Phase

1

# Installing and Configuring Microsoft SQL Server 2005

Many people are so excited to start working with their new software that they just start installing it without verifying they meet the prerequisites; you may have even done this yourself in the past. To ensure a successful installation of SQL Server 2005, though, you need to make sure you have the right hardware and software in place first; otherwise, you will end up with a mess. Once you have installed SQL Server 2005, you then need to configure it before you can let your users start working with it. The tasks in this phase will show you how to successfully install and configure SQL Server 2005.

# Task 1.1: Verifying Prerequisites and Installing the Default Instance

In this task, you will verify that your machine meets the prerequisites for installing SQL Server 2005, and then you will install the default instance.

## Scenario

You are the database administrator (DBA) for a midsize company with offices in various cities throughout the United States and Canada. The company has decided to use SQL Server 2005 for data storage and retrieval, and you have been asked to install the software and get it running.

As an experienced DBA, you understand the importance of installing the software right the first time, because if you install SQL Server incorrectly or on the wrong hardware, it will work slowly or not at all. Therefore, you have decided to verify the prerequisites and then install the software.

## Scope of Task

### Duration

This task should take less than one hour.

### Setup

This task requires little setup. All you need is access to a copy of SQL Server 2005 Enterprise Edition and a computer that meets the requirements to run it.

### Caveat

It seems redundant, but SQL Server 2005 runs better on faster hardware. So, remember that the minimum requirements listed later in this task are just that, minimum. If you have access to a faster machine with more random access memory (RAM), use it.

## Procedure

In this task, you will verify that your computer meets the requirements for running SQL Server 2005 and then install the default instance.

### Equipment Used

Although several editions of SQL Server 2005 exist, you will be working with the Enterprise Edition in this book because it has the widest range of available features. You can download a 180-day trial copy of Enterprise Edition from the Microsoft website (`http://www.microsoft.com/sql`). You will also need access to a machine that meets the prerequisites for Enterprise Edition.

### Details

First, you must make certain your server meets the requirements for installing SQL Server 2005. Table 1.1 lists the prerequisites for installing the Standard, Developer, and Enterprise Editions.

**TABLE 1.1** Developer/Standard/Enterprise Edition Requirements

| Component | 32-bit | x64 | Itanium |
|---|---|---|---|
| Processor | 600MHz Pentium III–compatible or faster processor; 1GHz or faster processor recommended | 1GHz AMD Opteron, AMD Athlon 64, Intel Xeon with Intel EM64T support, Intel Pentium IV with EM64T support processor | 1GHz Itanium or faster processor |
| Memory | 512MB of RAM or more; 1GB or more recommended | 512MB of RAM or more; 1GB or more recommended | 512MB of RAM or more; 1GB or more recommended |
| Disk drive | CD or DVD drive | CD or DVD drive | CD or DVD drive |

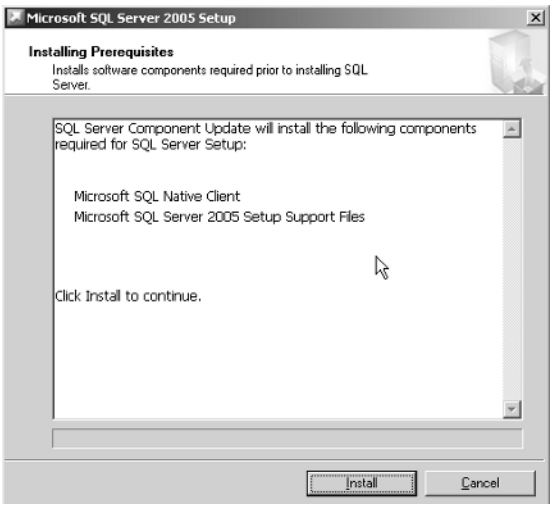
**TABLE 1.1** Developer/Standard/Enterprise Edition Requirements *(continued)*

| Component | 32-bit | x64 | Itanium |
|---|---|---|---|
| Hard disk space | Approximately 350MB of available hard disk space for the recommended installation with approximately 425MB of additional space for SQL Server BOL, SQL Server Mobile BOL, and sample databases | Approximately 350MB of available hard disk space for the recommended installation with approximately 425MB of additional space for SQL Server BOL, SQL Server Mobile BOL, and sample databases | Approximately 350MB of available hard disk space for the recommended installation with approximately 425MB of additional space for SQL Server BOL, SQL Server Mobile BOL, and sample databases |
| Operating system | Microsoft Windows 2000 Server with SP4 or newer; Windows 2000 Professional Edition with SP4 or newer; Windows XP with SP2 or newer; Windows Server 2003 Enterprise Edition, Standard Edition, or Datacenter Edition with SP1 or newer; Windows Small Business Server 2003 with SP1 or newer | Microsoft Windows Server 2003 Standard x64 Edition, Enterprise x64 Edition, or Datacenter x64 Edition with SP1 or newer; Windows XP Professional x64 Edition or newer | Microsoft Windows Server 2003 Enterprise Edition or Datacenter Edition for Itanium-based systems with SP1 or newer |

Second, after you have verified that your machine can handle SQL Server 2005, you can begin the installation. Follow these steps (which are for installing Enterprise Edition, but the steps are similar for all editions):

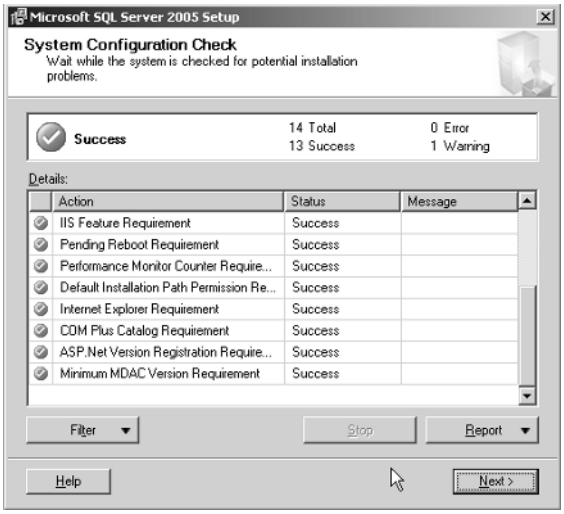
1. You need to create a service account, so create a user account named SqlServer, and make it a member of the Administrators local group. You can perform this task using one of these tools:
   - On a Windows member server or on Windows XP, use Computer Management.
   - On a Windows domain controller, use Active Directory Users and Computers.
2. Insert the SQL Server CD, and wait for the automenu to open.
3. Under Install, choose Server Components ➢ Tools ➢ Books Online ➢ Samples.
4. You will then be asked to read and agree with the end user license agreement (EULA); check the box to agree, and click Next.

5. If your machine does not have all the prerequisite software installed, the setup will install them for you at this time. Click Install if you are asked to do so. When complete, click Next.

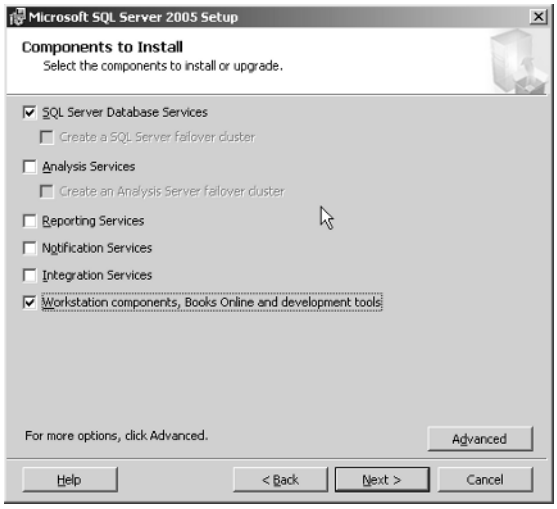



6. Next you will see a screen telling you that the setup is inspecting your system's configuration again, and then the welcome screen appears. Click Next to continue.

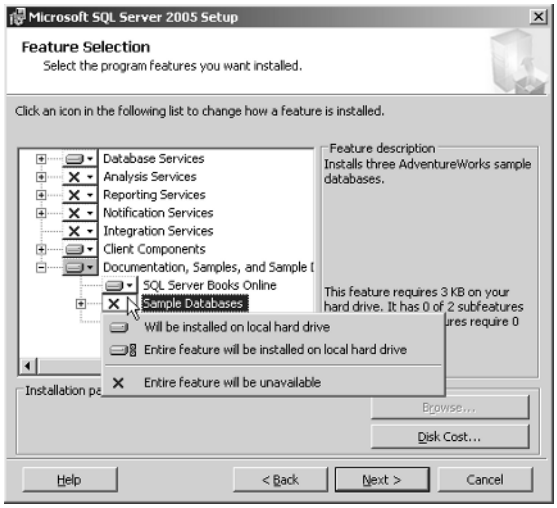
7. Another, more in-depth system configuration screen appears letting you know whether any configuration settings will prevent SQL Server from being installed. You need to repair errors (marked with a red icon) before you can continue. You can optionally repair warnings (marked with a yellow icon), which will not prevent SQL Server from installing. Once you have made any needed changes, click Next.



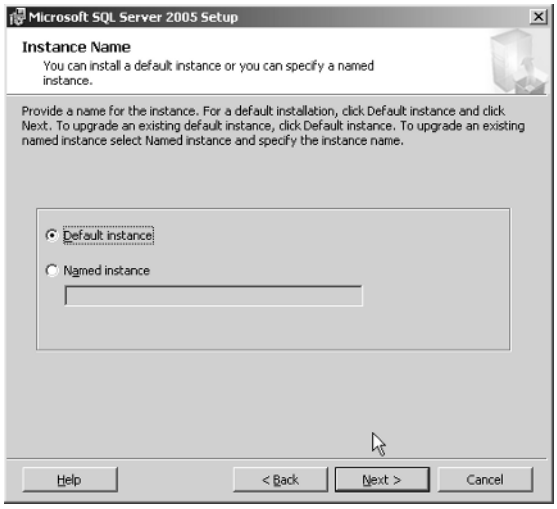
8. After a few configuration setting screens, you will be asked for your product key. Enter it, and click Next.
9. On the next screen, you need to select the components you want to install. Check the boxes next to the SQL Server Database Services option and the Workstation Components, Books Online and Development Tools option.



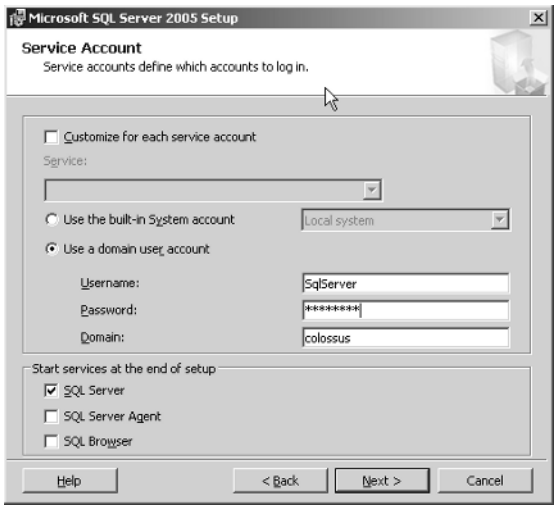

10. Click the Advanced button to view the advanced options for the setup.
11. Expand Documentation ➢ Samples ➢ Sample Databases. Then click the button next to Sample Databases, select Entire Feature Will Be Installed on Local Hard Drive, and then click Next. (This will install the AdventureWorks database you will be using later in this book.)



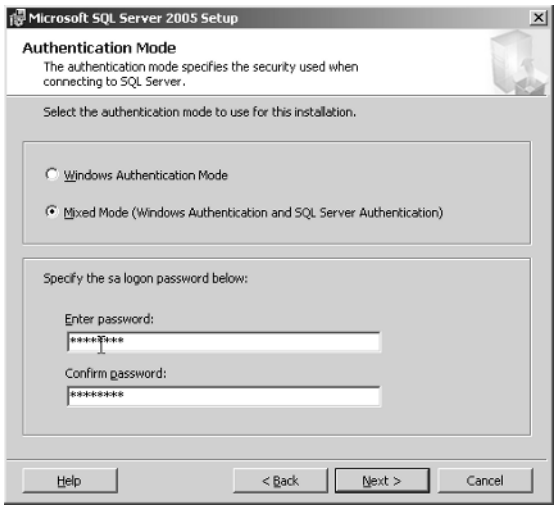
**12.** On the Instance Name screen, choose Default Instance, and click Next.



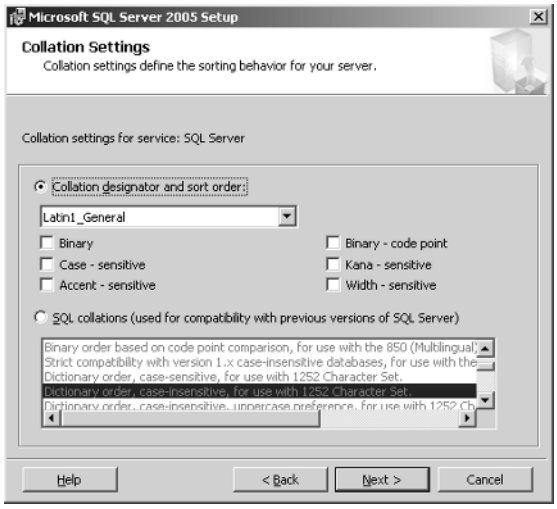

**13.** On the next screen, enter the account information for the service account you created in step 1. You will be using the same account for each service. When finished, click Next.



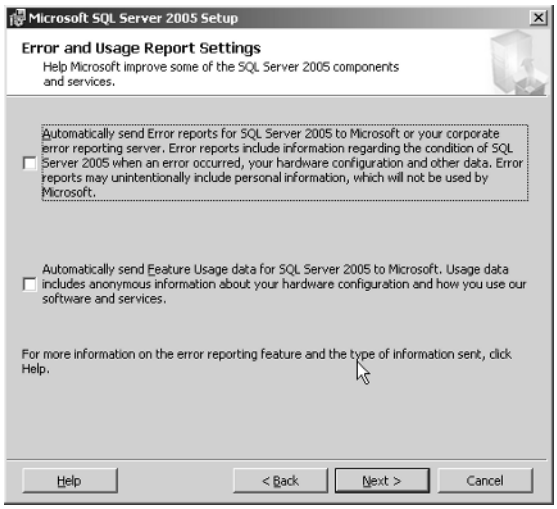
**14.** On the Authentication Mode screen, select Mixed Mode, enter a password for the sa account, and click Next.




**15.** Select the Latin1_General collation designator on the next screen, and click Next.



16. On the following screen, you can select to send error and feature usage information directly to Microsoft. You will not be enabling this function here. So, leave the defaults, and click Next.

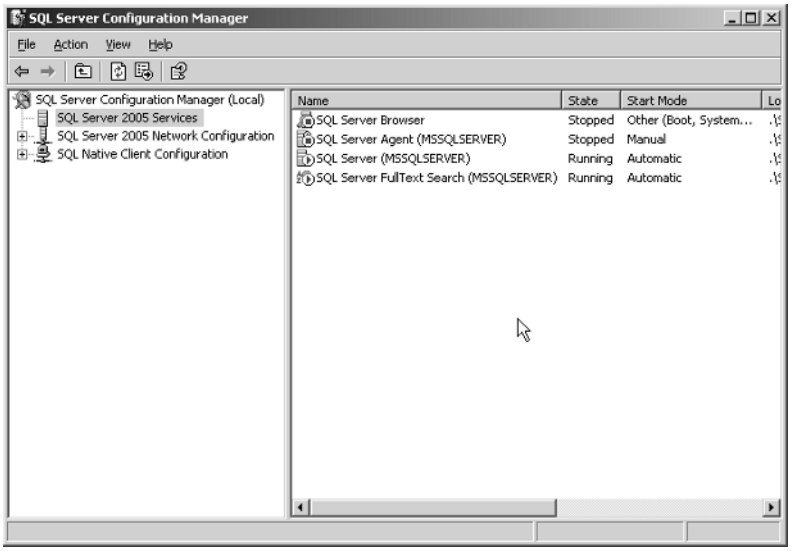



17. On the Ready to Install screen, review your settings, and then click Install.
18. The setup progress appears during the install process. When the setup is finished (which may take several minutes), click Next.
19. The final screen gives you an installation report, letting you know whether any errors occurred and reminding you of any post-installation steps to take. Click Finish to complete your install.
20. Reboot your system if requested to do so.

## Criteria for Completion

You have completed this task when you have a running instance of SQL Server 2005 installed on your system. To verify this, select Start ➢ All Programs ➢ Microsoft SQL Server 2005 ➢ Configuration Tools ➢ SQL Server Configuration Manager. Select SQL Server 2005 Services, and check the icons. If the icon next to SQL Server (MSSQLServer) service is green, then your installation is a success (see Figure 1.1).

**FIGURE 1.1** Check SQL Server Configuration Manager to see whether your services are running after installation.




# Task 1.2: Installing a Second Instance

Using a technique called *instancing*, you can have more than one copy of SQL Server 2005 running on the same computer at the same time. Each instance has its own set of system databases and its own security system in place. This is useful if you need a server for production and another for testing but you do not have enough hardware for two separate physical machines or you need to have two or more systems with disparate security settings. Throughout this book, you will be using a named instance in order to get some practice with the more advanced features of SQL Server 2005 (such as replication). In this task, you will install a second instance of SQL Server 2005 on the same machine as the default instance.

## Scenario

You are the database administrator (DBA) for a midsize company with offices in various cities throughout the United States and Canada. You know you need an instance of SQL Server 2005 for testing new service packs, new database schemas, and the like, but your company does not have the budget for new hardware at this time. The only way for you to have a test copy of SQL Server is to install a named instance.

## Scope of Task

### Duration

This task should take less than one hour.

### Setup

Again, all you need for this task is the machine you used in Task 1.1 and the same copy of SQL Server 2005 you used in Task 1.1.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will install a second instance of SQL Server on the same machine used in Task 1.1.

### Equipment Used

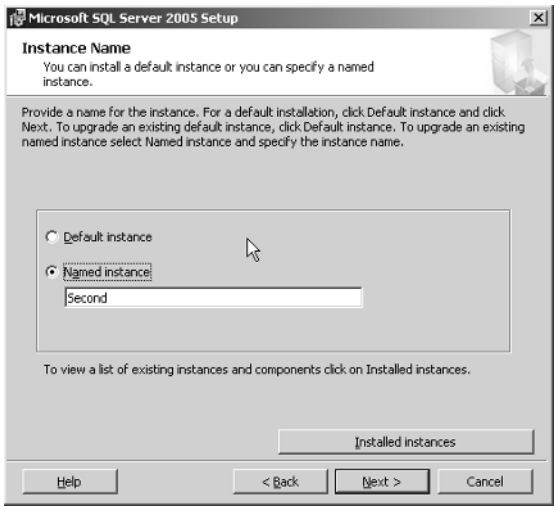
All you need for this task is the machine you used in Task 1.1 and the same copy of SQL Server 2005 you used in Task 1.1.

### Details

Follow these steps to create a second instance of SQL Server on the same machine as the default instance:

1. Insert the SQL Server CD, and wait for the automenu to open.
2. Under Install, choose Server Components ➢ Tools ➢ Books Online ➢ Samples.
3. You will then be asked to read and agree with the end user license agreement (EULA); check the box to agree, and click Next.
4. If your machine does not have all the prerequisite software installed, the setup will install them for you at this time. Click Install if you are asked to do so. When complete, click Next.
5. Next you will see a screen telling you the setup is inspecting your system's configuration again, and then the welcome screen appears. Click Next to continue.
6. Another, more in-depth system configuration screen appears letting you know whether any configuration settings will prevent SQL Server from being installed. You need to repair errors (marked with a red icon) before you can continue. You can optionally repair warnings (marked with a yellow icon), which will not prevent SQL Server from installing. Once you have made any needed changes, click Next.

7. After a few configuration setting screens, you will be asked for your registration information. Enter it, and click Next.
8. On the next screen, you need to select the components you want to install. Check the box next to SQL Server Database Services, and click Next.
9. On the Instance Name screen, choose Named Instance, and in the text box enter **Second**. Then click Next.

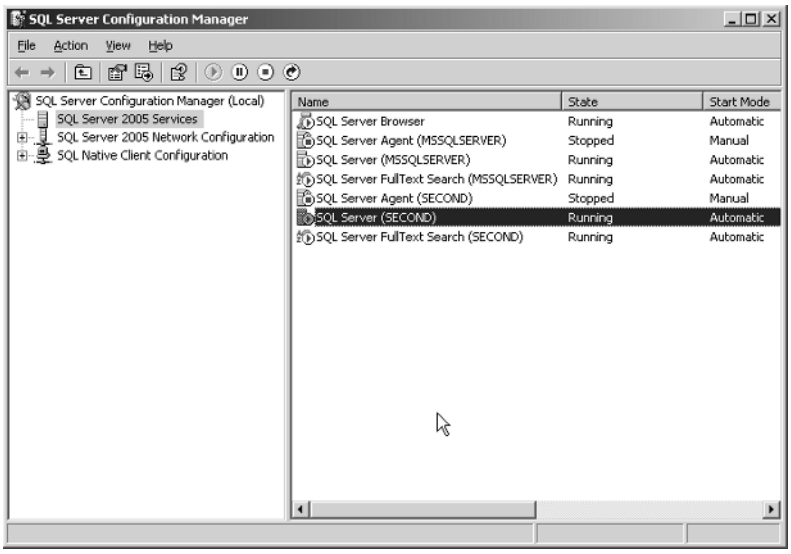



10. On the next screen, enter the account information for the service account you created in step 1 of Task 1.1. You will be using the same account for each service. When finished, click Next.
11. On the Authentication Mode screen, select Mixed Mode, enter a password for the sa account, and click Next.
12. Select the Latin1_General collation designator on the next screen, and click Next.
13. On the following screen, you can select to send error and feature usage information directly to Microsoft. You will not be enabling this function here. So, leave the defaults, and click Next.
14. On the Ready to Install screen, you can review your settings, and then click Install.
15. The setup progress appears during the install process. When the setup is finished (which may take several minutes), click Next.
16. The final screen gives you an installation report, letting you know whether any errors occurred and reminding you of any post-installation steps to take. Click Finish to complete your install.
17. Reboot your system if requested to do so.

## Criteria for Completion

You have completed this task when you have a second running instance of SQL Server 2005 installed on your system. To verify this, select to Start ➢ Microsoft SQL Server 2005 ➢ Configuration Tools ➢ SQL Server Configuration Manager. Select SQL Server 2005 Services, and refer to the icons. If the icon next to SQL Server (Second) instance is green, then your installation is a success (see Figure 1.2).

**FIGURE 1.2** Check SQL Server Configuration Manager to see whether your services are running for the Second instance.




# Task 1.3: Designing and Creating a Database

SQL Server 2005 stores your database information in two types of files: one or more database files and one or more transaction log files. As a database administrator (DBA), it is your duty to design and create databases to store user data and other objects. As part of your role as a database creator, you must decide how large to make these database files and what type of growth characteristics they should have, as well as their physical placement on your system.

In this task, you will decide where to put the data and log files and then create a database.

## Scenario

You are the DBA for a midsize company with offices in various cities throughout the United States and Canada. You have just installed a new instance of SQL Server, and now you need to create a database to hold data for your sales department.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

All you need for this task is access to the machine you installed SQL Server 2005 on in Task 1.1.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a database that will hold data for the sales department. You will use this database in later tasks for storing other database objects as well.

### Equipment Used

All you need for this task is access to the machine you installed SQL Server 2005 on in Task 1.1.

### Details

First, you must decide where to put the data and log files. Here are some guidelines to use:

- Data and log files should be on separate physical drives so that, in case of a disaster, you have a better chance of recovering all data.
- Transaction logs are best placed on a RAID-1 array because this has the fastest sequential write speed.
- Data files are best placed on a RAID-5 array because they have faster read speed than other RAID-arrays.
- If you have access to a RAID-10 array, you can place data and log files on it because it has all the advantages of RAID-1 and RAID-5.

Next, you must decide how big your files should be. Data files are broken down into 8KB pages and 64KB extents (eight contiguous pages). To figure out how big your

database will need to be, you must figure out how big your tables will be. You can do that using these steps:

1. Calculate the space used by a single row of the table.
   a. To do this, add the storage requirements for each datatype in the table.
   b. Add the null bitmap using this formula: `null_bitmap = 2 + ((number of columns + 7) /8)`.
   c. Calculate the space required for variable length columns using this formula: `variable_datasize = 2 + (num_variable_columns X 2) + max_varchar_size`.
   d. Calculate the total row size using this formula: `Row_Size = Fixed_Data_Size + Variable_Data_Size + Null_Bitmap + Row_Header`.

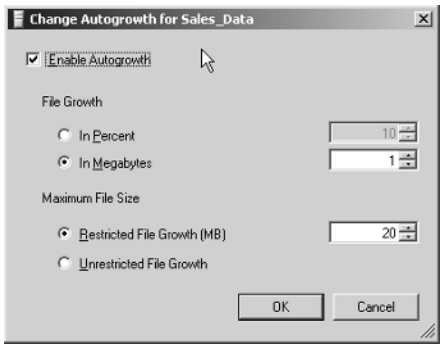

The row header is always 4 bytes.

2. Calculate the number of rows that will fit on one page. Each page is 8,192 bytes with a header, so each page holds 8,096 bytes of data. Therefore, calculate the number of rows using this formula: `8096 ÷ (RowSize + 2)`.
3. Estimate the number of rows the table will hold. No formula exists to calculate this; you just need to have a good understanding of your data and user community.
4. Calculate the total number of pages that will be required to hold these rows. Use this formula: `Total Number of Pages = Number of Rows in Table / Number of Rows Per Page`.

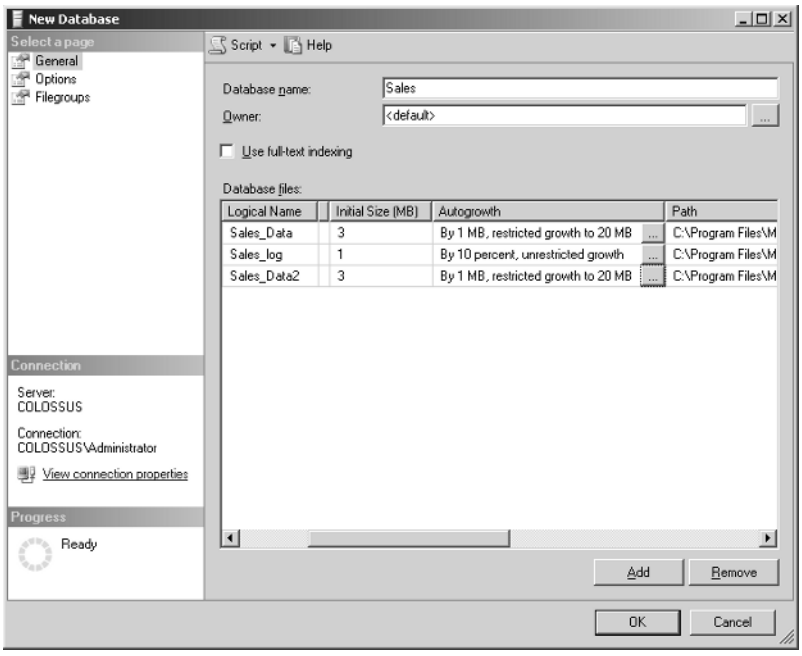
Once you have decided where to put your files and how big they should be, follow these steps to create a database named Sales (you will be creating the files on a single drive for simplicity):

1. Start SQL Server Management Studio by selecting Start ➢ Programs ➢ Microsoft SQL Server 2005 ➢ Management Studio.
2. Connect to your default instance of SQL Server.
3. Expand your Databases folder.
4. Right-click either the Databases folder in the console tree or the white space in the right pane, and choose New Database from the context menu.
5. You should now see the General tab of the Database properties sheet. Enter the database name **Sales**, and leave the owner as <default>.
6. In the data files grid, in the Logical Name column, change the name of the primary data file to **Sales_Data**. Use the default location for the file, and make sure the initial size is 3.

7. Click the ellipsis button (the one with three periods) in the Autogrowth column for the `Sales_Data` file. In the dialog box that opens, check the Restricted File Growth radio button, and restrict the filegrowth to 20MB.




8. To add a secondary data file, click the Add button, and change the logical name of the new file to **Sales_Data2**. Here too use the default location for the file, and make sure the initial size is 3.
9. Restrict the filegrowth to a maximum of 20MB for `Sales_Data2` by clicking the ellipsis button in the Autogrowth column.
10. Leave all of the defaults for the `Sales_Log` file.

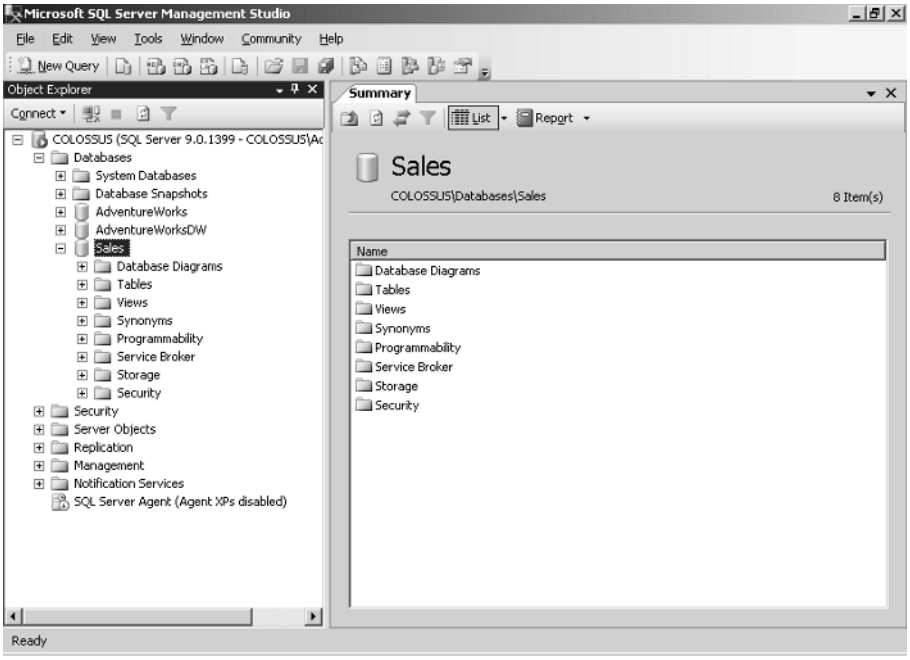



11. Click OK when you are finished. You should now have a new Sales database.

## Criteria for Completion

You have completed this task when you have a database named Sales that you can see in SQL Server Management Studio, as shown in Figure 1.3.

**FIGURE 1.3** You should see a database named Sales in SQL Server Management Studio.




# Task 1.4: Designing and Creating a Table

All of the data in a database is logically organized into tables. Each table in turn is divided into columns and rows (or fields and records). Each table holds a grouping of related data. For example, if you are creating a database to hold sales information, you may have one table for available products, another table for orders placed, another for salesperson territories, and so on.

To create a table, you need to know what data will be stored in the table so you can decide which columns should be in the table and what datatype each column should be. These decisions are different for each database in each company. In this task, you will be creating some tables in your Sales database.

## Scenario

You have just created a database for your sales department, and now you need to create some tables to hold customer, product, and order data.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

All you need for this task is access to the machine you installed SQL Server 2005 on in Task 1.1 and the Sales database you created in Task 1.3.

### Caveat

Remember that this is just an exercise. In the real world, you would probably have multiple tables for each of these categories. For example, you would have an OrderHeader table and an OrderDetails table to store multiple line items for a single order. You are creating a single table for each category in this task only for the sake of simplicity.

## Procedure

In this task, you will create three tables in your Sales database: a Products table, a Customers table, and an Orders table.

### Equipment Used

All you need for this task is access to the machine you installed SQL Server 2005 on in Task 1.1 and the Sales database you created in Task 1.3.

### Details

You will be creating three tables based on these criteria, as shown in Table 1.2, Table 1.3, and Table 1.3.

**TABLE 1.2** Products Table Fields

| Field Name | Datatype | Contains |
|---|---|---|
| ProdID | Int, Identity | A unique ID number for each product that can be referenced in other tables to avoid data duplication |

**TABLE 1.2** Products Table Fields *(continued)*

| Field Name | Datatype | Contains |
|---|---|---|
| Description | Nvarchar(100) | A brief text description of the product |
| InStock | Int | The amount of product in stock |

**TABLE 1.3** Customers Table Fields

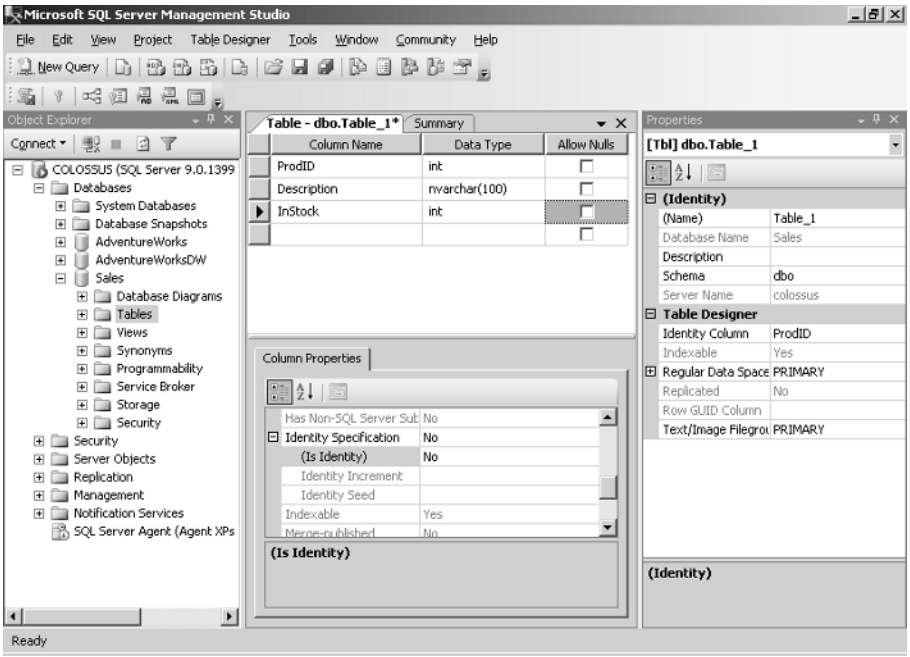
| Field Name | Datatype | Contains |
|---|---|---|
| CustID | Int, Identity | A unique number for each customer that can be referenced in other tables |
| Fname | Nvarchar(20) | The customer's first name |
| Lname | Nvarchar(20) | The customer's last name |
| Address | Nvarchar(50) | The customer's street address |
| City | Nvarchar(20) | The city where the customer lives |
| State | Nchar(2) | The state where the customer lives |
| Zip | Nchar(5) | The customer's ZIP code |
| Phone | Nchar(10) | The customer's phone number without hyphens or parentheses (to save space, those will be displayed but not stored) |

**TABLE 1.4** Orders Table Fields

| Field Name | Datatype | Contains |
|---|---|---|
| CustID | Int | References the customer number stored in the Customers table so you don't need to duplicate the customer information for each order placed |
| ProdID | Int | References the Products table so you don't need to duplicate product information |
| Qty | Int | The amount of product sold for an order |
| OrdDate | Smalldatetime | The date and time the order was placed |

Follow these steps to create the Products table in the Sales database:

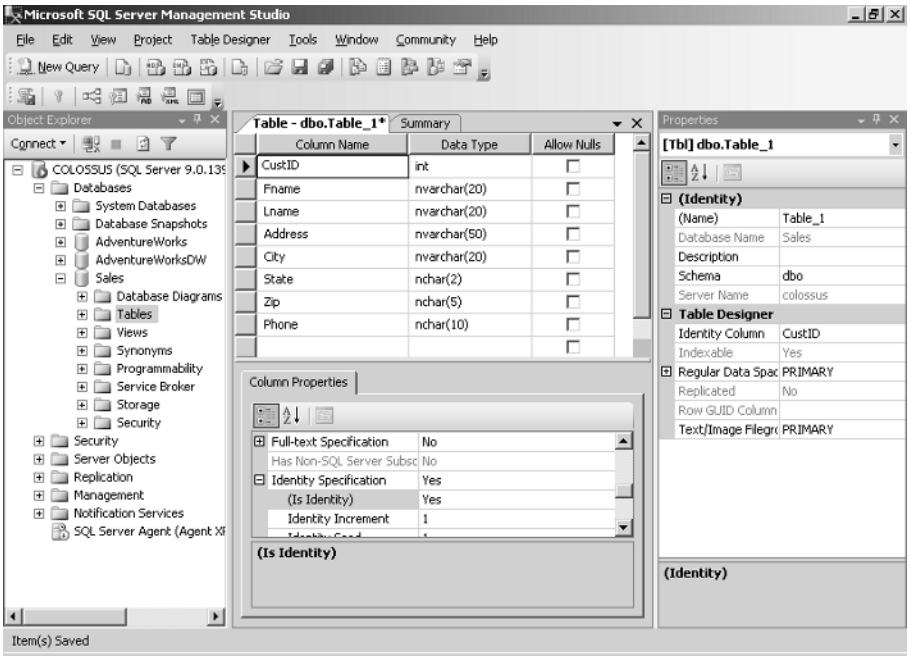
1. Open SQL Server Management Studio. In Object Explorer, expand your server, and expand Databases ➢ Sales.
2. Right-click the Tables icon, and select New Table to open the table designer.
3. In the first row, under Column Name, enter **ProdID**.
4. Just to the right of that, under Data Type, select Int.
5. Make certain Allow Nulls isn't checked. The field can be completely void of data if this option is checked, and you don't want that here.
6. In the bottom half of the screen, under Column Properties, in the Table Designer section, expand Identity Specification, and change (Is Identity) to Yes.
7. Just under ProdID, in the second row under Column Name, enter **Description**.
8. Just to the right of that, under Data Type, enter **nvarchar**(100).
9. Make certain Allow Nulls is cleared.
10. Under Column Name in the third row, enter **InStock**.
11. Under Data Type, select Int.
12. Uncheck Allow Nulls.



13. Click the Save button on the left side of the toolbar (it looks like a floppy disk).
14. In the Choose Name box that opens, enter **Products**, then click OK.
15. Close the table designer by clicking the *X* in the upper-right corner of the window.

Now, follow these steps to create the Customers table:

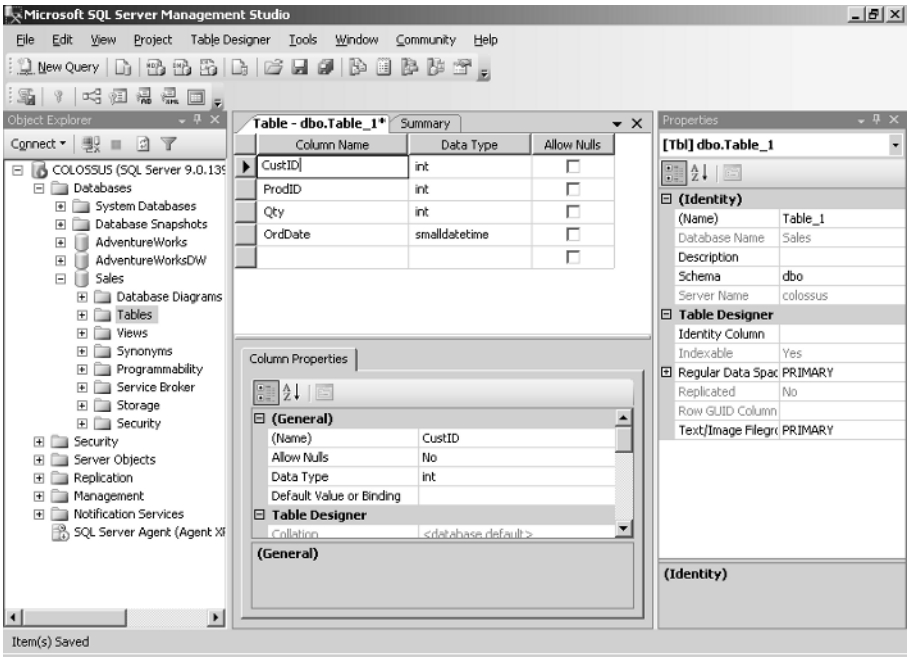
1. Right-click the Tables icon, and select New Table to open the table designer.
2. In the first row, under Column Name, enter **CustID**.
3. Under Data Type, select Int.
4. Make certain Allow Nulls isn't checked.
5. Under Column Properties, in the Table Designer section, expand Identity Specification, and change (Is Identity) to Yes.
6. Just under CustID, in the second row under Column Name, enter **Fname**.
7. Just to the right of that, under Data Type, enter **nvarchar**(20).
8. Make certain Allow Nulls is unchecked.
9. Using the parameters displayed earlier, fill in the information for the remaining columns. Don't allow nulls in any of the fields.



10. Click the Save button.
11. In the Choose Name box that opens, enter **Customers**, then click OK.
12. Close the table designer.

Now follow the same steps to create the Orders table:

1. Right-click the Tables icon, and select New Table to open the table designer.
2. In the first row, under Column Name, enter **CustID**.
3. Under Data Type, select Int.
4. Make certain Allow Nulls isn't checked.
5. This won't be an identity column like it was in the Customers table, so don't make any changes to the Identity Specification settings.
6. Just under CustID, in the second row under Column Name, enter **ProdID** with a datatype of int. Don't change the Identity Specification settings. Don't allow null values.
7. Just below ProdID, create a field named Qty with a datatype of int that doesn't allow nulls.
8. Create a column named OrdDate with a datatype of smalldatetime. Don't allow null values.

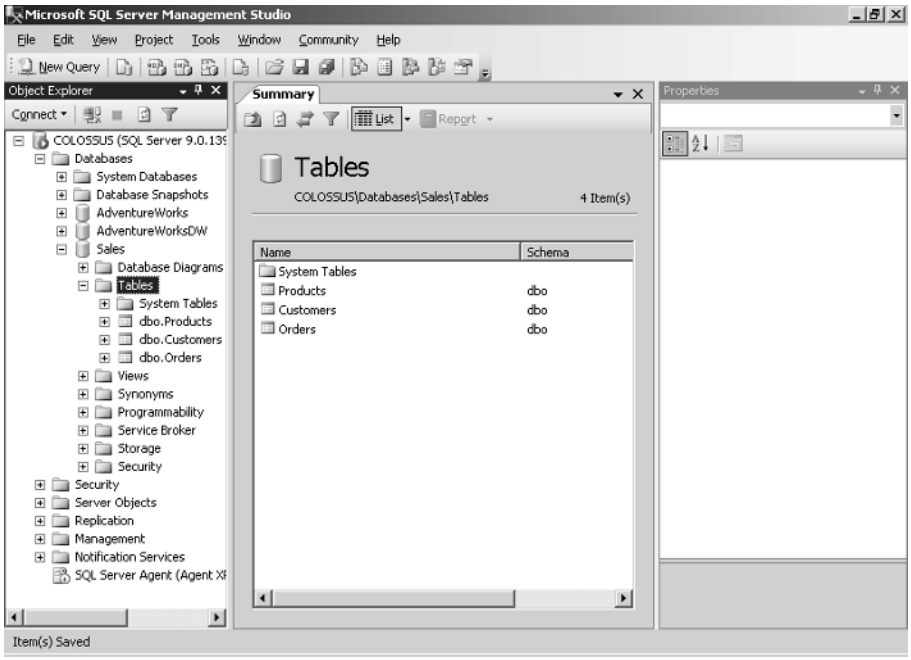



9. Click the Save button.
10. In the Choose Name box that opens, enter **Orders**, then click OK.
11. Close the table designer.

## Criteria for Completion

You have completed this task when you have the Products, Customers, and Orders tables in your database with the columns defined in the exercise. To verify this, just expand your Sales database in SQL Server Management Studio and then expand Tables; you should see all three tables, as shown in Figure 1.4.

**FIGURE 1.4** You should see the Products, Customers, and Orders tables in your Sales database.




# Task 1.5: Designing and Creating a Constraint

When you first create a table, it's wide open to your users. It's true that they can't violate datatype restrictions by entering characters in an int type field and the like, but that is really the only restriction. It's safe to say you probably want more restrictions than that. For example, you probably don't want your users to enter **XZ** for a state abbreviation in a State field (because

XZ isn't a valid state abbreviation), and you don't want them entering numbers for someone's first name. You need to restrict what your users can enter in your fields, which you can do by using constraints. In this task, you will create constraints on your Customers table in the Sales database.

## Scenario

You have just created tables in your new Sales database, one of which holds customer information. You want to make certain that users enter only valid ZIP codes in the Zip field of the Customers table, so you decide to create a constraint to restrict the data that can be entered in the field.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this exercise, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a constraint on the Customers table to prevent users from entering invalid ZIP codes. Specifically, you will prevent users from entering letters; they can enter only numbers.

### Equipment Used

For this exercise, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

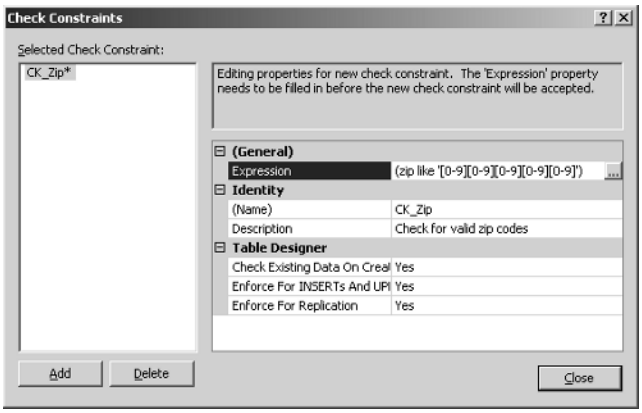
### Details

Follow these steps to create a constraint on the Zip field of the Customers table:

1. In Object Explorer, expand the Sales database, and expand Tables ➢ dbo.Customers.
2. Right-click Constraints, and click New Constraint.
3. In the New Constraint dialog box, enter **CK_Zip** in the (Name) text box.

4. In the Description text box, enter **Check for valid zip codes**.
5. To create a constraint that will accept only five numbers that can be zero through nine, enter the following code in the Expression text box:

```
(zip like '[0-9][0-9][0-9][0-9][0-9]')
```




6. Click Close.
7. Click the Save button at the top left of the toolbar.
8. Close the table designer (which was opened when you started to create the constraint).

## Criteria for Completion

This task is complete when you have a constraint that prevents users from entering letters in the Zip field of the Customers table. To test your new constraint, follow these steps:

1. In SQL Server Management Studio, and click the New Query button.
2. Enter the following code in the query window:

```
USE Sales
INSERT Customers
VALUES ('Greg','Scott','111 Main','Provo'
,'UT','88102','5045551212')
```

3. Click the Execute button just above the query window to execute the query, and notice the successful results.
4. To see the new record, click the New Query button, and execute the following code:

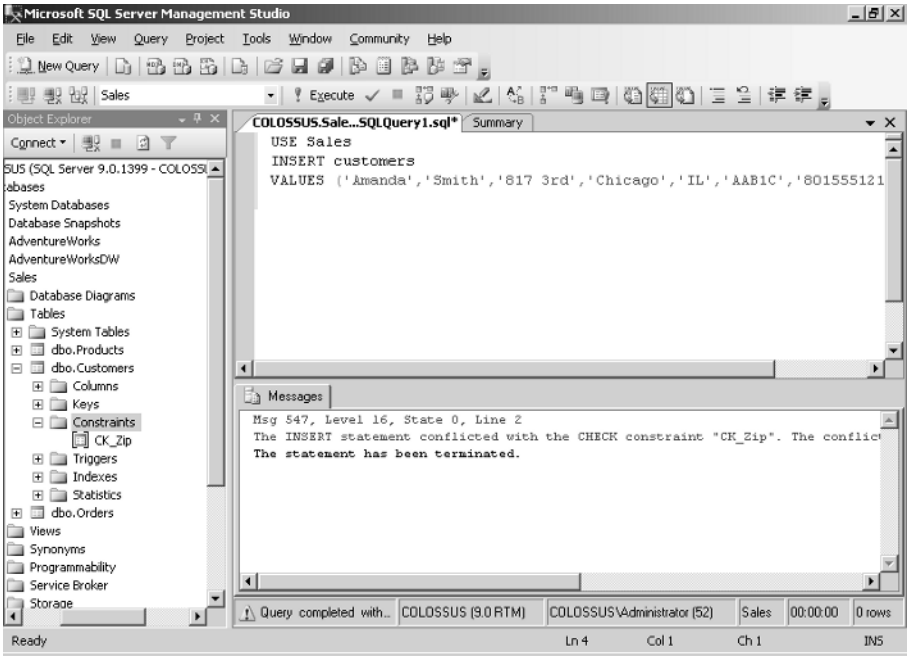
```
SELECT * FROM customers
```

5. Notice that the record now exists with a CustID of 1 (because of the identity property discussed earlier, which automatically added the number for you).

6. To test the check constraint by adding characters in the Zip field, click the New Query button, and execute the following code (note the letters in the Zip field):

```
USE Sales
INSERT customers
VALUES ('Amanda','Smith','817 3rd','Chicago','IL','AAB1C','8015551212')
```

7. Notice in the results pane that the query violated a constraint and so failed.




# Task 1.6: Designing and Creating a View

Microsoft describes a *view* as a virtual table or a stored SELECT query. Views do not hold any data themselves; they represent the data that is stored in a table. Of course, views offer more advantages than just looking at the data stored in a table. For instance, you may want to see only a subset of records in a large table, or you may want to see data from multiple tables in a single query. Both of these are good reasons to use a view. In this task, you will create a view based on the Person.Contact table in the AdventureWorks database.

## Scenario

You have a database that has been in use for some time and has a number of records in the Contacts table. Your users have asked you to break this data down for them by area code. You

have decided that the easiest way to accomplish this goal is to create a view that displays only a subset of data from the table based on the area code.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this exercise, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database that is installed with the sample data.

### Caveat

Realistically, views won't be this simplistic in the real world; this is just to keep the exercise simple.

## Procedure

In this task, you will create a view based on the Person.Contact table in the AdventureWorks database. Specifically, you will create a view that displays only those customers in the 398 area code.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database that is installed with the sample data.

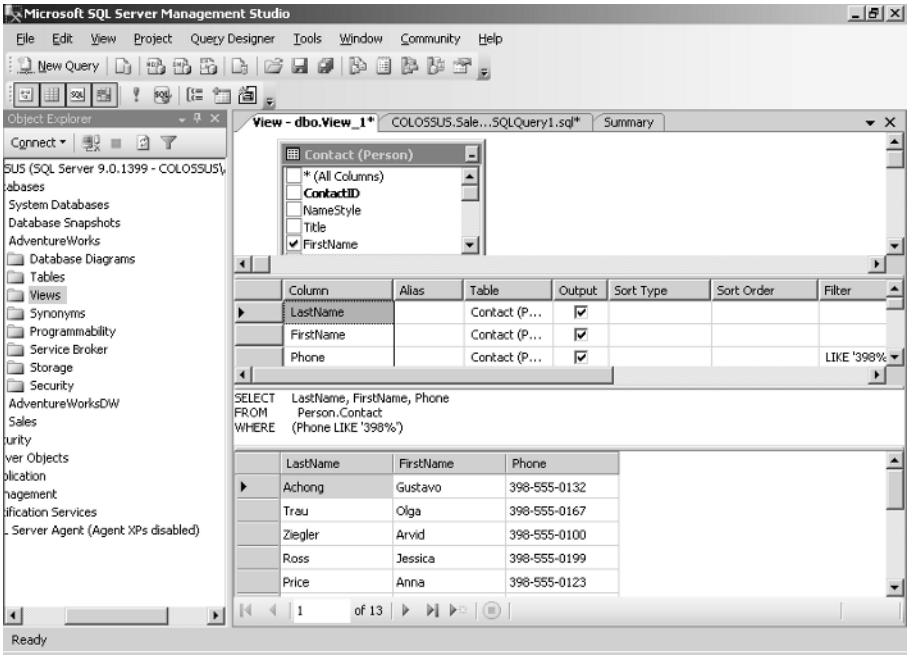
### Details

Follow these steps to create the Contacts_in_398 view:

1. Open SQL Server Management Studio by selecting it from the Microsoft SQL Server 2005 group under Programs on your Start menu, and connect with Windows Authentication if requested.
2. In Object Explorer, expand your server, and then expand Databases ➢ AdventureWorks. Right-click Views, and select New View.
3. In the Add Table dialog box, select Contact (Person), and click Add.
4. Click Close, this opens the view designer.
5. In the Transact-SQL syntax editor text box, under the column grid, enter the following:

```
SELECT LastName, FirstName, Phone
FROM Person.Contact
WHERE (Phone LIKE '398%')
```

6. Click the Execute button (the red exclamation point) on the toolbar to test the query.




7. Choose File ➢ Save View – dbo.View_1.
8. In the Choose Name dialog box, enter **Contacts_in_398**, and click OK.
9. To test the view, click the New Query button, and execute the following code:

```
USE AdventureWorks
SELECT * FROM dbo.Contacts_in_398
```

10. To verify that the results are accurate, open a new query, and execute the code used to create the view:

```
USE AdventureWorks
SELECT lastname, firstname, phone from Person.Contact
WHERE phone LIKE '398%'
```

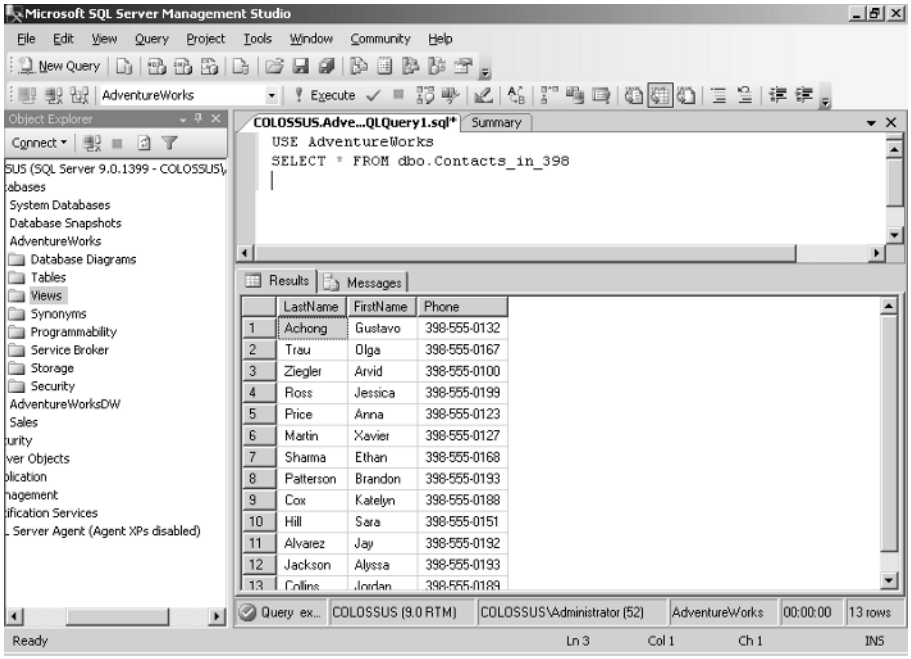
## Criteria for Completion

This task is complete when you can query the Contacts_in_398 view and retrieve the correct results. To test your view, execute this code:

```
USE AdventureWorks
SELECT * FROM dbo.Contacts_in_398
```

You should see the results shown in Figure 1.5.

**FIGURE 1.5** You should see only those contacts in the 398 area code.




# Task 1.7: Designing and Creating a Stored Procedure

A *stored procedure* is a query that is stored in a database on SQL Server rather than being stored in the front-end code on the client machine (also called an *ad hoc query*). Several good reasons exist for doing this.

One advantage of using stored procedures is that when SQL Server receives a query, it compiles it. In other words, it reads the query, looks at such things as `JOIN`s and `WHERE` clauses, and compares that query with all the available indexes to see which index (if any) would return data to the user fastest. Once SQL Server determines which indexes will function best, it creates a set of instructions telling SQL Server how to run the query (called an *execution plan*), which is stored in memory. Ad hoc queries must be compiled nearly every time they're run whereas stored procedures are precompiled. This means stored procedures have gone through the compilation process and have a plan waiting in memory; therefore, they execute faster than ad hoc queries.

Another advantage of using stored procedures is that they can make database management easier. For example, if you need to modify an existing query and that query is stored on the users' machines, you must make those changes on all of the users' machines. If you store the query centrally on the server, as a stored procedure, you need to make the changes only once, at the server. This can save you a great deal of time and effort.

Stored procedures can also accept variable input and produce variable output. This means you can use variable placeholders for input and output data, which is extremely useful for making stored procedures dynamic.

In this task, you will create a stored procedure that accepts an input parameter and returns data from the Production.Product table in the AdventureWorks database.

## Scenario

You have a database that has been in use for some time and has a number of records in the Product table. Your users frequently query the table for data based on the SellStartDate column. To speed up execution time and enhance ease of management, you have decided to create a stored procedure that accepts a date as an input parameter and returns a result set of the most frequently queried columns.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database that is installed with the sample data.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a stored procedure based on the Production.Product table in the AdventureWorks database. Specifically, you will create a stored procedure that returns product data based on an input parameter.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database that is installed with the sample data.

### Details

Follow these steps to create the Production.Show_Products stored procedure:

1. Open SQL Server Management Studio. In Object Explorer, expand your server, and then expand Databases ➢ AdventureWorks ➢ Programmability.
2. Right-click the Stored Procedures icon, and select New Stored Procedure to open a new query window populated with a stored procedure template.
3. In the Transact-SQL syntax box, change the code to look like this:

```
CREATE PROCEDURE Production.Show_Products
   @Date datetime
AS
BEGIN
   SELECT Name, Color, ListPrice, SellStartDate
   FROM Production.Product
   WHERE SellStartDate > @Date
   ORDER BY SellStartDate, Name
END
GO
```

4. Click the Execute button on the toolbar to create the procedure.
5. Close the query window.

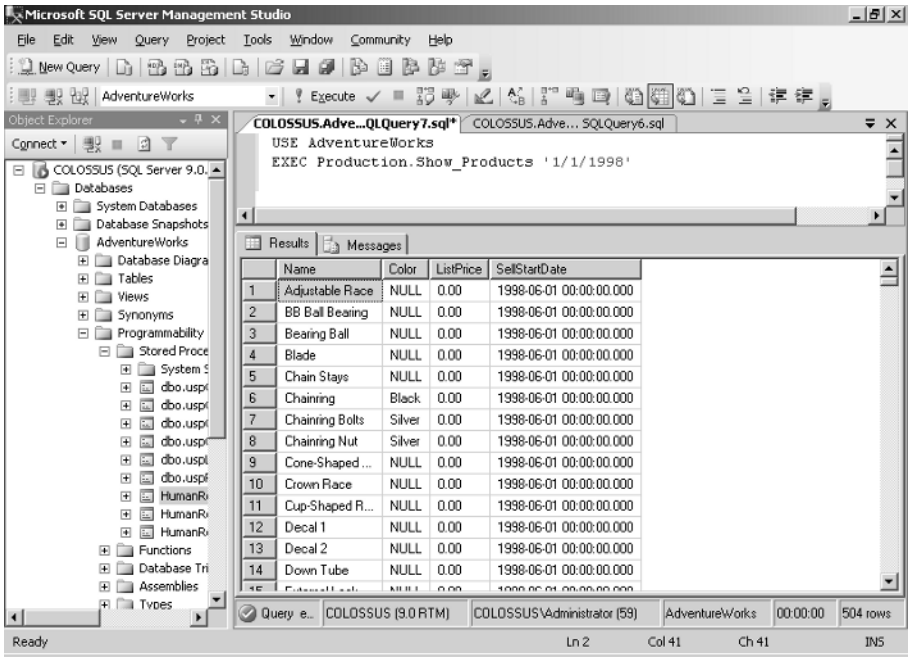
## Criteria for Completion

This task is complete when you can execute the Production.Show_Products stored procedure using an input parameter and have it return the correct results. Execute the following code in a new query window to test your stored procedure:

```
USE AdventureWorks
EXEC Production.Show_Products '1/1/1998'
```

You should see the results shown in Figure 1.6.

**FIGURE 1.6** You should see only those products that are available after January 1, 1998.




# Task 1.8: Designing and Creating an *INSERT* Trigger

As I discuss later in this book, you can control access to data by assigning permissions for objects (such as tables) to users. You can use these permissions to prevent users from inserting new data or modifying or deleting existing data. If you want more granular control than that, you can use triggers.

A *trigger* is a collection of SQL statements that looks and acts a great deal like a stored procedure. The only real difference between the two is that a trigger can't be executed directly; they are activated (or *fired*) when a user executes a Transact-SQL statement. Data Manipulation Language (DML) triggers fire on `INSERT`, `UPDATE`, and

DELETE statements; and Data Definition Language (DDL) triggers fire on CREATE, ALTER, and DROP statements.

INSERT triggers fire every time someone tries to create a new record in a table using the INSERT command. As soon as a user tries to insert a new record into a table, SQL Server copies the new record into a table in the database called the *trigger table* and a special table stored in memory called the *inserted* table. SQL Server then determines whether the data being inserted into your table meets the criteria defined in your trigger. If so, the record is inserted; if not, the record is not inserted.

You can even record what your users are doing by adding the RAISERROR() command to your trigger, which will write an error to the Windows event log. As you will see later in this book, the SQL Server Agent service can read this log entry and fire an alert, which can be used to send e-mail and text messages and even execute subsequent code.

In this task, you will create an INSERT trigger that prevents users from overselling a product in the Products table of the Sales database. If a user tries to do so, you will use the RAISERROR() command to write an event to the Windows event log.

## Scenario

You have created a database for your sales department that contains information about orders customers have placed. The sales manager needs to know what is in stock at all times, so she would like to have the quantity of an item that is in stock decremented automatically when a user places an order. You have decided that the best way to accomplish this is by using a trigger on the Orders table.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Products table you created in Task 1.4.

### Caveat

When executing a trigger, SQL Server uses two special tables: inserted and deleted. The inserted table holds new records that are about to be added to the table, and the deleted table holds records that are about to be removed. You will be using the inserted table in this task.

## Procedure

In this task, you will create a trigger that automatically decrements the InStock quantity in the Products table when a user places an order.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Products table you created in Task 1.4.

### Details

First, you need to add some data to the Customers and Products tables:

1. Open SQL Server Management Studio by selecting it from the Microsoft SQL Server 2005 group in Programs on the Start menu, and log in using either Windows Authentication or SQL Server Authentication.
2. Open a new SQL Server query window, and enter and execute the following code to populate the Customers table with customer information:

```
USE Sales
INSERT customers
VALUES ('Andrea','Elliott','111 Main','Oakland','CA','94312','7605551212')
INSERT customers
VALUES ('Tom', 'Smith', '609 Georgia', 'Fresno', 'CA', '33045', '5105551212')
INSERT customers
VALUES ('Janice', 'Thomas', '806 Star', 'Phoenix', 'AZ', '85202',
'6021112222')
```

3. To populate the Products table with product and inventory information, enter and execute the following code:

```
INSERT Products
VALUES ('Giant Wheel of Brie', 200)
INSERT Products
VALUES ('Wool Blankets', 545)
INSERT Products
VALUES ('Espresso Beans', 1527)
INSERT Products
VALUES ('Notepads', 2098)
```

4. Close the query window.

Next, follow these steps to create the new InvUpdate trigger:

1. In Object Explorer, expand your server, and then expand Databases ➢ Sales ➢ Tables ➢ dbo.Orders.

2. Right-click the Triggers folder, and select New Trigger.
3. In the Transact-SQL syntax box, enter and execute the following code to create the trigger:

```
CREATE TRIGGER dbo.InvUpdate
   ON  dbo.Orders
   FOR INSERT
AS
BEGIN
UPDATE p
SET p.instock = (p.instock - i.qty)
FROM Products p JOIN inserted i
ON p.prodid = i.prodid
END
GO
```

4. Close the query window.

## Criteria for Completion

This task is complete when you have a trigger that automatically updates the InStock column of the Products table whenever a record is inserted in the Orders table. To test your new trigger, follow these steps:

1. Open a new SQL Server query, and execute the following code to verify the InStock quantity for item 1 (it should be 200):

```
USE Sales
SELECT prodid, instock
FROM Products
```

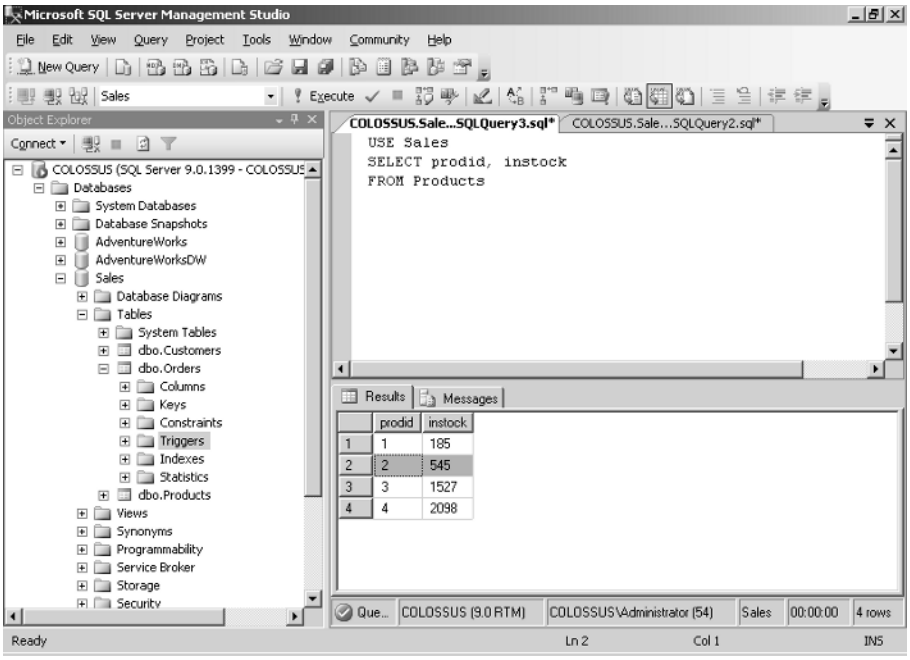
2. To cause the `INSERT` trigger to fire, you need to insert a new record in the Orders table. To do this, open a new query window, and enter and execute the following code, which assumes you're selling 15 quantities of product 1 to customer ID 1 on today's date (`GETDATE()` is used to return today's date):

```
USE Sales
INSERT Orders
VALUES (1,1,15,getdate())
```

3. To verify that the `INSERT` trigger fired and removed 15 from the InStock column of the Products table, click the New Query button, and enter and execute the following code:

```
USE Sales
SELECT prodid, instock
FROM Products
```

4. Notice that the exact quantity you sold customer 1 (15) was subtracted from the total InStock quantity of product ID 1. You now have 185 instead of 200.




5. Close the query windows.

# Task 1.9: Designing and Creating a *DELETE* Trigger

Ordinarily, when a user executes a DELETE statement, SQL Server removes the record from the table, and the record is never heard from again. That behavior changes when you add a DELETE trigger to the table. With a DELETE trigger in place, SQL Server moves the record being deleted to a logical table in memory called *deleted*; the records aren't entirely gone, and you can still reference them in your code. This comes in handy for complex business logic.


The special deleted table can easily be compared to the Recycle Bin in the Windows operating system, where deleted files are moved before they're deleted from the system. The biggest difference is that the deleted table is automatically purged of records after a transaction is complete whereas the Recycle Bin must be purged manually.

Suppose you want to keep your users from deleting customers who have more than $10,000 in credit with your company. Without a DELETE trigger in place, a user could successfully delete any record they wanted, regardless of the amount of credit the customer had. With a DELETE trigger in place, however, SQL Server places the record in question in the deleted table, so you can still reference the credit limit column and base the success of the transaction on the value therein.

In this task, you will create a DELETE trigger that prevents users from deleting customers based in Arizona.

## Scenario

You have created a database for your sales department that contains information about orders customers have placed. One of your biggest customers is in Arizona, and they have recently had to stop placing regular orders with you, which makes them look inactive. Your sales manager is concerned that someone new may accidentally delete important information about this customer, and she has asked you to prevent that from happening. You have decided that the best way to accomplish this is by using a trigger on the Customers table.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

### Caveat

When executing a trigger, SQL Server uses two special tables named *inserted* and *deleted*. The inserted table holds new records that are about to be added to the table, and the deleted table holds records that are about to be removed. You will be using the deleted table in this task.

## Procedure

In this task, you will create a DELETE trigger that prevents users from deleting customers based in Arizona.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Products table you created in Task 1.4.

### Details

Follow these steps to create the new AZDel trigger:

1. In Object Explorer, expand your server, and then expand Databases ➢ Sales ➢ Tables ➢ dbo.Orders.
2. Right-click the Triggers folder, and select New Trigger.
3. In the Transact-SQL syntax box, enter and execute the following code to create the trigger:

```
CREATE TRIGGER dbo.AZDel
   ON  dbo.Customers
   FOR DELETE
AS
BEGIN
IF (SELECT state FROM deleted) = 'AZ'
BEGIN
PRINT 'Cannot remove customers from AZ'
PRINT 'Transaction has been cancelled'
ROLLBACK
END
END
GO
```

4. Close the query window.

## Criteria for Completion

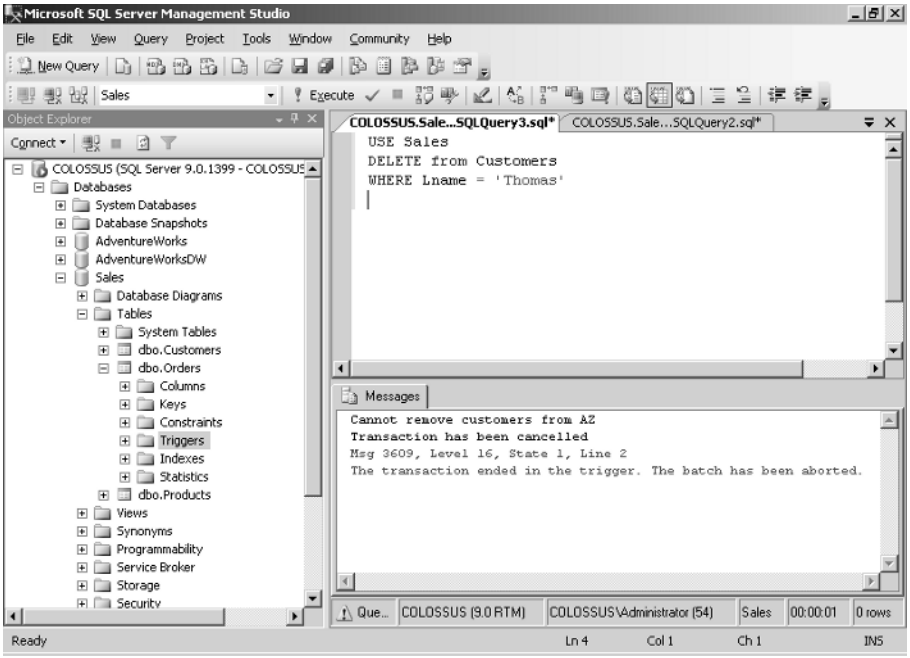
This task is complete when you have a trigger that prevents you from deleting customers who reside in Arizona. To test your new trigger, follow these steps:

1. Open a new SQL Server query, and execute the following code to verify you have customers from Arizona (for example, Janice Thomas should be in Arizona):

```
USE Sales
SELECT * FROM customers
```

2. To cause the DELETE trigger to fire, try to delete Janice from the Customers table. To do this, open a new query, and enter and execute the following code (you should see an error message upon execution):

```
USE Sales
DELETE from Customers
WHERE Lname = 'Thomas'
```




3. To verify that Janice has not been deleted, enter and execute the following code (you should still see Janice):

```
USE Sales
SELECT * FROM customers
```

4. Close the query window.

# Task 1.10: Designing and Creating an *UPDATE* Trigger

As discussed in the previous task, a trigger is a collection of SQL statements that fires when a user performs an action against a record in a table. An `UPDATE` trigger fires when a user executes an `UPDATE` statement against a table.

What makes this type of trigger unique is that it uses the same functionality as both `DELETE` and `INSERT` triggers. When an `UPDATE` statement is executed, SQL Server actually deletes the record in question from the table and holds it in the deleted table; it then inserts a new record in the table and holds a duplicate copy in the inserted table. So, both the inserted and updated tables are available to `UPDATE` triggers.

In this task, you will create an UPDATE trigger that prevents users from overselling a product in the Products table of the Sales database. If a user tries to do so, you will use the RAISERROR() command to write an event to the Windows event log.

## Scenario

You have created a database for your sales department that contains product information. Some of the new sales personnel have accidentally oversold some product, and the sales manager has asked you for a technical means to prevent this in the future. You have decided to put a trigger in place that prevents users from selling any amount of product that will set the InStock column to a negative value. If they try to execute such a transaction, you have opted to use the RAISERROR() command to write an event to the Windows event log for tracking and reporting purposes.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Products table you created in Task 1.4.

### Caveat

When executing a trigger, SQL Server uses two special tables named *inserted* and *deleted*. The inserted table holds new records that are about to be added to the table, and the deleted table holds records that are about to be removed. You will be using the inserted table in this task.

## Procedure

In this task, you will create a trigger that prevents users from updating a record in the Products table if that update would set the InStock column to a negative amount. If they try to execute such a query, you will use the RAISERROR() command to write an event to the Windows event log.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Products table you created in Task 1.4.

### Details

Follow these steps to create the new CheckStock trigger:

1. Expand your server, and then expand Databases ➢ Sales ➢ Tables ➢ dbo.Products.
2. Right-click the Triggers folder, and select New Trigger.
3. In the Transact-SQL syntax box, enter and execute the following code to create the trigger:

```
CREATE TRIGGER dbo.CheckStock
   ON  dbo.Products
   FOR UPDATE
AS
BEGIN

   IF (SELECT InStock from inserted) < 0
   BEGIN
      PRINT 'Cannot oversell Products'
      PRINT 'Transaction has been cancelled'
      ROLLBACK
      RAISERROR('Cannot oversell products', 10, 1) WITH LOG
   END
END
GO
```

4. Close the query window.

## Criteria for Completion

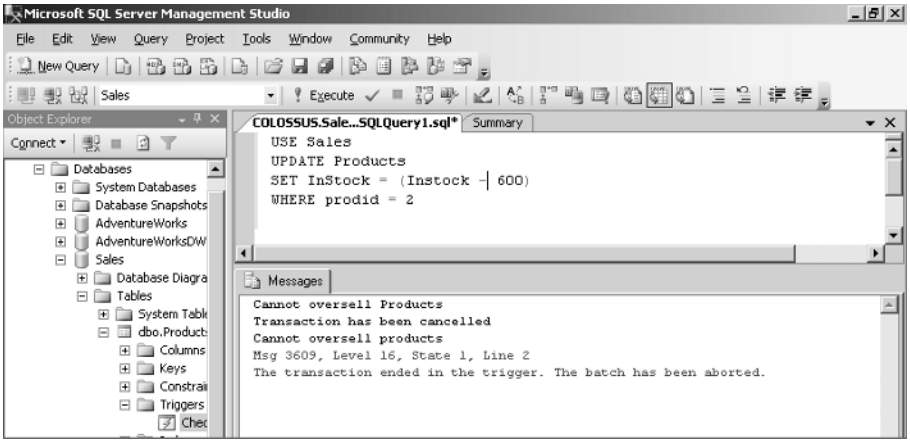
This task is complete when you have a trigger that prevents you from entering a negative value in the InStock column of the Products table in the Sales database and also writes an event to the Windows event log if you try. To test your new trigger, follow these steps:

1. Open a new SQL Server query, and execute the following code to verify the quantity in stock on available products (product ID 2 should have 545 in stock currently):

```
USE Sales
SELECT prodid, instock FROM Products
```

2. To cause the UPDATE trigger to fire, you'll try to sell 600 units of product ID 2 (wool blankets) to a customer. Open a new SQL Server query, and enter and execute the following code (you should see an error message upon execution):

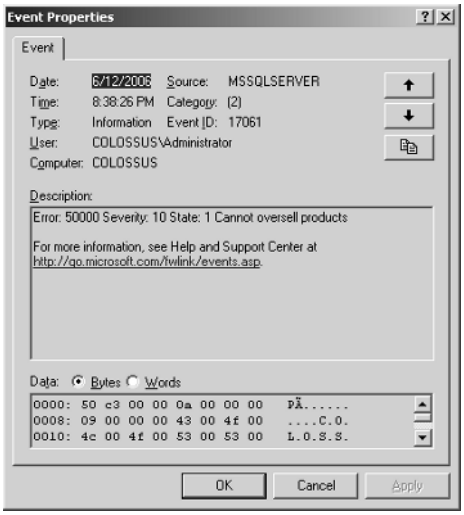
```
USE Sales
UPDATE Products
SET InStock = (Instock - 600)
WHERE prodid = 2
```




3. To verify that the transaction was disallowed and that you still have 545 wool blankets in stock, click the New Query button, and enter and execute the following code (you should still see 545 of product ID 2):

```
USE Sales
SELECT prodid, instock FROM Products
```

4. Now, open Event Viewer, and look in the Application log for the event written by `RAISERROR()`.




5. Close the query window.

# Task 1.11: Designing and Creating an *INSTEAD OF* Trigger

In Task 1.6, I discussed views, which you can use to display only a few of the columns in a table, only a subset of the rows in a table, or data from more than one table at a time. This works great when you want to see just the data, but you can have problems when you try to modify data through a view.

Because views may not display all the columns in a table, data modification statements can fail. For example, suppose you have a Customers table like the one in your Sales database that contains customer information such as name, address, city, state, ZIP code, and so on. Then suppose you have created a view that displays all the columns except the City field. If you try to update the Customers table through the new view, the update will fail because the City field (which is a required field) is not available through the view. Using an `INSTEAD OF` trigger can make this type of update successful.

In this task, you will create an `INSTEAD OF` trigger that can insert a value that is not available through a view into a table. To accomplish this, you will first create a view that does not display the City column (which is a required column for updates), and then you will try to update through this column. Next you will create an `INSTEAD OF` trigger that can insert the missing value for you, after which you will try the insert again.

## Scenario

You have created a database for your sales department with a table that contains customer information. Your sales manager has asked you to make it easier for sales representatives to find the customer data they need, so you have decided to create a view that displays only the necessary data. You need to make sure the sales representatives can insert new data through the view, and because the view does not show all the required columns, you need to create an `INSTEAD OF` trigger to modify the `INSERT` statement so that `INSERT` statements on the view will succeed.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a view that shows only a subset of columns from the Customers table in the Sales database. Then you will create a trigger that intercepts an INSERT statement and modifies the statement so that all required columns are filled in.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

### Details

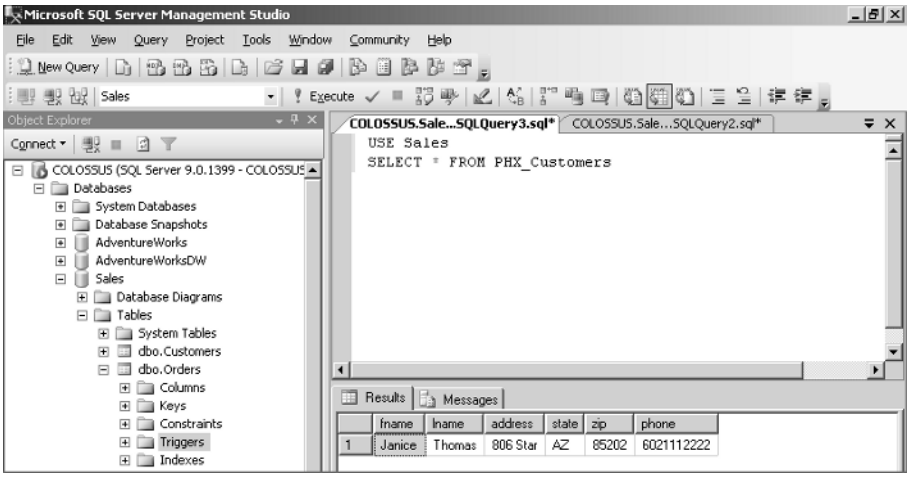
First, you need to create a view based on the Customers table that does not display the City field (which is a required field for an INSERT). Follow these steps to create the PHX_Customers view:

1. In SQL Server Management Studio, open a new query, and enter and execute the following code:

```
USE Sales
GO
CREATE VIEW PHX_Customers AS
SELECT fname, lname, address, state, zip, phone
FROM Customers
WHERE City = 'Phoenix'
```

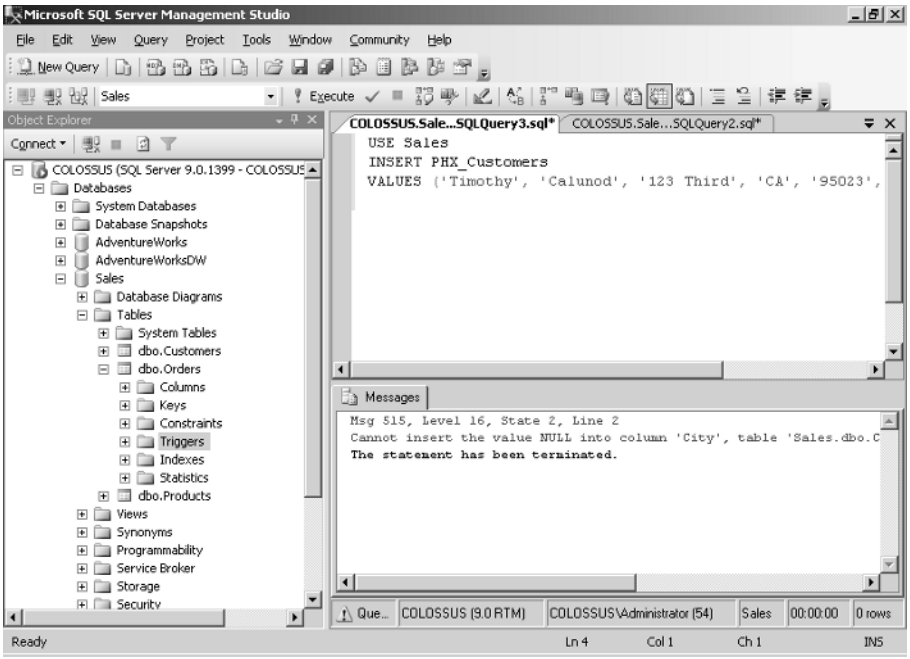
2. To verify that the view displays only the columns you want, click the New Query button, and enter and execute the following query:

```
USE Sales
SELECT * FROM PHX_Customers
```



3. Now you will try to insert a new customer through the view. Select New Query with Current Connection from the Query menu, and enter and execute the following code:

```
USE Sales
INSERT PHX_Customers
VALUES ('Timothy', 'Calunod', '123 Third', 'CA', '95023', '9252221212')
```




Next, you can create an `INSTEAD OF` trigger that inserts the missing value for you when you insert through the view:

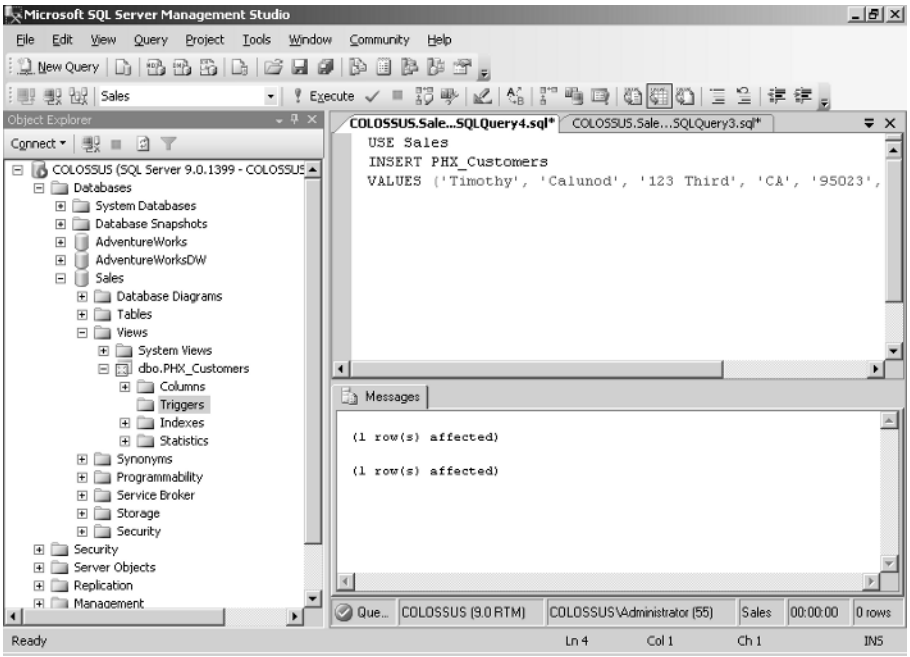
1. Expand your server, and then expand Databases ➢ Sales ➢ Views ➢ dbo.PHX_Customers.
2. Right-click the Triggers folder, and select New Trigger.
3. In the Transact-SQL syntax box, enter and execute the following code to create the trigger:

```
CREATE TRIGGER Add_City ON PHX_Customers
INSTEAD OF INSERT
AS
DECLARE
 @FNAME VARCHAR(20),
 @LNAME VARCHAR(20),
 @ADDR VARCHAR(50),
 @CITY VARCHAR(20),
 @STATE NCHAR(2),
```

```
@ZIP CHAR(5),
@PHONE CHAR(10)

SET @CITY = 'Phoenix'

SET @FNAME = (SELECT FNAME FROM INSERTED)
SET @LNAME = (SELECT LNAME FROM INSERTED)
SET @ADDR = (SELECT ADDRESS FROM INSERTED)
SET @STATE = (SELECT STATE FROM INSERTED)
SET @ZIP = (SELECT ZIP FROM INSERTED)
SET @PHONE = (SELECT PHONE FROM INSERTED)

INSERT CUSTOMERS
VALUES(@FNAME, @LNAME, @ADDR, @CITY, @STATE, @ZIP, @PHONE)
```

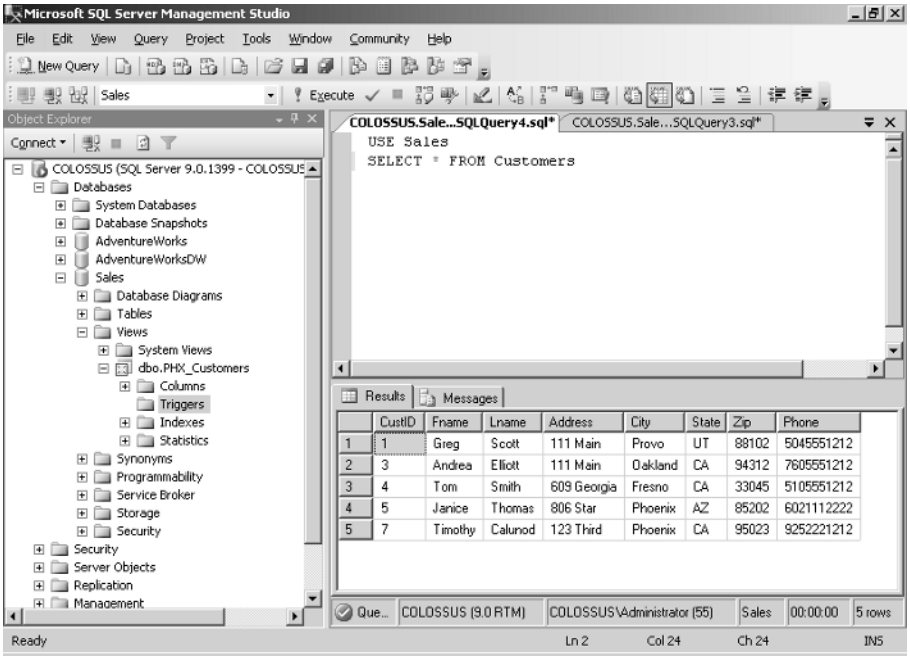
4. To test the trigger, enter and execute the same code from step 3 in the previous series of steps:

```
USE Sales
INSERT PHX_Customers
VALUES ('Timothy', 'Calunod', '123 Third', 'CA', '95023', '9252221212')
```



5. To verify that the data was inserted into the Customers table and that the City column was populated, select New Query with Current Connection from the Query menu, and enter and execute the following query:

```
USE Sales
SELECT * FROM Customers
```




6. Close the query windows.

## Criteria for Completion

This task is complete when you have an `INSTEAD OF` trigger that intercepts an `INSERT` statement on the PHX_Customers database and adds a value to insert into the City field.

# Task 1.12: Designing and Creating a User-Defined Function

A *function* is a grouping of Transact-SQL statements that you can reuse. SQL Server has a large number of built-in functions, but these may not meet all of your needs. For this reason, SQL Server gives you the ability to create your own functions, called *user-defined functions*,

to perform any tasks you may need. This is especially useful for performing complex calculations in your business logic. You can create several types of functions:

- Scalar functions return a single value or the type specified in the `RETURN` statement, such as a single int value.
- Table-valued functions return a table datatype.
- CLR functions are special functions created using any language available in the .NET Framework (that is, C# or Visual Basic .NET). These can return scalar or table values.

In this task, you will create a user-defined function that determines which customer has placed the largest order for a given product.

## Scenario

You have created a database for your sales department with a table that contains order information. Your sales manager wants to know the largest number of items for a given product on a single order. She wants to run this calculation every week, so you decide to create a user-defined function for ease of use and management.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Orders and Customers tables you created in Task 1.4.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a user-defined function that calculates the largest number of items for a given product on a single order. First you will add some orders to the database, and then you will create a function to calculate the sales; finally, you will verify the function.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Orders and Customers tables you created in Task 1.4.

## Details

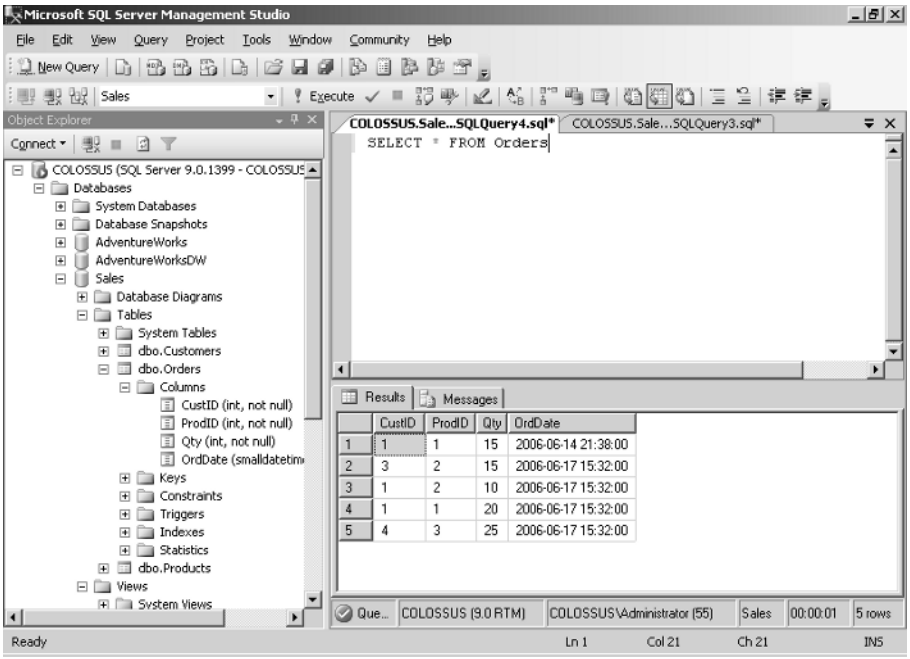
First, you need to place some orders in the Orders table so the function has some data to work with:

1. Open a new SQL Server query, and execute the following code to insert some new records in the Orders table:

```
USE Sales
INSERT Orders
VALUES (3,2,15,getdate())
GO
INSERT Orders
VALUES (1,2,10,getdate())
GO
INSERT Orders
VALUES (1,1,20,getdate())
GO
INSERT Orders
VALUES (4,3,25,getdate())
GO
```

2. To verify that the new orders exist, enter and execute this code in a new query window:

```
USE Sales
SELECT * FROM Orders
```



Next, you are ready to create a user-defined function to calculate the largest number of items for a given product on a single order:

1. In SQL Server Management Studio, open a new query, and enter and execute the following code:

```
USE Sales
GO
CREATE FUNCTION ItemOrderCount
  (@ProductID int)
RETURNS int
AS
BEGIN
  RETURN (SELECT MAX(Qty) FROM Orders WHERE ProdID = @ProductID)
END
```

I'm using the `GO` statement here because the `CREATE FUNCTION` statement must be the first statement in a batch. `GO` separates the statements into separate batches.

## Criteria for Completion

This task is complete when you have created a function that tells you the largest amount of items for a specific product placed on a single order. To test your function, execute this code in a new query window:

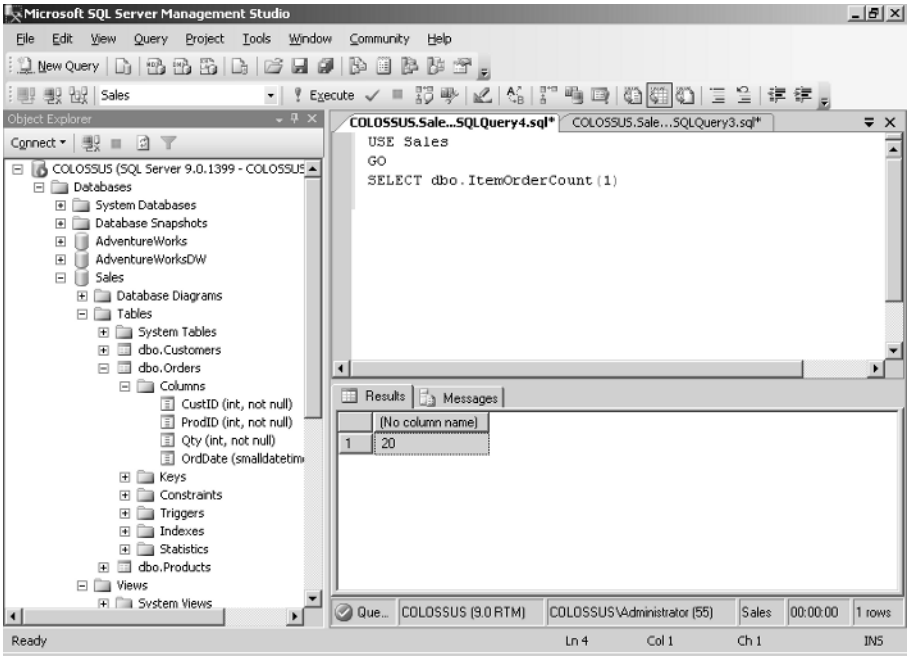
```
USE Sales
GO
SELECT dbo.ItemOrderCount(1)
```

As shown in Figure 1.7, you should get a value of 20; this was the most items placed on a single order for product ID 1.

# Task 1.13: Designing and Creating a Clustered Index

SQL Server stores data on the hard disk in 8KB pages inside the database files. By default, these pages and the data they contain aren't organized in any way. To bring order to this chaos, you must create an index. You can create two types of indexes on a table: clustered and nonclustered.

**FIGURE 1.7** The ItemOrderCount() function should return a value of 20 for product ID 1.




*Clustered indexes* physically rearrange the data that users insert in your tables. The arrangement of a clustered index on disk is comparable to that in a dictionary, because they both use the same storage paradigm. If you needed to look up a word in the dictionary—for example, *satellite*—how would you do it? You would turn right to the *S* section of the dictionary and continue through the alphabetically arranged list until you found the word *satellite*. The process is similar with a clustered index; a clustered index on a Lastname column would place *Adams* physically before *Burns* in the database file. This way, SQL Server can more easily pinpoint the exact data pages it wants.

It might help to visualize an index in SQL Server as an upside-down tree. In fact, the index structure is called a *B-tree* (binary-tree) structure. At the top of the B-tree structure, you find the *root page*; it contains information about the location of other pages further down the line called *intermediate-level pages*. These intermediate pages contain yet more key values that can point to still other intermediate-level pages or data pages. The pages at the bottom of a clustered index, the *leaf pages*, contain the actual data, which is physically arranged on disk to conform to the constraints of the index.


Because all the data is arranged physically in the database, clustered indexes are a good choice when users search for a range of data.

On a table without a clustered index in place (called a *heap*), new data is inserted at the end of the table, which is the bottom of the last data page. If none of the data pages has any room, SQL Server allocates a new extent and starts filling it with data. Because you've told SQL Server to physically rearrange your data by creating a clustered index, SQL Server no longer has the freedom to stuff data wherever it finds room. The data must physically be placed in order. To accomplish this, you need to leave some room at the end of each data page on a clustered index. This blank space is referred to as the *fill factor*. A higher fill factor gives less room, and a lower fill factor gives more room. If you specify a fill factor of 70, for example, the data page is filled with 70 percent data and 30 percent blank space. If you specify 100, the data page is filled to nearly 100 percent, having room for only one record at the bottom of the page.

When you need to insert data into a page that has become completely full, SQL Server performs a *page split*. This means SQL Server takes approximately half the data from the full page and moves it to an empty page, thus creating two half-full pages, leaving plenty of room for new data.


Because clustered indexes physically rearrange the data, you can have only one clustered index per table.

In this task, you will create a clustered index on the Customers table of the Sales database.

## Scenario

You have created a database for your sales department with a table that contains customer information. Your sales representatives have started complaining about slow access times. You know that the sales representatives look up customers based on their ZIP codes quite often, so you decide to create a clustered index on the Zip column of the Customers table to improve data access times.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers tables you created in Task 1.4.

### Caveat

Because this is just a test system and Customers is a small table with little data, you will not see a significant improvement in data access time.

## Procedure

In this task, you will create a clustered index on the Customers table based on the Zip column.

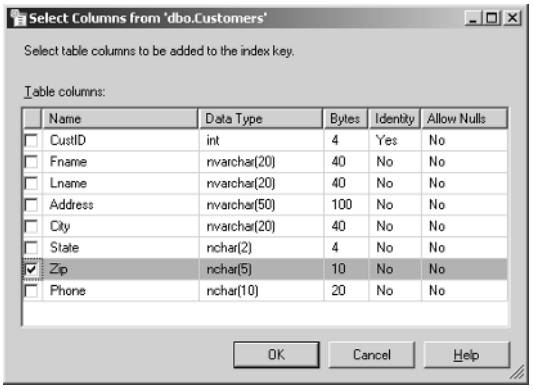
### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

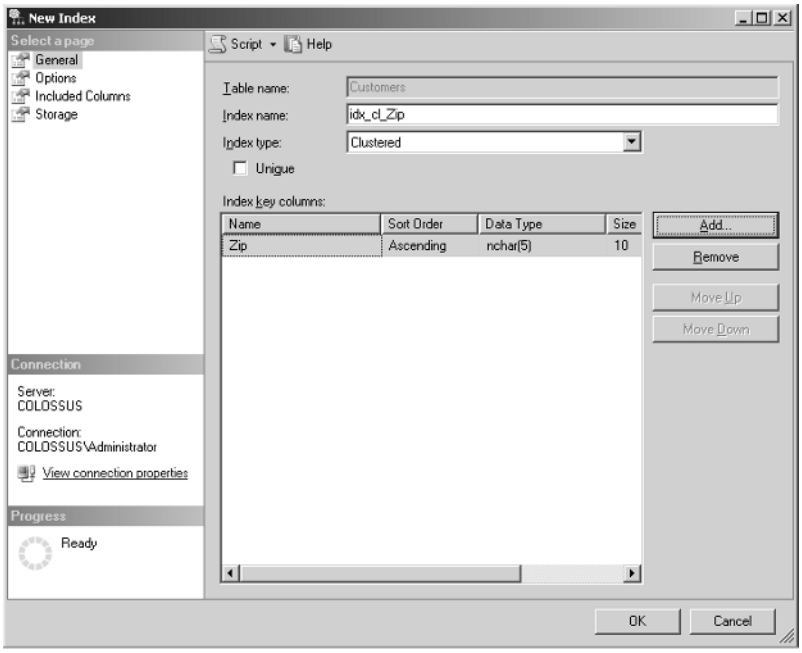
### Details

To create the new idx_cl_Zip clustered index, follow these steps:

1. Open SQL Server Management Studio, and connect using Windows Authentication.
2. In Object Explorer, expand your server, and then expand Databases ➢ Sales ➢ Tables ➢ dbo.Customers.
3. Right-click Indexes, and select New Index.
4. In the Index name box, enter **idx_cl_Zip**.
5. Select Clustered for the index type.
6. Click the Add button next to the Index Key Columns grid.
7. Check the box next to the Zip column.



8. Click OK to return to the New Index dialog box.




9. Click OK to create the index.

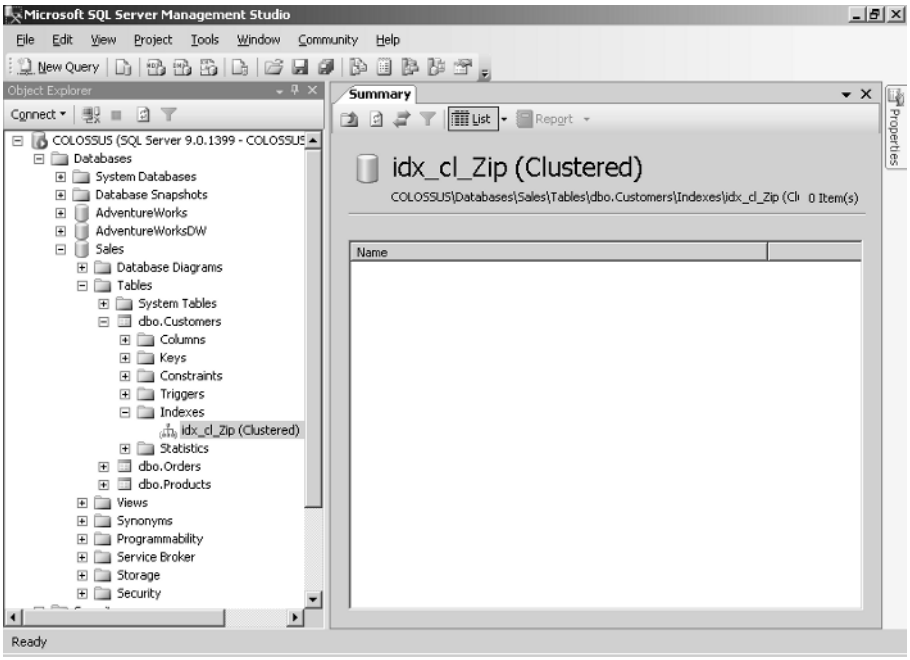
## Criteria for Completion

This task is complete when you have a clustered index on the Zip column of the Customers table in the Sales database. To verify that the index is there, expand Databases ➢ Sales ➢ Tables ➢ dbo.Customers ➢ Indexes, and you should see the idx_cl_Zip index listed, as shown in Figure 1.8.

# Task 1.14: Designing and Creating a Nonclustered Index

Like its clustered cousin, a *nonclustered* index is a B-tree structure having a root page, intermediate levels, and a leaf level. However, two major differences separate the index types. The first is that the leaf level of the nonclustered index doesn't contain the actual data; it contains pointers to the data stored in data pages. The second big difference is that the nonclustered index doesn't physically rearrange the data. It functions more like the index at the back of a topically arranged book.

**FIGURE 1.8** The new idx_cl_Zip clustered index shows up in Object Explorer after it has been created.




When you search for data on a table with a nonclustered index, SQL Server first queries the sysindexes table looking for a record that contains your table name and a value in the indid column from 2 to 251 (0 denotes a heap, and 1 is for a clustered index). Once SQL Server finds this record, it looks at the root column to find the root page of the index (just like it did with a clustered index). Once SQL Server has the location of the root page, it can begin traversing the B-tree in search of your data.

If you're searching a nonclustered index that is based on a heap (a table with no clustered index in place), SQL Server uses the pointer in the leaf-level page to jump right to the data page and return your data.

If your table has a clustered index in place, the nonclustered index leaf level contains a pointer to the clustered index key value. This means once SQL Server is done searching your nonclustered index, it has to traverse your clustered index as well. SQL Server searches two indexes because it can actually be faster when you consider how data is updated through a nonclustered index.

When inserting data using a nonclustered index on a heap, SQL Server inserts the data wherever it finds room and adds a new key value that points to the new record of the associated index pages.

When you insert data into a table with a nonclustered index and a clustered index in place, SQL Server physically inserts the data where it belongs in the order of the clustered index and

updates the key values of the nonclustered index to point to the key values of the clustered index. When one of the data pages becomes full and you still have more data to insert, a page split occurs, which is why the key values of the nonclustered index point to the clustered index instead of the data pages themselves.

When you're using a nonclustered index without a clustered index in place, each index page contains key values that point to the data. This pointer contains the location of the extent, as well as the page and record number of the searched-for data. If a page split occurred and the nonclustered index didn't use clustered index key values, then all the key values for the data that had been moved would be incorrect because all the pointers would be wrong. The entire nonclustered index would need to be rebuilt to reflect the changes. However, because the nonclustered index references the clustered index key values (not the actual data), all the pointers in the nonclustered index will be correct even after a page split has occurred, and the nonclustered index won't need to be rebuilt. That is why you reference the key values of a clustered index in a nonclustered index.


You can have 249 nonclustered indexes per table, though in practice you will not have nearly that many.

In this task, you will create a nonclustered index on the Customers table of the Sales database.

## Scenario

You have created a database for your sales department with a table that contains customer information. You want to make sure data access is as fast as possible. After some investigation, you have discovered that your sales representatives consistently search for customer information by searching for a last name. Because they search only for a single record at a time, you decide to create a nonclustered index on the Lname column of the Customers table to improve data access times.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

### Caveat

Because this is just a test system and Customers is a small table with little data, you will not see a significant improvement in data access time.

## Procedure

In this task, you will create a nonclustered index on the Customers table based on the Zip column.

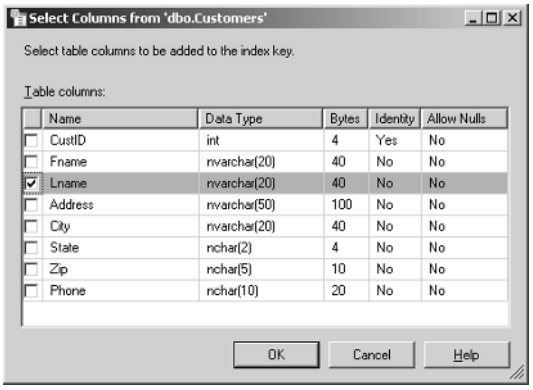
### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the Sales database you created in Task 1.3, and the Customers table you created in Task 1.4.

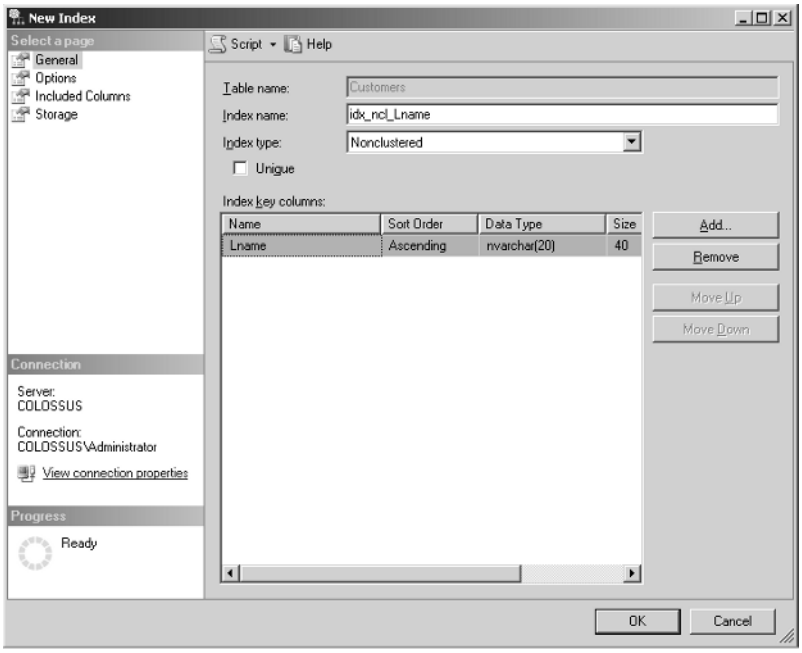
### Details

To create the new idx_ncl_Lname nonclustered index, follow these steps:

1. Open SQL Server Management Studio, and connect using Windows Authentication.
2. In Object Explorer, expand your server, and then expand Databases ➢ Sales ➢ Tables ➢ dbo.Customers.
3. Right-click Indexes, and select New Index.
4. In the Index Name box, enter **idx_ncl_Lname**.
5. Select Nonclustered for the index type.
6. Click the Add button next to the Index Key Columns grid.
7. Check the box next to the Lname column.



8. Click OK to return to the New Index dialog box.




9. Click OK to create the index.

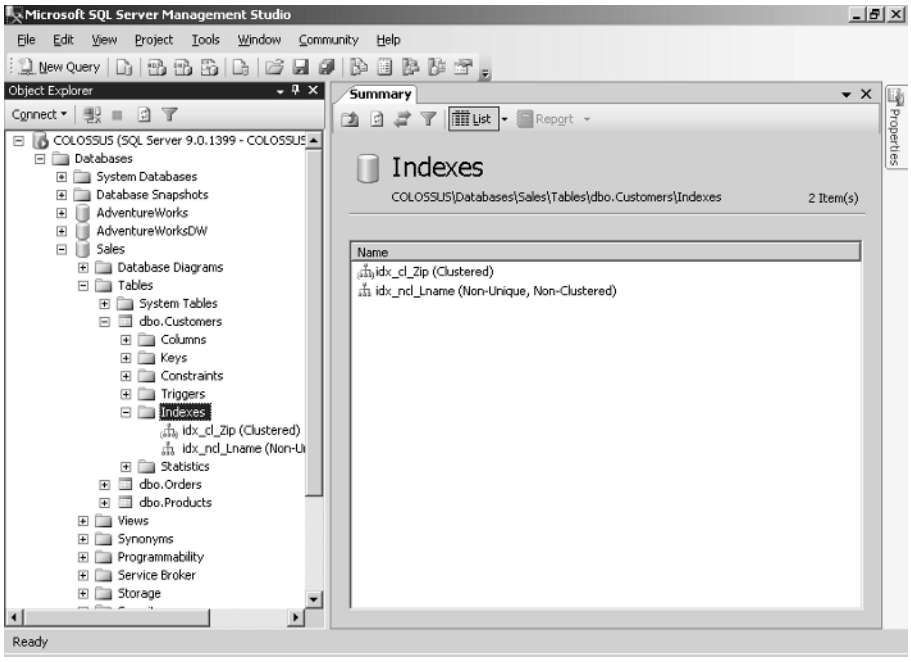
## Criteria for Completion

This task is complete when you have a nonclustered index on the Lname column of the Customers table in the Sales database, as shown in Figure 1.9. To verify that the index is there, expand Databases ➢ Sales ➢ Tables ➢ dbo.Customers ➢ Indexes, and you should see the idx_ncl_Lname index listed.

# Task 1.15: Designing and Creating a Full-Text Index

Databases are excellent containers for all sorts of data, including massive amounts of text. Many companies, in fact, have entire libraries of corporate documents stored in their databases. New datatypes, such as nvarchar(max), were devised to store such large amounts of text. Because the standard SELECT statement wasn't designed to handle such large amounts of text, something else had to be devised—something more robust. Enter *full-text search*.

**FIGURE 1.9** The new idx_ncl_Lname nonclustered index shows up in Object Explorer after it has been created.




Full-text search is a completely separate program that runs as a service (called the SQL Server FullText Search service, or msftesq) and can index all sorts of information from most of the BackOffice (or even non-Microsoft) products. Thus, when you perform a full-text search, you are telling SQL Server to make a request of the FullText Search service. To perform a full-text search, you need only use the `CONTAINS`, `CONTAINSTABLE`, `FREETEXT`, or `FREETEXTTABLE` clause in your `SELECT` query.


You can find a detailed discussion of `CONTAINS`, `CONTAINSTABLE`, `FREETEXT`, and `FREETEXTTABLE` in *Mastering SQL Server 2005* (Sybex, 2006).

Before you can start using this powerful tool, you need to configure it. The first step you need to take is to create a full-text index. You create full-text indexes with SQL Server tools, such as SQL Server Management Studio, but you maintain them with the FullText Search service; and they are stored on the disk as files separate from the database. To keep the full-text indexes organized, they are stored in catalogs in the database. You can create as many catalogs in your databases as you'd like to organize your indexes, but these catalogs cannot span databases.

In this task, you will create a full-text index on the Production.Document table of the AdventureWorks database.

## Scenario

One of the databases that your company has been using for some time contains a table that stores large documents. Your users have been using standard SELECT statements to access the documents in this table, but that method is proving too slow now that the table is starting to grow. You want to make sure users can query the table and get results as quickly and easily as possible, so you decide to create a full-text index on the table.

## Scope of Task

### Duration

This task should take approximately 20 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database that is installed with the sample data.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a full-text catalog and index on the Production.Document table of the AdventureWorks database.

### Equipment Used

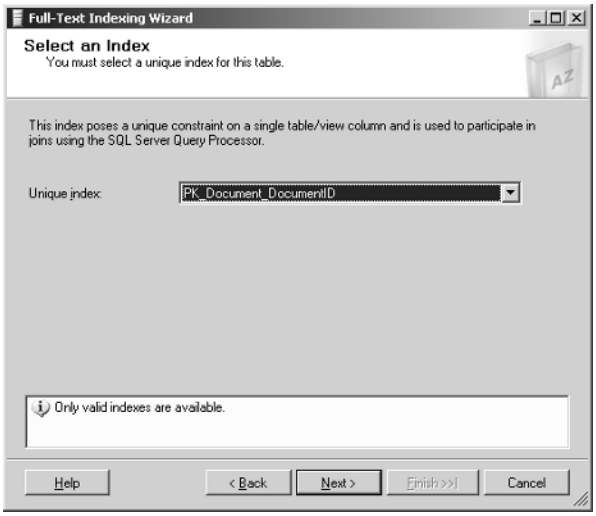
For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database that is installed with the sample data.

### Details

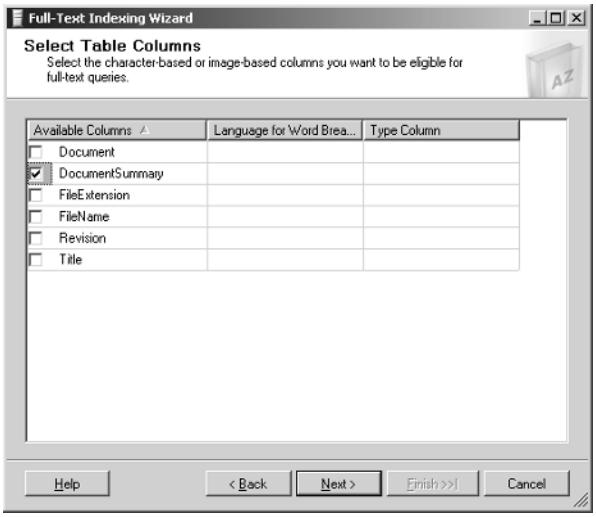
To create the new full-text catalog and index on the Production.Document table, follow these steps:

1. Open SQL Server Management Studio, and in Object Explorer, expand Databases ➢ AdventureWorks ➢ Tables.
2. Right-click Production.Document, move to Full-Text Index, and click Define Full-Text Index.

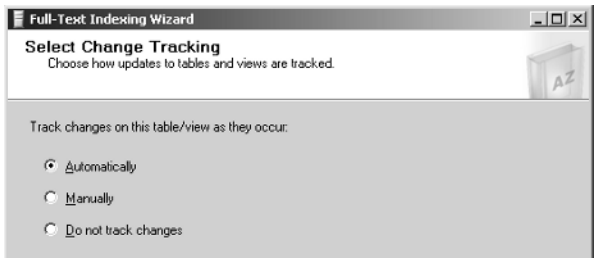
3. On the first screen of the Full-Text Indexing Wizard, click Next.
4. Each table on which you create a full-text index must already have a unique index associated with it for the FullText Search service to work. In this instance, select the default PK_Document_DocumentID index, and click Next.



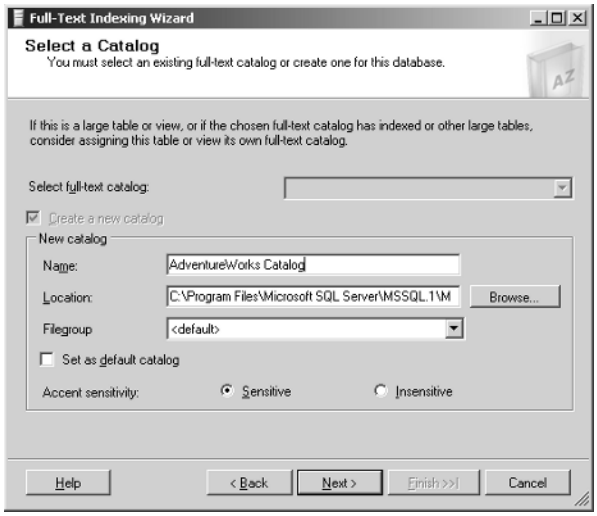

5. On the next screen, you are asked which column you want to full-text index. DocumentSummary is the only nvarchar(max) column in the table, so it is the best candidate; select the box next to it, and click Next.



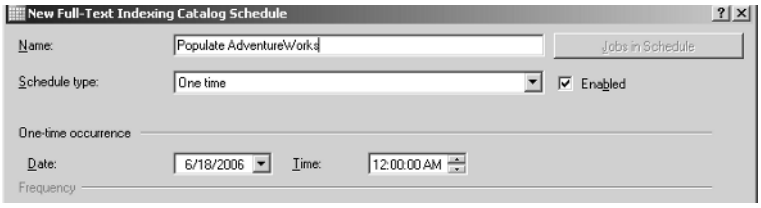
6. On the next screen, you are asked when you want changes to the full-text index applied:
   - Automatically means the full-text index is updated with every change made to the table. This is the fastest, least-hassle way to keep full-text indexes up-to-date, but it can tax the server because it means the changes to the table and index take place all at once.
   - Manually means changes to the underlying data are maintained, but you will have to schedule index population yourself. This is a slightly slower way to update the index, but it is not as taxing on the server because changes to the data are maintained but the index is not updated immediately.
   - Do Not Track Changes means changes to the underlying data are not tracked. This is the least taxing, and slowest, way to update the full-text index. Changes are not maintained, so when the index is updated, the FullText Search service must read the entire table for changes before updating the index.
7. Choose Automatically, and click Next.



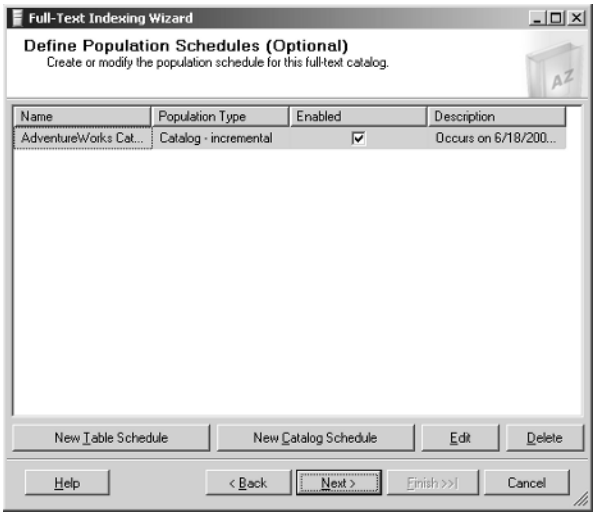

8. The next screen asks you to select a catalog. You'll need to create one here, because you don't have any available. In the Name field, enter **AdventureWorks Catalog**. You can also select a filegroup to place the catalog on; leave this as default, and click Next.



9. On the next screen, you are asked to create a schedule for automatically repopulating the full-text index. If your data is frequently updated, you will want to do this more often, maybe once a day. If it is read more often than it is changed, you should repopulate less frequently. You can schedule population for a single table or an entire catalog at a time. Here, you will set repopulation to happen just once for the entire catalog by clicking the New Catalog Schedule button.
10. On the New Schedule Properties screen, enter **Populate AdventureWorks**, and click OK.




11. When you are taken back to the Full-Text Indexing Wizard, click Next.



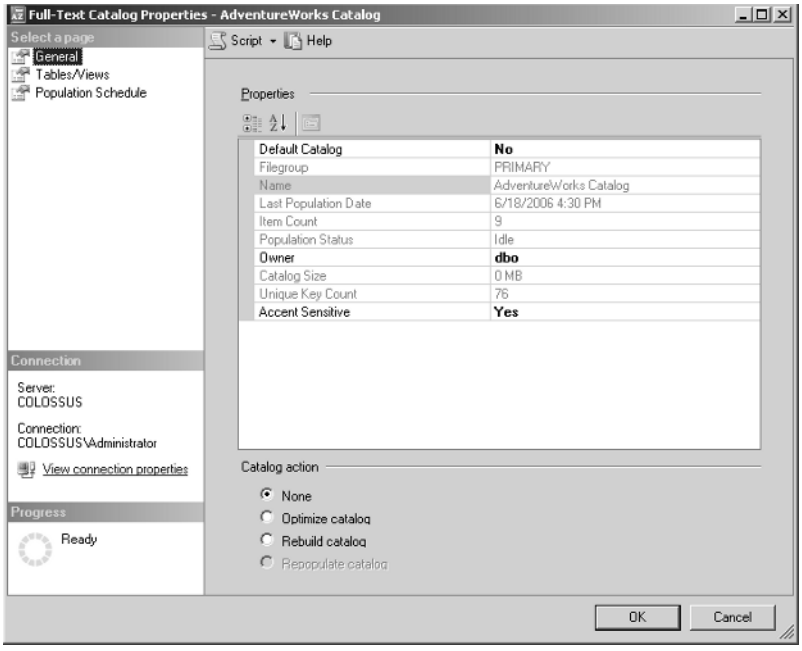

12. On the final screen of the wizard, you are given a summary of the choices you have made. Click Finish to create the index.

## Criteria for Completion

This task is complete when you have a full-text index on the Production.Document table of the AdventureWorks database. To verify that the index is there, follow these steps:

1. To see your new catalog and index, in Object Explorer expand AdventureWorks ➢ Storage ➢ Full Text Catalogs.

2. Double-click the AdventureWorks catalog to open its properties.




# Task 1.16: Creating a Windows Login

It is vitally important to protect the information stored in your databases. Realizing this, the team at Microsoft implemented a world-class security system in SQL Server 2005. The foundation of this security system is the login. Two types of logins exist: Windows and Standard. To fully understand how they work, you need to know about authentication modes.

An *authentication mode* is how SQL Server processes usernames and passwords. SQL Server 2005 provides two such modes: Windows Authentication mode and Mixed mode.

In Windows Authentication mode, a user can sit down at her computer, log in to the Windows domain, and gain access to SQL Server using the Kerberos security protocol. Although an in-depth discussion of Kerberos is beyond the scope of this book, here is a brief overview of how this security protocol works:

1. When the user logs in, Windows performs a DNS lookup to locate a Key Distribution Center (KDC).
2. The user's machine logs in to the domain.
3. The KDC issues a special security token called a Ticket Granting Ticket (TGT) to the user.
4. To access the SQL Server, the user's machine presents the TGT to the SQL Server; if the ticket is accepted, the user is allowed access.

The main advantage of Windows Authentication mode is that users don't have to remember multiple usernames and passwords. This mode also allows you to apply Windows password policies, which perform such tasks as expire passwords, require a minimum length for passwords, keep a history of passwords, and so on.

One of the disadvantages is that only users with Windows accounts can open a trusted connection to SQL Server. This means someone like a Novell client can't use Windows Authentication mode because they don't have a Windows account. If it turns out that you have such clients, you'll need to implement Mixed mode.

Mixed mode allows both Windows Authentication and SQL Server Authentication. SQL Server Authentication works as follows:

1. A user logs in to his network, Windows or otherwise.
2. The user opens a nontrusted connection to SQL Server using a username and password other than those used to gain network access. It's called a *nontrusted* connection because SQL Server doesn't trust the operating system to verify the user's password.
3. SQL Server matches the username and password entered by the user to an entry in the syslogins table.

The primary advantage is that anyone—Mac users, Novell users, Unix users, and the like—can gain access to SQL Server using Mixed mode. You could also consider this to be a second layer of security, because if someone hacks into the network in Mixed mode, it doesn't mean they have automatically hacked into SQL Server at the same time.

Ironically, multiple passwords can be a problem as well as an advantage. When users have multiple sets of credentials, they tend to write them down and thus breach the security system you have worked so hard to create.

In this task, you will create Windows logins for several Windows user and group accounts.

## Scenario

You have just installed a new SQL Server at your company, and you need to make sure the right people have access. Some of these users will be using Windows accounts, so you have decided to create Windows logins in SQL Server for these users.

## Scope of Task

### Duration

This task should take approximately 90 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1.

### Caveat

You will need administrative access to the machine you will be using for this task, because you will be creating new groups and user accounts on the machine. This task also works best if performed on Windows 2003.

## Procedure

In this task, you will create several Windows user and group accounts and then create Windows logins for these accounts in SQL Server.

### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1.

### Details

You need to create the Windows user and group accounts in Computer Management (or Active Directory Users and Computers if you are in a domain with Domain Admin rights):

1. Open Computer Management in the Administrative Tools group under Programs on the Start menu, expand Local Users and Groups, click Users, and then select Action ➢ New User.
2. Create six new users with the criteria in the following list:

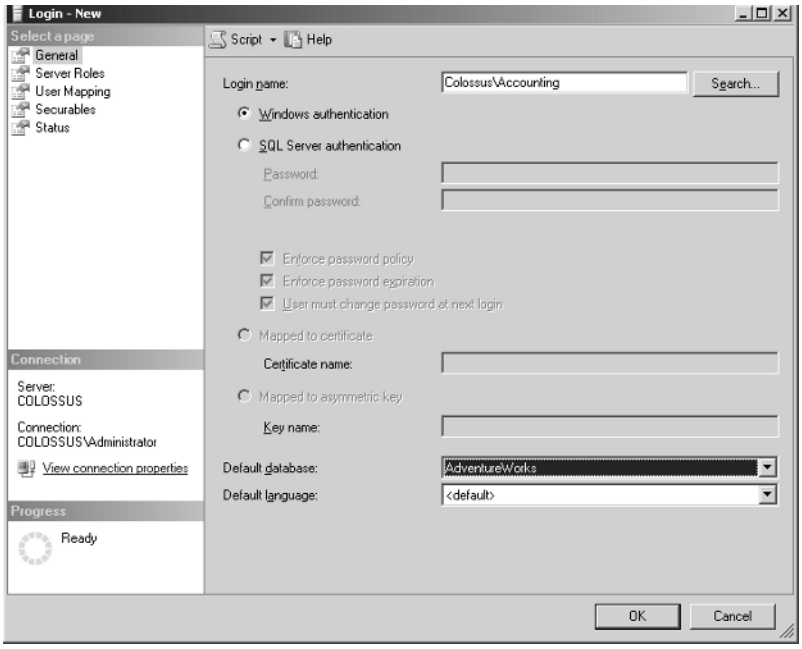
| Username | Description | Password | Must Change | Never Expires |
|---|---|---|---|---|
| MorrisL | IT | Password1 | Deselect | Select |
| RosmanD | Administration | Password1 | Deselect | Select |
| JohnsonK | Accounting | Password1 | Deselect | Select |
| JonesB | Accounting | Password1 | Deselect | Select |
| ChenJ | Sales | Password1 | Deselect | Select |
| SamuelsR | Sales | Password1 | Deselect | Select |

3. While in Computer Management, create a Local group called Accounting.
4. Add the new users you just created whose Description value is Accounting.
5. While still in Computer Management, create a Local group named Sales.
6. Add all the users whose Description value is Sales.
7. Open Local Security Policy from the Administrative Tools group under Programs on the Start menu.
8. Expand Local Policies, and click User Rights Assignment.

9. Double-click the Allow Log on Locally right, and click Add User or Group.
10. Select the Everyone group, click OK, and then click OK again (on a production machine this is not a best practice; this is only for this exercise).
11. Close the Local Policies tool, and open SQL Server Management Studio.

With your user accounts and groups created, you're ready to create Windows logins for these accounts in SQL Server:

1. Open SQL Server Management Studio, expand your server, and then expand Security ➢ Logins.
2. Right-click Logins, and select New Login.
3. In the Login Name box, enter ***Sqldomain*\Accounting** (the name of the Local group created earlier).
4. Under Defaults, select AdventureWorks as the default database.




5. On the User Mapping page, select the Map check box next to AdventureWorks to give your user access to the default database.
6. Click OK to create the login.
7. Right-click Logins, and select New Login.
8. In the Login Name box, enter ***Sqldomain*\Sales** (the name of the Local group created earlier).
9. Under Defaults, select AdventureWorks as the default database.

10. On the User Mapping page, select the Map check box next to AdventureWorks to give your user access to the default database.
11. Click OK to create the login.
12. Right-click Logins, and select New Login.
13. Enter ***Sqldomain*\RosmanD** in the Login Name field.
14. Under Defaults, select AdventureWorks as the default database.
15. On the User Mapping page, select the Permit check box next to AdventureWorks to give your user access to the default database.
16. Click OK to create the login.
17. Right-click Logins, and select New Login.
18. Enter ***Sqldomain*\MorrisL** in the Login Name field.
19. Under Defaults, select AdventureWorks as the default database.
20. On the User Mapping page, select the Permit check box next to AdventureWorks to give your user access to the default database.
21. Click OK to create the login.

## Criteria for Completion

This task is complete when you have successfully created the Windows login accounts specified in the task. To test this, first you'll log in as a member of one of the groups you created, and then you'll log in as a specific user:

1. Log out of Windows, and log back in as JonesB.
2. Open a new SQL Server query in SQL Server Management Studio, and select Windows Authentication from the Authentication drop-down list.
3. Close SQL Server Management Studio, log out of Windows, and log back in as RosmanD.
4. Open a new SQL Server query in SQL Server Management Studio, and select Windows Authentication from the Authentication drop-down list.

# Task 1.17: Creating a Standard Login

In Task 1.16, you learned that only clients with a Windows account can make trusted connections to SQL Server (where SQL Server trusts Windows to validate the user's password). If the user (such as a Macintosh or Novell client) for whom you're creating a login can't make a trusted connection, you must create a Standard login for them. In this task, you'll create two Standard logins that you'll use later in the phase.

Although you can create Standard logins in Windows Authentication mode, you won't be able to use them. If you try, SQL Server will ignore you and use your Windows credentials instead.

## Scenario

You have just installed a new SQL Server at your company, and you need to make sure the right people have access. Some of these users run Macintosh and Linux, and they do not have Windows accounts, so you must create Standard logins in SQL Server for these users.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create two Standard logins in SQL Server.

### Equipment Used

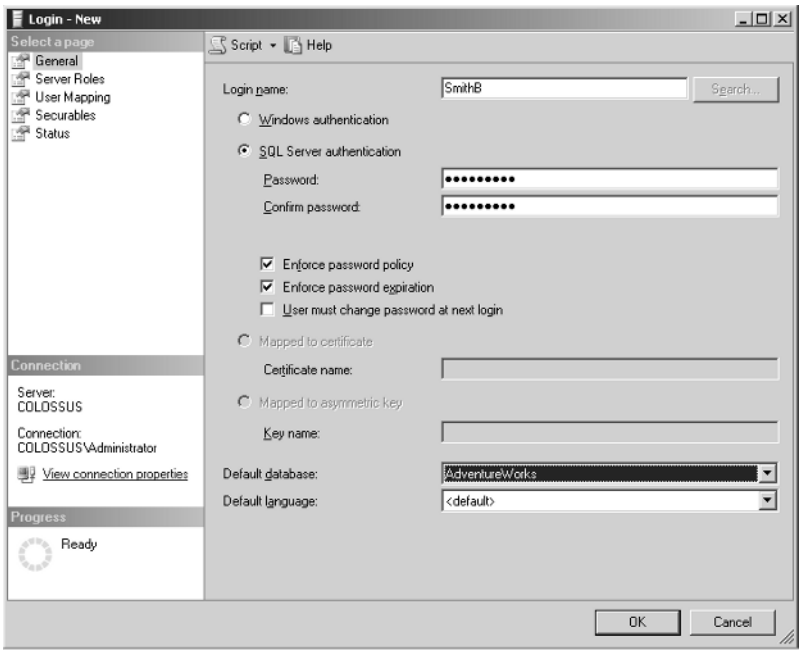
For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1.

### Details

Follow these steps to create the Standard logins in SQL Server:

1. Open SQL Server Management Studio, and expand your server by clicking the + sign next to the icon named after your server.
2. Expand Security, and then expand Logins.
3. Right-click Logins, and select New Login.
4. Select the SQL Server Authentication radio button.

5. In the Name box, enter **SmithB**.
6. In the Password text box, enter **Password1** (remember, passwords are case sensitive).
7. In the Confirm Password text box, enter **Password1** again.
8. Under Defaults, select AdventureWorks as the default database.
9. Uncheck the User Must Change Password at Next Login box.




10. On the User Mapping page, select the Map check box next to AdventureWorks to give your user access to the default database.
11. Click OK to create your new login.
12. Right-click Logins, and select New Login.
13. Select the SQL Server authentication radio button.
14. In the Name box, enter **GibsonH**.
15. In the Password text box, enter **Password1**.
16. In the Confirm Password text box, enter **Password1**.
17. Under Defaults, select AdventureWorks as the default database.
18. Uncheck the User Must Change Password at Next Login box.
19. Do not select the Permit check box next to AdventureWorks on the User Mapping page. You'll create a database user account later in this phase.
20. Click OK to create your new login.

## Criteria for Completion

This task is complete when you have successfully created the Standard login accounts specified in the task. To test this, follow these steps:

1. To test the SmithB login, click the New Query button in SQL Server Management Studio.
2. On the Query menu, hover over Connection, and then click Change Connection.
3. In the dialog box that opens, select SQL Server Authentication from the Authentication drop-down list.
4. In the Login Name box, enter **SmithB**.
5. In the Password box, enter **Password1**.


6. Click Connect to connect to AdventureWorks.

# Task 1.18: Assigning Logins to Fixed Server Roles

You need to make sure your users have only the permissions they need once they are logged in to SQL Server. You can accomplish this goal by assigning logins to fixed server roles, which are specifically designed to allow you to limit the amount of administrative access that a user has once logged in to SQL Server. Some users may be allowed to do whatever they want; other users may be able to manage security only. You can assign users any of eight server roles. The following list starts at the highest level and describes the administrative access granted:

**Sysadmin** Members of the sysadmin role have the authority to perform any task in SQL Server. Be careful whom you assign to this role, because people who are unfamiliar with SQL Server can accidentally create serious problems. This role is only for the database administrators (DBAs).

**Serveradmin** These users can set serverwide configuration options, such as how much memory SQL Server can use or how much information to send over the network in a single frame. They can also shut down the server. If you make your assistant DBAs members of this role, you can relieve yourself of some of the administrative burden.

**Setupadmin** Members here can install replication and manage extended stored procedures (these are used to perform actions not native to SQL Server). Give this role to the assistant DBAs as well.

**Securityadmin** These users manage security issues such as creating and deleting logins, reading the audit logs, and granting users permission to create databases. This too is a good role for assistant DBAs.

**Processadmin** SQL Server is capable of multitasking; that is, it can do more than one task at a time by executing multiple processes. For instance, SQL Server might spawn one process for writing to cache and another for reading from cache. A member of the processadmin group can end (or *kill*, as it's called in SQL Server) a process. This is another good role for assistant DBAs and developers. Developers especially need to kill processes that may have been triggered by an improperly designed query or stored procedure.

**Dbcreator** These users can create and make changes to databases. This may be a good role for assistant DBAs as well as developers (who should be warned against creating unnecessary databases and wasting server space).

**Diskadmin** These users manage files on disk. They perform tasks such as mirroring databases and adding backup devices. Assistant DBAs should be members of this role.

**Bulkadmin** Members of this role can execute the `BULK INSERT` statement, which allows them to import data into SQL Server databases from text files. Assistant DBAs should be members of this role.


If you don't want users to have any administrative authority, don't assign them to a server role. This limits them to being normal users.


Builtin\Administrators is automatically made a member of the sysadmin server role, giving SQL Server administrative rights to all your Windows administrators. Because not all your Windows administrators should have these rights, you may want to create a SQLAdmins group in Windows, add your SQL Server administrators to that group, and make the group a member of the sysadmin role. Afterward, you should remove Builtin\Administrators from the sysadmin role.

In this task, you will assign two of the logins you created in the previous two tasks to fixed server roles, thereby limiting their administrative authority.

## Scenario

You have just created several new logins on your SQL Server so your users can access the system. You need to ensure that these accounts have the right amount of administrative access, so you have decided to assign two of these logins to fixed server roles.

# Scope of Task

## Duration

This task should take approximately 15 minutes.

## Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the logins you created in Tasks 1.16 and 1.17.

## Caveat

This task doesn't have any caveats.

# Procedure

In this task, you will assign two of the logins you created to fixed server roles to limit their administrative access.

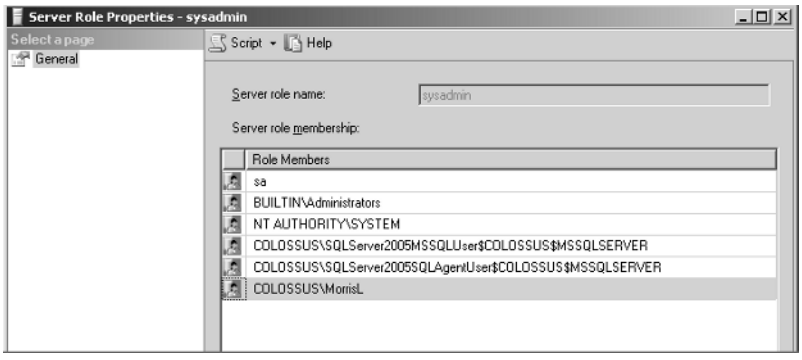
## Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the logins you created in Tasks 1.16 and 1.17.

## Details

Follow these steps to assign the logins to fixed server roles:

1. Open SQL Server Management Studio by selecting it from the SQL Server 2005 group under Programs on the Start menu, expand Security, and expand Server Roles.
2. Double-click the Sysadmin server role to open its properties.
3. Click Add, click Browse, select the check box next to *SqlDomain*\MorrisL, click OK, and then OK again.
4. MorrisL should now appear in the Role Members list.



5. Click OK to exit the Server Role Properties dialog box.
6. Double-click Serveradmin Server Role Properties.
7. Click Add, enter **GibsonH**, and click OK.
8. Click OK to exit the Server Role Properties dialog box.

## Criteria for Completion

This task is complete when you have successfully added MorrisL to the sysadmins fixed server role (as shown in Figure 1.10) and GibsonH to the serveradmin fixed server role (as shown in Figure 1.11).

**FIGURE 1.10** MorrisL should show up in the sysadmins Role Members list.

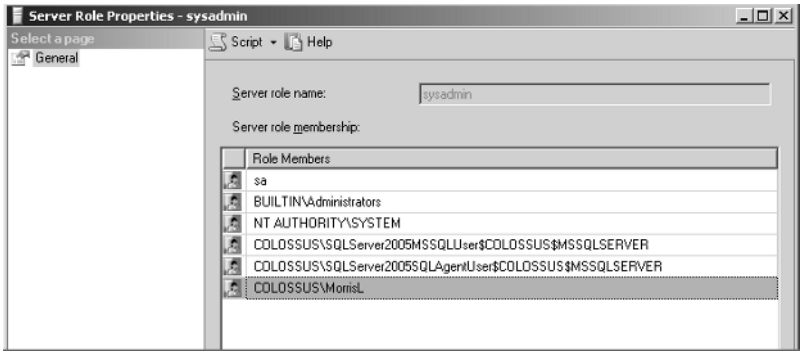



**FIGURE 1.11** GibsonH should show up in the serveradmin Role Members list.

# Task 1.19: Creating a Database User Mapping

Once you have created logins for your users so they can access SQL Server, you need to give them access to databases once they have logged in. You do so by creating database user mappings and then assigning permissions to those user mappings (I discuss permissions later in this book).

If you've looked at the existing database security, you may have noticed that two user mappings already exist in your databases when they are first created: DBO and guest. Members of the sysadmin fixed server role automatically become the database owner (DBO) user in every database on the system. In this way, they can perform all the necessary administrative functions in the databases, such as adding users and creating tables. The *guest user* is a catch-all database user mapping for people who have a SQL Server login but not a user account in the database. These users can log in to the server as themselves and access any database where they don't have a user mapping. The guest account should be limited in function, because anybody with a SQL Server login can use it.

In this task, you will create a database user mapping for GibsonH to access the AdventureWorks database.

## Scenario

You have just created several new logins on your SQL Server so your users can access the system. You didn't create user mappings for all the logins, though, so you need to create a user mapping for GibsonH to access the AdventureWorks database.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the GibsonH login you created in Task 1.17.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create a user mapping for GibsonH to access the AdventureWorks database.

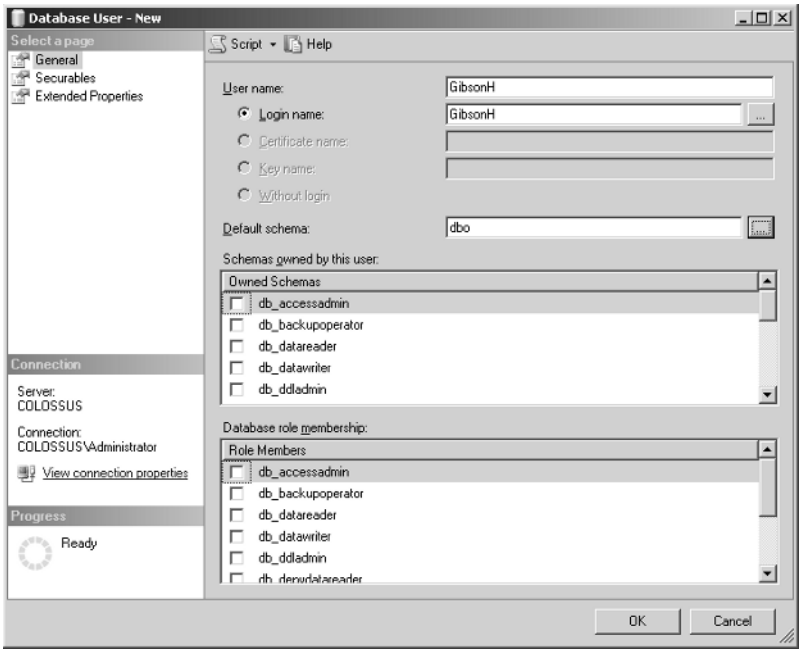
### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the GibsonH login you created in Task 1.17.

### Details

To create a user mapping for GibsonH in AdventureWorks, follow these steps:

1. Open SQL Server Management Studio, and expand your server.
2. Expand Databases by clicking the + sign next to the icon.
3. Expand the AdventureWorks database.
4. Expand Security, and click the Users icon.
5. Right-click Users, and select New User.
6. Click the ellipsis button next to the Login Name box, and click Browse. View all the available names; note that only logins you've already created are available.
7. Select the check box next to GibsonH, and click OK twice.
8. Enter **GibsonH** in the User Name box and **dbo** in the Default Schema box.

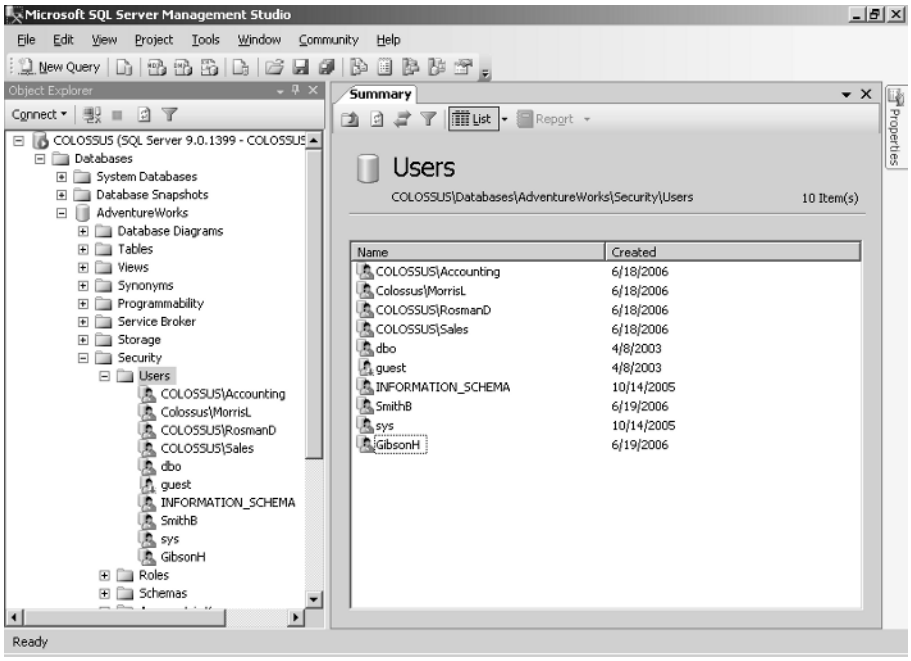



9. Click OK to create the GibsonH database user account.

## Criteria for Completion

This task is complete when you have successfully created a user mapping for GibsonH in the AdventureWorks database, as shown in Figure 1.12.

**FIGURE 1.12** GibsonH should show up in Users list of the AdventureWorks database.




# Task 1.20: Assigning User Mappings to Fixed Database Roles

In SQL Server when several users need permission to access a database, it's much easier to give them all permissions as a group rather than try to manage each user separately. That is what *database roles* are for—granting permissions to groups of database users, rather than granting permissions to each database user separately. Three types of database roles exist: fixed, custom, and application.

*Fixed* database roles have permissions already applied; that is, all you have to do is add users to these roles, and the users inherit the associated permissions. (This is different from

custom database roles, as you'll see later.) You can use several fixed database roles in SQL Server to grant permissions:

**db_owner** Members of this role can do everything the members of the other roles can do as well as some administrative functions.

**db_accessadmin** These users have the authority to say who gets access to the database by adding or removing users.

**db_datareader** Members here can read data from any table in the database.

**db_datawriter** These users can add, change, and delete data from all the tables in the database.

**db_ddladmin** Data Definition Language (DDL) administrators can issue all DDL commands; this allows them to create, modify, or change database objects without viewing the data inside.

**db_securityadmin** Members here can add and remove users from database roles, and they can manage statement and object permissions.

**db_backupoperator** These users can back up the database.

**db_denydatareader** Members can't read the data in the database, but they can make schema changes (for example, add a column to a table).

**db_denydatawriter** These users can't make changes to the data in the database, but they're allowed to read the data.

**Public** The purpose of this group is to grant users a default set of permissions in the database. All database users automatically join this group and can't be removed.


Because all database users are automatically members of the Public database role, you need to be cautious about the permissions that are assigned to the role.

In this task, you will add user mappings to fixed database roles.

## Scenario

You have just created several new logins on your SQL Server and created user mappings for them in the AdventureWorks database. You now need to make sure these users have only the necessary permissions, so you need to add some of them to fixed database roles.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the GibsonH and SmithB logins you created in Task 1.17, the SmithB user mapping you created in Task 1.17, and the GibsonH user mapping you created in Task 1.19.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will add the SmithB user mapping to the db_denydatawriter fixed database role and the GibsonH user mapping to the db_denydatareader fixed database role.

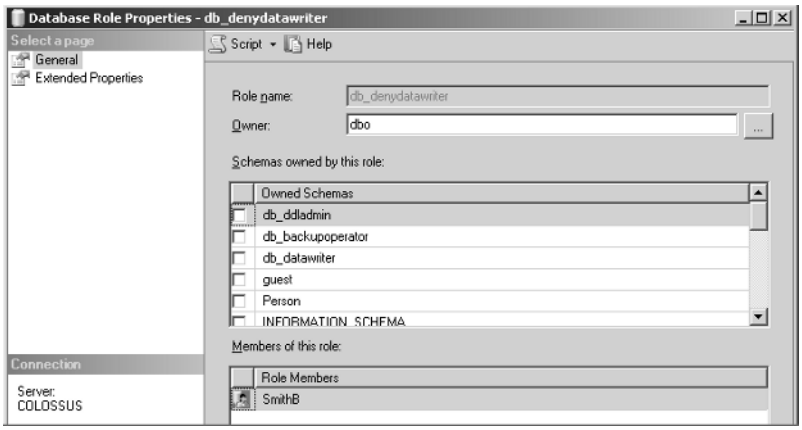
### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the GibsonH and SmithB logins you created in Task 1.17, the SmithB user mapping you created in Task 1.17, and the GibsonH user mapping you created in Task 1.19.

### Details

To add these user mappings to the fixed database roles, follow these steps:

1. Open SQL Server Management Studio, expand your server, and then expand Databases ➢ AdventureWorks.
2. Expand Security, then Roles, and then Database Roles.
3. Right-click db_denydatawriter, and select Properties.
4. Click Add.
5. Enter **SmithB** in the Enter Object Names to Select box, and click OK.



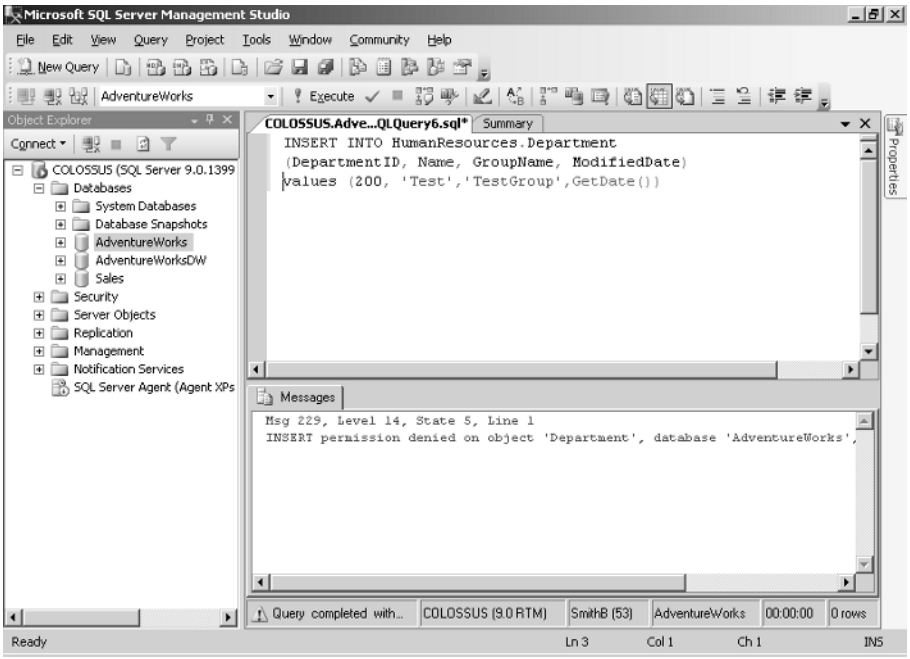
6. Click OK again to return to SQL Server Management Studio.
7. Right-click db_denydatareader, and select Properties.
8. Click Add.
9. Enter **GibsonH** in the Enter Object Names to Select box, and click OK. Then click OK again.

## Criteria for Completion

This task is complete when you have successfully added the SmithB user mapping to the db_denydatawriter fixed database role and the GibsonH user mapping to the db_denydatareader fixed database role. Follow these steps to verify you have been successful:

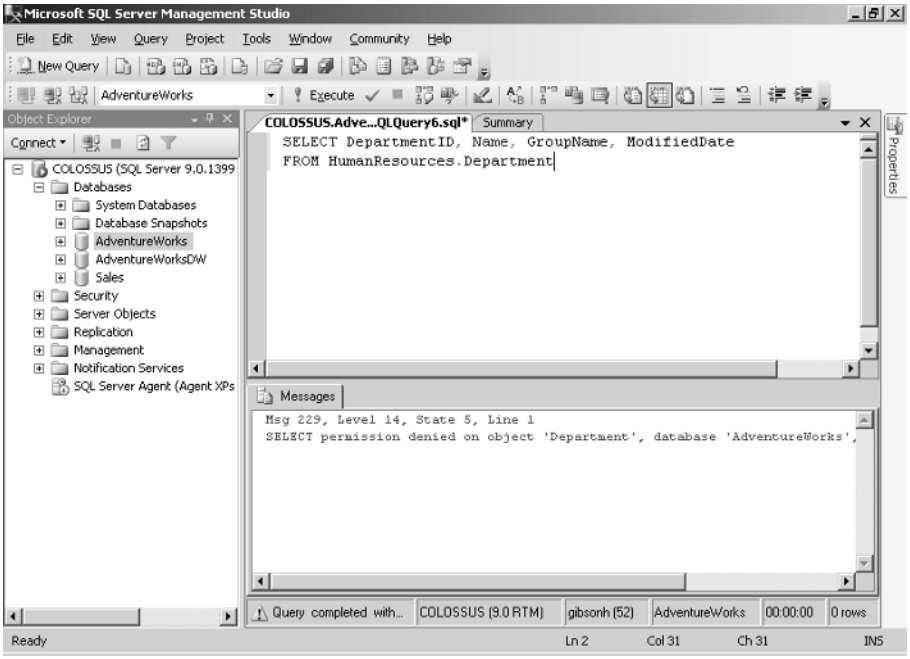
1. Open a new SQL Server query in SQL Server Management Studio.
2. On the Query menu, hover over Connection, and then click Change Connection.
3. Select SQL Server Authentication from the Authentication list box.
4. In the User Name box, enter **SmithB**; in the Password box, enter **Password1,** and click Connect.
5. Enter and execute the following query, which tries to update information in the HumanResources.Department table (it fails because SmithB is a member of the db_denydatawriter role):

```
INSERT INTO HumanResources.Department (DepartmentID, Name, GroupName,
ModifiedDate) values (200, 'Test','TestGroup',GetDate())
```



6. On the Query menu, hover over Connection, and then click Change Connection.
7. Select SQL Server Authentication from the Authentication list box.
8. In the User Name box, enter **GibsonH**; in the Password box, enter **Password1,** and click Connect.
9. Enter and execute the following query, which tries to read data from the Human-Resources.Department table (it fails because GibsonH is a member of the db_ denydatareader role):

```
SELECT DepartmentID, Name, GroupName, ModifiedDate  FROM
HumanResources.Department
```




10. Close the query window.

# Task 1.21: Creating a Custom Database Role

Although quite a few fixed database roles are at your disposal, they will not always meet your security needs. For example, you may have several users who need Select, Update, and Execute permissions in your database and nothing more. Because none of the fixed database roles give that set of permissions, you need to create a *custom* database role. When you create this new

role, you assign permissions to it and then assign users to the role; the users inherit whatever permissions you assign to that role. That is different from the fixed database roles, where you don't need to assign permissions but just need to add users.


You can make your custom database roles members of other database roles. This is referred to as *nesting roles.*

In this task, you will create a custom database role, assign permissions to it, and add user mappings to the new custom role.

## Scenario

You have just created several new logins on your SQL Server and created user mappings for them in the AdventureWorks database. You have added some of these users to fixed database roles, but other users require combinations of permissions that none of the fixed database roles offers. To assign the appropriate permissions to these users, you decide to create a custom database role.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the AdventureWorks database, and the RosmanD login you created in Task 1.16.

### Caveat

In the real world, you would not create a custom database role for just one user, and you would grant more than a single permission.

## Procedure

In this task, you will create a custom database role that grants the members permission to select data from the HumanResources.Department table in the AdventureWorks database. You will then add the RosmanD user mapping to the new custom database role.

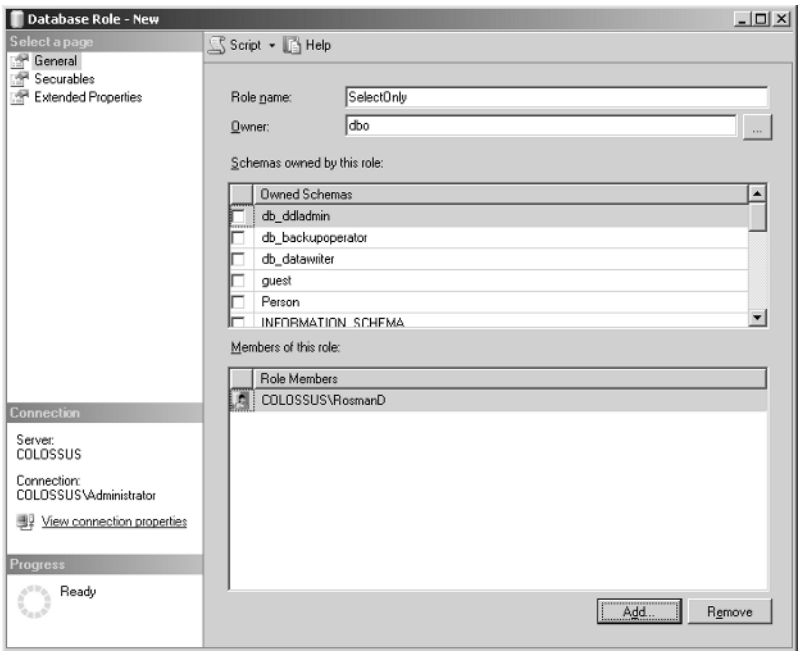
## Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the AdventureWorks database, and the RosmanD login you created in Task 1.16.

## Details

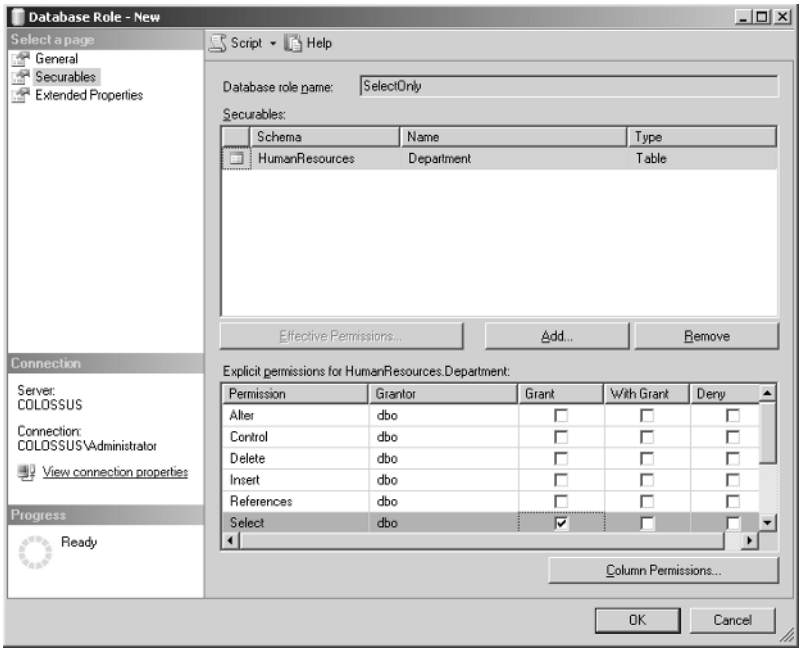
To create a new custom database role and add users to it, follow these steps:

1. Open SQL Server Management Studio, expand your server, and then expand Databases ➤ AdventureWorks.
2. Expand Security and then Roles.
3. Right-click Database Roles, and select New Database Role.
4. In the Role Name box, enter **SelectOnly**, and enter **dbo** in the Owner box.
5. Add ***Sqldomain*\RosmanD** to the Role Members list.




6. On the Securables page, click Add under the Securables list box, select the Specific Objects radio button, and click OK.
7. Click the Objects Type button, select Tables, and click OK.

8. Click Browse, select the HumanResources.Department check box, click OK, and then click OK again.
9. In the Explicit Permissions for HumanResources.Department list, check the Grant check box next to Select, and click OK.



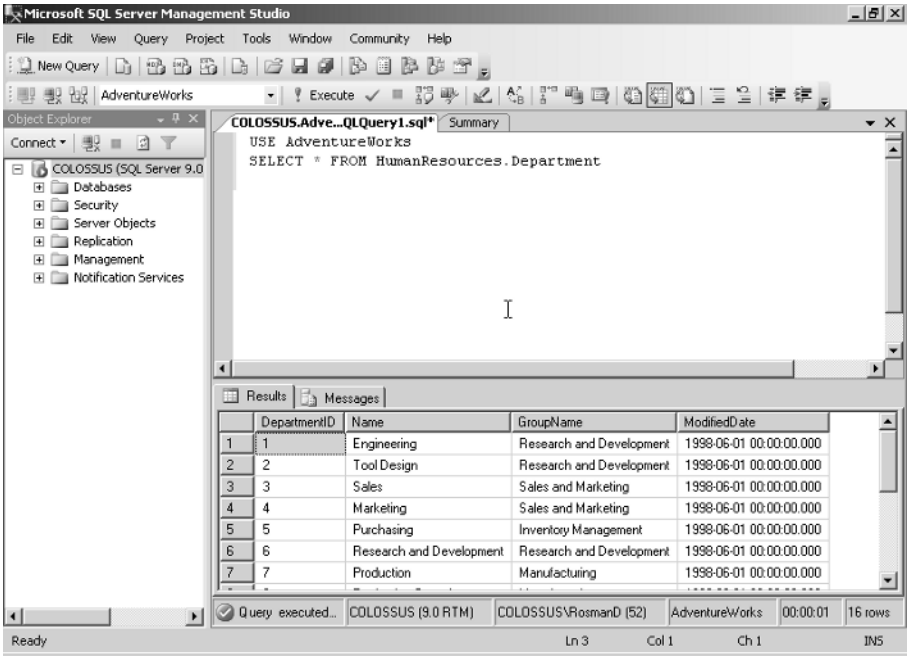

10. Click OK to create the role and return to SQL Server Management Studio.

## Criteria for Completion

This task is complete when you have successfully created a new custom database role, assigned permissions, and added the RosmanD user mapping to the new role. Follow these steps to verify you have been successful:

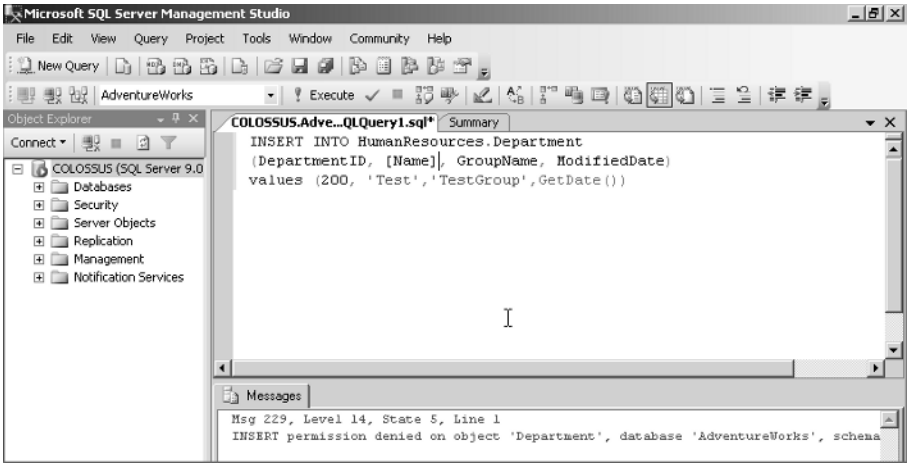
1. Close all programs, log out of Windows, and log back in as RosmanD.
2. Open a new SQL Server query in SQL Server Management Studio, and connect using Windows Authentication.
3. Notice that the following query succeeds because RosmanD is a member of the new SelectOnly role:

```
USE AdventureWorks
SELECT * FROM HumanResources.Department
```




4. Now notice the failure of the next query because RosmanD is a member of a role that is allowed to select only:

```
INSERT INTO HumanResources.Department (DepartmentID, [Name], GroupName,
ModifiedDate) values (200, 'Test','TestGroup',GetDate())
```




5. Close all programs, log out of Windows, and log back in as yourself.

# Task 1.22: Creating an Application Role

Many companies have developed custom applications in-house for manipulating their data; your company may have developed such an application. Such applications can take months, or even years, to build and perfect, so it goes without saying that you want your employees to use the custom application rather than find their own ways to modify the data. That is where application roles come in to play.

An *application* role is a special role that is activated with a password. By implementing this special role, your users can't access data using just their SQL Server login and database account; they must use the proper application. Here is how it works:

1. Create an application role, and assign it permissions.
2. Users open the approved application and are logged in to SQL Server.
3. To enable the application role, the application executes the `sp_setapprole` stored procedure (which is written into the application at design time).

Once the application role is enabled, SQL Server no longer sees users as themselves; it sees users as the application and grants them application role permissions.

In this task, you will create and assign permissions to an application role.

## Scenario

Your company has written a custom application for manipulating data in one of your databases. This application has taken hundreds of staff hours and has cost hundreds of thousands of dollars, so management has insisted employees use this custom application to access the database and nothing else. You have decided the best way to accomplish this goal is to create an application role, which your developers can hard code into their application.

## Scope of Task

### Duration

This task should take approximately 15 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database.

### Caveat

This task doesn't have any caveats.

## Procedure

In this task, you will create an application role that grants the members permission to select data from the HumanResources.Department table in the AdventureWorks database.

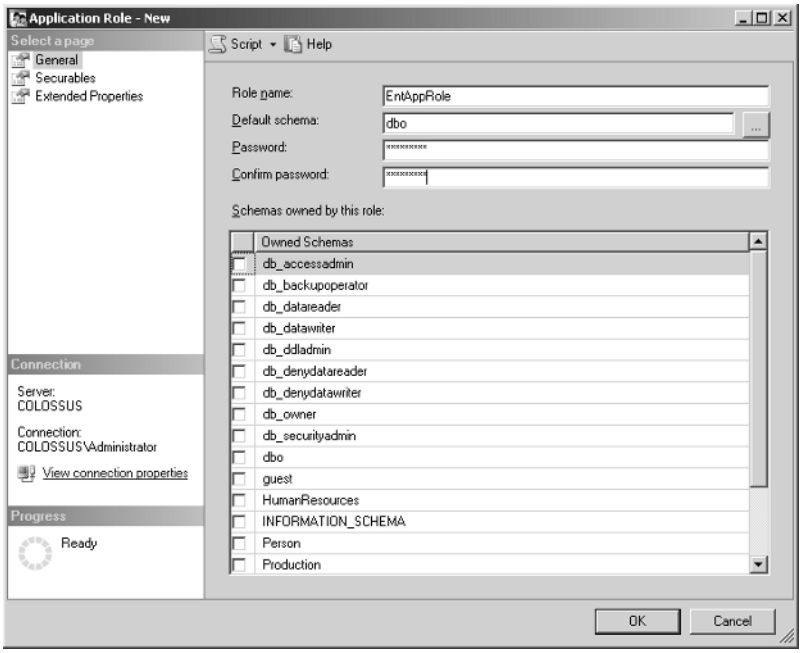
### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1 and the AdventureWorks database.

### Details

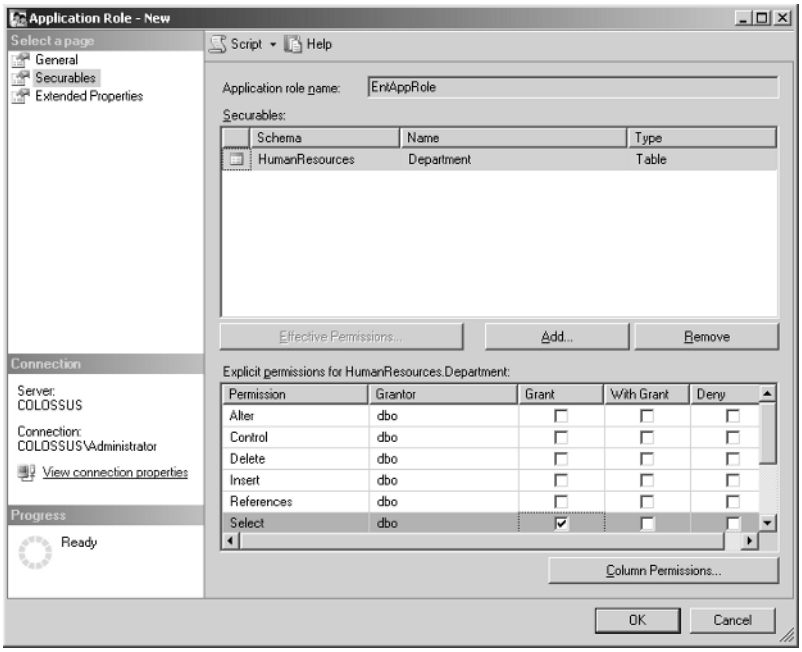
To create an application role and assign permissions to it, follow these steps:

1. Open SQL Server Management Studio, and expand Databases ➢ AdventureWorks ➢ Security.
2. Right-click Application Roles, and select New Application Role.
3. In the Role Name box, enter **EntAppRole**.
4. Enter **dbo** in the Default Schema box.
5. In the Password and Confirm Password boxes, enter **Password1**.




6. On the Securables page, click Add under the Securables list box, select the Specific Objects radio button, and click OK.

7. Click the Objects Type button, select Tables, and click OK.
8. Click Browse, select the HumanResources.Department check box, click OK, and then click OK again.
9. In the Permissions for HumanResources.Department list, select the Grant check box next to Select, and click OK to create the role.

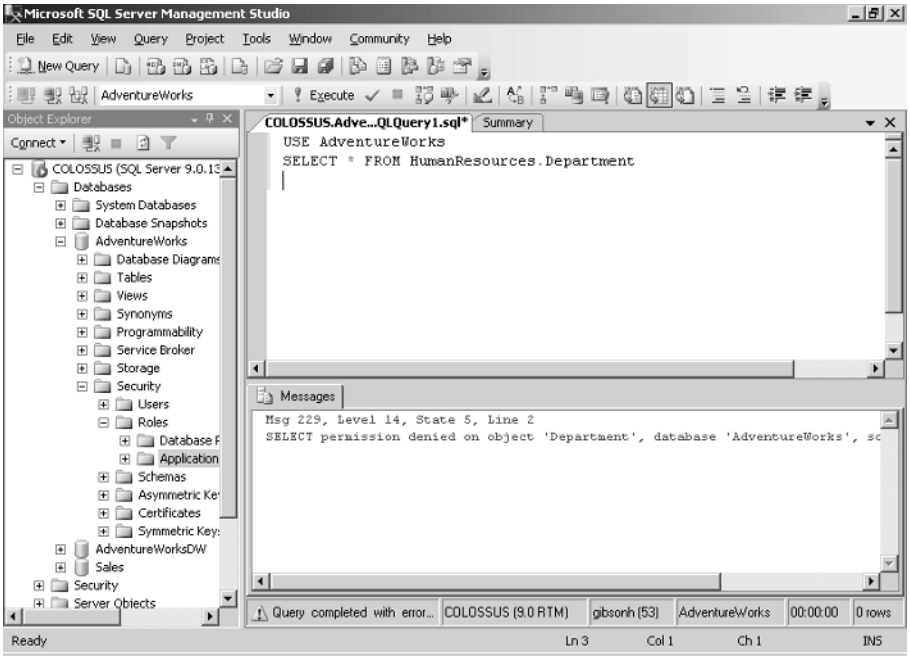



## Criteria for Completion

This task is complete when you have successfully created a new application role and assigned permissions. Follow these steps to verify you have been successful:

1. Open a new SQL Server query in SQL Server Management Studio.
2. On the Query menu, hover over Connection, and click Change Connection.
3. Connect using SQL Authentication with GibsonH as the username and Password1 as the password.
4. Enter and execute the following query. Notice that it fails because GibsonH has been denied Select permissions because of membership in the db_denydatareader database role (assigned in Task 1.20):

```
USE AdventureWorks
SELECT * FROM HumanResources.Department
```




5. To activate the application role, execute the following query:

```
sp_setapprole @rolename='EntAppRole', @password='Password1'
```

6. Clear the query window, and execute the following query. Notice that the query is successful this time. This is because SQL Server now sees you as EntAppRole, which has Select permission.

```
SELECT * FROM HumanResources.Department
```

7. Close the query window.

# Task 1.23: Assigning Permissions

In the past few tasks, you assigned permissions to users by adding them to roles. To understand how these roles work, you need a more complete understanding of what permissions are and how they work.

Any object to which SQL Server regulates access is referred to as a *securable*. Securables can fall under three scopes:

- Server scope
  - Server
  - Endpoint

  - SQL Server login
  - SQL Server login mapped to a Windows login
  - SQL Server login mapped to a certificate
  - SQL Server login mapped to an asymmetric key
- Database scope
  - Database users
  - Database users mapped to a Windows login
  - Database users mapped to a certificate
  - Database users mapped to an asymmetric key
  - Database roles
  - Application roles
  - Assemblies
  - Message type
  - Service contract
  - Service
  - Full-text catalog
  - DDL events
  - Schema
- Schema scope
  - Table
  - View
  - Function
  - Procedure
  - Queue
  - Type
  - Rule
  - Default
  - Synonym
  - Aggregate

All of these objects are secured by applying permissions. You can work with two types of permissions: statement and object permissions.

*Statement* permissions have nothing to do with the actual data; they allow users to create the structure that holds the data. It's important not to grant these permissions haphazardly, because doing so can lead to such problems as wasted server resources. It's best to restrict statement permissions to database administrators (DBAs), assistant DBAs, and developers. These are some good examples of statement permissions:

- Create Database
- Create Table
- Create View
- Create Procedure
- Create Index
- Create Rule
- Create Default


When you create a new database, a record is added to the sysdatabases system table, which is stored in the master database. Therefore, you can grant the `CREATE DATABASE` statement only on the master database.

The second type of permissions is *object* permissions, which control how users work with the actual data. Using object permissions, you can control who is allowed to read from, write to, or otherwise manipulate your data. The 12 object permissions are as follows:

**Control** This permission gives the principal ownership-like capabilities on the object and all objects under it in the hierarchy. For example, if you grant a user Control permission on the database, then the user has Control permission on all the objects in the database, such as tables and views.

**Alter** This permission allows users to create, alter, or drop the securable and any object under it in the hierarchy. The only property the user can't change is ownership.

**Take Ownership** This allows the user to take ownership of an object.

**Impersonate** This permission allows one login or user to impersonate another.

**Create** As the name implies, this permission lets a user create objects.

**View Definition** This permission allows users to see the Transact-SQL syntax that was used to create the object being secured.

**Select** When granted, this permission allows users to read data from the table or view. When granted at the column level, it lets users read from a single column.

**Insert** This permission allows users to insert new rows into a table.

**Update** This permission lets users modify existing data in a table but not add new rows to or delete existing rows from a table. When this permission is granted on a column, users can modify data in that single column.

**Delete** This permission allows users to remove rows from a table.

**References** Tables can be linked together on a common column with a foreign-key relationship, which is designed to protect data across tables. When two tables are linked with a foreign key, this permission allows the user to select data from the primary table without having Select permission on the foreign table.

**Execute** This permission allows users to execute the stored procedure where the permission is applied.

All the permissions in SQL Server can exist in one of three states:

**Grant** Granting allows users to use a specific permission. For instance, if you grant SmithB Select permission on a table, then SmithB can read the data within. You know a permission has been granted when the Allow check box is selected next to the permission in the permissions list.

**Revoke** A revoked permission isn't specifically granted, but a user can inherit the permission if it has been granted to another role of which they are a member. That is, if you revoke the Select permission from SmithB, then SmithB can't use it. If, however, SmithB is a member of a role that has been granted Select permission, SmithB can read the data just as if SmithB had the Select permission. A permission is revoked when neither Allow nor Deny boxes are selected next to a permission.

**Deny** If you deny a permission, the user doesn't get the permission—no matter what. If you deny SmithB Select permission on a table, even if SmithB is a member of a role with Select permission, then SmithB can't read the data. You know a permission has been denied when the Deny check box is selected next to the permission in the permissions list.

In this task, you'll get some hands-on experience with assigning permissions and changing the states of those permissions.

## Scenario

You have just installed a new SQL Server for your company and want to make sure you fully understand how permissions work, so you have decided to test permissions on your test server before assigning them to users on production databases.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the SmithB account you created in Task 1.17, and the AdventureWorks database.

### Caveat

In a production environment, for ease of management, you will create custom roles and assign permissions to those roles. You will usually assign permissions to a specific user only when you want to deny a permission to that user.

## Procedure

In this task, you will assign permissions to the SmithB user mapping. You will then change the permission state and test the effects of the change.

### Equipment Used

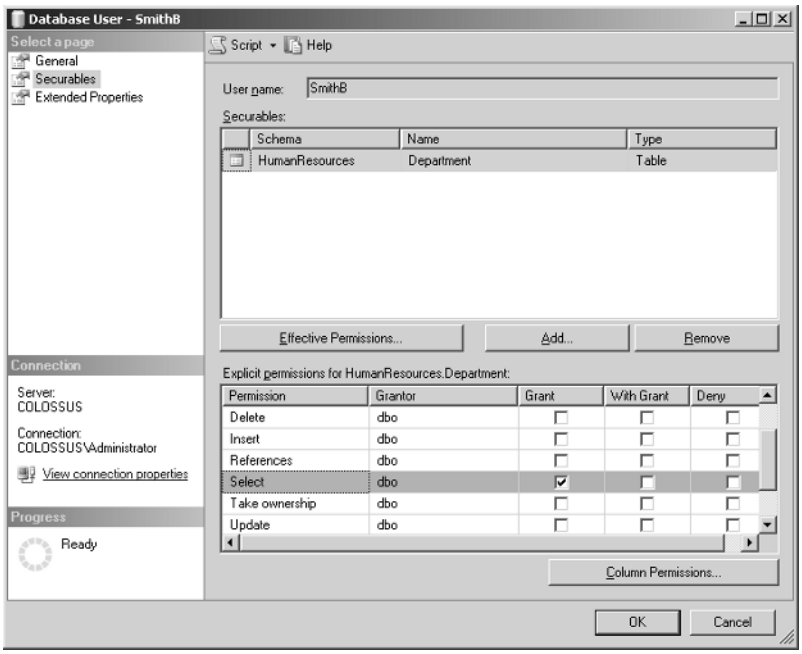
For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1, the SmithB account you created in Task 1.17, and the AdventureWorks database.

### Details

To create assign and modify permissions for SmithB, follow these steps:

1. Open SQL Server Management Studio, expand your server, and then expand Databases ➢ AdventureWorks ➢ Security.
2. Expand Users, right-click SmithB, and select Properties.
3. On the Securables page, click Add under the Securables list box, select the Specific Objects radio button, and click OK.
4. Click the Objects Type button, select Tables, and click OK.
5. Click Browse, select the HumanResources.Department check box, and click OK twice.

6. In the Permissions for HumanResources.Department list, select the Grant check box next to Select, and click OK.

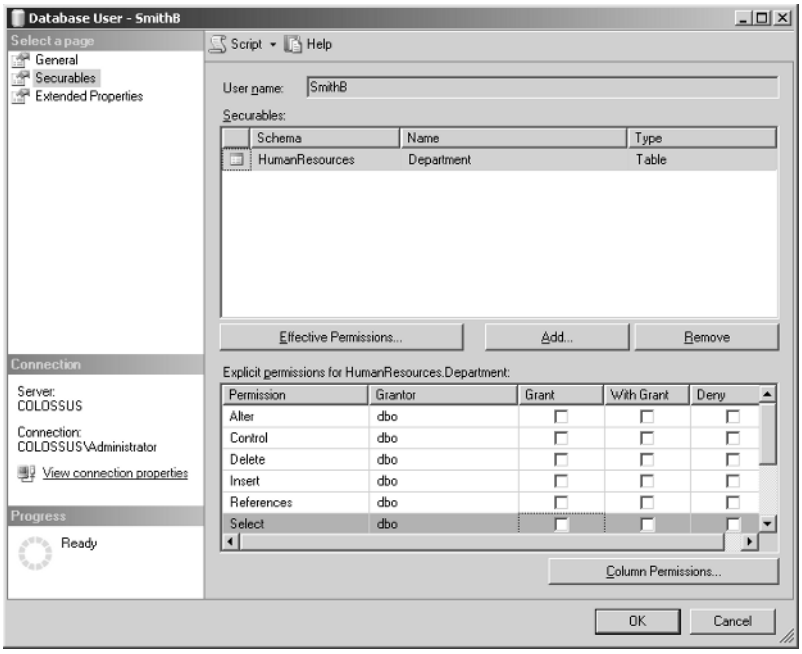



7. Open a new SQL Server query in SQL Server Management Studio.
8. On the Query menu, hover over Connection, and then click Change Connection.
9. Select SQL Server Authentication from the Authentication list box.
10. In the User Name box, enter **SmithB**; in the Password box, enter **Password1,** and click Connect.
11. Execute the following query. It's successful because SmithB has Select permission on the HumanResources.Department table.

```
USE AdventureWorks
SELECT * FROM HumanResources.Department
```

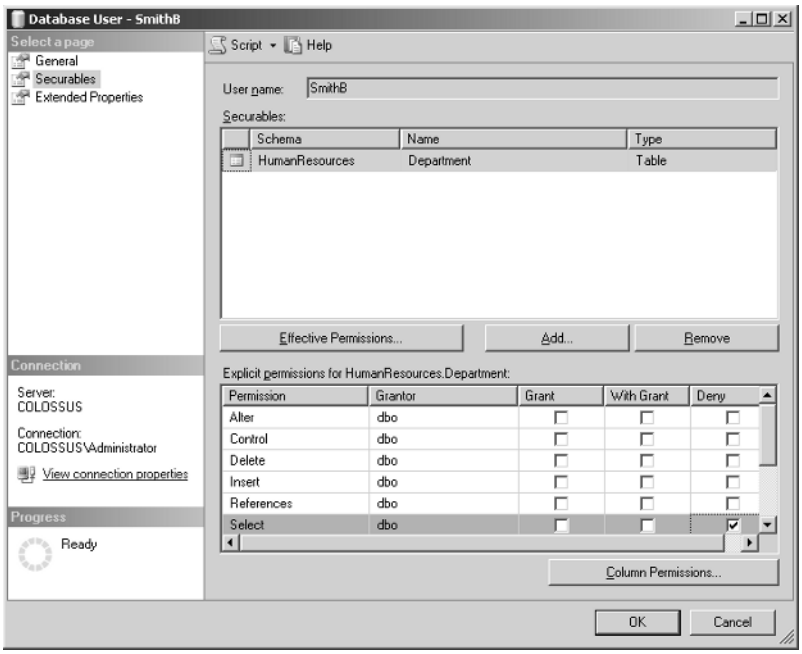
12. Right-click SmithB under Users in the AdventureWorks database, and select Properties.
13. On the Securables page, click Add under the Securables list box, select the Specific Objects radio button, and click OK.
14. Click the Objects Type button, select Tables, and click OK.

15. Click Browse, select the HumanResources.Department check box, and click OK.
16. In the Permissions for HumanResources.Department list, uncheck the Grant check box next to Select (this revokes the permission), and click OK.




17. Return to the query window, and execute the query in step 11. It fails because SmithB doesn't have explicit Select permission.
18. Right-click SmithB under Users in the AdventureWorks database, and select Properties.
19. Under Role Membership, select the check box next to the db_datareader role.
20. Return to the query window, and rerun the query from step 11. Now it fails because SmithB does not have the permission expressly applied and is not a member of a role that has this permission.
21. Right-click SmithB under Users in the AdventureWorks database, and select Properties.
22. On the Securables page, click Add under the Securables list box, select the Specific Objects radio button, and click OK.
23. Click the Objects Type button, select Tables, and click OK.
24. Click Browse, select the HumanResources.Department check box, and click OK.

25. In the Permissions for HumanResources.Department list, select the Deny check box next to Select, and click OK.




26. Return to the query window, and again run the query from step 11. It fails this time because you've specifically denied SmithB access.

## Criteria for Completion

This task is complete when you have successfully assigned the permissions as described in the "Details" section.

# Task 1.24: Configuring Encrypted Connections

Thus far, you've seen how to protect your data from intruders by granting access and applying permissions to objects. But when someone legitimately accesses the server and starts transferring data, it travels over the network. If you need really robust security, you can go so far as to encrypt the data as it travels between the client and the server over the network. That way, if anyone is reading your network traffic, they will not be able to interpret the data.

Just like a secure web page, SQL Server uses Secure Sockets Layer (SSL) encryption. This means that to encrypt your connections to SQL Server, you first need to get a certificate, which you can get from one of the major vendors such as VeriSign or can install Windows Certificate services and supply your own.


If you want to test encryption but do not have a certificate, SQL Server can generate a self-signed certificate for you to use.


Do not use a self-signed certificate in production; they do not provide strong security and are susceptible to attack. They are for testing and development only.

Although using encryption can help protect your data in a highly secured environment, using encryption has a few drawbacks:

- An extra network round-trip is required at connect time.
- Packets sent from the application to the instance of SQL Server must be encrypted by the client Net-Library and decrypted by the server Net-Library.
- Packets sent from the SQL Server instance to the application must be encrypted by the server Net-Library and decrypted by the client Net-Library.

This means encryption can negatively impact SQL Server's performance, so use encryption only when necessary.

In this task, you'll configure SQL Server to use encryption using a self-signed certificate.

## Scenario

You have just installed a new SQL Server for your company and created a database that holds e-commerce data. For your customers' convenience, your company has decided to store credit card information so users do not have to enter it every time they place an order on your site. You know you need to use the highest level of security possible, so you have decided to configure SQL Server to require encrypted connections.

## Scope of Task

### Duration

This task should take approximately 30 minutes.

### Setup

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1.

### Caveat

In production, you should not use a self-signed certificate. You are using one in this task for simplicity's sake.

## Procedure

In this task, you will configure SQL Server to use encrypted connections using a self-signed certificate.

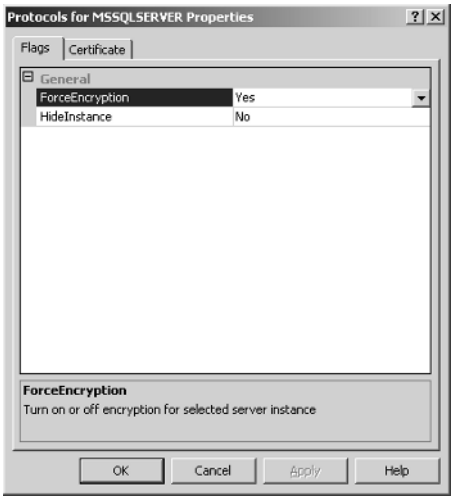
### Equipment Used

For this task, you need access to the machine you installed SQL Server 2005 on in Task 1.1.

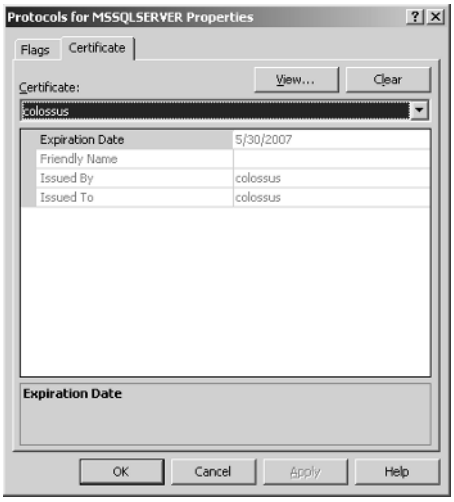
### Details

To configure the server to force clients to use encrypted connections, follow these steps:

1. In SQL Server Configuration Manager, expand SQL Server 2005 Network Configuration, right-click Protocols for <server instance>, and then select Properties.
2. In the Protocols for <instance name> Properties dialog box, on the Flags tab, change ForceEncryption to Yes.




3. Click OK twice to close the warning dialog box that opens.
4. Restart the SQL Server service.
5. Again, right-click Protocols for <server instance>, and then select Properties.
6. In the Protocols for <instance name> Properties dialog box, on the Certificate tab, select the desired certificate (in this case the self-signed certificate) from the Certificate drop-down list, and then click OK.




7. Click OK to close the warning dialog box that opens.
8. Restart the SQL Server service.

Next, you need to configure the clients to request encrypted connections to the server. Here's how:

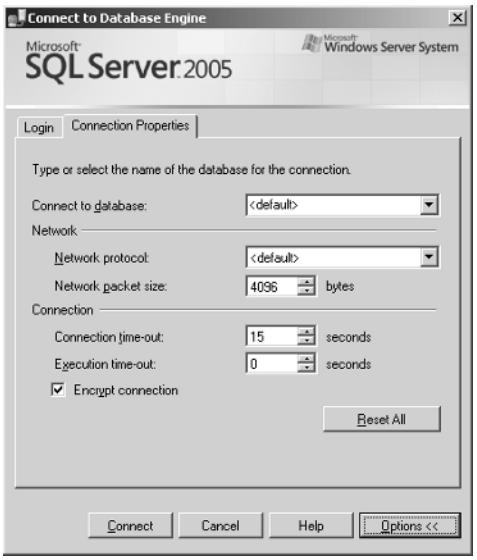
1. In SQL Server Configuration Manager, right-click SQL Native Client Configuration, and select Properties.
2. On the Flags tab, in the Force Protocol Encryption box, select Yes, and then click OK to close the dialog box.


3. Restart the SQL Server service.

## Criteria for Completion

This task is complete when you can connect to SQL Server using an encrypted connection. To verify this, follow these steps:

1. Open a new SQL Server query in SQL Server Management Studio.
2. On the Query menu, hover over Connection, and then click Change Connection.
3. Select Windows Authentication from the Authentication list box.
4. Click the Options button, and check the Encrypt Connection box.




5. Click Connect to make the connection.