **setupadmin** Can manage linked servers and startup procedures
- **sysadmin** Can perform any server activity

By assigning user accounts to fixed server roles, you enable users to perform the administrative tasks those roles have permissions for. Depending on your needs, this setup might be preferable to having DBAs use the same administrative account. Like database roles, fixed server roles are much easier to maintain than individual permissions, but fixed server roles cannot be modified. You can assign a user fixed server role by following the steps listed next.

1. In Enterprise Manager, expand a server group, expand a server, expand the Security folder, and click Server Roles. Right-click the fixed server role you want to add the user to and choose Properties from the shortcut menu. This invokes the Server Role Properties window, shown in Figure 34-21.

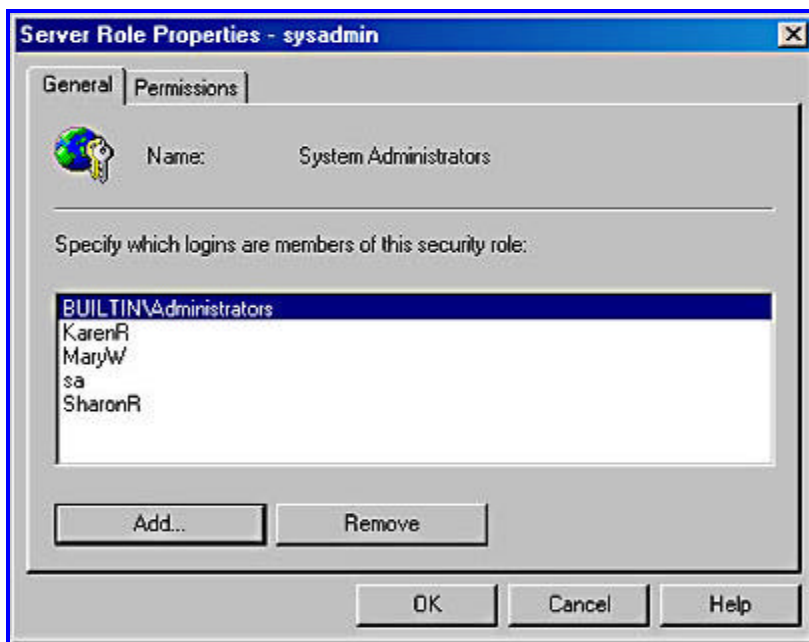


Figure 34-21. The Server Role Properties window.

2. To add a user account to the role, first click Add. This invokes the Add Members dialog box, in Figure 34-22, in which you select new role members.

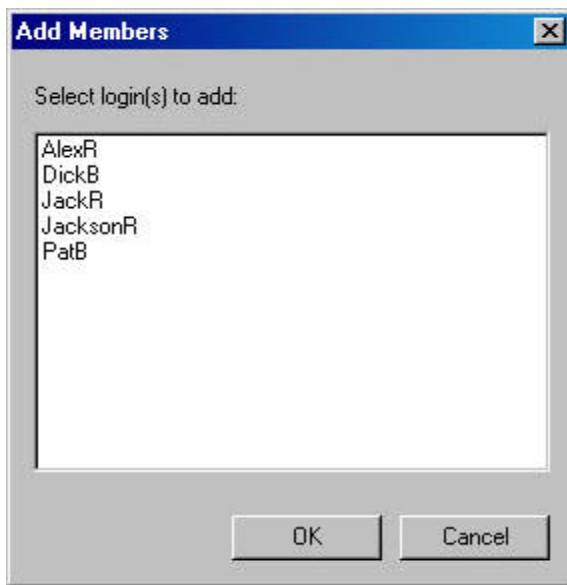


Figure 34-22. *The Add Members dialog box.*

3. After you have selected the users that you want to add to the fixed server role, click OK. This will return you to the Server Role Properties window. Click OK to add the user to the security role.

[3.4](#)

Security Account Delegation

SQL Server 2000 builds on the Windows 2000 security features by using the Kerberos security model. (Information about the Kerberos security model can be found in [Chapter 2](#).) SQL Server 2000 uses the Kerberos protocol to support mutual authentication between the client and the server. This allows the security credentials of a client to be passed between computers so that the client can connect to servers; when a new server is accessed, the server can proceed by using the credentials of the impersonated client. This credential sharing is known as security account delegation.

Let's look at an example of security account delegation. Suppose a client connects to ServerA as NTDOMAIN\AlexR, and ServerA connects to ServerB. ServerB then knows that the connection identity is NTDOMAIN\AlexR. This alleviates the need for the client to log in to ServerB.

If you want to use security account delegation, all servers that you are connecting to must be running Windows 2000 with Kerberos support enabled, and you must be using Active Directory services. The following options must be set in Active Directory services for delegation to work:

- **Account Is Sensitive And Cannot Be Delegated** This option must not be selected for the user requesting delegation.
- **Account Is Trusted For Delegation** This option must be selected for the service account of SQL Server 2000.
- **Computer Is Trusted For Delegation** This option must be selected for the server running an instance of SQL Server 2000.

Configuring SQL Server

Prior to using security account delegation, you must configure SQL Server 2000 to accept the Delegation enforces mutual authentication. To use security account delegation, SQL Server 2000 must have a Service Principal Name (SPN) assigned by the Windows 2000 account domain administrator. SPN must be assigned to the service account of the SQL server on that computer. The SPN is necessary prove that SQL Server is verified on the particular server and at the particular socket address by the Windows 2000 account domain administrator. You can have your domain administrator establish an for SQL Server by using the Setspn utility, which is accessible through the Windows 2000 Resource Kit.

To create an SPN for SQL Server, you run the following command:

```
setspn -A MSSQLSvc/Host:port serviceaccount
```

Here is an example of using this command:

```
setspn -A MSSQLSvc/MyServer.MyDomain.MyCompany.com sqlaccount
```

MORE INFO

For more information about the Setspn utility, see the Windows 2000 documentation.

You must also be using TCP/IP to use security account delegation. You cannot use named pipes the SPN targets a particular TCP/IP socket. If you are using multiple ports, you must have an SPN for each port.

You can enable delegation by using the LocalSystem account. SQL Server will self-register at service startup and automatically register the SPN. This option is easier than enabling delegation by using a domain user account. However, when SQL Server shuts down, the SPNs will be unregistered for the LocalSystem account. To enable delegation to run under the LocalSystem account, run the following command in the Setspn utility:

```
setspn -A MSSQLSvc/Host:port serviceaccount
```

NOTE

If you change a service account in SQL Server 2000, you must delete any previous SPNs were defined and create new ones.

[3.4](#)

Summary

In this chapter, you've learned about SQL Server user and security management. You've seen how both database logins and database user accounts are used to allow access to the database, and you've learned how to create and manage logins and database users. You've also learned how database roles can make user management easier by allowing a set of permissions to be assigned to a group of users and by allowing those permissions to be modified in a single location. And you've learned about a special group of roles called fixed server roles, which are used to assign administrative permissions to users and

Finally we looked at a security enhancement included in SQL Server 2000 that enables security accounts to be securely passed between servers in a Windows 2000 environment. In [Chapter 35](#), you'll learn about SQL stored procedures and optimizing your queries.