# Beginning SQL Server 2005 for Developers

## From Novice to Professional

Robin Dewson

Apress®

**Beginning SQL Server 2005 for Developers**



ISBN-13 (pbk): 978-1-59059-588-6

ISBN-10 (pbk): 1-59059-588-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1



Lead Editors: Matthew Moodie, Tony Davis
Technical Reviewer: Jasper Smith
Additional Material: Cristian Lefter
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editor: Ami Knox
Assistant Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Susan Glinert
Proofreaders: Lori Bring, Nancy Sixsmith
Indexer: Broccoli Information Management
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.



The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 4

# Security

Security is important—more so, in fact, than design, creation, and performance. If your database had no security measures in place, absolutely anyone could come along and steal or corrupt the data, causing havoc to you and your company. And not just in one database, but on every database in every server.

Security can be enforced in many ways on a SQL Server: by Windows itself through Windows authentication; by restricting users' access to sensitive data through views; or by specifically creating users, logins, and roles that have explicit levels of access.

This chapter covers some parts of security, although it is impossible to talk about every area of security, mainly because we haven't seen much of SQL Server's contents yet! In Chapter 1 we looked at the difference between Windows authentication and SQL Server authentication, so already you know your options with regard to the type of security you might wish to use. So what is next?

First of all, you need to understand what **users**, **roles**, and **logins** are.

## Logins

The only way anyone can connect to SQL Server is via a login. As discussed in Chapter 1, this doesn't necessarily mean that every user has to have a specific login within SQL Server itself. With Windows authentication, if a user belongs to a specific Windows group, just by belonging to that group, providing that group is contained within SQL Server, the account will have access to SQL Server.

When a database is created, initially only the database owner has any rights to complete any task on that database, whether that be to add a table, insert any data, or view any data. This was the case when we first created our `ApressFinancial` database in Chapter 3. It is only when the database owner grants permissions to other users that they gain extra access to complete tasks.

It is common practice to create a Windows group and place Windows user accounts into that group. This is how we wish to work with our `ApressFinancial` system, and so we will create some Windows groups for it. We will group logins depending on which department we are dealing with and what we want to allow each group to do. We will allow some groups to add new financial products, other groups to add customers, and, finally, a group set up for batch processes to add interest and financial transactions. We will create a few of these groups so that later in the book we can see security in action.

In Chapter 1, I mentioned that you should log in as an administrator account to install SQL Server. This would mean that you are in the `BUILTIN/Administrators` group, which is a group defined for the local computer that contains Windows users accounts with administrator rights. We can therefore already connect to SQL Server with this login, which includes VMcGlynn. AJMason could not log in, though. However, by adding this account to a group we will be creating, and then adding that group to SQL Server, we will see how they both can.

---

■ **Note** The process we are about to go through would be the same if we were adding a single user.

---

## Try It Out: Creating a Group

1. Navigate to your Control Panel, then select Administrative Tools ➤ Computer Management.

2. This brings up a screen that shows different aspects of your computer management. We are interested in selecting Local Users and Groups ➤ Groups. When you do so, you will see that there are already several groups within your computer, as shown in Figure 4-1, as well as a large number of groups already defined for the use of SQL Server. These groups differ from groups that we will be defining for accessing the data.

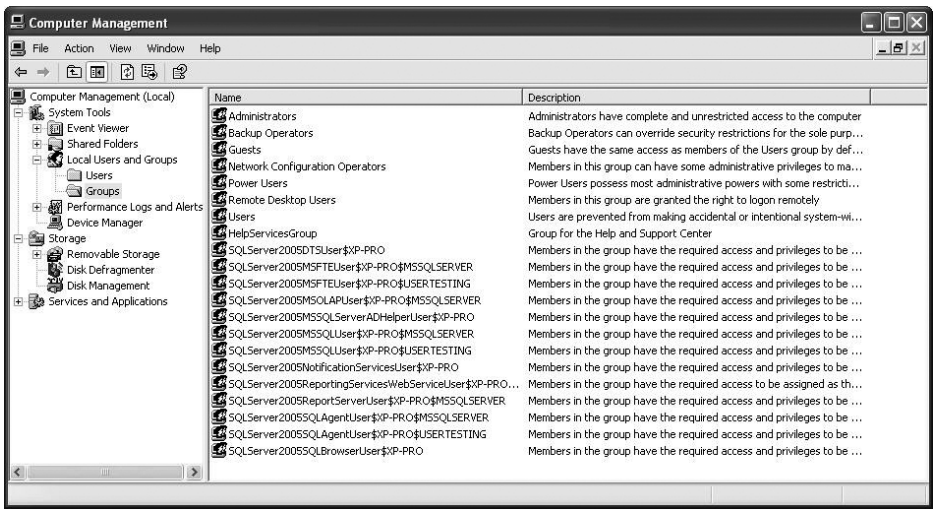


**Figure 4-1.** *List of groups on the computer*

3. AJMason is a product controller and can add new corporate financial products. Right-click Groups and select New Group. This will bring up the New Group screen, as shown in Figure 4-2, where we can add our grouping for our product controllers. `Apress_Product_Controllers` is the group we'll use in this chapter.


**Figure 4-2.** *Adding the first group for our application*

4. By clicking Add, we can then add all the Windows user accounts that we wish to be part of this group. We can either type **AJMason** or click Advanced, which brings up a selection dialog box. Clicking the Check Names button adds the user to the group.

If AJMason was on your company network, you would have to prefix the name with the domain name. For example, if you had a network domain called Apress and AJMason was on that domain (as opposed to your local computer and therefore your local domain as is the case for our example), then you would type **Apress\AJMason**. Figure 4-3 shows AJMason is on the XP-PRO local domain.

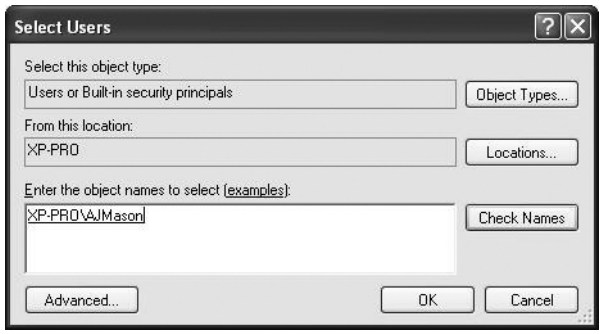


**Figure 4-3.** *AJMason found, ready to add to our group*

5. Click OK and then click the Create button on the New Group screen. Once you have created the group, you should close the New Group dialog box, as we don't want to create any more groups at the moment. This brings us back to the Computer Management dialog box, where we see our new group added, as shown in Figure 4-4.

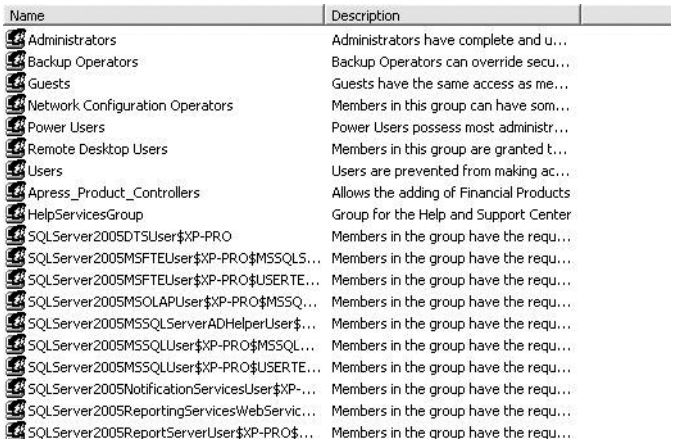


**Figure 4-4.** *New group added*

6. We now need to add this group to SQL Server. Open SQL Server Management Studio and navigate to Security/Logins within the Object Explorer. Once there, click New Login, which will bring up the dialog box shown in Figure 4-5.

7. Click Search to display the Select User or Group dialog box where we will begin our search for our group, as shown in Figure 4-6. This is very similar to the previous search box we saw but has been defined to search for a user or built-in security principal. However, by default, the search will not search for groups. You need to click Object Types and ensure the Groups option is checked on the screen that comes up.

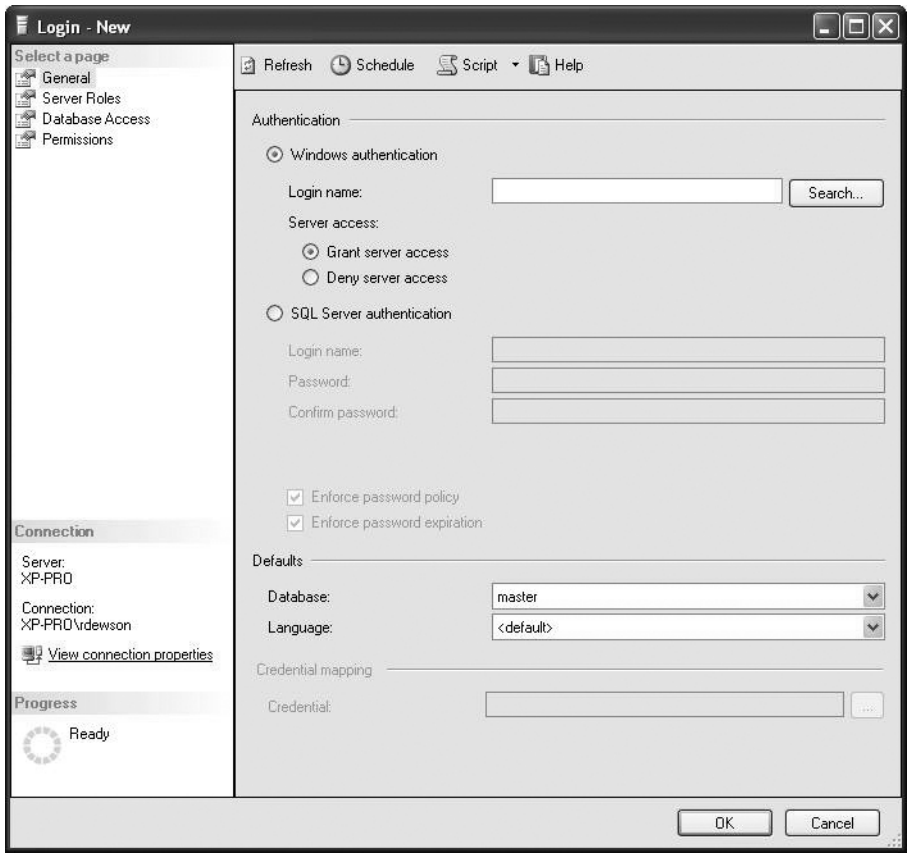


**Figure 4-5.** *Creating a new login*

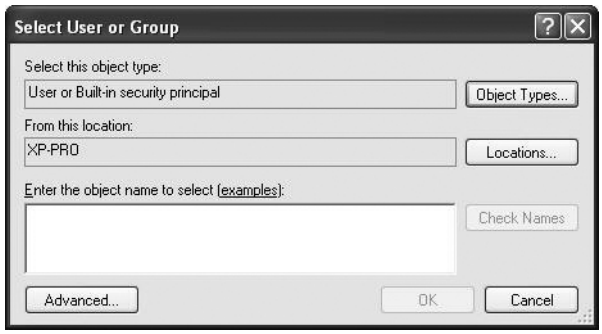


**Figure 4-6.** *Searching for groups*

8. This will allow you to click Advanced, which will then let you complete the search for the group you want. Highlight this group, `Apress_Product_Controllers` in this case, as shown in Figure 4-7, and click OK.

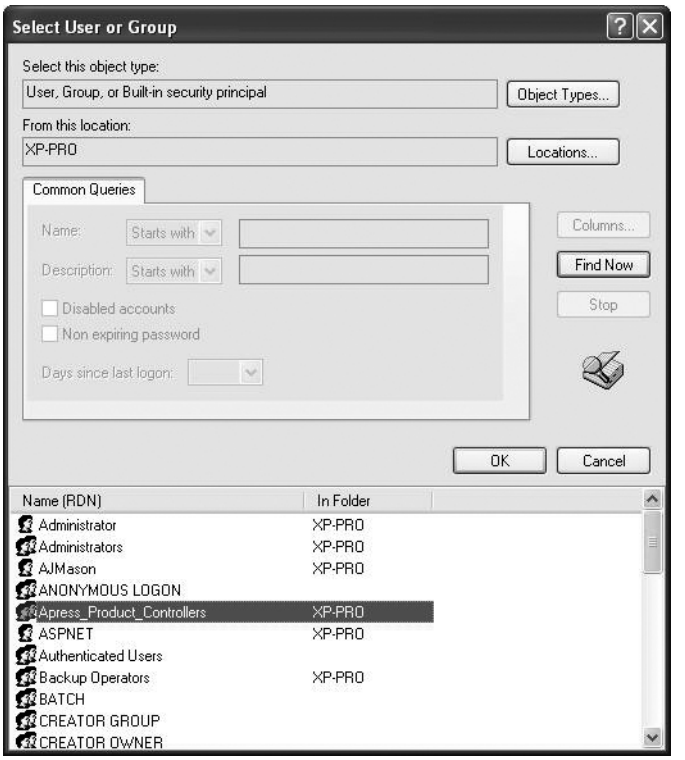


**Figure 4-7.** *Finding the Apress_Product_Controllers group*

**9.** This brings us back to the Select User or Group dialog box where we will see our group has been added, as shown in Figure 4-8. We can then click OK.

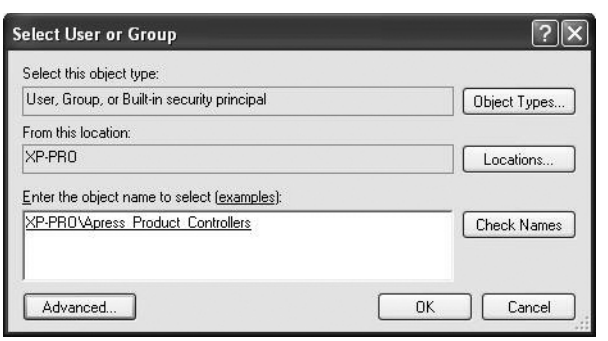


**Figure 4-8.** *Group found, ready for adding*

■ **Note** We are now back at the new login screen where the group will be populated. If we clicked OK at this point, this would only allow the group to connect to SQL Server and nothing else. Members of this group would therefore not be able to do anything.

10. So we need to give this group access to the databases we wish to allow them to use. It is vital that you only allow users or groups of users access to the resources they need and don't use the "allow everything, it's easier" approach that I have seen on my travels. We only want our users to see the `ApressFinancial` database, so we select that database on the Users mapped to this login section of the screen shown in Figure 4-9. For the moment, click the Script button. (When you select this option, it doesn't matter which of the three options you choose when selecting where to put the T-SQL.) We will come back to logins in the next section when we examine roles.

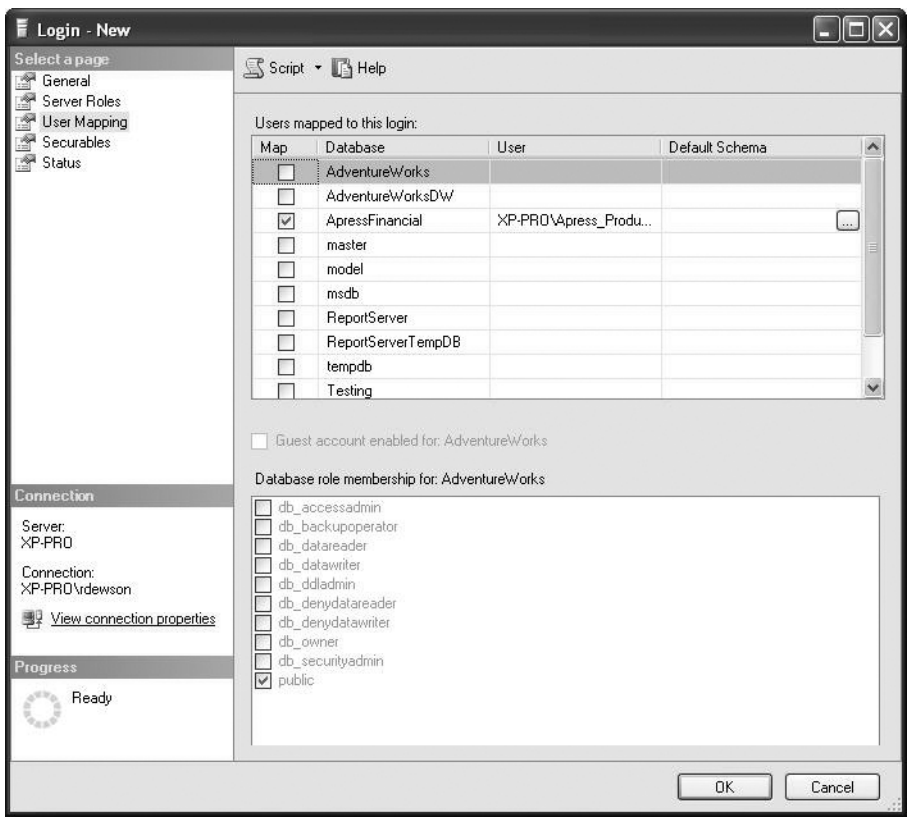


**Figure 4-9.** *Giving a login access to a database*

11. The SQL generated from Figure 4-9 follows. We will look at it in more detail in a moment when we examine more closely adding a login.

```
USE [master]
GO
CREATE LOGIN [XP-PRO\Apress_Product_Controllers]
FROM WINDOWS WITH DEFAULT_DATABASE=[master]
GO
USE [ApressFinancial]
GO
CREATE USER [XP-PRO\Apress_Product_Controllers]
FOR LOGIN [XP-PRO\Apress_Product_Controllers]
GO
```

12. Going back to SQL Server Management Studio, you can see in Figure 4-10 that we have moved to the Status page. Here we can grant or deny access to SQL Server for a Windows account, SQL Server login, or in our case Windows group. The second set of options is for enabling or disabling SQL Server logins. The final set of options, specific to SQL Server authentication, allows an account to be unlocked after it has been locked out.

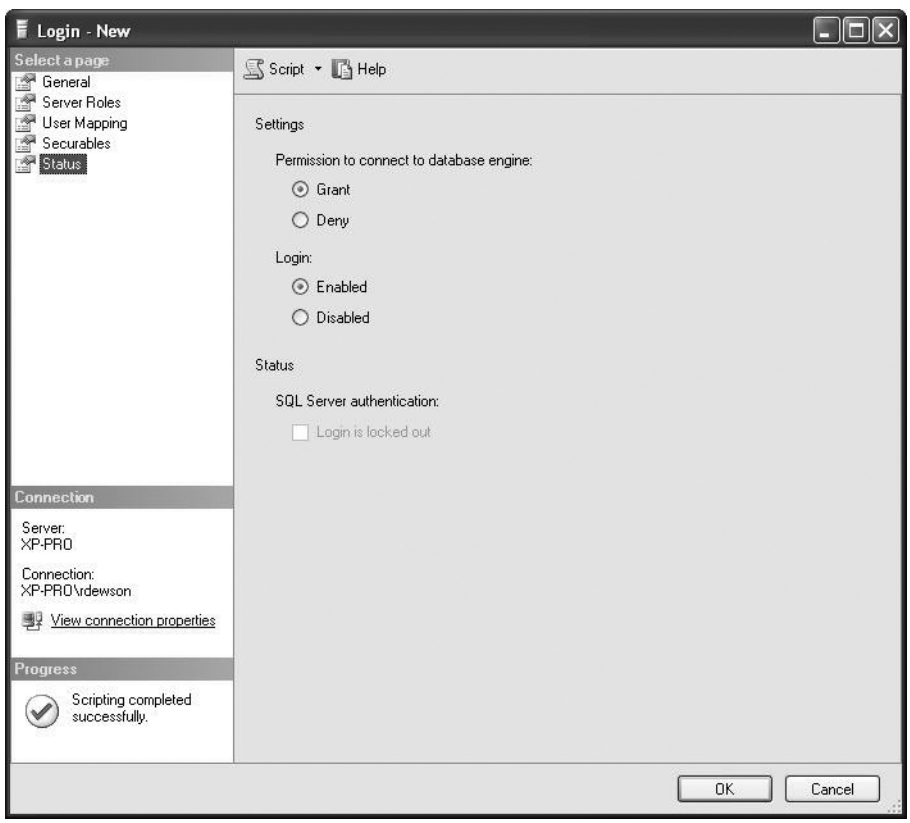


**Figure 4-10.** *Login status*

13. We can now click OK to add the group. This will complete the addition to SQL Server.

Now that we have created the new group and placed it within SQL Server, we could now switch the user account to AJMason and successfully connect. However, as AJMason, we would only be able to explore the `ApressFinancial` database we created in Chapter 3.

As I mentioned at the start of this discussion, the process would be the same if you wished to add a single user.

For SQL Server authentication, each user needs to be added separately. The process is very similar to that for adding users with Windows authentication, but you must specify a password expiration and enforce password complexity. This will force the Windows password policies for expiration and complexity that exist on this account to apply to this login's password.

So now that we have added a login graphically, the same can be achieved via a query pane using T-SQL code. We saw the code generated previously, and we will use it as the basis of our next login creation. This is a very straightforward process, so let's take a look at it next.

## Try It Out: Programmatically Working with a Login

1. From SQL Server, select New Query ➤ Database Engine Query. This should bring up an empty query pane similar to the one we saw in Chapter 2.

2. We want to add a second login group. We have available two different methods, and which one we use depends on whether we are going to use Windows authentication or SQL Server authentication. Our first example takes a look at the Windows authentication method. Locate the code from Steps 10 and 11 in the previous "Try It Out: Creating a Group" section (it is repeated below for ease of reference).

```
CREATE LOGIN [XP-PRO\Apress_Product_Controllers]
FROM WINDOWS
WITH DEFAULT_DATABASE=[master],
DEFAULT_LANGUAGE=[us_english]
GO
USE [ApressFinancial]
GO
CREATE USER [XP-PRO\Apress_Product_Controllers]
FOR LOGIN [XP-PRO\Apress_Product_Controllers]
GO
```

3. We can now alter this to create a group that will be defined for users wishing to view customers and their information, probably used in call centers, for example, for the Corporate edition of our software. Also, this time we are going to set the database that will be connected to by default, to our `ApressFinancial` database. Before entering the following code, we will of course need to add the new group, `Apress_Client_Information`, within our Computer Management icon found in the Administrative tools of the Control Panel first (see the "Try It Out: Creating a Group" section earlier for more on this). Once you've done this, enter the following code in a new Query Editor window. (Don't execute it yet.)

```
CREATE LOGIN [XP-PRO\Apress_Client_Information]
FROM WINDOWS
WITH DEFAULT_DATABASE=[ApressFinancial],
DEFAULT_LANGUAGE=[us_english]
GO
```

The format of this syntax is straightforward. In this case, `CREATE LOGIN` instructs SQL Server that you want to create a new login called `XP-PRO\Apress_Client_Information`, where XP-PRO is the name of the network domain in which the `Apress_Client_Information` group can be found. You should change the prefix to match your own setup. Here the definition appears surrounded with optional square brackets in case of spaces in the name.

Next the keywords `FROM WINDOWS` inform SQL Server that you are creating a login with Windows authentication. After that you define the name of the database that the login will connect to when a connection is made using `WITH DEFAULT_DATABASE`. Finally, the second option specifies the default language the connection will use, although it is possible at any time to alter the language using the Set Language option. This will allow this group to connect to SQL Server.

4. Once you have placed the code in your query pane, you can execute it by pressing either Ctrl+E or F5, or clicking the Execute button on the toolbar. Once it finishes executing, you should see the new login in the Security node within the Object Explorer on the left, as shown in Figure 4-11. If you right-click the new login and select Properties, you will see the same screen and details as we saw when we created the login graphically.


**Figure 4-11.** *Both logins created*

5. We can then give the login access to SQL Server or disable it by using the `ALTER LOGIN` command. It is also possible to alter the login's default database. In our graphical example, if you check back to Figure 4-5, you will see that the default database was called master. It would be better for the login to connect to the correct database. The following code informs SQL Server that it should connect our login to the `ApressFinancial` database by default, rather than the `master` database as defined previously. Remember to change the prefix as appropriate.

```
ALTER LOGIN [XP-PRO\Apress_Product_Controllers]
WITH DEFAULT_DATABASE=ApressFinancial
```

6. The final piece in the jigsaw is to grant the Windows account access to the database, which will then allow the login to use the `ApressFinancial` database. To do this, we need to switch from the `master` database to the `ApressFinancial` database with the `USE` keyword followed by the name of the database.

Using `CREATE USER`, we can then specify the name of the user we want in our database. The standard procedure is to use the same name as the login, which makes life so much easier when maintaining the system in general. We then use `FOR LOGIN` to define which server login we want to map to this database user.

```
USE ApressFinancial
GO
CREATE USER [XP-PRO\Apress_Client_Information]
FOR LOGIN [XP-PRO\Apress_Client_Information]
GO
```

## Server Logins and Database Users

As you now know, there are two steps to complete, whether you want to create a SQL Server authentication–based login or a Windows authentication–based login. The first is a server login, which was the first part of creating a login that we went through. A server login is one that, when used, can connect only to the server itself. It cannot use any of the user databases within SQL Server. The second step was creating the database user; in the graphical section that we looked at first, this is when we selected the databases we wanted to use.

Within SQL Server, permissions can be granted at multiple levels, including the server and database level. Examples of server-level permissions include creating new logins or managing server properties. Examples of database permissions include being able to read data from a table or being able to create new tables. One server login can be associated with multiple users in different databases. Generally, when using Windows authentication, a database username is the same as the login name, but this does not have to be the case. It does, however, simplify administration. In this book, we will mostly be dealing with database-level permissions, but we will briefly examine server roles in the following section.

# Roles

Three different types of roles exist within SQL Server: fixed server roles, database roles (which refers to the general roles included during installation of SQL Server; component-specific roles such as those for Reporting Services that are added when the component is installed; and user-defined roles), and application roles.

## Fixed Server Roles

Within SQL Server, specific predefined roles are set up to allow certain tasks and to restrict other tasks. Someone with the right permissions, such as a system administrator, can assign these roles to any user ID or group of user IDs within SQL Server.

If you look at the Server Roles node in the Object Explorer, you will see a list of roles as shown in Figure 4-12. But what do they mean? You get a little hint if you move to the Server Roles node within SQL Server Management Studio.


**Figure 4-12.** *Fixed server roles*

■ **Note** It is not possible to create your own server role.

These roles, available for anyone to use across the server, can perform the following tasks:

- `bulkadmin`: Run `BULK INSERT` statements.
- `dbcreator`: Create, alter, or drop databases as well as restore them.
- `diskadmin`: Administer disk files.
- `processadmin`: Kill a login running T-SQL code.
- `securityadmin`: Manage logins including passwords for SQL logins and login permissions.
- `serveradmin`: Administrate the server and carry out tasks such as changing options and even starting and shutting down the server.
- `setupadmin`: Work with more than one server with which they are linked and manage the linked server definitions.
- `sysadmin`: Perform any activity.

Server roles are static objects. They contain groups of actions that operate at the server level rather than at the database level. When creating a new login, you could assign these server roles to it if you wanted the login to carry out server actions as well as any database-related actions, if required.

If your Windows account belongs to the `BUILTIN/Administrators` group, then it automatically belongs to the `sysadmin` server role. You can check this yourself by highlighting the `sysadmin` server role, right-clicking it, and selecting Properties to bring up the dialog box shown in Figure 4-13. You should see `BUILTIN/Administrators` listed. As more logins are created, they can be added to this role via the Add button.

Although we are not going to alter this for our example database, having Windows XP administrators automatically being administrators for SQL Server can be a bit of a security issue. Many companies batten down their computers so that no user is an administrator of his or her local machine. By doing this, they stop people adding their own software, shareware, games, or whatever to a machine that is administrated and looked after by a support team.

This helps keep the machine stable, and throughout your organization everyone will know that a piece of software developed on one machine will work on any other. Therefore, users won't have administrator rights on their XP machine and won't have those rights in SQL Server. This is not the case in all organizations. By leaving the `Administrators` group in the `sysadmin` role, everyone who has administrator rights on their PC will have system administrator rights within SQL Server. As the owner of the database, you have now lost control of the security and development of your SQL Server database.

---

■ **Note** Because this book assumes that we're using either a standalone PC or a secure set of users, it is safe to keep the `Administrators` group. However, you will find that this group is usually removed from database setups to keep the security of the database intact. However, it is worth keeping in mind that before removing the login, or removing it from the sysadmin role, that you should set up a new group or user as a system administrator to prevent locking yourself out.

---

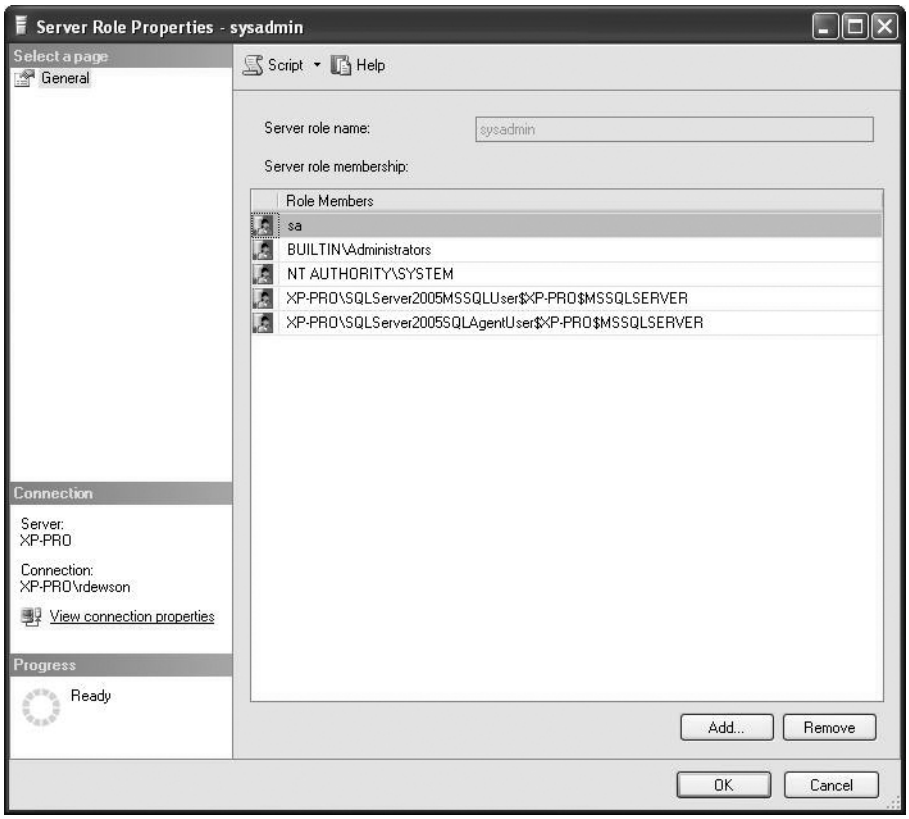


**Figure 4-13.** *Members of the sysadmin role*

## Database Roles

Database roles deal with actions that are performed at the database level. Actions within SQL Server can be grouped into different types of actions.

Following are the existing database roles installed with SQL Server and what they can or cannot do:

- `dbo/db_owner`: Specifies the owner of the database
- `db_accessadmin`: Can manage access to a database for logins
- `db_backupoperator`: Can back up the database
- `db_datareader`: Can read data from all user-defined tables
- `db_datawriter`: Can perform any write actions to user tables
- `db_ddladmin`: Can perform Data Definition Language (DDL) actions like creation of tables
- `db_denydatareader`: Cannot read data from user tables

- db_denydatawriter: Cannot write data from user tables
- db_securityadmin: Can modify database role membership and manage permissions
- public: Can see any database objects that are created with public, or full rights, access (Every user that you create will belong to the public database role.)

Although you will put the existing database roles to use, you'll find it helpful to create new database roles, a common task in SQL Server, when you want to be very specific about permissions particular users have. You do this by creating a specific database role, and then adding the Windows accounts/Windows groups/SQL Server logins to your role. If you wanted to group several groups together, then you might create a new role.

## Application Roles

Databases are written for applications. However, not all databases exist for just one application. Application roles allow you to define one role for accessing a database based on the application that is connecting, rather than having security for different groups of users or single users. Let's look at an example.

Consider a central database that holds client data. This database is in turn accessed from the sales order department, which has its own separate database. The client database is also accessed from the debt recovery department, which also has its own database.

As a database administrator, you may set up user groups for each application. Say you have a Debt Recovery group and a Sales Order Processing group. Debt Recovery would want to see information that was hidden from the Sales Order group, such as how in debt a customer is. But what if a user, such as AJMason, worked in both the debt recovery and sales order departments, in two different part-time jobs, for instance? While working as part of the Sales Order group, AJMason could see information that was not pertinent to that group.

You can set up an application role for Sales Order and another for Debt Recovery, thus removing the conflict of having two different sets of security settings for the one user. Also, when users move departments, you are not wasting time revoking one set of roles to give them a new set of roles for their new department.

An application role overrides any user security settings and is created for giving an application access to SQL Server. Therefore, the Sales Order Processing application would define the access for anybody using it.

An application role has no users; it is used when you wish to define what an application can access within your database and what it cannot. We need to create an application role for examples shown later in this book, so let's do this now.

### Try It Out: Creating a New Application Role

1. Navigate to the ApressFinancial database, expand the Security node, right-click Roles, and select New Application Role. In the dialog box that appears, enter a useful role name and a password as shown in Figure 4-14. This role will be for the banking application through which users will want to look at checks, cash withdrawals, etc.

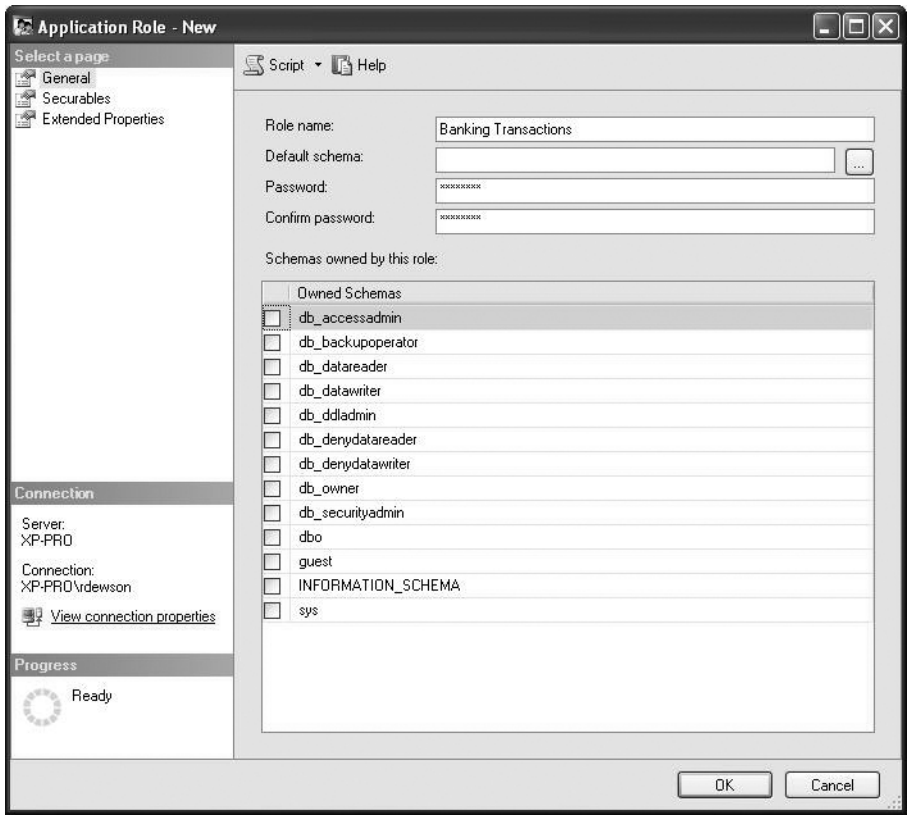


**Figure 4-14.** *Creating a new application role*

2. Click Securables in the Object Explorer on the left-hand side, and then click Add Objects. This is how we begin to define what objects we want to assign to this role.

3. In the Add Objects dialog box that appears, leave the options as they are shown in Figure 4-15 and click OK.

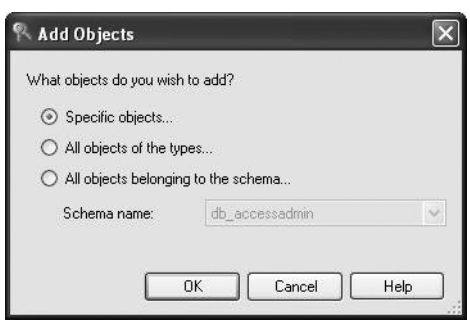


**Figure 4-15.** *Selecting the type of objects to add*

4. As we don't have much within our database that we can give permissions to just now, select Databases, as shown in Figure 4-16, and click OK. We are going to give this application authority to access our database, but not the ability to do anything (mainly because we don't have anything it can do yet).

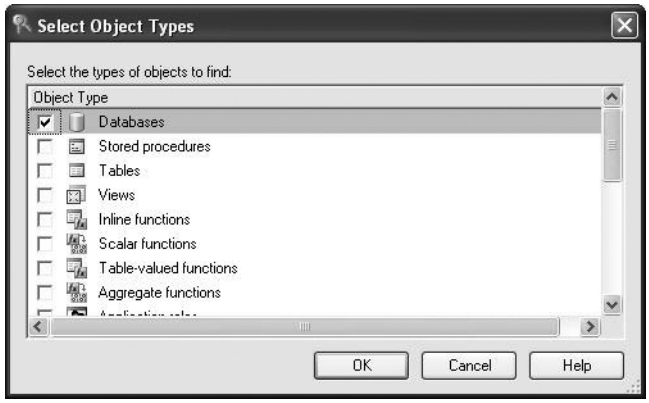


**Figure 4-16.** *Selecting the database*

5. We now see a list of all the databases within the server. As shown in Figure 4-17, select `ApressFinancial` as this is the only database this role will access.

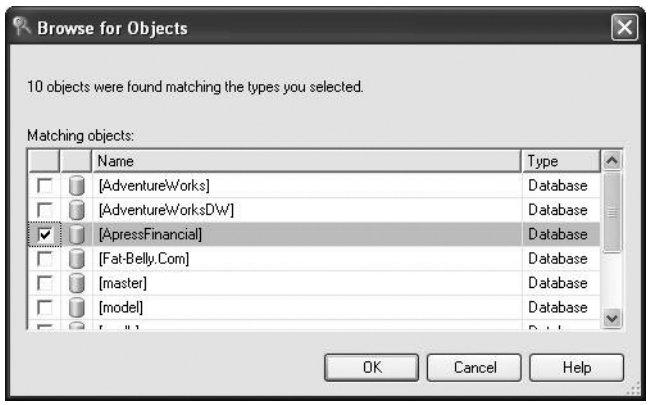


**Figure 4-17.** *ApressFinancial database selected*

6. This brings us back to the Securables screen where we can allow or deny specific actions, as you see in Figure 4-18. Leave everything unchecked for the moment; we will come back to this later in the book when we look at stored procedures in Chapter 10.

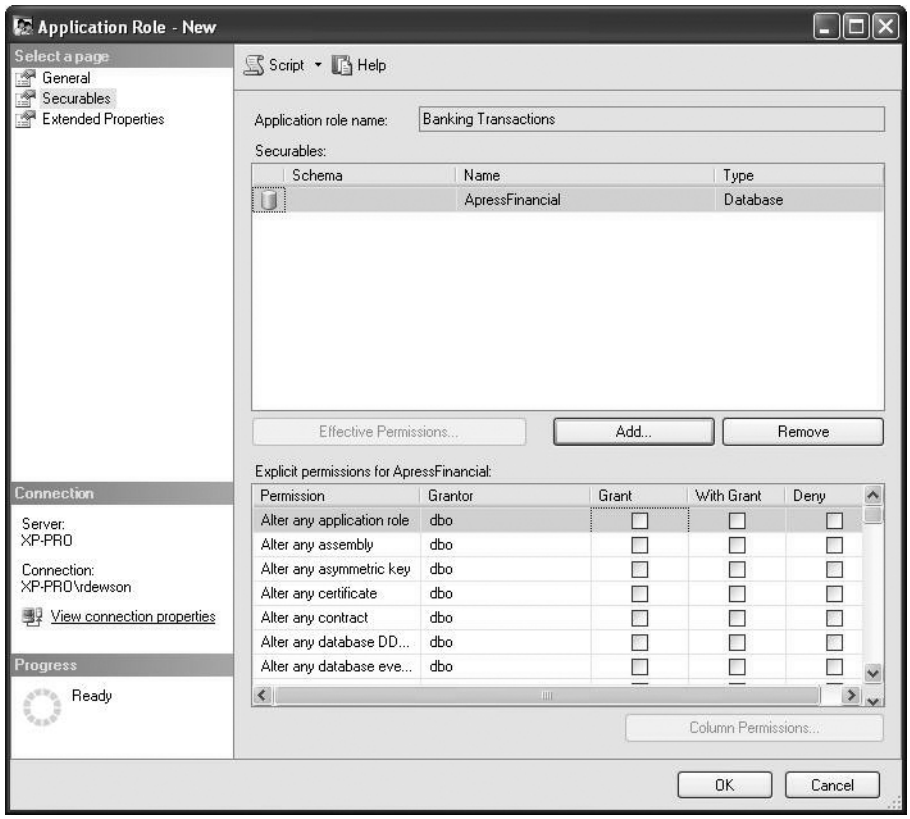


**Figure 4-18.** *Application roles explicit permission settings*

7. Click OK to finish creating our application role.

# Schemas

In the following chapters, we will be creating SQL Server objects to hold and work with our data. We could create these objects so that each could be seen as its own small part of the overall solution. It would make for better organization, though, if objects that could be seen as subsets of the whole solution were grouped together. For example, in our example, we could group share details and share prices together as share information, or group the financial transactions the customer makes using the transactions and transaction types tables together. These groupings could then be used as the basis of security to the underlying data for when a SQL Server connection tries to access the data. These groupings we have just talked about exist in SQL Server 2005 and are called **schemas**. Therefore, a schema is a method of creating a group and placing objects within that group, which can then be used to grant or revoke permissions as a group to SQL Server connections.

Prior to SQL Server 2005, each object was owned by a user account. Whenever a user left, quite often it would mean moving the ownership of objects for that user's account to a new account. As you can imagine, in a very large system this could take hours or even days to complete. Now objects are owned by schemas, and the many objects that exist will be contained within one schema in our very large system. Although a schema will still be owned by a SQL Server account, as we will see when we take a look at the syntax in a moment, the number of schemas should be a fraction of the number of objects, even in very large systems, and therefore moving ownership will be easier and faster.

So by having a schema within our solution and assigning objects to that schema, not only are we improving security, but we are also grouping logical units of the database together, making our solution easier to understand and use.

To create a schema is very simple, and the syntax is defined as follows:

```
CREATE SCHEMA schema_name AUTHORIZATION owner_name
```

We can now see this in action.

## Try It Out: Creating Schemas and Adding Objects

1. Open up a Query Editor window so we can create our first schema. This schema will be used to keep all our transaction details together. Enter the following code:

   ```
   USE ApressFinancial
   GO
   CREATE SCHEMA TransactionDetails AUTHORIZATION dbo
   ```

2. When we execute this by pressing F5 or Ctrl+E, or clicking the Execute button, we should see it successfully complete. This will have created a new schema where the owner of the schema is the `dbo` user. This means that any login with `sysadmin` privileges will automatically have access to this schema because they map to the `dbo` user in all databases. If you execute the code successfully, you'll see the following message:

   ```
   Command(s) completed successfully.
   ```

3. We can then create further schemas for other groupings, such as one for share details or customer details including products. Enter the following code:

   ```
   CREATE SCHEMA ShareDetails AUTHORIZATION dbo
   GO
   CREATE SCHEMA CustomerDetails AUTHORIZATION dbo
   ```

4. Execute this code, which will add the schemas to the database. If you execute the code successfully, you'll see the following message:

   ```
   Command(s) completed successfully.
   ```

5. If we move to the Object Explorer, we can see our schemas in place, as shown in Figure 4-19.

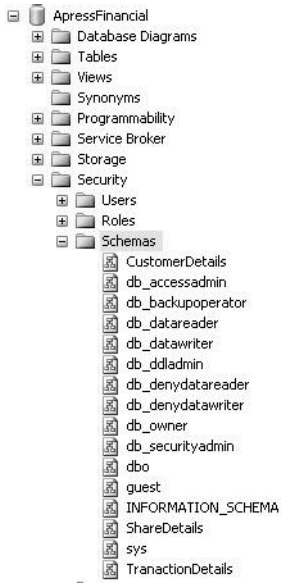


**Figure 4-19.** *Schemas applied*

# Before You Can Proceed with Your Solution

You have now created a database and gained an understanding of the different roles in SQL Server. Before you can proceed and create objects such as tables, you need to clear a couple of obstacles. After the database, the next objects you will create are tables, which you will learn about in the next chapter. So what security considerations do you need to check before you can do this?

First of all, you must be using a database user account that has the authority to add tables to the specific database you are required to, in this case `ApressFinancial`. This is the next security issue we will tackle, and you should keep in mind what you learned in the previous chapter about accounts and roles. You also need to have access to that specific database. Let's look at the issue of access to the database if you are using a user ID that did not create the database.

The database was re-created at the very end of the previous chapter under user ID `XP-PRO\RDewson`. The user who created the database is the database owner, also known as `dbo`. So how can you check who created the database if you did not? At the end of the last chapter, I asked you to create the database under your own user ID, which, if you followed the instructions so far and you are a local administrator of the machine SQL Server is installed on, you should have the right privileges within SQL Server to do.

If you are working with SQL Server already installed on an XP/W2K(3) machine, you need to ensure that your user ID is set up as an administrator user ID, as demonstrated in Chapter 1, or set up specifically as an administrator within SQL Server.

This next section will demonstrate how you check the identity of the database owner.

## Try It Out: Checking the Database Owner

1. Ensure that SQL Server Management Studio is open.
2. Navigate to the database that you wish to check on, in this case `ApressFinancial`.
3. Click the `ApressFinancial` database node in the Object Explorer on the left-hand side of the screen once, and then right-click.
4. Select Properties to bring up the Database Properties dialog box shown in Figure 4-20. On the General tab, you will see an item named Owner. This is the fully qualified XP/Win2K account preceded by the domain or local machine name.

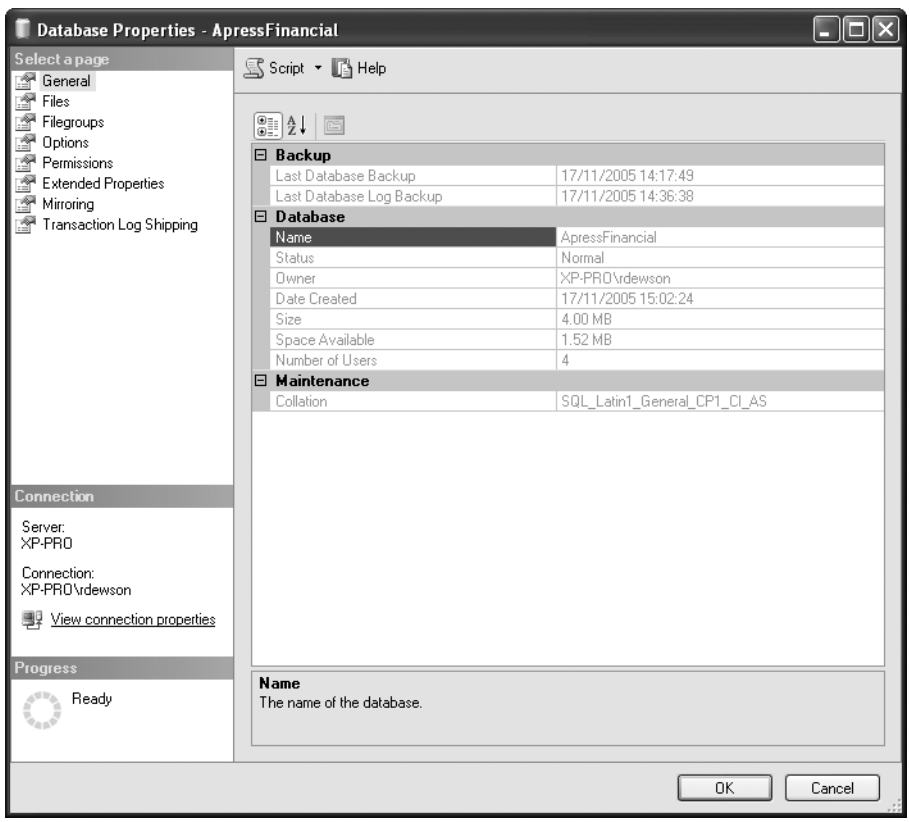


**Figure 4-20.** *Database Properties*

5. Click Cancel to close this dialog box.

Ownership of tables and other database objects is just as important. If you create a table using the same login ID as that which you created the database with, or use a logon ID that is a

member of the `sysadmin` role that is also implicitly mapped to the `dbo` user in the database, the table will have a default schema of `dbo`. However, if you logged in with a different user ID, the table would have that user's default schema as the prefix to the table name, replacing the `dbo` prefix.

Now that we know who the database owner is, it is up to that user, or another user who has system administration rights (in other words, a login that has the `sysadmin` server role or has the `db_owner` database role), to allow any other specified user the ability to create tables within the database. We have a user called AJMason who is not a system administrator, but a developer. Recall we created this user in Chapter 1, and that this user could not log in to SQL Server.

The next section will go through a scenario where, as a developer, AJMason has no rights to create any new items. However, we will rectify this situation in the next section, where we will alter AJMason so that he can connect to SQL Server and create a table.

## Try It Out: Allowing a User to Create a Table

1. Log on to SQL Server as a `sysadmin`. Create a new login by clicking the Logins node on the Server Security node. This brings up the new login screen, which we can populate with the login name of the user by typing in the details of the login, as shown in Figure 4-21. We are also going to allow this user to connect to `ApressFinancial` by default when he or she logs in.

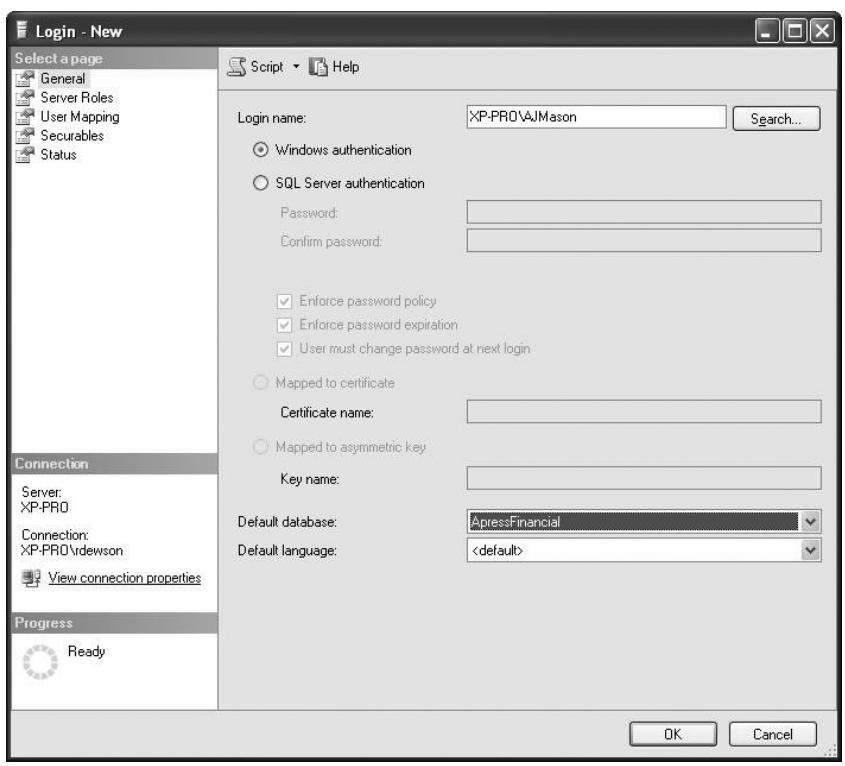


**Figure 4-21.** *New login*

2. We are not going to assign this user any server roles, but we are going to assign this user to the `db_owner` role, as you see in Figure 4-22. This will allow the user to create tables as well as create and work with other objects and data. We could have selected `db_ddladmin`, but this would only have allowed the user to create objects and not create data.

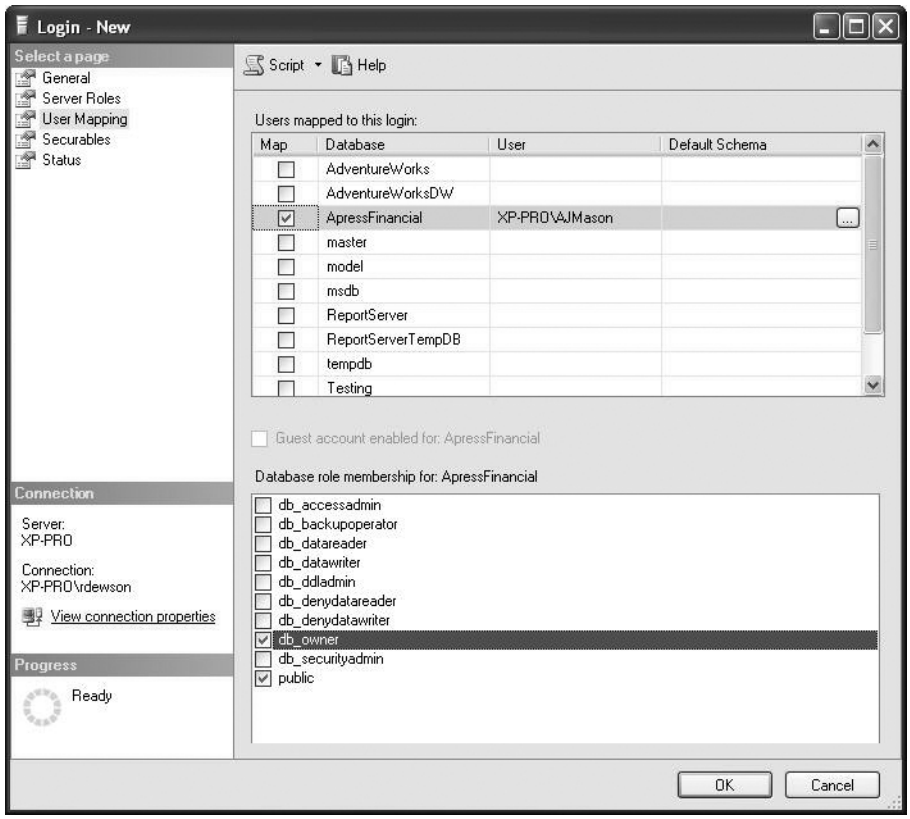


**Figure 4-22.** *New login with database access*

3. We now click OK, which will create not only a server login, but also a database user in `ApressFinancial` for AJMason, as shown in Figure 4-23.

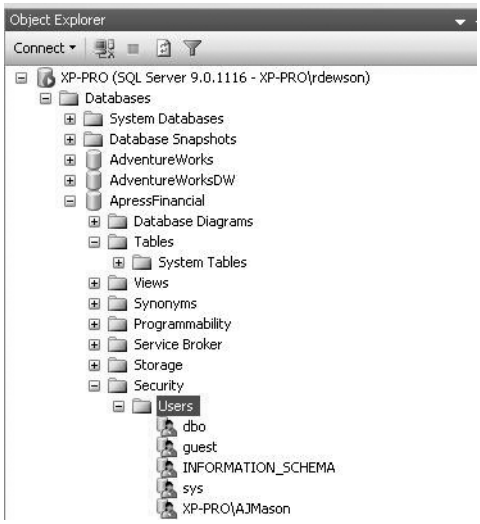


**Figure 4-23.** *User login accounts*

AJMason is now in a position to log in to SQL Server and create tables in the `ApressFinancial` database.

# Summary

There is a great deal to cover concerning security and its different aspects. I would like to just recap everything that we have seen just for one last time to ensure that you understand how everything fits together.

Before you can connect to SQL Server, an administrator of the SQL Server installation must give you permission to connect. In a Windows authentication setup, the administrator would either allow your Windows account or a group that contains your Windows account to connect to SQL Server. He or she can do this by either using the GUI and creating a login via the Security node or using the `CREATE LOGIN ... FROM WINDOWS` T-SQL statement. If you are in a SQL Server authentication setup, then a user ID and password would be created within SQL Server, again either via the Security/Logins node or by using the `CREATE LOGIN ... PASSWORD = 'password'` syntax.

Once a connection has been made, you can create a user login within the database using the `CREATE USER ...` syntax. This will allow either the Windows account or the SQL Server login access to the database.

It is then possible to place the user into a role: either a predefined role or, more likely, a custom role that you create. This role can be used to determine what can and cannot be accessed within SQL Server tables, views, stored procedures, and any other object. Therefore, a role allows groups of users in one statement to be granted or revoked access to objects within SQL Server. Without roles, as new people join and as old people leave, or people move between departments, you would need to grant or revoke privileges as required—quite an onerous task.

Finally, when creating objects. as you will see in the next few chapters, these objects are owned by schemas. This allows for groups of objects to belong to a specific schema rather than a specific user login. This also reduces the overhead of granting privileges and allows the grouping of objects that belong together, making your application easier to understanding.

This chapter continued our coverage of security within SQL Server 2005. At this point in the book, you now know about SQL Server authentication and Windows authentication, and you have discovered how to control access to databases. Even during the installation process, the `sa` login and password enforcement were discussed on that special account. Our discussions on security are by no means finished because there are still several areas that we need to explore together, which we will do as we go through the book.

Security is the most important part of ensuring that your organization continues to have the ability to work. A security breach could result in lost income and will certainly mean that many people will be unable to do their work. It can also lead to unfulfilled orders, backlogs, or even fraudulent transactions. Regardless of whether you have the most well-designed database or the most poorly performing application ever, if you allow the wrong person into the wrong database, the result will be catastrophic.