Chapter 2
How to use Management Studio

Before you start the exercises…

Before you start these exercises, you need to install SQL Server and the SQL Server Management Studio. The procedures for doing both of these tasks are provided in appendix A of the book.

In addition, your instructor will provide you with scripts that contain starting code for many of these exercises. By default, these scripts will be in a directory named MGS Exercise Starts. If your instructor has changed the name of the directory or makes the scripts available individually, adjust the following instructions accordingly.

Exercises

In these exercises, you'll use SQL Server Management Studio to create the MyGuitarShop database, review the tables in the MyGuitarShop database, and enter SQL statements and run them against this database.

Create the database

1. Start SQL Server Management Studio and open a connection using either Windows or SQL Server authentication.
2. Open the script file named CreateMyGuitarShop.sql that's in the MGS Exercise Starts directory by clicking the Open File button in the toolbar and then using the resulting dialog box to locate and open the file.
3. Execute the entire script by clicking the Execute button in the SQL Editor toolbar or by pressing F5. When you do, the Messages tab indicates whether the script executed successfully.

Review the database

4. In the Object Explorer window, expand the node for the database named MyGuitarShop so you can see all of the database objects it contains. If it isn't displayed in the Object Explorer window, you may need to right-click on the name of the connection and select Refresh to display it.
5. View the data for the Categories and Products tables.
6. Navigate through the database objects and view the column definitions for the Categories and Products tables.

Enter and run some test SQL statements

7. Open a new Query Editor window by clicking the New Query button in the toolbar. Then, select the MyGuitarShop database from the Available Databases dropdown menu (Ctrl + U) to choose it as the default database.
8. Enter and run this SQL statement:

```
SELECT ProductName FROM Products
```

9. Delete the e at the end of ProductName and run the statement again. This shows you what an error caused by a typo looks like.
10. Open another Query Editor window and then enter and run this statement:

```
SELECT COUNT(*) AS NumberOfProducts
FROM Products
```

Open and run scripts

11. Open the script named ProductDetails.sql that's in the MGS Exercise Starts directory. Note that this script contains just one SQL statement. Then, run the statement.
12. Open the script named ProductSummary.sql that's in the MGS Exercise Starts directory. Note that this opens another Query Editor window. Then, run the statement.
13. Open the script named ProductStatements.sql that's in the MGS Exercise Starts directory. Notice that this script contains two SQL statements that end with semicolons.
14. Press the F5 key or click the Execute button to run both of the statements in the script. Note that this displays the results in two Results tabs. Make sure to view the results of both SELECT statements.
15. Exit from SQL Server Management Studio.

Chapter 3
How to retrieve data from a single table

Exercises

Enter and run your own SELECT statements

In these exercises, you'll enter and run your own SELECT statements.

1. Write a SELECT statement that returns four columns from the Products table: ProductCode, ProductName, ListPrice, and DiscountPercent. Then, run this statement to make sure it works correctly.

   Add an ORDER BY clause to this statement that sorts the result set by list price in descending sequence. Then, run this statement again to make sure it works correctly. This is a good way to build and test a statement, one clause at a time.

2. Write a SELECT statement that returns one column from the Customers table named FullName that joins the LastName and FirstName columns.

   Format this column with the last name, a comma, a space, and the first name like this:

   ```
   Doe, John
   ```

   Sort the result set by last name in ascending sequence.

   Return only the customers whose last name begins with a letter from M to Z.

3. Write a SELECT statement that returns these column names and data from the Products table:

ProductName	The ProductName column
ListPrice	The ListPrice column
DateAdded	The DateAdded column

   Return only the rows with a list price that's greater than 500 and less than 2000.

   Sort the result set in descending sequence by the DateAdded column.

4. Write a SELECT statement that returns these column names and data from the Products table:

ProductName	The ProductName column
ListPrice	The ListPrice column
DiscountPercent	The DiscountPercent column
DiscountAmount	A column that's calculated from the previous two columns
DiscountPrice	A column that's calculated from the previous three columns

   Sort the result set by discount price in descending sequence.

5. Write a SELECT statement that returns these column names and data from the OrderItems table:

ItemID	The ItemID column
ItemPrice	The ItemPrice column
DiscountAmount	The DiscountAmount column
Quantity	The Quantity column
PriceTotal	A column that's calculated by multiplying the item price by the quantity
DiscountTotal	A column that's calculated by multiplying the discount amount by the quantity
ItemTotal	A column that's calculated by subtracting the discount amount from the item price and then multiplying by the quantity

   Only return rows where the ItemTotal is greater than 500.

   Sort the result set by item total in descending sequence.

Work with nulls and test expressions

6. Write a SELECT statement that returns these columns from the Orders table:

OrderID	The OrderID column
OrderDate	The OrderDate column
ShipDate	The ShipDate column

   Return only the rows where the ShipDate column contains a null value.

7. Write a SELECT statement without a FROM clause that creates a row with these columns and values:

Price	100
TaxRate	.07
TaxAmount	The price multiplied by the tax rate
Total	The price plus the tax amount

   To calculate the fourth column, add the expressions you used for the first and third columns.

Chapter 4

How to retrieve data from two or more tables

Exercises

1. Write a SELECT statement that joins the Categories table to the Products table and returns these columns: CategoryName, ProductName, ListPrice.

   Sort the result set by CategoryName and then by ProductName in ascending order.

2. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: FirstName, LastName, Line1, City, State, ZipCode.

   Return one row for each address for the customer with an email address of allan.sherwood@yahoo.com.

3. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: FirstName, LastName, Line1, City, State, ZipCode.

   Code the join so only addresses that are the shipping address for a customer are returned.

4. Write a SELECT statement that joins the Customers, Orders, OrderItems, and Products tables. This statement should return these columns: LastName, FirstName, OrderDate, ProductName, ItemPrice, DiscountAmount, and Quantity.

   Use aliases for the tables.

   Sort the final result set by LastName, OrderDate, and ProductName.

5. Write a SELECT statement that returns the ProductName and ListPrice columns from the Products table.

   Return one row for each product that has the same list price as another product. (Hint: Use a self-join to check that the ProductID columns aren't equal but the ListPrice column is equal.)

   Sort the result set by ProductName.

6. Write a SELECT statement that returns these two columns:

CategoryName	The CategoryName column from the Categories table
ProductID	The ProductID column from the Products table

   Return one row for each category that has never been used. (Hint: Use an outer join and only return rows where the ProductID column contains a null value.)

7. Use the UNION operator to generate a result set consisting of three columns from the Orders table:

ShipStatus	A calculated column that contains a value of SHIPPED or NOT SHIPPED
OrderID	The OrderID column
OrderDate	The OrderDate column

   If the order has a value in the ShipDate column, the ShipStatus column should contain a value of SHIPPED. Otherwise, it should contain a value of NOT SHIPPED.

   Sort the final result set by OrderDate.

Chapter 5
How to code summary queries

Exercises

1. Write a SELECT statement that returns these columns:

   The count of the number of orders in the Orders table

   The sum of the TaxAmount column in the Orders table

2. Write a SELECT statement that returns one row for each category that has products. The statement should return these columns:

   The CategoryName column from the Categories table

   The number of products in each category

   The list price of the most expensive product in each category

   Sort the result set so the category with the most products appears first.

3. Write a SELECT statement that returns one row for each customer that has orders. The statement should return these columns:

   The EmailAddress column from the Customers table

   The total price of the items the customer has ordered (Hint: multiply the item price in the OrderItems table by the quantity in the OrderItems table)

   The total discount applied to the items the customer has ordered (Hint: multiply the discount amount column in the OrderItems table by the quantity in the OrderItems table)

   Sort the result set in descending sequence by the price total for each customer.

4. Write a SELECT statement that returns one row for each customer that has orders. The statement should return these columns:

   The EmailAddress column from the Customers table

   A count of the number of orders for each customer

   The total amount for the orders for each customer (Hint: First, subtract the discount amount from the price. Then, multiply by the quantity.)

   Return only those rows where the customer has more than 1 order.

   Sort the result set in descending sequence by the sum of the line item amounts.

5. Modify the solution to exercise 4 so it only counts and totals line items that have an ItemPrice value that's greater than 400.

6. Write a SELECT statement that answers this question: What is the total amount ordered for each product? Return these columns:

   The product name from the Products table

   The total amount for each product in the OrderItems table (Hint: You can calculate the total by subtracting the discount amount from the item price and then multiplying it by the quantity)

   Use the ROLLUP operator to include a row that gives the grand total.

7. Write a SELECT statement that answers this question: Which customers have ordered more than one type of product? Return these columns:

   The email address from the Customers table

   The count of distinct products from the customer's orders

Chapter 6
How to code subqueries

Exercises

1. Write a SELECT statement that returns the same result set as this SELECT statement, but don't use a join. Instead, use a subquery in a WHERE clause that uses the IN operator.

```
SELECT DISTINCT CategoryName
FROM Categories c JOIN Products p
  ON c.CategoryID = p.CategoryID
ORDER BY CategoryName
```

2. Write a SELECT statement that answers this question: Which products have a list price that's greater than the average list price for all products?

   Return the ProductName and ListPrice columns for each product.

   Sort the results by the ListPrice column in descending sequence.

3. Write a SELECT statement that returns the CategoryName column from the Categories table.

   Return one row for each category that has never been assigned to any product in the Products table. To do that, use a subquery introduced with NOT EXISTS.

4. Write a SELECT statement that returns three columns: EmailAddress, OrderID, and the order total for each order. To do this, you can group the result set by the EmailAddress and OrderID columns. Then, you can calculate the order total from the columns in the OrderItems table.

   Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the EmailAddress column.

5. Write a SELECT statement that returns the name and discount percent for each product that has a unique discount percent. In other words, don't include products that have the same discount percent as another product.

   Sort the results by the ProductName column.

6. Use a correlated subquery to return one row per customer with each customer's oldest order (the one with the earliest date). Each row should include these three columns: EmailAddress, OrderID, and OrderDate.

Chapter 7
How to insert, update, and delete data

Exercises

1. Write an INSERT statement that adds a category named "Brass" to the Categories table. (Hint: Remember that you do not need to specify a value for identity columns.)
2. Write an UPDATE statement that modifies the row you just added to the Categories table. This statement should change the CategoryName from "Brass" to "Woodwinds", and it should use the CategoryID column to identify the row.
3. Write a DELETE statement that deletes the row you added to the Categories table in exercise 1. This statement should use the CategoryID column to identify the row.
4. Write an INSERT statement that adds a row to the Products table with the following values:

ProductID:	The next automatically generated ID
CategoryID:	4
ProductCode:	dgx_640
ProductName:	Yamaha DGX 640 88-Key Digital Piano
Description:	Long description to come.
ListPrice:	799.99
DiscountPercent:	0 *(this is the default)*
DateAdded:	The current date and time.

   Use a column list for this statement.
5. Write an UPDATE statement that modifies the product you added in exercise 4. This statement should change the DiscountPercent column from 0% to 35%.
6. Write a DELETE statement that deletes the row in the Categories table that has an ID of 4. Then, run the statement. When you do, it will produce an error since the category has related rows in the Products table. To fix that, precede the DELETE statement with another DELETE statement that deletes all products in this category from the Products table.
7. Write an INSERT statement that adds a row to the Customers table with the following values:

EmailAddress:	rick@raven.com
Password:	*(empty string)*
FirstName:	Rick
LastName:	Raven

   Use a column list for this statement.
8. Write an UPDATE statement that modifies the Customers table. Change the password to "secret" for the customer with an email address of rick@raven.com.
9. Write an UPDATE statement that modifies the Customers table. Change the password to "reset" for every customer in the table.

10. Open the script named CreateMyGuitarShop.sql that's in the MGS Exercise Starts directory. Then, run this script. That should restore the data that's in the database. If an error message is displayed indicating that the database is in use, you'll need to close and restart Management Studio and then run the script again.

Chapter 8
How to work with data types

Exercises

1. Write a SELECT statement that returns these columns from the Products table:

   The ListPrice column

   A column that uses the CAST function to return the ListPrice column with 1 digit to the right of the decimal point

   A column that uses the CONVERT function to return the ListPrice column as an integer

   A column that uses the CAST function to return the ListPrice column as an integer

2. Write a SELECT statement that returns these columns from the Products table:

   The DateAdded column

   A column that uses the CAST function to return the DateAdded column with its date only (year, month, and day)

   A column that uses the CAST function to return the DateAdded column with its full time only (hour, minutes, seconds, and milliseconds)

   A column that uses the CAST function to return the DateAdded column with just the month and day

3. Write a SELECT statement that returns these columns from the Orders table:

   A column that uses the CONVERT function to return the OrderDate column in this format: MM/DD/YY. In other words, use two digits for the month, day, and year, and separate each date component with slashes.

   A column that uses the CONVERT function to return the OrderDate column with the date, and the hours and minutes on a 12-hour clock with an am/pm indicator.

   A column that uses the CONVERT function to return the OrderDate column with just the time in a 24-hour format, including the milliseconds.

Chapter 9
How to use functions

Exercises

1. Write a SELECT statement that returns these columns from the Products table:

   The ListPrice column

   The DiscountPercent column

   A column named DiscountAmount that uses the previous two columns to calculate the discount amount and uses the ROUND function to round the result to 2 decimal places. (Note: Even though the result will be rounded, you may still have more than two zeroes after the decimal place as the data type does not change with ROUND.)

2. Write a SELECT statement that returns these columns from the Orders table:

   The OrderDate column

   A column that returns only the four-digit year that's stored in the OrderDate column

   A column that returns only the day of the month that's stored in the OrderDate column

   A column that returns the result from adding thirty days to the OrderDate column

3. Write a SELECT statement that returns these columns from the Orders table:

   The CardNumber column

   The length of the CardNumber column

   The last four digits of the CardNumber column

   When you get that working right, add the column that follows to the result set. This is more difficult because the column requires you to nest functions within functions.

   A column that displays the last four digits of the CardNumber column in this format: XXXX-XXXX-XXXX-1234. In other words, use Xs for the first 12 digits of the card number and actual numbers for the last four digits of the number.

4. Write a SELECT statement that returns these columns from the Orders table:

   The OrderID column

   The OrderDate column

   A column named ApproxShipDate that's calculated by adding 2 days to the OrderDate column

   The ShipDate column

   A column named DaysToShip that shows the number of days between the order date and the ship date

   When you get that working right, add a WHERE clause that retrieves just the orders placed in January 2023.

Chapter 10
How to design a database

Exercises

1. Create a database diagram that shows the relationships between the seven tables in the MyGuitarShop database. (Hint: The Administrators table is not related to the other six tables.)
2. Create a database diagram for a database that stores information about what files the users of a website download where each file is related to a product.

   Each user must have an email address, first name, and last name.

   Each user can have one or more downloads.

   Each download must have a filename and download date/time.

   Each product can be related to one or more downloads.

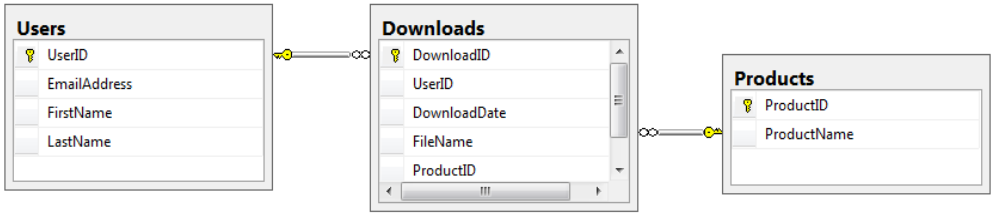
   Each product must have a name.

Chapter 11

How to create and maintain a database with SQL

Exercises

1. Write a script that adds an index to the MyGuitarShop database for the zip code field in the Addresses table.
2. Write a script that implements the following design to create a database named MyWebDB:



   Include a statement to drop the MyWebDB database if it already exists.

   Include statements to create and select the MyWebDB database.

   In the Downloads table, the UserID and ProductID columns are foreign keys.

   Choose appropriate data types for each column in the tables. (Hint: Use the int, varchar, and datetime2 data types.)

   Define the EmailAddress and ProductName columns so that each value must be unique.

   In the Users table, the EmailAddress and LastName columns are required, but the FirstName column is not.

   Include any indexes that you think are necessary.

3. Write a script that adds rows to the database that you created in exercise 2.

   Add two rows to the Users table and two rows to the Products table. You can choose any names you like or use names that return the same results as in the example table below.

   Add three rows to the Downloads table: one row for user 1 and product 1; one for user 2 and product 1; and one for user 2 and product 2. Use the GETDATE function to insert the current date and time into the DownloadDate column.

   Write a SELECT statement that joins the three tables and retrieves the data from these tables like this:

	EmailAddress	FirstName	LastName	DownloadDate	Filename	ProductName
1	johnsmith@gmail.com	John	Smith	2020-03-09 11:09:49.7400000	petals_are_falling.mp3	Local Music Vol 1
2	janedoe@yahoo.com	Jane	Doe	2020-03-09 11:09:49.7400000	turn_signal.mp3	Local Music Vol 1
3	janedoe@yahoo.com	Jane	Doe	2020-03-09 11:09:49.7400000	one_horse_town.mp3	Local Music Vol 2

   Sort the results by the email address in descending order and the product name in ascending order.

4. Write a script that uses ALTER TABLE statements to add the following new columns to the Products table created in exercise 2:

   Add one column for product price that provides for up to three digits to the left of the decimal point and two to the right. This column should have a default value of 9.99.

   Add one column for the date and time that the product was added to the database.

5. Write a script that uses an ALTER TABLE statement to modify the Users table created in exercise 2 so the FirstName column cannot store null values and can store a maximum of 20 characters.

   In the same script, code an UPDATE statement that attempts to insert a row with a null value in the FirstName column. It should fail due to the new not null constraint.

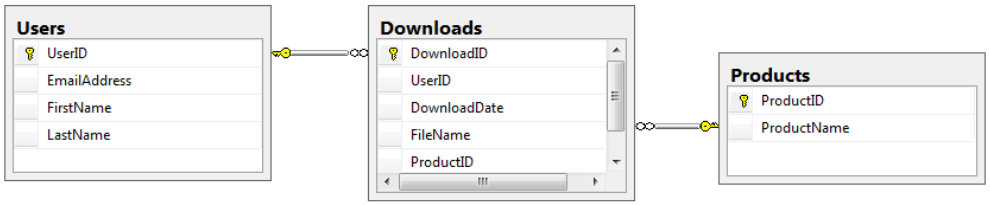
   Code another UPDATE statement that attempts to insert a first name that's longer than 20 characters. It should fail due to the length of the column.

Chapter 12

How to create and maintain a database with Management Studio

Exercises

1. Use the Management Studio to create a new database named MyWebDB using the default settings. (If the database already exists, use the Management Studio to delete it and then recreate it.)
2. Use the Management Studio to create the following tables and relationships.



3. Define the UserID column in the Users table, the ProductsID column in the Products table, and the DownloadID column in the Downloads table as primary keys and identity columns.
4. In the Downloads table, set the UserID and ProductID columns as foreign keys.
5. Choose appropriate data types for each column in the tables. (Hint: Use the int, varchar, and datetime2 data types.)
6. Set the EmailAddress and ProductName columns so that each value must be unique.
7. Define the columns so none of them allow null values.
8. Use the Management Studio to create indexes for the foreign keys in the Downloads table.
9. Use the Management Studio to create a unique index on the EmailAddress and ProductName columns and a regular index on the DownloadDate column.

Chapter 13
How to work with views

Exercises

1. Create a view named CustomerAddresses that shows the shipping and billing addresses for each customer in the MyGuitarShop database.

   This view should return these columns from the Customers table: CustomerID, EmailAddress, LastName and FirstName.

   This view should return columns with the following names created from the Addresses table: BillLine1, BillLine2, BillCity, BillState, BillZip, ShipLine1, ShipLine2, ShipCity, ShipState, and ShipZip.

   Use the BillingAddressID and ShippingAddressID columns in the Customers table to determine which addresses are billing addresses and which are shipping addresses.

   (Hint: You can join the Addresses table to the Customers table twice, once for each type of address).

2. Write a SELECT statement that returns these columns from the CustomerAddresses view that you created in exercise 1: CustomerID, LastName, FirstName, BillLine1.

3. Write an UPDATE statement that updates the CustomerAddresses view you created in exercise 1 so it sets the first line of the shipping address to "1990 Westwood Blvd." for the customer with an ID of 8.

4. Create a view named OrderItemProducts that returns columns from the Orders, OrderItems, and Products tables.

   This view should return these columns from the Orders table: OrderID, OrderDate, TaxAmount, and ShipDate.

   This view should return these columns from the OrderItems table: ItemPrice, DiscountAmount, FinalPrice (the discount amount subtracted from the item price), Quantity, and ItemTotal (the quantity times the final price).

   This view should return the ProductName column from the Products table.

5. Create a view named ProductSummary that uses the view you created in exercise 4 to return some summary information about each product.

   Each row should include these columns: ProductName, OrderCount (the number of times the product has been ordered), and OrderTotal (the total sales for the product).

6. Write a SELECT statement that uses the view that you created in exercise 5 to get the total sales for each of the five best selling products.

   Sort the results by the total sales in descending order.

Chapter 14
How to code scripts

Exercises

1. Write a script that declares a variable and sets it to the count of all products in the Products table. If the count is greater than or equal to 7, the script should display a message that says, "The number of products is greater than or equal to 7". Otherwise, it should say, "The number of products is less than 7".
2. Write a script that uses variables to store (1) the count of all of the products in the Products table and (2) the average list price for those products. If the product count is greater than or equal to 7, the script should print a message that displays the values of both variables. Otherwise, the script should print a message that says, "The number of products is less than 7".
3. Write a script that calculates the common factors between 10 and 20. To find a common factor, you can use the modulo operator (%) to check whether a number can be evenly divided into both numbers. Then, this script should print lines that display the common factors like this:

```
Common factors of 10 and 20
1
2
5
```

4. Write a script with a TRY…CATCH statement that attempts to insert a new category named "Guitars" into the Categories table. If the insert is successful, the script should display this message:

```
SUCCESS: Record was inserted.
```

   If the insert is unsuccessful, the script should use the ERROR_NUMBER and ERROR_MESSAGE functions to display a message similar to this one:

```
FAILURE: Record was not inserted.

Error 2627: Violation of UNIQUE KEY constraint
'UQ__Categori__8517B2E0A87CE853'. Cannot insert duplicate key
in object 'dbo.Categories'. The duplicate key value is
(Guitars).
```

Chapter 15

How to code stored procedures, functions, and triggers

Exercises

1. Write a script that creates a stored procedure named spInsertCategory. The procedure should accept one parameter for a category name, which it uses to add a new row to the Categories table.

   In the same script, code two EXEC statements to test your newly created procedure. One should succeed and one should fail. (Hint: Note that the Categories table doesn't allow duplicate category names.)

2. Write a script that creates a function named fnDiscountPrice that calculates the discount price of an item in the OrderItems table (the discount amount subtracted from the item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.

   In the same script, code a SELECT statement that returns the ItemID, ItemPrice, and DiscountAmount columns, as well as the value returned by the fnDiscountPrice function for each item.

3. Write a script that creates a function named fnItemTotal that calculates the total amount of an item in the OrderItems table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, use the fnDiscountPrice function that you created in exercise 2, and return the value of the total for that item.

   In the same script, code a SELECT statement that returns the ItemID, ItemPrice, and DiscountAmount columns, the value returned by the fnDiscountPrice function for each item, the quantity of each item, and the value returned by the fnItemTotal function.

4. Write a script that creates a stored procedure named spInsertProduct that inserts a row into the Products table. This stored procedure should accept a parameter for each of these columns: CategoryID, ProductCode, ProductName, ListPrice, and DiscountPercent.

   This stored procedure should set the Description column to an empty string, and it should set the DateAdded column to the current date.

   If the value for the ListPrice column is a negative number, the stored procedure should raise an error that indicates that this column doesn't accept negative numbers. Similarly, the procedure should raise an error if the value for the DiscountPercent column is a negative number.

   Code two EXEC statements that test this procedure, one which succeeds and one which fails.

5. Write a script that creates a stored procedure named spUpdateProductDiscount that updates the DiscountPercent column in the Products table. This procedure should have input parameters for the product ID and the discount percent.

If the value for the DiscountPercent column is a negative number, the stored procedure should raise an error that indicates that the value for this column must be a positive number.

Code two EXEC statements that test this procedure, one which succeeds and one which fails.

6. Create a trigger named Products_UPDATE that checks the new value for the DiscountPercent column of the Products table. This trigger should raise an error with an appropriate message if the discount percent is greater than 100 or less than 0.

   If the new discount percent is between 0 and 1, this trigger should modify the new discount percent by multiplying it by 100. That way, a discount percent of .2 becomes 20.

   Test this trigger with an UPDATE statement that sets the discount percent to .25.

7. Create a trigger named Products_INSERT that inserts the current date for the DateAdded column of the Products table if the value for that column is null.

   Test this trigger with an INSERT statement that does not include a value for the DateAdded column.

8. Create a table named ProductsAudit. This table should have all columns of the Products table, except the Description column. Also, it should have an AuditID column for its primary key, and the DateAdded column should be changed to DateUpdated.

   Create a trigger named Products_UPDATE. This trigger should insert the old data about the product into the ProductsAudit table after the row is updated and set the DateUpdated column to the current date and time.

   Test this trigger with an appropriate UPDATE statement.

9. Re-run the script named CreateMyGuitarShop.sql to restore the database to its original state.

Chapter 16

How to manage transactions and locking

Exercises

1. Write a script that includes two SQL statements coded as a transaction to delete the row with a customer ID of 8 from the Customers table. To do this, you must first delete all addresses for that customer from the Addresses table.

   If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

2. Write a script that includes these statements coded as a transaction:

```
INSERT Orders
VALUES (3, GETDATE(), '10.00', '0.00', NULL, 4,
  'American Express', '378282246310005', '04/2026', 4);

SET @OrderID = @@IDENTITY;

INSERT OrderItems
VALUES (@OrderID, 6, '415.00', '161.85', 1);

INSERT OrderItems
VALUES (@OrderID, 1, '699.00', '209.70', 1);
```

   Here, the @@IDENTITY variable is used to get the order ID value that's automatically generated when the first INSERT statement inserts an order.

   If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

Chapter 17
How to manage database security

Exercises

1. Write a script that creates a user-defined database role named OrderEntry in the MyGuitarShop database. Give INSERT and UPDATE permission to the new role for the Orders and OrderItems tables. Give SELECT permission for all user tables.
2. Write a script that (1) creates a login ID named "RobertHalliday" with the password "HelloBob!"; (2) sets the default database for the login to the MyGuitarShop database; (3) creates a user named "RobertHalliday" for the new login; and (4) assigns the user to the OrderEntry role you created in exercise 1.
3. Write a script that uses dynamic SQL and a cursor to loop through each row of the Administrators table. For each administrator:

   Create a login ID that consists of the administrator's first and last name with no space between.

   Set a password of "TempPa$$1" for the new login ID.

   Set the default database for the new login ID to MyGuitarShop.

   Create a user for the login ID with the same name as the login ID.

   Assign the user to the OrderEntry role you created in exercise 1.

   (Hint: Begin the script by declaring two variables, @DynamicSQL and @LoginName.)
4. Using Management Studio, create a login ID named "RBrautigan" with the password "RBra99999", and set the default database to the MyGuitarShop database. Then, grant the login ID access to the MyGuitarShop database, create a user for the login ID named "RBrautigan", and assign the user to the OrderEntry role you created in exercise 1.

   Note: If you get an error that says "The MUST_CHANGE option is not supported", you can deselect the "Enforce password policy" option for the login ID.
5. Write a script that removes the user-defined database role named OrderEntry. (Hint: This script should begin by removing all users from this role.)
6. Write a script that does the following:

   Creates a schema named Admin.

   Transfers the table named Addresses from the dbo schema to the Admin schema.

   Assigns the Admin schema as the default schema for the user named RobertHalliday that you created in exercise 2.

   Grants all standard privileges except for REFERENCES and ALTER to RobertHalliday for the Admin schema.

Chapter 18
How to use Azure Data Studio

Exercises

In these exercises, you'll use Azure Data Studio to review the tables in the MyGuitarShop database and to enter SQL statements and run them against this database.

Review the database

1. Start Azure Data Studio.
2. In the Connections tab, expand the node for the database named MyGuitarShop so you can see all of the database objects it contains. If it isn't displayed in the Connections tab, you may need to click on the Refresh button to display it.
3. View the data for the Categories and Products tables.
4. Navigate through the database objects and view the column definitions for at least the Categories and Products tables.

Enter and run SQL statements

5. Open a new Query Editor window by right-clicking on the database and selecting New Query.
6. Enter and run this SQL statement:

```
SELECT ProductName FROM Products
```

7. Delete the e at the end of ProductName and run the statement again. Note the error number and the description of the error.
8. Open another Query Editor window and then enter and run this statement:

```
SELECT COUNT(*) AS NumberOfProducts
FROM Products
```

Open and run scripts

9. Open the script named ProductDetails.sql that's in the MGS Exercise Starts directory. Note that this script contains just one SQL statement. Then, run the statement.
10. Open the script named ProductSummary.sql that's in the MGS Exercise Starts directory. Note that this opens another Query Editor window.
11. Open the script named ProductStatements.sql that's in the MGS Exercise Starts directory. Notice that this script contains two SQL statements that end with semicolons.
12. Press the F5 key or click the Run button to run both of the statements in this script. Note that this displays the results in two Results panes. Make sure to view the results of both SELECT statements.
13. Exit from Azure Data Studio.