LESSON 5: Administering a Database

After you create the new table for the database that contains the second Washington store's inventory and sales data, your boss asks you to ensure that the store's employees are logging in to the correct database for their store data. She also requests that you work on and implement a backup plan for each store's information. Per your boss's request, you look at implementing both a full backup and an incremental backup schedule, as well as defining a new security schema.

The usefulness of a database depends in large part on the security of the information it contains. Therefore, as a database administrator, you must have a clear understanding of how permissions are granted so that users can access only certain tables or databases. In addition, you must be familiar with how and when to back up and restore a database.

Securing Databases

THE BOTTOM LINE

All database administrators must understand the need to secure a database, what objects can be secured, and what objects should be secured, as well as the importance of user accounts and roles.

It's common practice to first develop a database and then worry about the security of that database. Although there's no point in applying security while a database is in the process of being designed, the project will ultimately be beneficial if you consider your security plan sooner rather than later. Security, like every other aspect of a database project, must be carefully designed, implemented, and tested. Also, because security may affect the execution of some procedures, it must be taken into account when a project's code is being developed.

A simple security plan with only a few roles and all IT users designated as sysadmins may suffice for a small organization, but larger organizations—such as the military, banks, or international corporations—require a more complex security plan that's designed and implemented with heavy security in place. Regardless of an organization's size, the end result of its ***database security*** should be to ensure that users' assigned rights and responsibilities are enforced through a security plan.

Within a database, a ***permission*** is used to grant an entity (such as a user) access to an object (such as another user or a database). The security model within Microsoft SQL Server is very complex, so great thought must be given to applying appropriate user roles and permissions. The SQL Server security model is based on what are referred to as "securables"; in this model, different objects (defined as databases, tables, logins, users, and roles) can be granted permissions to access different securables.

A ***login*** or logon is the process by which individual access to a computer system is controlled by identification of the user through the credentials he or she provides. The most common login method involves supplying both a username and password. A ***user account*** is a logical representation of a person within an electronic system.

It is important to be aware of the rights and permissions associated with each object in a database because it's possible to inadvertently grant administrative rights to objects or users that should not have them. Within SQL Server, users are assigned to roles, which may in turn grant permission to objects, as illustrated in Figure 5-1. Note that each object has an owner, and ownership also affects permissions.

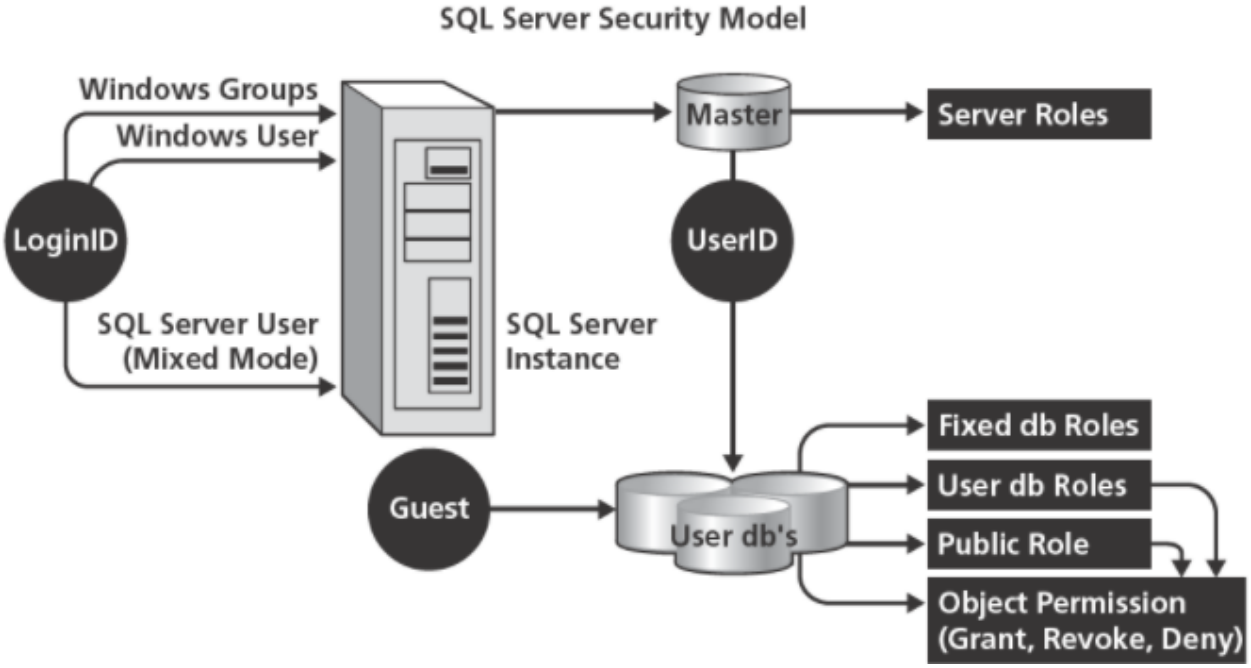

Figure 5-1 Planning roles and permissions

An overview of the SQL Server security model shows that users are first authenticated to the server, followed by the databases, and finally the objects within the databases. In the diagram, the circles represent how the user is identified.

Understanding Server-Level Security

CERTIFICATION READY
What roles are assigned to a Microsoft SQL Server, and what roles are assigned to a SQL database?
5.1

In the security model for a SQL Server, there are three different methods by which a user can be initially identified.

The three different methods by which a user can be initially identified include:

- Windows user login
- Membership in a Windows user group
- SQL Server-specific login (if the server uses mixed-mode security)

TAKE NOTE*
Users can login to a SQL Server using a Windows domain login, a username login, or a SQL Server login.

It is important to remember that at the SQL Server level, where the database resides, users are known by their login names. This can be a SQL Server login, a Windows domain login, or a username login.

Once a user logs into the server and is subsequently verified, that user now has whatever server-level administration rights he or she has been granted via fixed server roles. (These concepts will be discussed in greater depth later in this lesson.)

Remember, if you add a user to the ***sysadmin*** role, that user now has full access to every server function, database, and object for that server. With full access, the user can now grant other users permission to all server securables, and he or she can perform a variety of system-level actions, such as adding his or her network login ID to be mapped to a specific database user ID. Thus, the sysadmin role is a powerful one, and you must be sure not to grant it to the wrong user login. Users who lack the sysadmin level of access can't alter database server configurations or grant access where they shouldn't be able to. It is possible for users who have not been granted direct access to a database to gain access using the "guest" user account—and with this account, they can make limited changes within the database server.

Understanding Database-Level Security

Even though a user may belong to a fixed database role and have certain administrativelevel permissions, he or she still cannot access data without first being granted permission to database objects (e.g., tables, stored procedures, views, functions).

All users are automatically members of the public standard database role, but user-defined roles are custom roles that serve as groups. These roles may then be granted permission to a database object, and users may be assigned to a database user-defined role.

Each object's permission is assigned through granting, denying, or revoking user login permissions:

TAKE NOTE*

Certain database fixed roles can also affect object access, such as the right to read to and write from the database.

- Granting permission means that a user can access the object.
- Denying permission overrides a granted permission.
- Revoking a permission removes the permission that has been assigned, regardless of wh permission.

A user may have multiple permission paths to an object (e.g., individually, through a standard database role, and through the public role). If any of these paths are denied, then the user is blocked from accessing the object.

CERTIFICATION READY
What is the primary permission that gives a user full permission to all databases?
What is the primary permission that gives a user full permission to only a single database?
5.1

Understanding Windows Security

Because SQL Server is an environment within the Windows Server system, one of your primary security concerns should be ensuring that the Windows Server itself is secure.

Because SQL Server databases often support websites, you need to be sure that all firewalls and other Internet server applications are detailed and considered when constructing your security plan. You must also be familiar with the different types of SQL Server service accounts, as well as the basics of Windows authentication.

UNDERSTANDING SQL SERVER SERVICE ACCOUNTS

It is important to note that the SQL Server process itself requires permission to access files and directories, and it therefore requires a Windows account. Three different types of accounts are available for the SQL Server service account:

- **Local user account:** If you find that access to the network is not actually required, this is the perfect option to consider because a local user account cannot be used outside the server environment.
- **Local system account:** If you are using a single-server installation, you may wish to choose this account, because the SQL Server can use the local system account of the operating system for permission to the machine. The only drawback of using this account login is that it fails to provide the necessary network security credentials for databases because it has privileges inside the operating system that the administrator's account does not. This creates a potential security hole.

- **Domain user account:** This is the recommended login account because the SQL Server can then use the Windows account specifically created for it. You can then grant administrator rights to the SQL Server account.

UNDERSTANDING WINDOWS AUTHENTICATION

Authentication is the act of establishing or confirming a user or system identity. Windows Authentication mode is superior to mixed mode because users need not learn yet another password and because this mode leverages the security design of the network.

Using Windows Authentication means that users must have a valid Windows account in order to be recognized by SQL Server. The Windows SID (security identifier) is passed to SQL Server. Windows Authentication is very robust in that it will authenticate not only Windows users, but also users within Windows user groups.

When a Windows user group is accepted as a SQL Server login, any Windows user who is a member of that group can be authenticated by SQL Server. Access, roles, and permissions can be assigned to the Windows user group, and they will apply to any user in that group.

SQL Server also knows the actual Windows username for each user, so the application can gather audit information at both the user level and the group level.

ADDING A NEW WINDOWS LOGIN

Windows users are created and managed in various places in different versions of Windows. In Windows Vista and newer versions, local users can be managed by selecting Control Panel > Administrative Tools > Computer Management. Domain users are managed with tools such as the Active Directory Users and Computers snap-in.

Once users exist in the Windows user list or the Windows domain, SQL Server can recognize them.

ADD A NEW LOGIN TO SQL SERVER

GET READY. To add a new login to SQL Server through Object Explorer, follow these steps:

1. In SSMS, open and right-click the Security folder, select New and Select User, as shown in Figure 5-2.

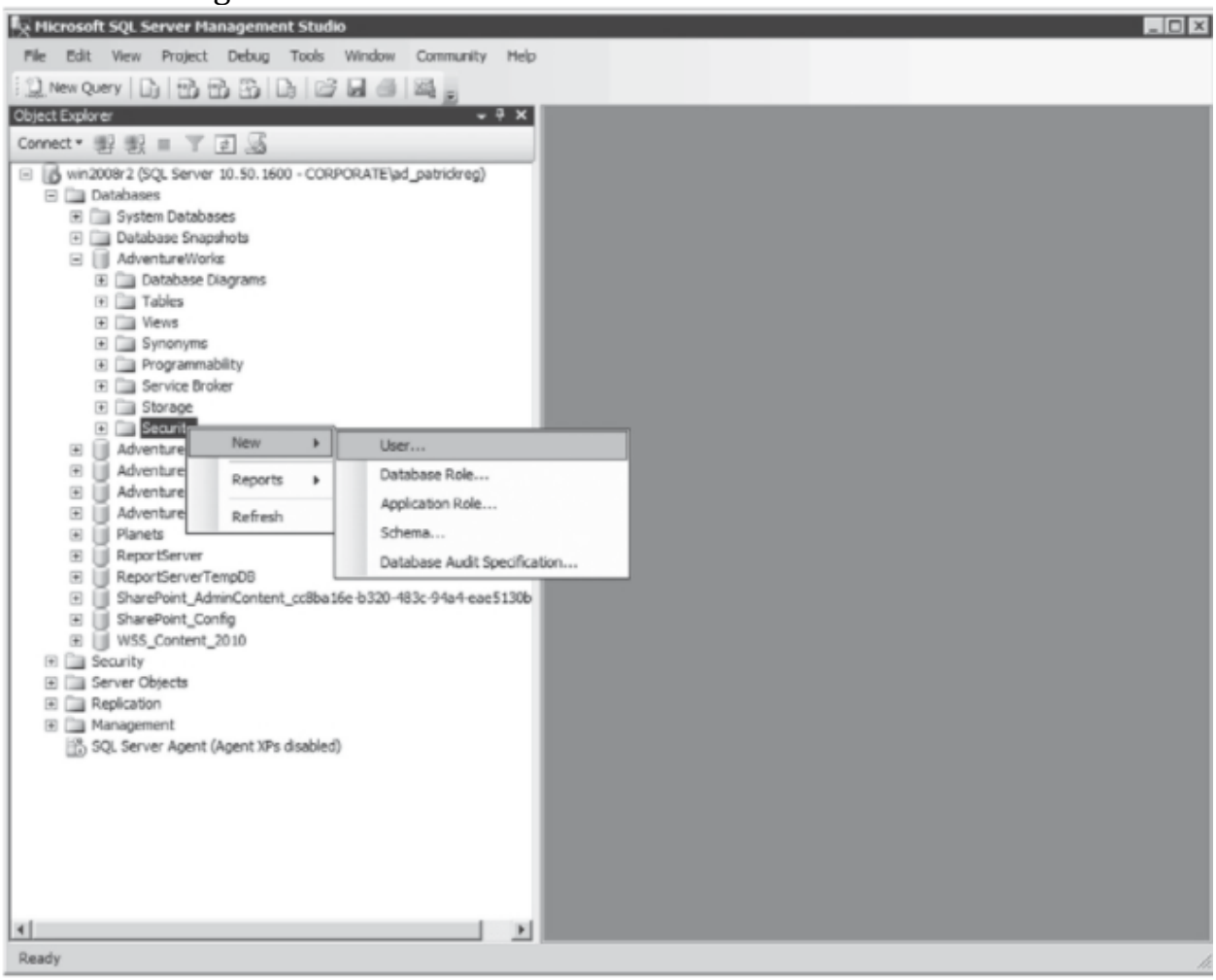

Figure 5-2 Adding a new user

2. In the General page of the Database User, (as shown in Figure 5-3) type in the name of the user or you can use the Search (…) button to locate the Windows user.

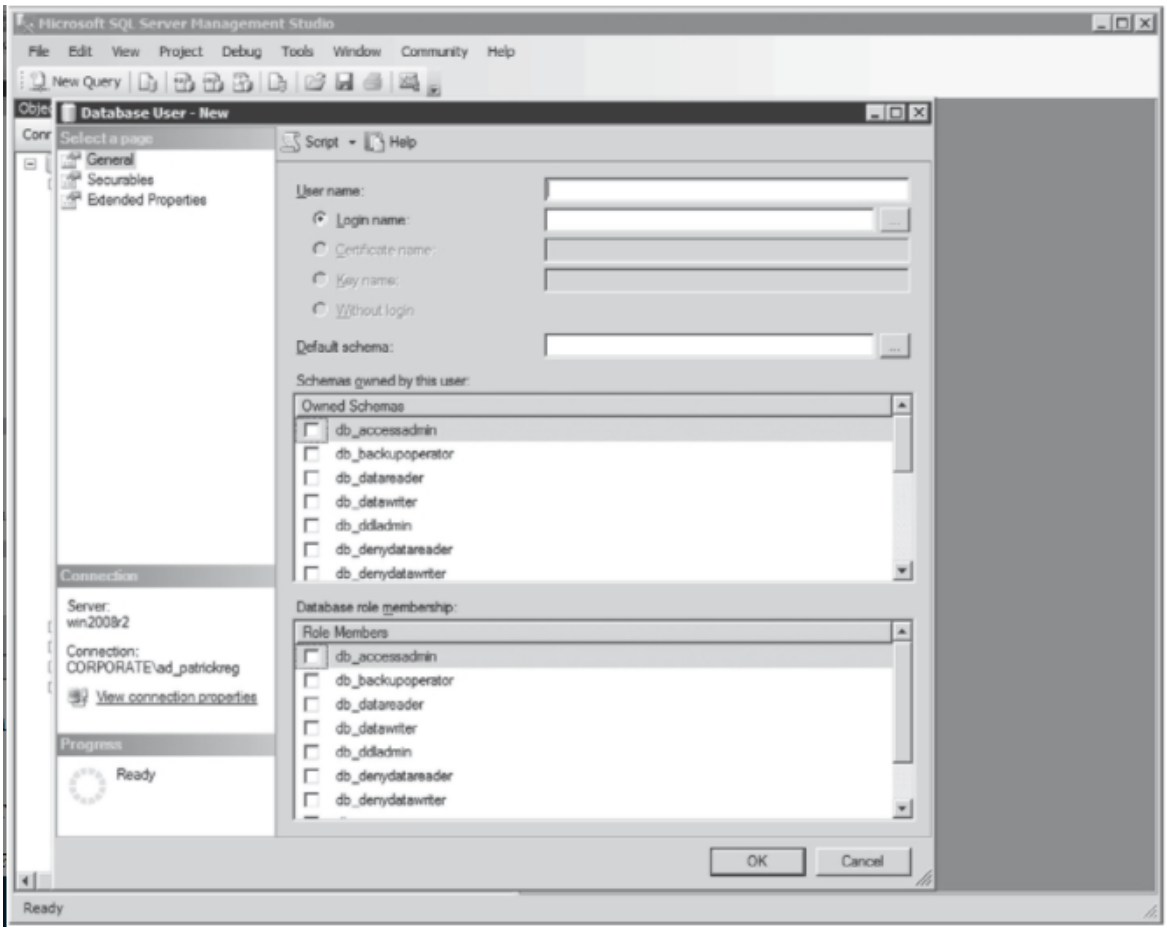

Figure 5-3 Choosing a user name

3. You may enter a username or group name or use the Browse button to search for a user, as shown in Figures 5-4 and 5-5. Windows users are managed and assigned to different Windows groups using the Computer Management tool.

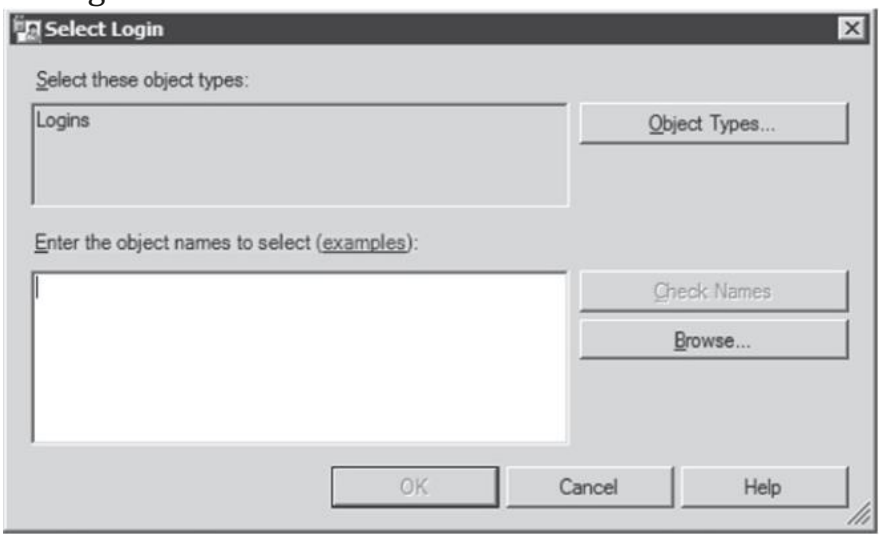

Figure 5-4 Browsing for existing users

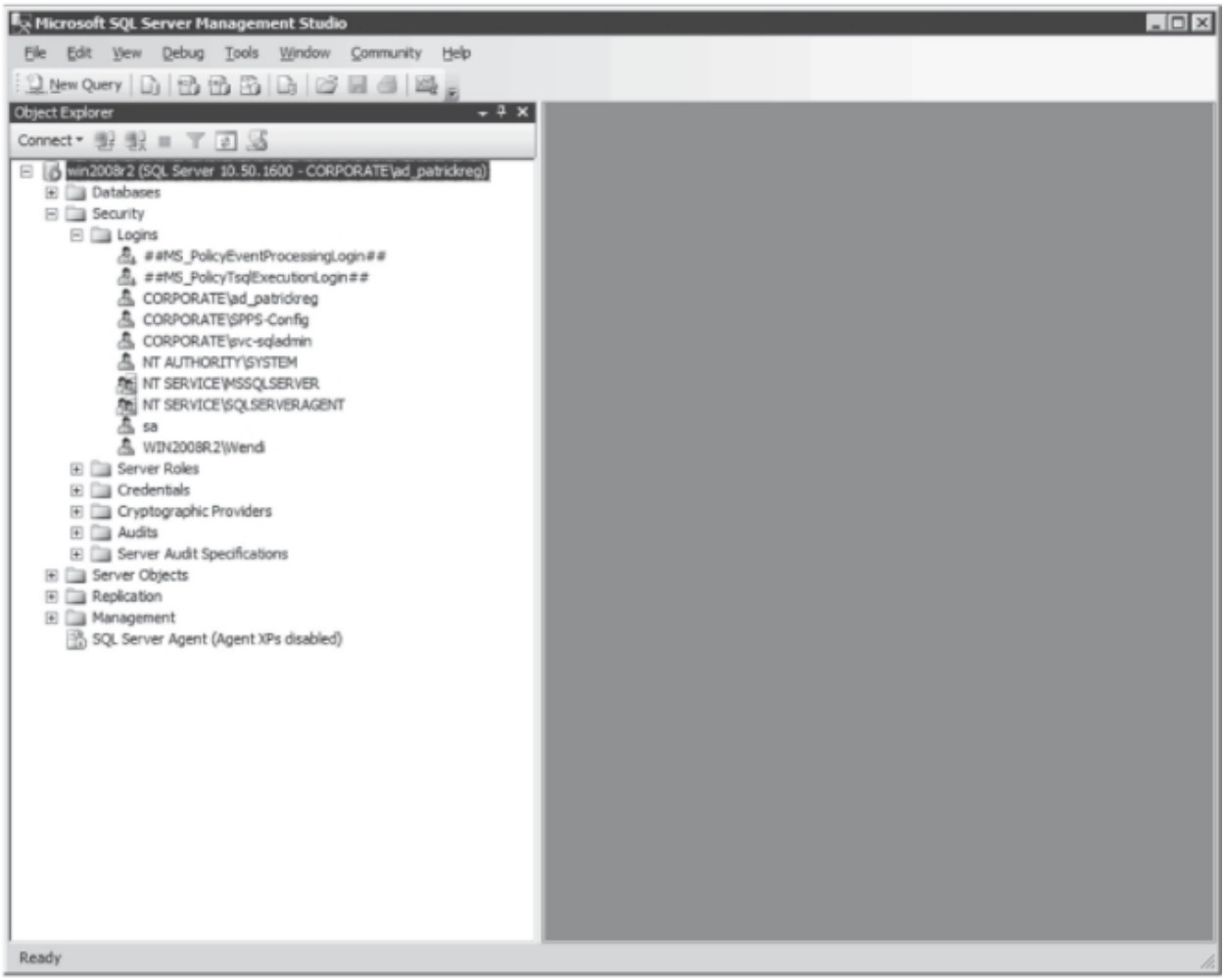

Figure 5-5 SQL Server logins

4. Click OK to save and close the Database User – New dialog box

The user may be assigned a default database and language at the bottom of the SQL Server Login Properties dialog box, but note that assigning a default database to a user does not automatically grant access to that database. The user may be granted access to databases in the Database Access tab.

To create a login using Transact-SQL syntax so that you can add a Windows user or group, run the `CREATE LOGIN` command. Be sure to use the full Windows username, including the domain name, of the user you are trying to add, as follows:

```
CREATE LOGIN 'XPS\Joe'
```

If you want to create and edit user logins at the server level, use the General page of the Login/ New Dialog box. The Login dialog box is also used to manage existing users. To access the Login dialog box, just double-click the user.

TAKE NOTE*
A user can be granted access to databases in the Database Access tab.

REMOVING A WINDOWS LOGIN

A Windows login can be removed from SQL Server through SSMS. To do so, select the security directory (much as you did to create a new user login) in Object Browser, then use the menu to find and delete the desired user (as shown in Figure 5-6). Of course, this doesn't delete the user from Windows; it only removes the user from SQL Server.

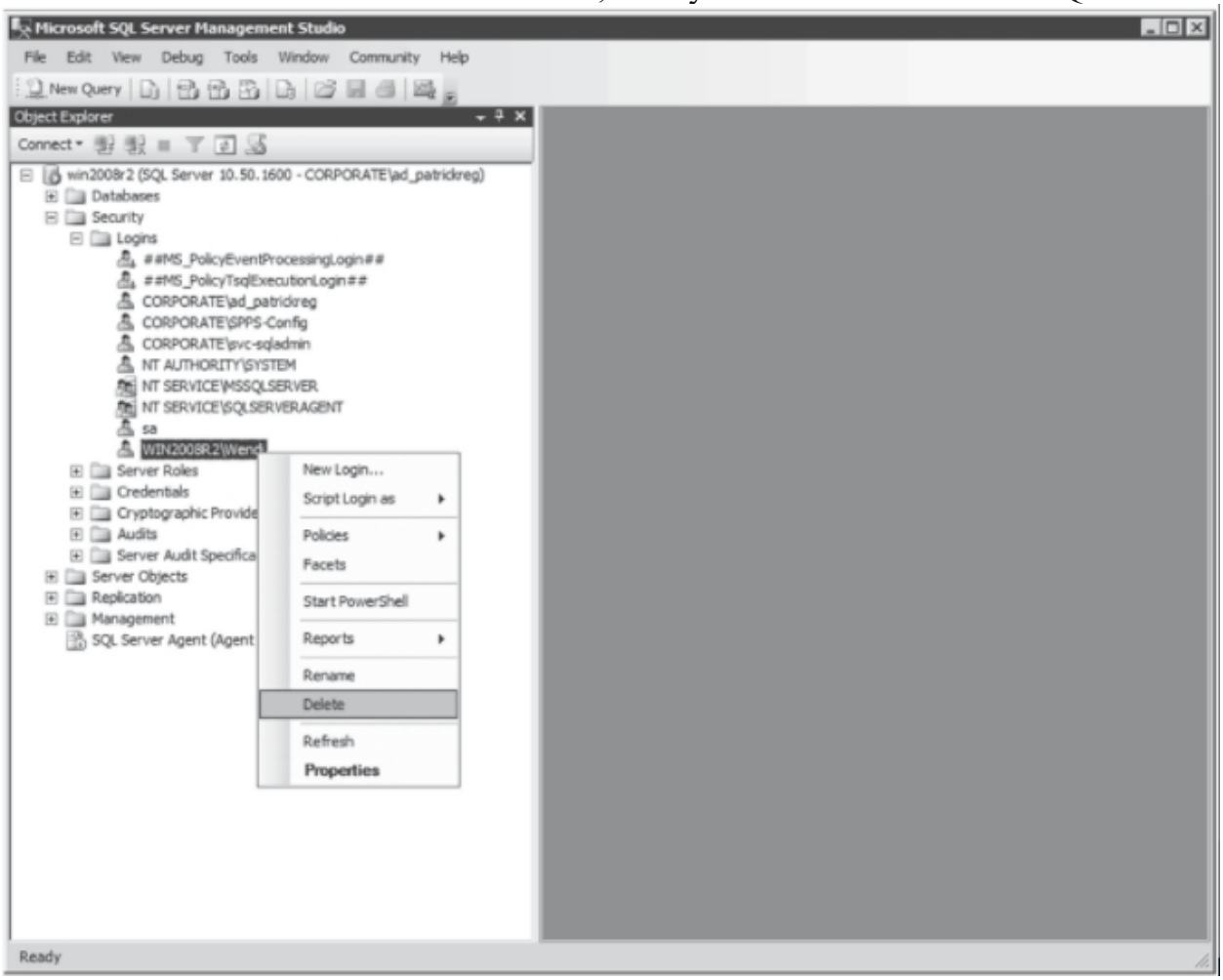

Figure 5-6 Deleting an existing user from SQL Server

To remove a Windows user or group from SQL Server, you can also use the `DROP LOGIN` command, as in the following example:

```
DROP LOGIN 'XPS\Joe'
```

After running this command, the Windows user or group will continue to exist in Windows but will no longer be recognized by SQL Server.

Understanding SQL Authentication

SQL servers also support mixed mode, which allows you to connect to a SQL server using Windows authentication or SQL Server authentication. A SQL Server login account and related passwords are defined on the SQL server and are not related to Active Directory or Windows accounts.

Associated with SQL authentication is the ***sa account***. The sa account is the built-in SQL administrator account associated with SQL authentication. Because SQL Authentication is less secure than Windows logins, avoiding mixed mode is recommended; however, it is available for backward compatibility.

Understanding Database Server Roles

There are three kinds of database server roles: fixed roles, the public role, and user-defined roles. Each of these roles is explored in greater depth in this section.

UNDERSTANDING FIXED SERVER ROLES

SQL Server includes fixed, predefined ***server roles***. Primarily, these roles grant permission to perform certain server-related administrative tasks. A user may belong to multiple server roles.

The following fixed server roles are best used for delegating certain server administrative tasks:

- **Bulkadmin:** Can perform bulk insert operations.
- **Dbcreator:** Can create, alter, drop, and restore databases.
- **Diskadmin:** Can create, alter, and drop disk files.
- **Processadmin:** Can kill a running SQL Server process.
- **Securityadmin:** Can manage the logins for the server.
- **Serveradmin:** Can configure the server-wide settings, including setting up full-text searche
- **Setupadmin:** Can configure linked servers, extended stored procedures, and the startup sto
- **Sysadmin:** Can perform any activity in the SQL Server installation, regardless of any oth overrides denied permissions on an object.

The one user that SQL Server automatically creates during installation of the software is BUILTINS/Administrator, which includes all Windows users in the Windows Administration group and allows a choice of what groups or users are added during setup. The BUILTINS/ Administrators user can be deleted or modified as desired after installation.

If you add a user to the sysadmin role group, that user must reconnect to the SQL Server instance in order for the full capabilities of the sysadmin role to take effect.

Fixed server roles are set in SSMS in the Server Roles page of the Login Properties dialog box (see Figure 5-7).

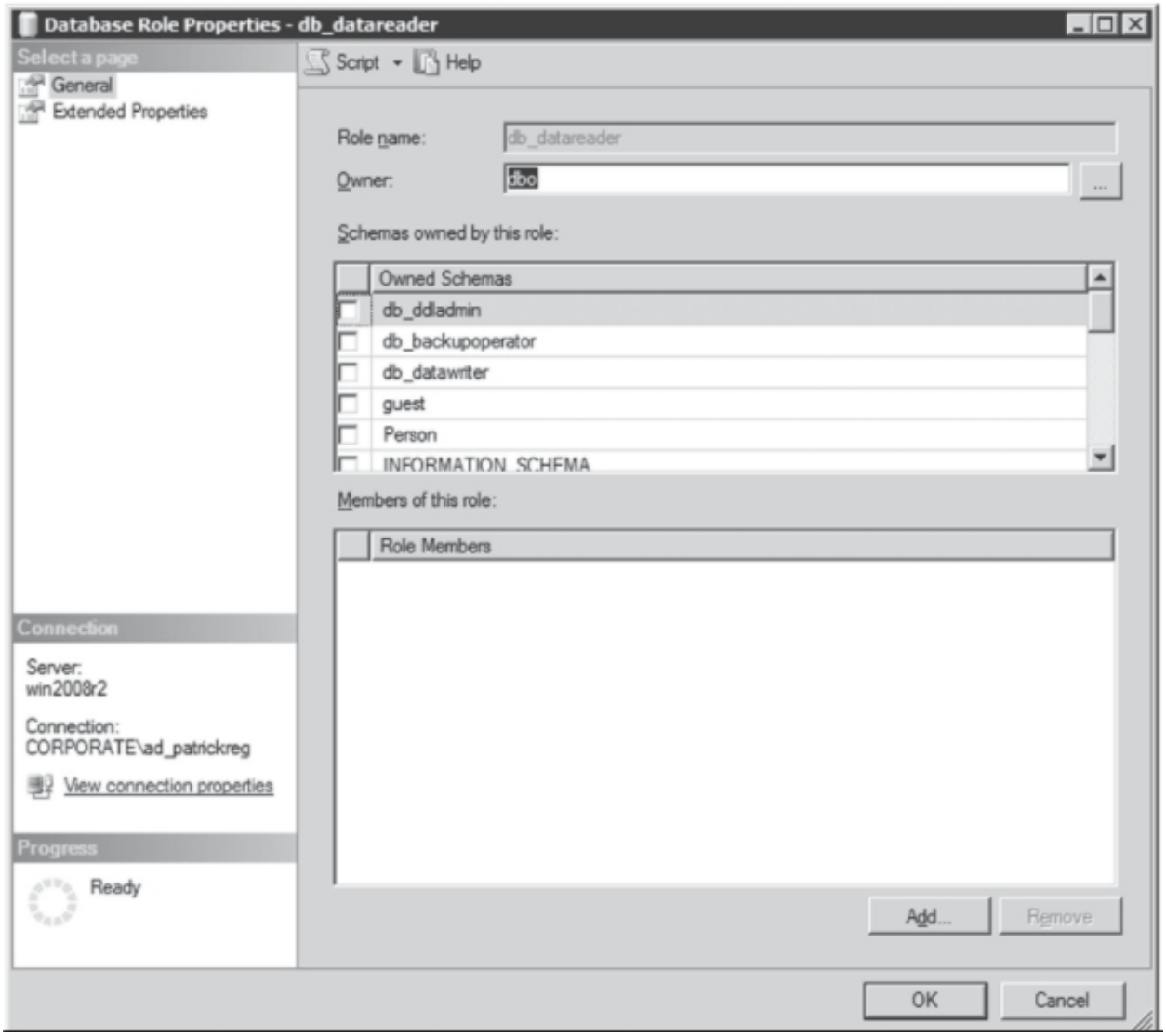

Figure 5-7 Database Role Properties dialog box

In Transact-SQL syntax, you can assign a user to different server roles by way of a stored procedure, as follows:

```
sp_addsrvrolemember
[ @loginame = ] 'login',
[ @rolename = ] 'role'
```

For example, the following code adds the user login ''XPS\Lauren'' to the sysadmin role:

```
EXEC sp_addsrvrolemember 'XPS\Lauren', 'sysadmin'
```

UNDERSTANDING THE PUBLIC ROLE

The public role is a fixed role, but it can have object permissions like a standard role. Every user is automatically a member of the public role and cannot be removed, so the public role serves as a baseline or minimum permission level.

UNDERSTANDING USER-DEFINED ROLES

Because you cannot modify the permissions assigned to a fixed server role, you may need to grant individual server permissions to a user that are not defined by a fixed server role. Such user-defined roles are typically employed for users who need to perform specific database functions but to whom you don't want to grant a role that would permit them do more than what they need to.

Granting Access to a Database

Users must be explicitly granted access to any user database. Because this establishes a many-to-many relationship between logins and the database, you can manage database access from either the login side or the database side.

When a login is granted access to a database, that login is also assigned a database username, which may be the same as the login name or may be some other name by which the login will be known within the database.

To grant access to a database from the login side using Object Explorer, use the User Mapping page of the Login Properties form (shown in Figure 5-8).

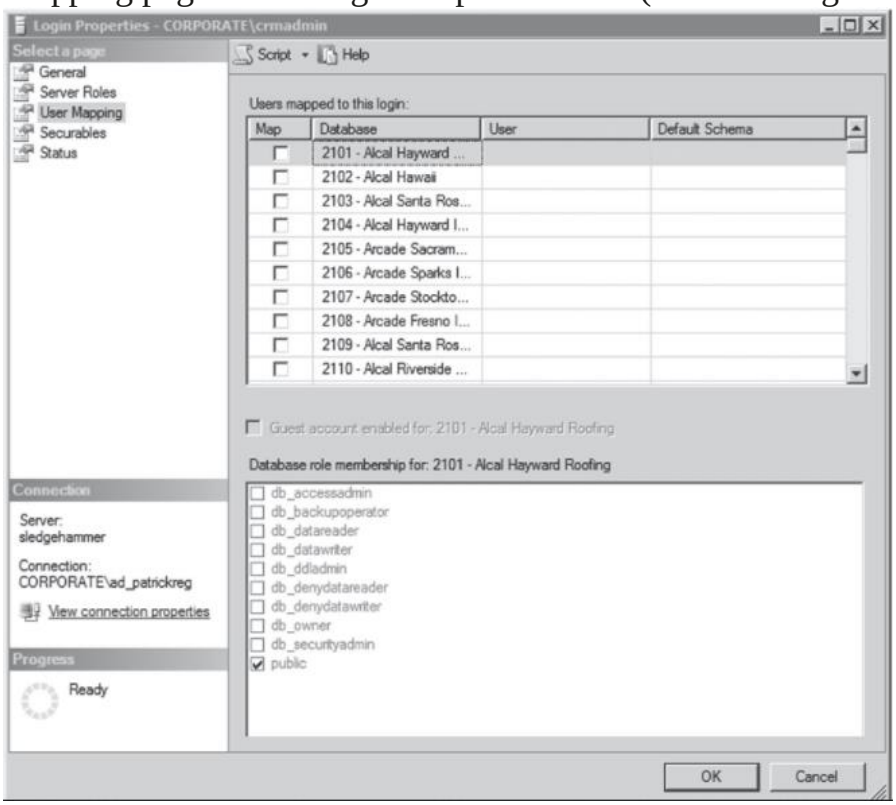

Figure 5-8 User Mapping page

To grant access from the database point of view, use the New User Context Menu command under the Database > Security > Users node to open the Database User-New form. Enter the login to be added in the Login Name field. To search for a login, use the ellipsis (. . .) button. In the User Name field, you must enter a name by which the user will be known within the database.

You can use the Login Properties form to grant a login access to any database and to assign database roles.

A Transact-SQL command is also available to grant database access to a user. This command must be issued from within the database to which the user is to be granted access. The first parameter in the command syntax is the server login, and the second is the optional database username, as in the following example:

```
USE Family

CREATE USER 'XPS\Lauren', 'LRN'
```

Lauren now appears in the list of database users as LRN. To remove Lauren's database access, the system-stored procedure DROP USER requires her database username, not her server login name, as follows:

```
USE Family

DROP USER 'LRN'
```

The Login dialog box can be used to add a new user to the database or to manage a current user.

UNDERSTANDING GUEST LOGIN ACCOUNTS

Any user who wishes to access a database but who has not been declared a user within the database is automatically granted the privileges of the ***guest user***, as long as the guest user account has been created. The guest user account is not actually created when a database is created; it must be specifically added either through SSMS or through a Transact-SQL statement, as shown here:

```
EXEC sp_adduser 'Guest'
```

Guest users must be removed from a database when they are no longer welcome, as they are a risk for a security breach.

UNDERSTANDING OBJECT SECURITY

If a user has access to a database, then permission to the individual database objects may be granted. Permission may be granted either directly to the user or to a standard role, with the user then assigned to the role.

Users may be assigned to multiple roles, so multiple security paths from a user to an object may exist.

Understanding Fixed Database Roles

SQL Server includes a few standard, or fixed, database roles. Like fixed server roles, these roles primarily organize administrative tasks. A user may belong to multiple fixed database roles.

In SQL Server, fixed database roles include the following:

- **db_accessadmin:** Authorizes a user to access the database, but not to manage database-level security.
- **db_backupoperator:** Allows a user to perform backups, checkpoints, and DBCC commands, but not restores. (Only server sysadmins can perform restores.)
- **db_datareader:** Authorizes a user to read all data in the database. This role is the equivalent of a grant on all objects, and it can be overridden by a deny permission.
- **db_datawriter:** Allows a user to write to all data in the database. This role is the equivalent of a grant on all objects, and it can be overridden by a deny permission.
- **db_ddladmin:** Authorizes a user to issue DDL commands (create, alter, drop).
- **db_denydatareader:** Permits a user to read from any table in the database. This overrides any object-level grant.
- **db_denydatawriter:** Blocks a user from modifying data in any table in the database. This overrides any object-level grant.
- **db_owner:** This is a special role that has all permissions in the database. This role includes all the capabilities of the other roles and differs from the dbo user role. This is not the database-level equivalent of the server sysadmin role because an object-level deny will override membership in this role.
- **db_securityadmin:** Permits a user to manage database-level security—including roles and permissions.

ASSIGNING FIXED DATABASE ROLES WITH SSMS

Fxed database roles can be assigned via SSMS using either of the following procedures:

- By adding the role to the user in the user's Database User Properties form, either as the user
- By adding the user to the role in the Database Role Properties dialog. To do so, select Roles u context menu to open the Properties form (see Figure 5-9).

ASSIGNING FIXED DATABASE ROLES WITH TRANSACT-SQL

In Transact-SQL code, you can add a user to a fixed database role by using the sp_addrole system stored procedure. For instance, the following example creates the database role auditors, which is owned by the db_securityadmin fixed database role:

```
USE AdventureWorks;

CREATE ROLE auditors AUTHORIZATION db_securityadmin;

GO
```

UNDERSTANDING APPLICATION ROLES

An application role is a database-specific role intended to allow an application to gain access regardless of its user. For example, if a specific Visual Basic (VB) program is used to search the Customer table and it doesn't handle user identification, that VB program can access SQL Server using a hard-coded application role. Thus, anyone using the VB application gains access to the database.

The Database Role Properties dialog box lists all users assigned to the current role.

ACCESSING THE DATABASE ROLES

GET READY. To add a user to a database role, follow these steps:

1. In SSMS, expand the database folder by clicking the appropriate plus (+) sign. Expand the Security folder, Expand Roles, and then expand the Database Roles folder.
2. Double-click the appropriate role to open the Properties dialog box.
3. To add or remove users from the role, use the Add and Remove buttons, respectively.

Understanding Object Permissions

Object permissions are permissions that allow a user to act on database objects, such as tables, stored procedures, and views.

Several specific types of object permissions exist:

- **Select:** The right to select data. Select permission can be applied to specific columns.
- **Insert:** The right to insert data.

- **Update:** The right to modify existing data. Update rights for which a `WHERE` clause is used require select rights as well. Update permission can be set on specific columns.
- **Delete:** The right to delete existing data.
- **DRI (References):** The right to create foreign keys with DRI.
- **Execute:** The right to execute stored procedures or user-defined functions.

Object permissions are assigned with the SQL DCL commands `GRANT` , `REVOKE` , and `DENY` . Permissions in SQL Server work just as they do in the operating system. SQL Server aggregates all the permissions a given user might have, whether assigned directly to the user or assigned through roles.

SQL Server gives the maximum of whatever permission has been granted. `DENY` is an exception to this rule, however. `DENY` functions as a trump card of sorts. In other words, if a `DENY` command has been issued anywhere, then, just as in Windows, the user is blocked. For instance, if a user can `SELECT` against a table directly assigned, but a role the user is a member of has a `DENY` for `SELECT` , then this user is blocked from issuing a `SELECT` against the table. Whether security is being managed from SSMS or from code, it's important to understand these three commands.

Granting object permission interacts with the server and database roles. The sysadmin server role is the ultimate security role, which has full access to all databases.
If a user does not have the sysadmin server role, the highest level object permission would be the Grant and Deny object permissions. However, Deny permission always has a higher priority than the Grant permission.

If your environment prohibits mixed-mode security, then the easiest way to check security is to right-click SQL Server Management Studio or Query Analyzer and use the `RUN AS` command to run as a different user; however, this requires the creation of dummy users in the Windows domain. Generally speaking, in a ''production'' Windows domain, most auditors would flag dummy users as an audit point. Because the workstations that belong to database administrators tend to belong in production domains, this recommendation won't work if the auditors are diligent.

MODIFY AN OBJECT'S PERMISSIONS

GET READY. To access an object's permission, follow these steps to modify an object's permissions:

1. In SSMS, open the database and open the object that you want to manage. The object could be tables, views, stored procedures, or user-defined functions.
2. In the Object Browser, right-click the object and select Properties to open the Properties dialog for that object type.

3. Click the Permissions page to open the Object Properties dialog.
4. To add a user, click the Search button. Type in the name of the user you wish to add or click the Browse button to select the user. Click the OK button to close the Select Users or Roles dialog box.
5. Select the appropriate Grant to Deny permission. See Figure 5-9.

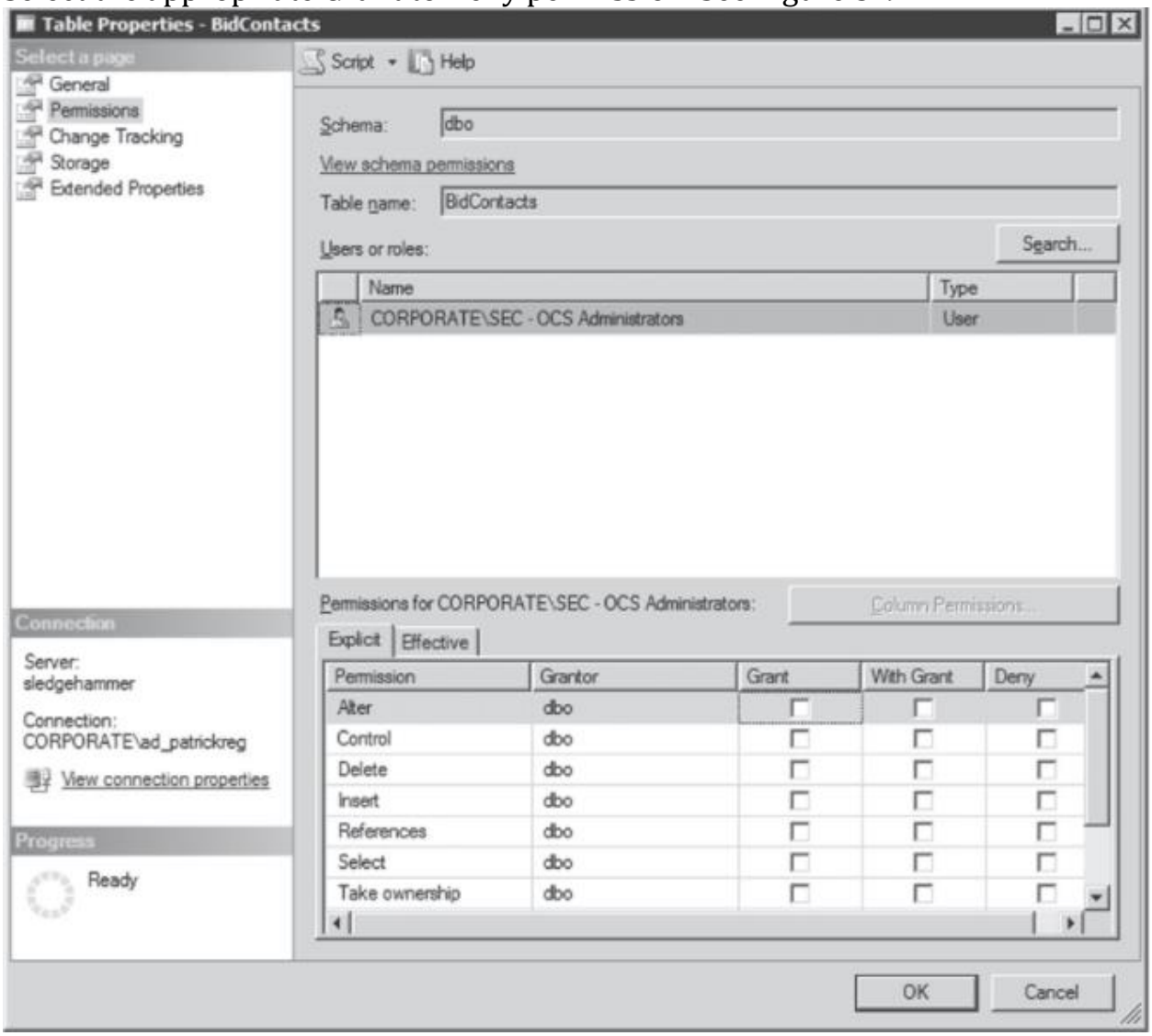

Figure 5-9 Granting Object Permissions

6. When done, click the OK button to close the Properties dialog box.

The top portion of the form is for selecting a user or role to assign or check permissions. The user must have access to the database to be selected.

As with setting statement permissions in the Database Properties Security tab, you can select grant, with grant, or deny. The object list at the top of the dialog box shows all the objects in the database. This list can be used to switch to other objects quickly without backing out of the form to the console and selecting a different object.

If the user or role has permission to a table, the Columns button opens the Column Permissions dialog. Select the user and click the button to set the columns permissions for that user. Only select and update permissions can be set at the column level, because inserts and deletes affect entire rows.

SETTING PERMISSIONS FROM THE USER LIST

Instead of granting the permission to a user from the properties of the object, you can also grant permissions to an object from the properties of the user. From the list of database users in SSMS, select a user and double-click, or select Properties from the right-click context menu. The Login Properties dialog box will appear, and it can be used to assign users to roles (as shown in Figure 5-10). The Securables page is used to assign or check object permissions. This dialog box is similar to the Permissions tab of the Database Object Properties dialog box.

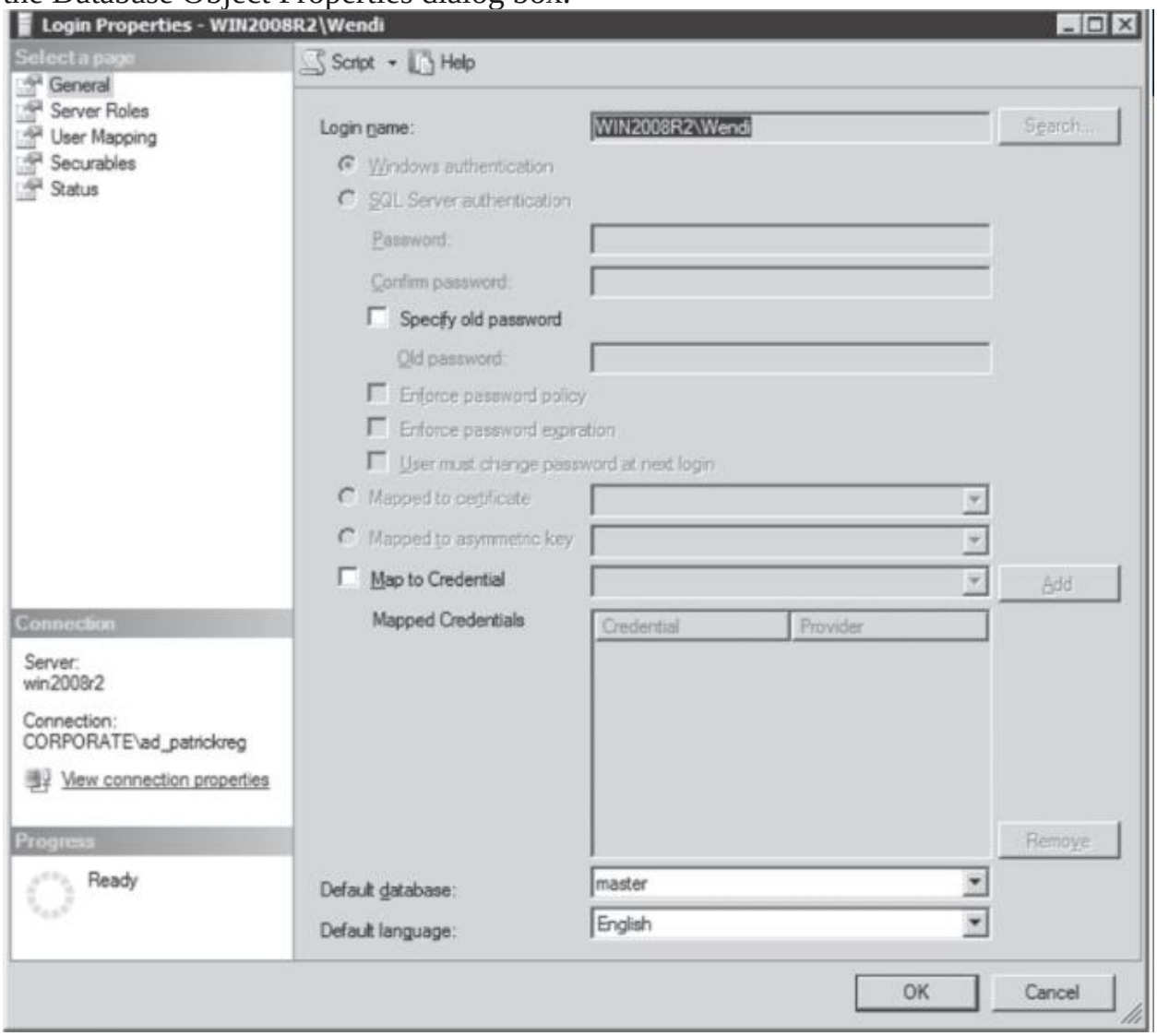

Figure 5-10 Login Properties dialog box

SETTING PERMISSIONS FROM THE ROLE LIST

The third way to control object permissions is from the database role. To open the Database Role Properties dialog box, double-click a role in the list of roles, or select Properties from the right-click context menu. The Database Role Properties dialog box can be used to assign users or other roles to a role, as well as to remove them from a role.

The Permissions button opens the Permissions dialog box for the role. This form operates like the other permission forms, except it is organized from the role's perspective.

GRANTING OBJECT PERMISSIONS WITH TRANSACT-SQL STATEMENTS

Using Transact-SQL, you can grant any object permissions to a database, table, view, or any other database object. The Transact-SQL statement to provide permission to an object for a specific user and his or her role is as follows:

```
GRANT Permission, Permission

ON Object

TO User/role, User/role

WITH GRANT OPTION
```

The assigned permission may be ALL, SELECT, INSERT, DELETE, REFERENCES, UPDATE, or EXECUTE.

The role or username refers to the database username, any user-defined public role, or the public role. For example, the following code grants select permission to Joe for the Person table:

```
GRANT Select ON Emails TO Joe
```

The next example grants all permissions to the public role for the Marriage table:

```
GRANT All ON Contacts TO dbcreator
```

Multiple users or roles and multiple permissions may be listed in the command. For example, the following code grants select and update permission to the guest user and to LRN:

```
GRANT Select, Update ON Emails to Guest, LRN
```

The `WITH GRANT` option provides the ability to grant permission for an object. For example, the following command grants Joe permission both to select from the Email table and to grant select permission to others:

```
GRANT Select ON Email TO Joe WITH GRANT OPTION
```

Managing Roles

Roles can be created, managed, and removed via SSMS or by executing Transact-SQL statements. Of course, you should create roles and assign users to those roles only when needed.

MANAGING ROLES WITH TRANSACT-SQL STATEMENTS

Creating standard roles with code involves using the sp_addrole system stored procedure. A role's name can be up to 128 characters and cannot include a backslash, be null, or be an empty string. By default, the roles will be owned by the dbo user. However, you can assign the role of owner by adding a second parameter. The following code creates the manager role:

```
CREATE ROLE 'Manager'
```

The counterpart to creating a role is removing it. A role may not be dropped if any users are currently assigned to it. The sp_droprole system stored procedure will remove the role from the database, as in the following example:

```
DROP ROLE 'Manager'
```

Once a role has been created, users may be assigned to the role by means of the sp_addrolemember system stored procedure. For instance, the following code sample assigns Joe to the manager role:

```
EXEC sp_addrolemember 'Manager', 'Joe'
```

Not surprisingly, the system stored procedure sp_droprolemember removes a user from an assigned role. Thus, the following code frees Joe from the drudgery of management:

```
EXEC sp_dropRoleMember 'Manager', 'Joe'
```

UNDERSTANDING HIERARCHICAL ROLE STRUCTURES

If your security structure is complex, then a particularly powerful permissions-organization technique is to design a hierarchical structure of standard database roles. In other words, you can nest user-defined database roles. For example:

- The worker role may have limited access.
- The manager role may have all worker rights plus additional rights to look up tables.
- The administrator role may have all manager rights plus the right to perform other database administration tasks.

To accomplish this type of design, follow these steps:

1. Create the worker role and set its permissions.
2. Create the manager role and set its permissions. Add the manager role as a user to the worker role.
3. Create the admin role. Add the admin role as a user to the manager role.

The advantage of this type of security organization is that a change in the lower level affects all upper levels. As a result, administration is required in only one location, rather than dozens of locations.

Understanding Ownership Chains

In SQL Server databases, users often access data by going through one or several objects. Ownership chains apply to views, stored procedures, and user-defined functions.

There are many occasions where a database object will access another database object. For example:

- A program might call a stored procedure that then selects data from a table.
- A report might select from a view, which then selects from a table.
- A complex stored procedure might call several other stored procedures.

In these cases, the user must have permission to execute the stored procedure or select from the view.

Whether the user also needs permission to select from the underlying tables depends on the ownership chain from the object the user called to the underlying tables.

If the ownership chain is unbroken from the stored procedure to the underlying tables, then the stored procedure can execute using the permission of its owner. The user only needs permission to execute the stored procedure, and the stored procedure can use its owner's permission to access the underlying tables. Thus, the user doesn't require permission to the underlying tables.

Ownership chains are great for developing tight security where users execute stored procedures but aren't granted direct permission to any tables.

If the ownership chain is broken, meaning the owners of one object and the next lower object are different, then SQL Server checks the user's permission for every object accessed.

Reviewing a Sample Security Model

To give a few examples of permissions using the OBXKites database, Table 5-1 lists the permission settings of the standard database roles. Table 5-2 lists a few of the users and their roles.

Table 5-1: Permission settings for OBXKites

STANDARD ROLE	HIERARCHICAL ROLE	PRIMARY FILEGROUP TABLES	STATIC FILEGROUP TABLES	OTHER PERMISSIONS
IT	Sysadmin server role			
Clerk				Execute permissions for several stored procedures that read from and update required day-to-day tables.
Admin	Db_owner Database fixed role			
Customer		Select permissions		

Table 5-2: Users and their roles for OBXKites

USER	DATABASE STANDARD ROLES
Sammy	Admin
Joe	Public
LRN	IT DBA
Clerk Windows group (Betty, Tom, Martha, and Mary)	Clerk

With this security model, the following users can perform the following tasks:

- Betty, as a member of the Clerk role, can execute the application that executes stored procedures to retrieve and update data. Betty can also run select queries as a member of the Public role.
- LRN, as the IT DBA, can perform any task in the database as a member of the Sysadmin server role.
- Joe can run select queries as a member of the Public role.
- As a member of the Admin role, Sammy can execute all stored procedures. He can also manually modify any table using queries. Furthermore, as a member of the Admin role that includes the Db_owner role, Joe can perform any database administrative task and select or modify data in any table.

- Only LRN can restore from the backups. Backups are discussed in the next part of the lesson.

Backing Up and Restoring Databases

THE BOTTOM LINE

In this section, you'll explore various backup types (such as full and incremental), the importance of backups, and how to restore a database.

The purpose of a database ***backup*** is to have something to restore if data is lost during a business's daily routine. For example, a user may accidentally delete a table, or a database administrator may need to ***restore*** multiple tables on different servers in order to combine them into one database. The need for a database backup and restore plan is both immediate and far reaching.

Understanding Recovery Models

SQL Server offers several recovery models for each database. The recovery models determine how much data loss in case the server has problems and you need to restore the data from backup.

SQL Server offers three recovery models. They are:

- Simple Recovery
- Full Recovery
- Bulk-Logged

Simple Recovery requires the least administration since the transaction log backups are truncated on a regular basis.

Full Recovery allows you to restore to a point in time since the logs files record all SQL transactions and the time they were performed. The disadvantages of Full Recovery mode is that the logs can grow a lot. Therefore, when you perform full backups, you will need to shrink and truncate the logs.

The least used model used is Bulk-Logged. It is a compromise between the two. It allows good performance while using the least log space. However, you cannot do a point-in-time recovery.

Understanding Database Backups

The scope of a backup of data (a ***data backup***) can be a whole database, a partial database, or a set of files or filegroups. For each of these, SQL Server supports full, differential, and incremental backups.

When you perform backups, you can choose the type of backup that is best for your environment. The type of backup is based on simplicity, time to perform a backup, and time to perform a restore.

- ***Full backup:*** A full backup contains all the data in a specific database or set of filegroups or files to allow recovering that data.
  CERTIFICATION READY
  What is the difference between a full backup and an incremental backup?
  5.2
- ***Differential backup:*** A differential backup is based on the latest full backup of the data. This is known as the ***base*** of the differential, or the differential base. A differential backup contains only the data that has changed since the differential base. Typically, differential backups that are taken fairly soon after the base backup are smaller and faster to create than the base of a full backup. Therefore, using differential backups can speed up the process of making frequent backups to decrease the risk of data loss. Usually, a differential base is used by several successive differential backups. At restore time, the full backup is restored first, followed by the most recent differential backup.
- ***Incremental backup:*** An incremental backup is based on the last backup of the data. An incremental backup contains only the data that has changed since the last full or incremental backup. Incremental backups are smaller and faster to create than full backups and differential backups. At restore time, the full backup is restored first, followed by each incremental backup following the full backup.

Over time, as a database is updated, the amount of data that is included in differential backups increases. This makes the backup slower to create and to restore. Eventually, another full backup must be created to provide a new differential base for another series of differential backups.

After the first data backup, under the full recovery model or bulk-logged recovery model, regular transaction log backups (or ***log backups***) are required. Each log backup covers the part of the transaction log that was active when the backup was created, and the log backup includes all log records that were not backed up in a previous log backup. Database backups are easy to use and are recommended whenever database size allows. Table 5-3 shows the types of database backups supported by SQL Server.

Table 5-3: Types of database backups supported by Microsoft SQL

BACKUP TYPE	DESCRIPTION
Database backup	A full backup of the whole database. Database backups represent the whole database at the time the backup finished.

BACKUP TYPE	DESCRIPTION
Differential database backups	A backup of all files in the database. This backup contains only the data that were modified since the most recent database backup of each file.

UNDERSTANDING PARTIAL AND DIFFERENTIAL BACKUPS

Partial and differential partial backups are designed to provide more flexibility for backing up databases that contain some read-only filegroups under the simple recovery model. However, these backups are supported by all recovery models. Table 5-4 shows the types of partial backups supported by SQL Server.

Table 5-4: Types of partial backups supported by Microsoft SQL

BACKUP TYPE	DESCRIPTION
Partial backup	A backup of all the full data in the primary filegroup, every read/write filegroup, and any optionally specified read-only files or filegroups. A partial backup of a read-only database contains only the primary filegroup.
Differential partial backup	A backup that contains only the data that were modified since the most recent partial backup of the same set of filegroups.

UNDERSTANDING FILE BACKUPS

The files in a database can also be backed up and restored individually. Using file backups can increase the speed of recovery by letting you restore only damaged files without restoring the rest of the database. For example, if a database consists of several files that are located on different disks and one disk fails, only the file on the failed disk will need to be restored. However, planning and restoring file backups can be complex; therefore, file backups should be used only where they clearly add value to your restore plan. Table 5-5 shows the types of file backups supported by SQL Server.

Table 5-5: Types of fi le backups

BACKUP TYPE	DESCRIPTION
File backup	A full backup of all the data in one or more files or filegroups. Important: Under the simple recovery model, file backups are basically restricted to read-only secondary filegroups. You can create a file backup of a read/write filegroup, but before you can restore the read/write file backup, you must set the

BACKUP TYPE	DESCRIPTION
	filegroup to read-only and take a differential read-only file backup.
Differential file backups	A backup of one or more files that contain data extents that were changed since the most recent full backup of each file. Note: Under the simple recovery model, this assumes that the data has been changed to read-only since the full backup.

Understanding Backup Devices

SQL Server backups are created on backup devices such as disk files or tape media. You can append new backups to any existing backups on a device, or you can overwrite any existing backups.

SCHEDULING BACKUPS

Performing a backup operation has minimal effect on transactions that are running; therefore, backup operations can be run during regular operations. During a backup operation, SQL Server copies the data directly from the database files to the backup devices. The data is not changed, and transactions that are running during the backup are never delayed. Therefore, you can perform a SQL Server backup with minimal effect on production workloads.

Understanding Database Restores

SQL Server supports a variety of restore scenarios, each of which is outlined in the following section.

Restore scenarios possible in SQL Server include the following:

- **Complete database restore:** Restores an entire database, beginning with a full database backup, which may be followed by restoring a differential database backup (and log backups).
- **File restore:** Restores a file or filegroup in a multi-filegroup database. After a full file restore, a differential file backup can be restored.
- **Page restore:** Restores individual pages.
- **Piecemeal restore:** Restores a database in stages, beginning with the primary filegroup and one or more secondary filegroups.
- **Recovery only:** Recovers data that is already consistent with the database and needs only to be made available.
- **Transaction log restore:** Under the full or bulk-logged recovery model, since the logs record each transaction, restoring from log backups is required to reach a desired recovery point.

- **Create a mirror database:** When you have a mirror database, you have duplicate databases on multiple servers. When information is written to one server, it is automatically replicated to the second server.
- **Create and maintain a standby server:** When you have a standby server, you are using an active-passive cluster that consists of two or more servers. When the active server fails, the passive server will become the active server, allowing for minimum downtime.

USING THE SSMS

To restore data through the graphical interface tool, follow these steps.

RESTORING DATA

GET READY. Before you begin, be sure to launch the SQL Server Management Studio application and connect to the database you wish to work with. Then, follow these steps:

1. After you connect to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**. Depending on the database, either select a user database or expand **System Databases** and then select a system database.
3. Right-click the database, point to **Tasks**, then click **Restore**.
4. Click **Database**, which opens the **Restore Database** dialog box.
5. On the **General** page, the name of the restoring database appears in the **To database** list box. To create a new database, enter its name in the list box.
6. In the **To a point in time** text box, either retain the default (*the most recent possible*) or select a specific date and time by clicking the browse button, which opens the **Point in Time Restore** dialog box.
7. To specify the source and location of the backup sets to restore, click one of the following options:
    a. **From database:** Enter a database name in the list box.
    b. **From device:** Click the browse button, which opens the **Specify Backup** dialog box. In the **Backup media** list box, select one of the listed device types. To select one or more devices for the **Backup location** list box, click **Add**.
    c. After you add the devices you want to the **Backup location** list box, click **OK** to return to the **General** page.
8. In the **Select the backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, change the selections in the grid. Any backups that depend on a deselected backup are deselected automatically.

9. To view or select the advanced options, click **Options** in the **Select a page** pane.
10. In the **Restore options** panel, you can choose any of the following options, if appropriate for your situation:
    a. Overwrite the existing database.
    b. Preserve the replication settings.
    c. Prompt before restoring each backup.
    d. Restrict access to the restored database.
11. Optionally, you can restore the database to a new location by specifying a new restore destination for each file in the **Restore the database files as** grid.
12. The **Recovery state** panel determines the state of the database after the restore operation. The default behavior isLeave the database ready to use by rolling back the uncommitted transactions. Additional transaction logs cannot be restored (RESTORE WITH RECOVERY). (Choose this option only if you are restoring all the necessary backups at this point.)
13. Alternatively, you can choose either of the following options:
    a. Leave the database non-operational, and do not roll back the uncommitted transactions. Additional transaction logs can be restored (RESTORE WITH NO RECOVERY).

CERTIFICATION READY
Do you understand database backups and restores?
5.2

    b. Leave the database in read-only mode. Undo uncommitted transactions, but save the undo actions in a standby file so that recovery effects can be reverted (RESTORE WITH STANDBY).

USING THE RESTORE COMMAND

The Transact-SQL RESTORE command enables you to perform the following restore scenarios:

- Restore an entire database from a full database backup (a complete restore).
- Restore part of a database (a partial restore).
- Restore specific files or filegroups to a database (a file restore).
- Restore specific pages to a database (a page restore).
- Restore a transaction log onto a database (a transaction log restore).
- Revert a database to the point in time captured by a database snapshot.

For example, to restore the database using the specified file, you would execute the following command:

```
RESTORE DATABASE name_of_database FROM DISK = ‘name of backup’
  GO
```

For example, to restore the AdventureWorks database using the C:\AdventureWorks.BAK backup, you would execute the following command:

```
RESTORE DATABASE AdventureWorks FROM DISK = ‘C:\AventureWorks.BAK’
  GO
```

For more information on using the RESTORE command, refer to the following website: http://msdn.microsoft.com/en-us/library/ms186858.aspx

SKILL SUMMARY

IN THIS LESSON, YOU LEARNED THE FOLLOWING:

- The end result of database security is to ensure that the rights and responsibilities given to users are enforced.
- A permission is used to grant an entity (such as a user) access to an object (such as another user or a database).
- A login or logon is the process by which individual access to a computer system is controlled by identification of the user using credentials provided by the user. The most common login method involves supplying a username and password.
- A user account is a logical representation of a person within an electronic system.
- Even though a user may belong to a fixed database role and have certain administrativelevel permissions, a user still cannot access data without first being granted permission to the database object themselves (e.g., tables, stored procedures, views, functions).
- Each object's permission is assigned by granting, revoking, or denying user login permissions.
- Authentication is the act of establishing or confirming a user or system identity.
- Windows Authentication mode is superior to mixed mode because users don't need to learn yet another password and because it leverages the security design of the network.
- The sa account is the built-in SQL administrator account associated with SQL authentication.
- SQL Server includes fixed, predefined server roles. Primarily, these roles grant permission to perform certain server-related administrative tasks.
- The sysadmin role can perform any activity in the SQL Server installation, regardless of any other permission setting. The sysadmin role even overrides denied permissions on an object.
- The public role is a fixed role, but it can have object permissions like a standard role. Every user is automatically a member of the public role and cannot be removed, so the public role serves as a baseline or minimum permission level.
- Users must be explicitly granted access to any user database.
- The db_owner is a special role that has all permissions in the database.
- An application role is a database-specific role intended to allow an application to gain access regardless of the user.
- The purpose of a database backup is to have something to restore if data is lost during a business's daily routine.
- A full backup contains all the data in a specific database or set of filegroups or files to allow recovering that data.
- Differential backup only backs up data since the last full backup.
- Incremental backup only backs up data since the last full or incremental backup.
- When you restore from differential backup, you must first restore the preceding full backup and then restore the last differential backup.
- When you restore from an incremental backup, you must first restore the preceding full backup and then restore each incremental backup since the full backup in order.