Manipulating Data

Lesson Objectives

By the end of this lesson, you will be able to:

- ☐ Understand the purpose and function of Data Manipulation Language (DML).
- ☐ Perform data entry tasks using the SSMS GUI.
- ☐ Create a database diagram.
- ☐ Insert data using the INSERT statement.
- ☐ Insert data using INSERT scripts.
- ☐ Select data using the SELECT statement.
- ☐ Update data using the UPDATE statement.
- ☐ Delete data using the DELETE statement.

Exam Objectives

1.3	Understand data manipulation language (DML)
3.1	Select data
3.2	Insert data
3.3	Update data
3.4	Delete data

Data Manipulation Language (DML)

Objective 1.3 3.2

As you learned earlier, SQL commands are generally grouped into four categories, each of which is considered a subset of SQL. ***Data Manipulation Language (DML)*** is the language that is used to insert data, and update and query a database. DML statements often perform mathematical and statistical calculations.

DML commands manipulate data and perform the CRUD operations (Create, Read, Update and Delete). These commands include:

INSERT	Used to insert new rows into a table. A variation of the INSERT statement allows you to insert the results of a query into a table.
SELECT	Used to select data from a database.
UPDATE	Used to change the values in a record.
DELETE	Used to delete records from a table.

All RDBMS programs use DML commands to manipulate data. In some programs you can enter the SQL code directly. In others, such as Microsoft Access, you can work with the data in a GUI and the SQL code is generated and executed behind the scenes.

We will briefly examine data manipulation techniques using the SQL Server Management Studio GUI, then the rest of the lesson will be devoted to writing and executing SQL statements in the Query window.

Data entry using the SSMS GUI

Microsoft SQL Server allows you to work with the data in your tables using a variety of methods. When you need to perform simple data entry to add or modify a relatively small number of records one at a time, it may be easiest to use SSMS graphic interface.

Performing data entry tasks using the SSMS GUI is similar to working with records in a spreadsheet. You simply click in a cell and start typing. You can move the pointer from cell to cell by pressing TAB or using the mouse. When you reach the end of a record, you can press TAB to move to the next new record or click in the row for the new record. Users who have performed data entry using the datasheet view in Microsoft Access will find the interface familiar.

Exercise 3-1: Data entry in the SSMS GUI

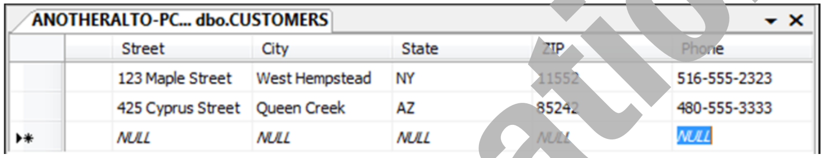
In this exercise, you will use the SSMS GUI to perform data entry tasks.

First, you will add records to tables.

1. Log on to SQL Server Management Studio.
2. In the Object Explorer, expand the **Databases** folder, expand the **HomeGrown** folder, then expand the **Tables** folder for the HomeGrown database.
3. In the Object Explorer, right-click **dbo.CUSTOMERS**, then select **Edit Top 200 Rows** to open the table for editing.
4. Click in the **CustomerID** field if necessary, type `1012`, press TAB, type `Helen`, press TAB, type `Johnson`, press TAB, type `123 Maple Street`, press TAB, type `West Hempstead`, press TAB, type NY, press TAB, type `11552`, press TAB, then type `516-555-2323`.

   Notice that a pencil icon displays in the record selector box at the left edge of the new record. The pencil icon indicates that a record contains changes that have not been saved.
5. Press TAB to save the changes to the first record and move the record pointer to the next row.
6. Type 1022, press TAB, type Andrew, press TAB, type Willis, press TAB, type 425 Cyprus Street, press TAB, type Queen Creek, press TAB, type AZ, press TAB, type 85242, press TAB, type 480-555-3333, then press TAB once more to save the second record and move the pointer to the next row.

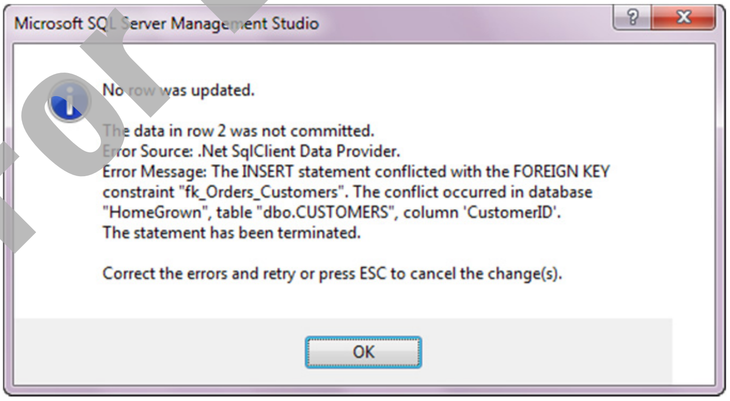
   Your table should appear as shown in Figure 3-1.


ANOTHERALTO-PC... dbo.CUSTOMERS

	Street	City	State	ZIP	Phone
	123 Maple Street	West Hempstead	NY	11552	516-555-2323
	425 Cyprus Street	Queen Creek	AZ	85242	480-555-3333
▸✱	NULL	NULL	NULL	NULL	NULL

Figure 3-1: Adding records to the CUSTOMERS table

7. In the Object Explorer, right-click **dbo.ORDERS**, then select **Edit Top 200 Rows** to open the table for editing in a new tab in the center pane.
8. Click in the OrderID field if necessary, press TAB to skip over the OrderID field, type `1012`, press TAB twice, type `09-19-2011`, then press TAB twice to enter a record for an order. Notice that a value is automatically entered into the OrderID field when you save the record. The OrderID field is set to auto-increment, and you cannot manually enter a value into the field.
9. Press TAB, type `1023`, press TAB twice, type `10-21-2011`, then press TAB twice to move the record point to the next row. Notice that SQL Server generates the error message shown in Figure 3-2 because of the foreign key constraint on the CustomerID field.


Microsoft SQL Server Management Studio

```
No row was updated.

The data in row 2 was not committed.
Error Source: .Net SqlClient Data Provider.
Error Message: The INSERT statement conflicted with the FOREIGN KEY
constraint "fk_Orders_Customers". The conflict occurred in database
"HomeGrown", table "dbo.CUSTOMERS", column 'CustomerID'.
The statement has been terminated.

Correct the errors and retry or press ESC to cancel the change(s).
```

Figure 3-2

   The value 1023 specified for the foreign key does not exist in the CUSTOMERS table, and therefore violates the foreign key constraint.
10. Click **OK** to close the error message, change the value in the CustomerID field to *1022*, then save the record.

Notice that the OrderID increments to 3 instead of 2. When you attempted to commit (save) the record the first time, SQL server entered the value *2* in the OrderID field. Because the row was not committed (due to the foreign key field constraint), and a new commit was attempted after you changed the data in the CustomerID field, the value in the OrderID field incremented again.

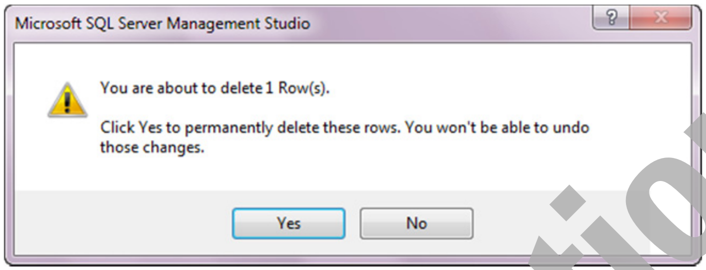
Next, you will modify (update) a record.

11. Click the tab for the **CUSTOMERS** table. Select the value in the LastName field for the first record and change the last name from *Johnson* to **Anderson** because Helen got married. Press SHIFT+ENTER to save the changes.

    Note that pressing SHIFT+ENTER accomplishes the same thing as tabbing through the fields of a record to move the record pointer to the next row.

Finally, you will delete records.

12. Click the tab for the **ORDERS** table, click the row selector for the second row to select the second row, right-click anywhere in the selected row and select **Delete**. SQL Server displays the message box shown in Figure 3-3 prompting you to confirm the deletion.


Microsoft SQL Server Management Studio

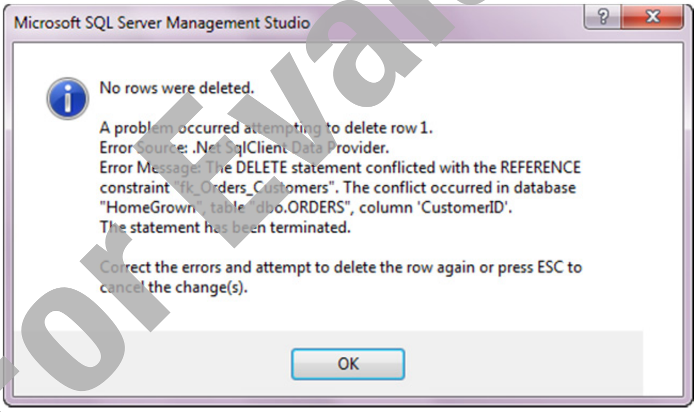
You are about to delete 1 Row(s).

Click Yes to permanently delete these rows. You won't be able to undo those changes.

Figure 3-3: Confirming a deletion

13. Click **Yes** to confirm and permanently delete the record. Notice that the one remaining record refers to Customer 1012.

14. Click the **CUSTOMERS** tab, click in the record selector for the first row, right-click in the selected row, select **Delete**, then click Yes to confirm the deletion. SQL Server displays the error message shown in Figure 3-4 because a record in the ORDERS table refers to Row 1 in the CUSTOMERS table.


Microsoft SQL Server Management Studio

No rows were deleted.

A problem occurred attempting to delete row 1.
Error Source: .Net SqlClient Data Provider.
Error Message: The DELETE statement conflicted with the REFERENCE constraint "fk_Orders_Customers". The conflict occurred in database "HomeGrown", table "dbo.ORDERS", column 'CustomerID'.
The statement has been terminated.

Correct the errors and attempt to delete the row again or press ESC to cancel the change(s).

Figure 3-4: Deletion error

MMM
Application Project 3-1: A Scripted Approach

15. Click **OK** to close the error message.

16. Click the **ORDERS** tab, delete the remaining record, then click the close button to close the tab for the ORDERS table.

17. In the **CUSTOMERS** table, delete the first record.

18. Delete the remaining record in the CUSTOMERS table, then close the tab for the CUSTOMERS table.

19. In the menu bar at the top of the SSMS window, select **File**, **Exit** to close the SQL Server window. If prompted, do not save any changes.

In this exercise, you performed basic data entry tasks in the SSMS GUI, and you observed the effects of foreign key constraints.

Re-creating the HomeGrown Database

Although you created several tables for the HomeGrown database in Lesson 2, you will drop and then re-create the entire database using SQL code provided in the Student Data folder. Re-creating the database and its objects will ensure that all exercise steps in this lesson will produce the expected results.

SQL code can be stored in text documents and pasted into the Query window for execution. Although code can be stored in a Word document, it is customary to save the code to a text document, with a .sql file name extension. Code stored in this manner is referred to as a SQL script.

As part of the re-creation of the database, you will also create a database diagram. A database diagram indicates how tables in a database are related to each other. Because these relationships are determined by the foreign key constraints, creating a database diagram can allow you to check and ensure that your tables are related in the way you intended.

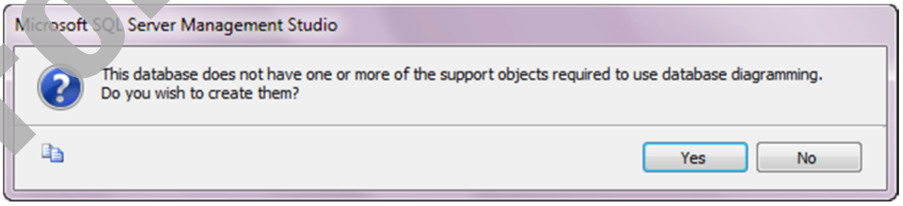
Exercise 3-2: Rebuilding the HomeGrown database

In this exercise, you will drop the HomeGrown database and then re-create it using a SQL script in the Student Data folder. You will also create a database diagram.

1. Log on to SQL Server Management Studio.
2. Expand the Databases folder, right-click the HomeGrown folder, then select Delete.
3. When the Delete Object window opens, click **OK** to confirm that you want to delete the HomeGrown database and all its objects.
4. In the menu bar at the top of the SSMS window, select **File**, **Open**, **File**, then in the Student Data folder on the desktop, navigate to the **Lesson_3** folder and double-click the **HomeGrown Creation Code.sql** file to open it in a new Query window. This simple text document contains the SQL code necessary to re-create the database.
5. Examine the code carefully, noting the foreign key constraints and the fields that cannot contain null values.

The settings used in the HomeGrown database are fairly permissive in that most fields are allowed to contain null values, and these settings have been chosen to speed data entry. In most real-world applications, the constraints would be tighter. Note however, that the ON DELETE and ON UPDATE clauses specify NO ACTION, which is the most restrictive setting. These settings have been chosen to illustrate certain points about how constraints affect data entry.

6. Click the **Execute** button in the SQL Editor toolbar to run the code and re-create the database and its tables.
7. Close the Query window without saving the changes.
8. In the Object Explorer, right-click the **Databases** folder, then select **Refresh** to show the newly created HomeGrown database in the folder list.
9. Expand the **HomeGrown** folder, then expand its **Tables** folder to view the objects and verify that all six tables were created.
10. In the Object Explorer, right-click the HomeGrown **Database Diagram** folder and select **New Database Diagram**. SQL Server displays the message shown in Figure 3-5.



This database does not have one or more of the support objects required to use database diagramming. Do you wish to create them?

Figure 3-5

11. Click **Yes** to specify that you want to create the support objects. SQL Server opens a database diagram window and the Add Table dialog box, shown in Figure 3-6.

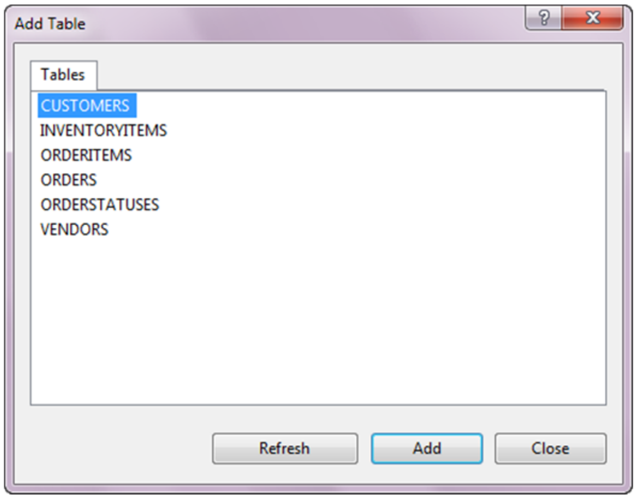


Figure 3-6: Add Table dialog box

12. In the Add Table dialog box, click the **Add** button six times (once for each table), then click the **Close** button.

Note: You can also click the first table in the Add Table dialog box, press and hold SHIFT, and then click the last table in the dialog box to select all six tables. Clicking the **Add** button would then add all selected tables to the diagram window at once.

13. Maximize the SQL Server window, then click the close button in the Object Explorer window and the Properties window to expand the workspace for the database diagram.

14. You can move a table in the diagram window by clicking the table once to select it, then dragging it into the desired position. Rearrange the tables in the diagram window until you can see how they are all related.

    Your database diagram should resemble the one shown in Figure 3-7.

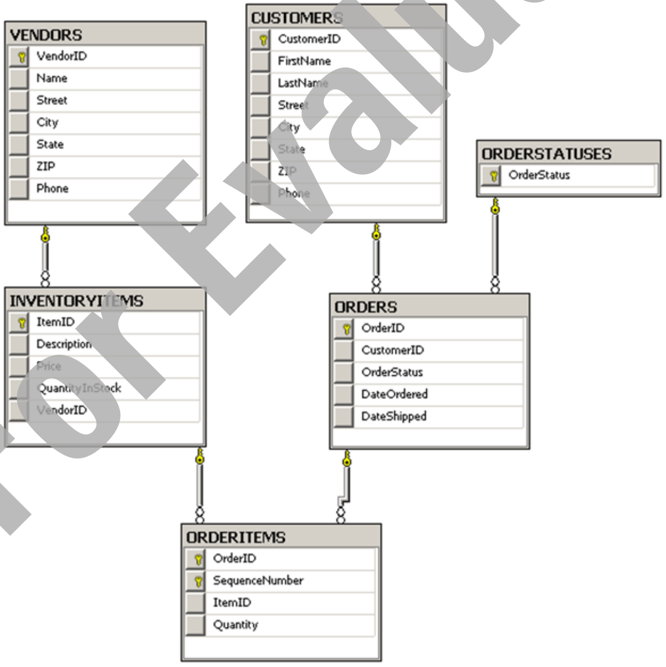


Figure 3-7: Database diagram for HomeGrown

Notice that database diagram shows the primary keys, and the one-to-many relationships in the database.

15. In the Standard toolbar, click the **Save** button and accept the default name, *Diagram_0*, by clicking **OK**.

16. In the menu bar, select **View**, **Object Explorer** to reopen the Object Explorer window.

17. In the menu bar, select **View**, **Properties Window** to reopen the Properties window.
18. Close the database diagram.
19. Minimize the SQL Server window.

In this exercise, you re-created the HomeGrown database and you created a database diagram.

MMM
Optional Exercise 3-1: Same Results Every Time

The INSERT Statement

Objective
3.2

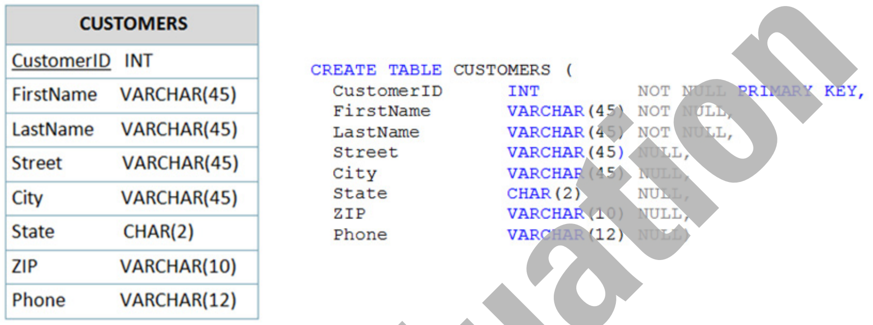
The SQL INSERT statement adds data to a table in a database, and the command has several variations.

The simplest form of the INSERT statement uses the following syntax:

INSERT INTO *table_name*
VALUES (*value1, value2, …*)

This form of the statement lists the values to be inserted into a new record after the VALUES keyword. The values must have the correct data types and must be listed in the same order as the fields in the table.

Consider the CUSTOMERS table and SQL creation code shown in Figure 3-8:



CUSTOMERS	
CustomerID	INT
FirstName	VARCHAR(45)
LastName	VARCHAR(45)
Street	VARCHAR(45)
City	VARCHAR(45)
State	CHAR(2)
ZIP	VARCHAR(10)
Phone	VARCHAR(12)

```
CREATE TABLE CUSTOMERS (
  CustomerID      INT          NOT NULL PRIMARY KEY,
  FirstName       VARCHAR(45)  NOT NULL,
  LastName        VARCHAR(45)  NOT NULL,
  Street          VARCHAR(45)  NULL,
  City            VARCHAR(45)  NULL,
  State           CHAR(2)      NULL,
  ZIP             VARCHAR(10)  NULL,
  Phone           VARCHAR(12)  NULL)
```

Figure 3-8: CUSTOMERS table layout and SQL creation code

You could use the following command to add a record to the CUSTOMERS table:

INSERT INTO CUSTOMERS
VALUES (1012, 'Helen', 'Anderson', '123 Maple Street', 'West Hempstead', 'NY', '11552', '516-555-2323')

In SQL, all non-numeric data values must be enclosed in single quotation marks, as shown in the sample command above. Numeric data types do not require the single quotation marks, and in fact will be treated as a CHAR data type if single quotation marks are used.

Notice that in the sample command, data is indicated for every field. You cannot skip fields when using this form of the INSERT statement, unless the field is set to auto-increment. You can, however, use the NULL keyword if the value of a particular column is unknown at the time of an insert operation. For example, you could use the following statement to enter data for a customer who does not have a phone number:

INSERT INTO CUSTOMERS
VALUES (1022, 'Andrew', 'Willis', '425 Cyprus Street', 'Queen Creek', 'AZ', '85242', NULL)

Note that the NULL keyword does not require single quotation marks.

If you defined a column using the NOT NULL constraint, however, you must enter a valid value for the column. In the CUSTOMERS table, for example, you could use the NULL keyword for the Street, City, State, ZIP and Phone fields only.

When you execute an INSERT statement in the Query window, the only visible result is a message at the bottom of the window indicating that the operation was successfully completed. To view the newly inserted record, you would need to execute a SELECT statement.

If an error occurs, an error message (or messages) displays at the bottom of the window, and the operation is aborted.

Specifying columns for an INSERT statement

MMM
Application Project 3-2: Insert Tab A into Slot B

The INSERT statement can also designate the columns into which values are to be inserted. Consider the following example.

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Phone)
VALUES (1080, 'Jill', 'Jackson', '714-555-6767')

The columns that were not named (Street, City, State and ZIP) will be given their default values for this record. The default value is any designated default for the columns or a NULL. If the column has no designated default value and does not accept nulls, you must supply a value for that column or the statement will generate an error.

Many SQL programmers prefer to always include column names in INSERT statements because it makes the statements clearer and easier to maintain.

Activity 3-1: Reading INSERT statements

In this activity, you will examine the structure of the CUSTOMERS table shown in Figure 3-8, and act as if you are the computer by executing the following four numbered INSERT statements on paper only. Write the results of each insert statement into the grid, or write the word "ERROR" into the grid if a particular statement will generate an error. The first one has been completed for you:

1. INSERT INTO CUSTOMERS
   VALUES (1030, 'Alice', 'Goldman', NULL, 'Mesa', 'AZ', NULL, NULL)
2. INSERT INTO CUSTOMERS
   VALUES (1034, 'Robert', 'Meyers', '29 River Street', 'Mesa', 'AZ', '85202', '480-555-7777')
3. INSERT INTO CUSTOMERS
   VALUES (1035, 'Anna', 'Billings', 'Mesa', 'AZ', '85203', '480-555-8888')
4. INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, City, ZIP)
   VALUES (1044, 'Melissa', 'Sharon', NULL, 'Flagstaff', '86002')

	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	*1030*	*Alice*	*Goldman*	*NULL*	*Mesa*	*AZ*	*NULL*	*NULL*
2								
3								
4								

When you are finished, compare your results with the rest of the class.

INSERT statements and auto-increment fields

Often tables include ID fields that are set to auto-increment. Just as you cannot type a value into an auto-increment field in the SSMS GUI, you cannot insert a value into an auto-increment field with an INSERT statement. Skip the auto-increment field when listing values or when naming the columns in the INSERT statement.

For example, consider the VENDORS table and SQL creation code shown in Figure 3-9:

VENDORS	
VendorID	INT
Name	VARCHAR(45)
Street	VARCHAR(45)
City	VARCHAR(45)
State	CHAR(2)
ZIP	VARCHAR(10)
Phone	VARCHAR(12)

```
CREATE TABLE VENDORS (
  VendorID  INT          NOT NULL PRIMARY KEY IDENTITY,
  Name      VARCHAR(45)  NOT NULL,
  Street    VARCHAR(45)  NULL,
  City      VARCHAR(45)  NULL,
  State     CHAR(2)      NULL,
  ZIP       VARCHAR(10)  NULL,
  Phone     VARCHAR(12)  NULL)
```

Figure 3-9: VENDORS table layout and SQL creation code

You can add a record to the VENDORS table with either of the following statements:

INSERT INTO VENDORS VALUES ('Good Earth Supplies', '21 Green Way', 'Ajo', 'AZ', '85321', '520-555-9997')

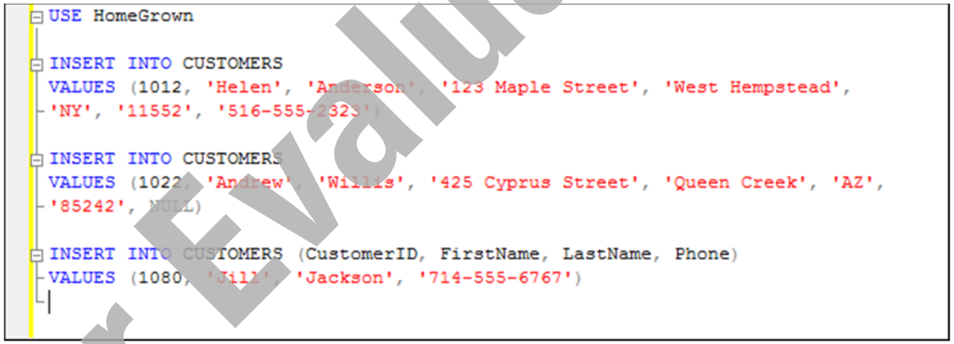
INSERT INTO VENDORS (Name, Street, City, State, ZIP, Phone)
VALUES ('GeoGiant', '1441 Scirocco ', 'Woodstock', 'NY', '12498', '845-555-7997')

Exercise 3-3: Adding records using INSERT statements

In this exercise, you will write and execute INSERT statements to add data to the HomeGrown database.

You enter and execute SQL statements in the Query window.

1. Restore the SQL Server window.
2. In the Standard toolbar, click the **New Query** button to open a Query window.
3. Type `USE HomeGrown`.
4. Press ENTER two or three times to add some white space.
5. Type
```
INSERT INTO CUSTOMERS
VALUES (1012, 'Helen', 'Anderson', '123 Maple Street', 'West Hempstead', 'NY',
'11552', '516-555-2323')
```
6. Press ENTER twice.
7. Type
```
INSERT INTO CUSTOMERS
VALUES (1022, 'Andrew', 'Willis', '425 Cyprus Street', 'Queen Creek', 'AZ',
'85242', NULL)
```
8. Press ENTER twice.
9. Type
```
INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Phone)
VALUES (1080, 'Jill', 'Jackson', '714-555-6767')
```
   Your screen should resemble the one shown in Figure 3-10.



```
USE HomeGrown

INSERT INTO CUSTOMERS
VALUES (1012, 'Helen', 'Anderson', '123 Maple Street', 'West Hempstead',
'NY', '11552', '516-555-2323')

INSERT INTO CUSTOMERS
VALUES (1022, 'Andrew', 'Willis', '425 Cyprus Street', 'Queen Creek', 'AZ',
'85242', NULL)

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Phone)
VALUES (1080, 'Jill', 'Jackson', '714-555-6767')
```

Figure 3-10: Writing INSERT Statements

10. Ensure that the CustomerID values you entered match the ones shown in Figure 3-10, as these will be important throughout the remainder of this lesson, then click the **Execute** button to insert the three new records.

    Because you executed three statements, the SQL Server message window lists three responses, each of which reads "(1 row(s) affected)."
11. In the Object Explorer, expand the necessary folders to view the tables in the HomeGrown database.
12. In the Object Explorer, right-click the **CUSTOMERS** table, then select **Edit Top 200 Rows** to view the newly inserted records.
13. Close the tab for the CUSTOMERS table.
14. Close the Query window without saving any changes.
15. Minimize the SQL Server window.

Using INSERT Scripts

As you may have already guessed, an INSERT script is simply a text document that contains a series of INSERT statements that you can copy and paste into the Query window for execution. Writing scripts allows you to carefully review the data before adding it to your tables.

When you execute INSERT scripts, you must keep foreign key constraints in mind so you add data to your tables in the correct order. For example, in the HomeGrown database, you could not enter data into the ORDERS table until you entered data into the CUSTOMERS and ORDERSTATUSES tables because the ORDERS table includes two foreign key constraints. One that refers to the CustomerID field in the CUSTOMERS table and one that refers to the OrderStatus field in the ORDERSTATUSES table.

To add data to the HomeGrown database, you can execute the listed INSERT scripts (which are included in the Student Data folder) in the following order:

1. CUSTOMERS-Inserts.sql
2. VENDORS-Inserts.sql
3. ORDERSTATUSES-Inserts.sql
4. INVENTORYITEMS-Inserts.sql
5. ORDERS-Inserts.sql
6. ORDERITEMS-Inserts.sql

MMM
Optional Exercise 3-2: Look Ma, No Hands

The order of the first three scripts does not matter, because these tables do not include foreign keys. However, all three of these scripts must be executed before you execute the last three.

Of the remaining three scripts, you must execute ORDERITEMS-Insert.sql last.

You can copy, paste and execute each script individually, or you can paste the contents of all the scripts into the Query window and execute the statements all at once. You can also paste all of the scripts into the Query window, but execute each statement one at a time by highlighting a statement; when you click the Execute button SQL Server will only run the highlighted statement.

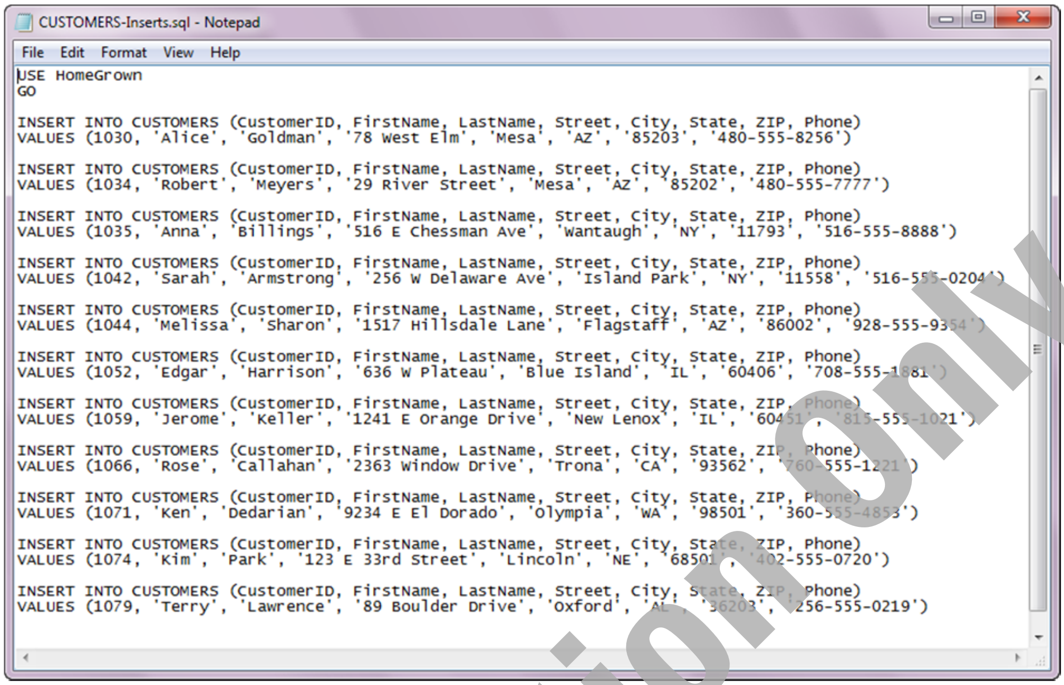
Exercise 3-4: Adding records using INSERT scripts

In this exercise, you will add data to the HomeGrown database using INSERT scripts.

Note: For the sake of expediency, a script named Table-Inserts.sql can be used to populate all the tables in the database at once. If you opt to use this method for adding records, use Table-Inserts.sql instead of CUSTOMERS-Inserts.sql in Steps 1- 7 of this exercise, and do not perform Step 8.

First, you will examine an INSERT script.

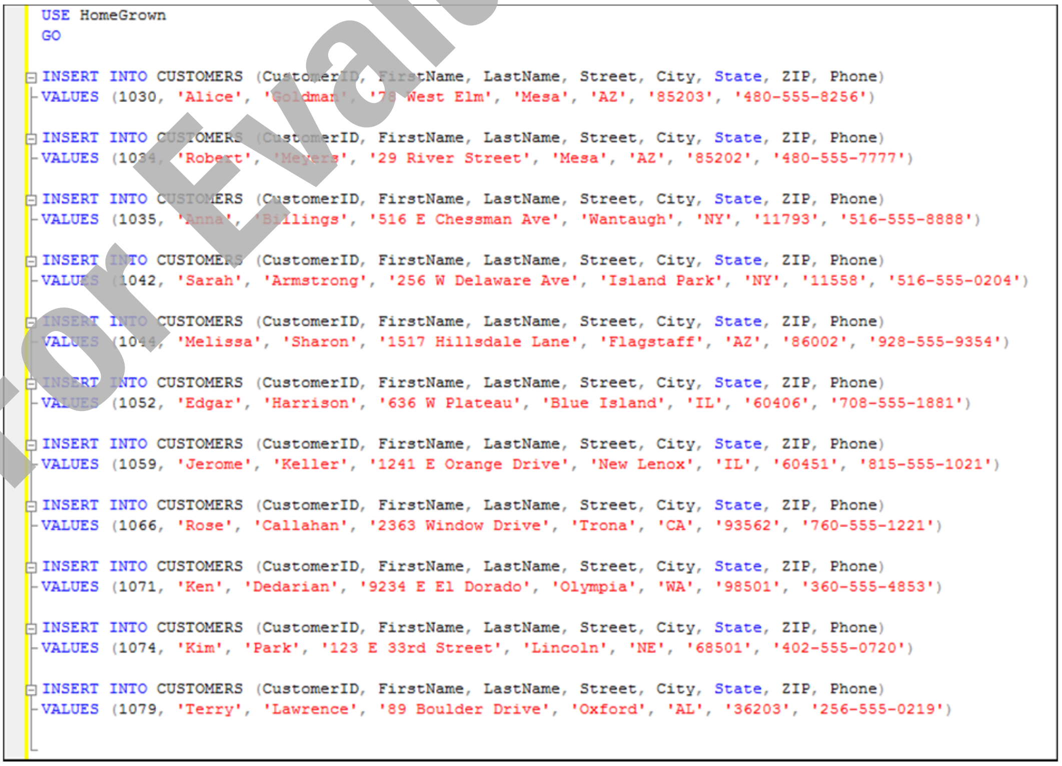
1. Navigate to the *Lesson_3* folder in the Student Data folder.
2. Right-click **CUSTOMERS-Inserts.sql**, select **Open With**, then double-click **Notepad** in the Open With dialog box. This step opens the file in the Notepad window. Although you can open a .sql file directly in SSMS (as you did in Exercise 3-2), files of this type can also be opened (and created) in a simple text editor. The file contains a series of INSERT statements, as shown in Figure 3-11.



```
USE HomeGrown
GO

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1030, 'Alice', 'Goldman', '78 West Elm', 'Mesa', 'AZ', '85203', '480-555-8256')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1034, 'Robert', 'Meyers', '29 River Street', 'Mesa', 'AZ', '85202', '480-555-7777')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1035, 'Anna', 'Billings', '516 E Chessman Ave', 'Wantaugh', 'NY', '11793', '516-555-8888')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1042, 'Sarah', 'Armstrong', '256 W Delaware Ave', 'Island Park', 'NY', '11558', '516-555-0204')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1044, 'Melissa', 'Sharon', '1517 Hillsdale Lane', 'Flagstaff', 'AZ', '86002', '928-555-9354')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1052, 'Edgar', 'Harrison', '636 W Plateau', 'Blue Island', 'IL', '60406', '708-555-1881')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1059, 'Jerome', 'Keller', '1241 E Orange Drive', 'New Lenox', 'IL', '60451', '815-555-1021')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1066, 'Rose', 'Callahan', '2363 Window Drive', 'Trona', 'CA', '93562', '760-555-1221')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1071, 'Ken', 'Dedarian', '9234 E El Dorado', 'Olympia', 'WA', '98501', '360-555-4853')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1074, 'Kim', 'Park', '123 E 33rd Street', 'Lincoln', 'NE', '68501', '402-555-0720')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1079, 'Terry', 'Lawrence', '89 Boulder Drive', 'Oxford', 'AL', '36203', '256-555-0219')
```

Figure 3-11: CUSTOMERS-Inserts.sql file

3. Close the INSERT script.
4. Restore the SQL Server window.
5. In the menu bar at the top of the SSMS window, select **File**, **Open**, **File**, then in the Student Data folder on the desktop, navigate to the **Lesson_3** folder and double-click the **CUSTOMERS-Inserts.sql** file to open it in a new Query window. Your Query window should appear as shown in Figure 3-12.



```
USE HomeGrown
GO

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1030, 'Alice', 'Goldman', '78 West Elm', 'Mesa', 'AZ', '85203', '480-555-8256')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1034, 'Robert', 'Meyers', '29 River Street', 'Mesa', 'AZ', '85202', '480-555-7777')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1035, 'Anna', 'Billings', '516 E Chessman Ave', 'Wantaugh', 'NY', '11793', '516-555-8888')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1042, 'Sarah', 'Armstrong', '256 W Delaware Ave', 'Island Park', 'NY', '11558', '516-555-0204')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1044, 'Melissa', 'Sharon', '1517 Hillsdale Lane', 'Flagstaff', 'AZ', '86002', '928-555-9354')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1052, 'Edgar', 'Harrison', '636 W Plateau', 'Blue Island', 'IL', '60406', '708-555-1881')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1059, 'Jerome', 'Keller', '1241 E Orange Drive', 'New Lenox', 'IL', '60451', '815-555-1021')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1066, 'Rose', 'Callahan', '2363 Window Drive', 'Trona', 'CA', '93562', '760-555-1221')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1071, 'Ken', 'Dedarian', '9234 E El Dorado', 'Olympia', 'WA', '98501', '360-555-4853')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1074, 'Kim', 'Park', '123 E 33rd Street', 'Lincoln', 'NE', '68501', '402-555-0720')

INSERT INTO CUSTOMERS (CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone)
VALUES (1079, 'Terry', 'Lawrence', '89 Boulder Drive', 'Oxford', 'AL', '36203', '256-555-0219')
```

Figure 3-12: INSERT script for CUSTOMERS table in Query window

6. Click the **Execute** button.
7. Close the Query window without saving the changes.
8. Repeat Steps 5 through 7 for the remaining scripts in the following order**:** **VENDORS-Inserts.sql**, **ORDERSTATUSES-Inserts.sql**, **INVENTORYITEMS-Inserts.sql**, **ORDERS-Inserts.sql**, **ORDERITEMS-Inserts.sql**.
9. In the Object Explorer, right-click the **CUSTOMERS** table, then select **Edit Top 200 Rows** to view the newly inserted records. The table should include 14 records.
10. Close the tab for the CUSTOMERS table.
11. Check the record count for the remaining tables. They should be as follows:
    INVENTORYITEMS - 41 records
    ORDERITEMS - 51 records
    ORDERS - 15 records
    ORDERSTATUSES - 4 records
    VENDORS - 6 records
12. Minimize the SQL Server window.

In this exercise, you used SQL INSERT scripts to add data to tables in a database.

Case Scenario 3-1: Yes, We Have That!

Case Scenarios

Now that you have created a database and tables for the Illuminated Manuscript book shop, write and execute INSERT statements to add the indicated records to the following tables:

Table	Records	Table	Records
CATEGORIES	Romance	AUTHORS	Kenneth Kozakis
	Adventure		Stephanie Brendenhall
	Horror		Ron Bartlow
	Humor		Julian Harrolds
	Science Fiction		Linda Centini
	Self-Help		Lyndel Georges
			Andrew Silver
			Patrick Kelley

You may execute your statements individually in the Query window, or save them to a text file and then copy, paste and execute them in the Query window.

After you have added your records, edit the two tables to view the newly added records.

When you are done inserting the listed data, create data for the remaining tables in the Illuminated database. Study the relational model closely to determine in which order you should add records. The foreign key constraints should prevent you from entering mismatched data.

After you have tried creating and entering data, delete your version of the Illuminated database. Recreate the Illuminated database, then re-create the necessary tables for the database by executing the statements in the **Lesson_3\Illuminated Creation Code.sql** file. Insert data into the tables by executing the statements in the **Lesson_3\Illuminated-Inserts.sql** file.

Selecting Data

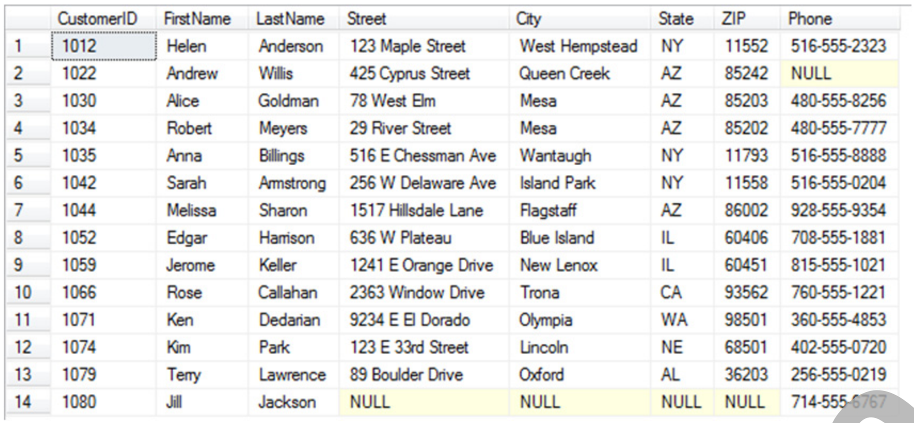
Objective
3.1

The SQL SELECT statement is a DML statement used to choose (select) or retrieve data from a database. In formal terms, selecting data is called executing a query. The records returned as a result of a query are called a result set, or a record set.

In its simplest format, the SELECT statement takes the following syntax:

SELECT * FROM *table_name*

In this form, the statement returns all the records from the named table. The asterisk (*) represents all the fields in the table. For example, if you executed the statement SELECT * FROM CUSTOMERS in the HomeGrown database, SQL Server would return the results shown in Figure 3-13.



	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	1012	Helen	Anderson	123 Maple Street	West Hempstead	NY	11552	516-555-2323
2	1022	Andrew	Willis	425 Cyprus Street	Queen Creek	AZ	85242	NULL
3	1030	Alice	Goldman	78 West Elm	Mesa	AZ	85203	480-555-8256
4	1034	Robert	Meyers	29 River Street	Mesa	AZ	85202	480-555-7777
5	1035	Anna	Billings	516 E Chessman Ave	Wantaugh	NY	11793	516-555-8888
6	1042	Sarah	Armstrong	256 W Delaware Ave	Island Park	NY	11558	516-555-0204
7	1044	Melissa	Sharon	1517 Hillsdale Lane	Flagstaff	AZ	86002	928-555-9354
8	1052	Edgar	Harrison	636 W Plateau	Blue Island	IL	60406	708-555-1881
9	1059	Jerome	Keller	1241 E Orange Drive	New Lenox	IL	60451	815-555-1021
10	1066	Rose	Callahan	2363 Window Drive	Trona	CA	93562	760-555-1221
11	1071	Ken	Dedarian	9234 E El Dorado	Olympia	WA	98501	360-555-4853
12	1074	Kim	Park	123 E 33rd Street	Lincoln	NE	68501	402-555-0720
13	1079	Terry	Lawrence	89 Boulder Drive	Oxford	AL	36203	256-555-0219
14	1080	Jill	Jackson	NULL	NULL	NULL	NULL	714-555-6767

*Figure 3-13: Results of SELECT * FROM CUSTOMERS statement*

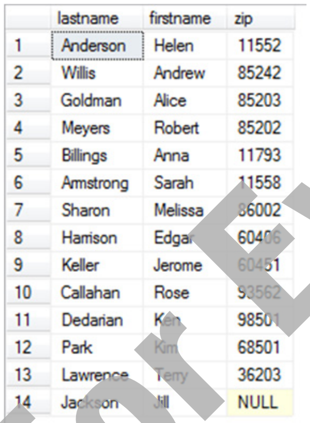
Note that queries may not always return the records in the same order. SQL provides mechanisms for ordering data. These mechanisms will be explored later in this lesson.

You can also list the fields you want to retrieve. For example, the following statement would return the same result as the first SELECT example:

SELECT CustomerID, FirstName, LastName, Street, City, State, ZIP, Phone
FROM CUSTOMERS

Every query must include the SELECT keyword indicating the columns that are to be retrieved (or the asterisk to indicate that all columns are to be retrieved) and a FROM keyword indicating the tables that are involved in the query. You can, of course, specify only specific columns. For example the following command will return the results shown in Figure 3-14.

SELECT LastName, FirstName, ZIP
FROM CUSTOMERS



	lastname	firstname	zip
1	Anderson	Helen	11552
2	Willis	Andrew	85242
3	Goldman	Alice	85203
4	Meyers	Robert	85202
5	Billings	Anna	11793
6	Armstrong	Sarah	11558
7	Sharon	Melissa	86002
8	Harrison	Edgar	60406
9	Keller	Jerome	60451
10	Callahan	Rose	93562
11	Dedarian	Ken	98501
12	Park	Kim	68501
13	Lawrence	Terry	36203
14	Jackson	Jill	NULL

Figure 3-14: Selecting specific columns

SELECT DISTINCT

The DISTINCT keyword is used to ensure that a query returns only unique (that is, distinct) values.

Suppose you needed a list of the states in which HomeGrown Garden Supplies has customers. The number of customers is not needed, just the states where the company has customers. The query could be written as follows.

SELECT State FROM CUSTOMERS

The query would return the following output:

```
NY
AZ
AZ
AZ
NY
NY
AZ
IL
IL
CA
WA
NE
AL
```

Note that the values NY, AZ and IL are repeated because several customers live in these states. To retrieve non-duplicate data, you can use the DISTINCT keyword, as in the following example.

```
SELECT DISTINCT State FROM CUSTOMERS
```

The query would return the following output:

```
AL
AZ
CA
IL
NE
NY
WA
```

Notice that the records are not necessarily returned in the order in which they are entered in the table.

SELECT WHERE

A WHERE clause is frequently added to a SELECT statement in order to apply a filter to the returned record set. In formal terms, a WHERE clause is used to add a predicate to a SQL statement When you use a WHERE clause in a SELECT statement, each row in the table is evaluated against the condition(s) in the predicate. Only the rows that meet the condition(s) in the predicate are included in the result set.

The following example illustrates a WHERE clause.

```
SELECT CustomerID, LastName, FirstName
FROM CUSTOMERS
WHERE CustomerID=1012
```

The query would return the following output in the HomeGrown database:

```
1012        Anderson        Helen
```

With the data currently in this specific CUSTOMERS table, the following SELECT statement would return the same results:

```
SELECT CustomerID, LastName, FirstName
FROM CUSTOMERS
WHERE LastName='Anderson'
```

Equality operators

Equality operators are used to compare two values. You have already seen the equal sign (=) operator used in a SQL predicate. SQL supports the following equality operators:

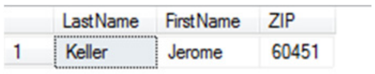
=	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

In some versions of SQL, the not equal to operator is written as (!=) instead of (<>).

As you have seen, you must enclose non-numeric values in single quotation marks when comparing them. Numeric values are not enclosed in single quotation marks.

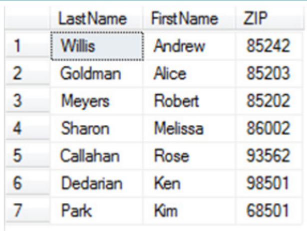
Review the following SELECT statements (executed against the CUSTOMERS table in the HomeGrown database) and their accompanying result sets to see how the operators are used:

```
SELECT LastName, FirstName, ZIP
FROM CUSTOMERS
WHERE ZIP = '60451'
```



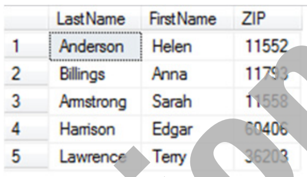
	LastName	FirstName	ZIP
1	Keller	Jerome	60451

```
SELECT LastName, FirstName, ZIP
FROM CUSTOMERS
WHERE ZIP > '60451'
```



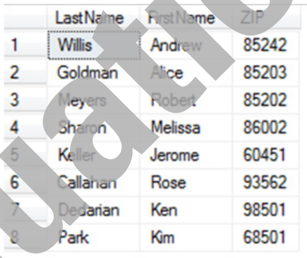
	LastName	FirstName	ZIP
1	Willis	Andrew	85242
2	Goldman	Alice	85203
3	Meyers	Robert	85202
4	Sharon	Melissa	86002
5	Callahan	Rose	93562
6	Dedarian	Ken	98501
7	Park	Kim	68501

```
SELECT LastName, FirstName, ZIP
FROM CUSTOMERS
WHERE ZIP < '60451'
```



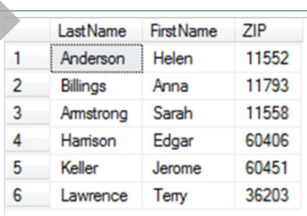
	LastName	FirstName	ZIP
1	Anderson	Helen	11552
2	Billings	Anna	11793
3	Armstrong	Sarah	11558
4	Harrison	Edgar	60406
5	Lawrence	Terry	36203

```
SELECT LastName, FirstName, ZIP
FROM CUSTOMERS
WHERE ZIP >= '60451'
```



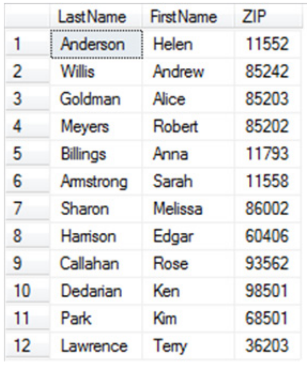
	LastName	FirstName	ZIP
1	Willis	Andrew	85242
2	Goldman	Alice	85203
3	Meyers	Robert	85202
4	Sharon	Melissa	86002
5	Keller	Jerome	60451
6	Callahan	Rose	93562
7	Dedarian	Ken	98501
8	Park	Kim	68501

```
SELECT LastName, FirstName, ZIP
FROM CUSTOMERS
WHERE ZIP <= '60451'
```



	LastName	FirstName	ZIP
1	Anderson	Helen	11552
2	Billings	Anna	11793
3	Armstrong	Sarah	11558
4	Harrison	Edgar	60406
5	Keller	Jerome	60451
6	Lawrence	Terry	36203

```
SELECT LastName, FirstName, ZIP
FROM CUSTOMERS
WHERE ZIP <> '60451'
```



	LastName	FirstName	ZIP
1	Anderson	Helen	11552
2	Willis	Andrew	85242
3	Goldman	Alice	85203
4	Meyers	Robert	85202
5	Billings	Anna	11793
6	Armstrong	Sarah	11558
7	Sharon	Melissa	86002
8	Harrison	Edgar	60406
9	Callahan	Rose	93562
10	Dedarian	Ken	98501
11	Park	Kim	68501
12	Lawrence	Terry	36203

If any of these operators is used to compare a particular value to a NULL value, an UNKNOWN is returned and the record is not included in the result set. In the previous examples, notice that record #14 for Jill Jackson is not returned in any of the result sets because her ZIP code is currently set to NULL. Note also that a NULL value is not treated as if it is the same as the value of 0 (zero).

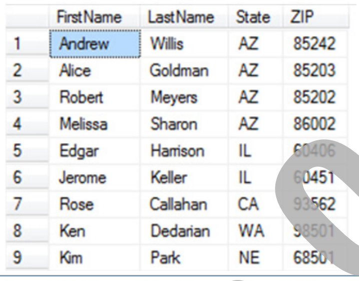
The AND operator

SQL supports the logical operators AND and OR.

The AND operator links together two or more conditional statements, and provides for increased filtering. When you use the AND operator, both expressions on either side of the AND must be evaluated to true for any given record to appear in the result set.

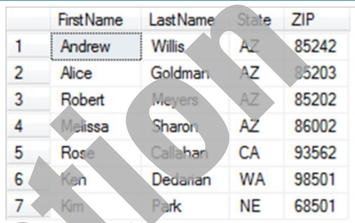
Consider the CUSTOMERS table again and the series of SQL statements and result sets. In the first example, nine records satisfy the condition of the expression in the WHERE clause: Each record must include a ZIP code greater than 6000.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE ZIP > '60000'
```



	FirstName	LastName	State	ZIP
1	Andrew	Willis	AZ	85242
2	Alice	Goldman	AZ	85203
3	Robert	Meyers	AZ	85202
4	Melissa	Sharon	AZ	86002
5	Edgar	Harrison	IL	60406
6	Jerome	Keller	IL	60451
7	Rose	Callahan	CA	93562
8	Ken	Dedarian	WA	98501
9	Kim	Park	NE	68501

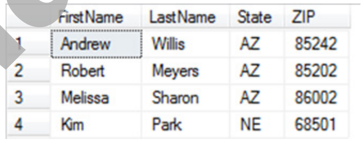
```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE ZIP > '60000'
AND State  <> 'IL'
```



	FirstName	LastName	State	ZIP
1	Andrew	Willis	AZ	85242
2	Alice	Goldman	AZ	85203
3	Robert	Meyers	AZ	85202
4	Melissa	Sharon	AZ	86002
5	Rose	Callahan	CA	93562
6	Ken	Dedarian	WA	98501
7	Kim	Park	NE	68501

Notice that in this result set, the records must meet the criteria specified in both expressions in the WHERE clause. Each record must include a ZIP code greater than 60000 and the State cannot be equal to IL.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE ZIP > '60000'
AND State  <> 'IL'
AND LastName >= 'H'
```



	FirstName	LastName	State	ZIP
1	Andrew	Willis	AZ	85242
2	Robert	Meyers	AZ	85202
3	Melissa	Sharon	AZ	86002
4	Kim	Park	NE	68501

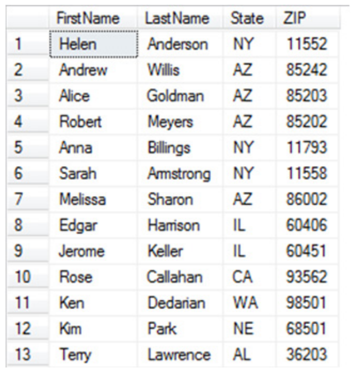
In this result set, the records must meet all three conditions specified by the expressions in the WHERE clause. Each record must include a ZIP code greater than 60000, the State cannot be equal to IL, and the LastName must begin with the letter “H” or higher. Note that you can fine tune a result set by adding expressions to the WHERE clause.

The OR operator

The OR operator is also used to help filter result sets. Instead of linking together conditional statements, the OR operator causes SQL to look for two separate conditions within the same query and return any rows meeting either of the conditions.

Consider the following statement and result set:

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE ZIP > '60000'
OR State  <> 'IL'
```



	FirstName	LastName	State	ZIP
1	Helen	Anderson	NY	11552
2	Andrew	Willis	AZ	85242
3	Alice	Goldman	AZ	85203
4	Robert	Meyers	AZ	85202
5	Anna	Billings	NY	11793
6	Sarah	Armstrong	NY	11558
7	Melissa	Sharon	AZ	86002
8	Edgar	Harrison	IL	60406
9	Jerome	Keller	IL	60451
10	Rose	Callahan	CA	93562
11	Ken	Dedarian	WA	98501
12	Kim	Park	NE	68501
13	Terry	Lawrence	AL	36203

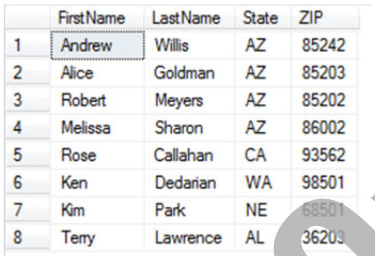
Notice that the records in this result set meet either the first condition (ZIP > '60000') or the second condition (State <>'IL').

The AND/OR combination

Conditional statements can be grouped together using parentheses (). Just as in other programming languages and mathematics, the parentheses are meant to group expressions that should be resolved as a group, and the result is treated as a single expression. By using parentheses, you can link conditions together to create robust queries.

For example, consider the following SELECT statement that employs an AND/OR combination.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE (ZIP > '60000' AND State <> 'IL')
OR State = 'AL'
```

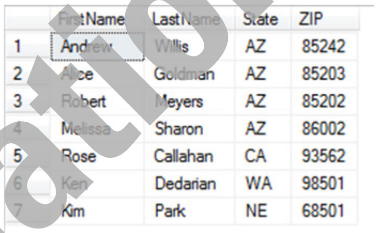


	FirstName	LastName	State	ZIP
1	Andrew	Willis	AZ	85242
2	Alice	Goldman	AZ	85203
3	Robert	Meyers	AZ	85202
4	Melissa	Sharon	AZ	86002
5	Rose	Callahan	CA	93562
6	Ken	Dedarian	WA	98501
7	Kim	Park	NE	68501
8	Terry	Lawrence	AL	36203

Notice that the result set includes records that meet both of the conditions in the first clause, and the one record that meets the condition in the second clause. In this example, `ZIP > '60000' AND State <> 'IL'` is evaluated first. SQL looks at each record and determines whether ZIP > '60000' and whether State <> 'IL'. If both expressions are true, the entire statement within the parentheses returns a true.

Grouped differently, the same expressions can retrieve different results, as shown in the following example.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE ZIP > '60000'
AND (State <> 'IL' OR State = 'AL')
```



	FirstName	LastName	State	ZIP
1	Andrew	Willis	AZ	85242
2	Alice	Goldman	AZ	85203
3	Robert	Meyers	AZ	85202
4	Melissa	Sharon	AZ	86002
5	Rose	Callahan	CA	93562
6	Ken	Dedarian	WA	98501
7	Kim	Park	NE	68501

Truth tables

A truth table is a table that illustrates the relationships of the logical operators AND and OR. A truth table is made up of two columns with 1s and 0s or Ts and Fs for True and False. As you learned in the previous section:

- In a logical AND, all items must be true to get a "True" result.
- In a logical OR, only one of the items must be true to get a “True" result.
- The order of the items has not influence on the result of a truth table.

The truth tables for logical AND and OR are shown in Figure 3-15.

Logical AND

ITEM1	ITEM2	RESULT
T	T	T
T	F	F
F	T	F
F	F	F

Logical OR

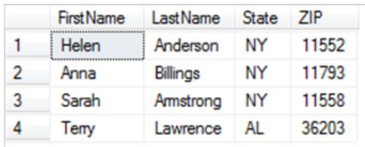
ITEM1	ITEM2	RESULT
T	T	T
T	F	T
F	T	T
F	F	F

Figure 3-15: Truth tables for Logical AND and Logical OR

The NOT keyword

The NOT keyword is used to reverse an expression's value. For example, consider the following statement and its result set.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE NOT ZIP > '60000'
```



	FirstName	LastName	State	ZIP
1	Helen	Anderson	NY	11552
2	Anna	Billings	NY	11793
3	Sarah	Amstrong	NY	11558
4	Terry	Lawrence	AL	36203

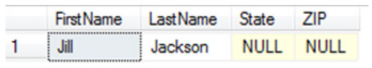
The NOT keyword is placed before the column's name, not after it. In English, it would be natural to say "ZIP code is not greater than 60000." However, using this syntax would generate an error.

The IS NULL statement

You can use the ***IS NULL*** statement to test for null values. The statement will return either true (a ***NULL*** value is present) or false.

Consider the following statement and its result set.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE State IS NULL
```



	FirstName	LastName	State	ZIP
1	Jill	Jackson	NULL	NULL

Conversely, using the NOT keyword with the IS NULL statement can return all rows that do not contain a NULL value for the specified field. The syntax for such a statement would be:

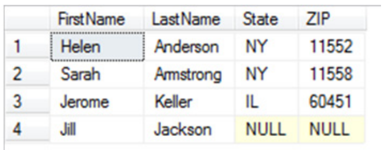
```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE State IS NOT NULL
```

The IN operator

The SQL IN operator is used to select data that matches a list of values. For example, if you wanted to retrieve customer data for two or more customers, you could use the IN operator to specify a list of customer names, and the query would retrieve rows reflecting every customer name included in the list.

Consider the following statement and its result set.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE LastName IN ('Anderson','Armstrong','Keller',
'Jackson')
```

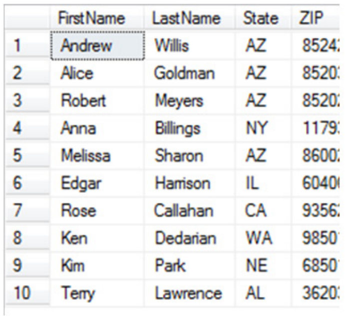


	FirstName	LastName	State	ZIP
1	Helen	Anderson	NY	11552
2	Sarah	Armstrong	NY	11558
3	Jerome	Keller	IL	60451
4	Jill	Jackson	NULL	NULL

Note: The WHERE clause in this statement (WHERE LastName IN ('Anderson', 'Armstrong', 'Keller', 'Jackson') is equivalent to (WHERE LastName = 'Anderson' OR LastName = 'Armstrong' OR LastName = 'Keller' OR LastName = 'Jackson'). The IN operator is preferred because it is more efficient, and for large databases this make a big difference in performance.

Conversely, the NOT IN operator can be used to eliminate records from a result set, as shown by the following statement and its result set.

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE LastName NOT IN ('Anderson','Armstrong','Keller',
'Jackson')
```

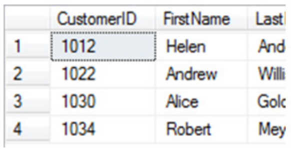


	FirstName	LastName	State	ZIP
1	Andrew	Willis	AZ	8524
2	Alice	Goldman	AZ	8520
3	Robert	Meyers	AZ	8520
4	Anna	Billings	NY	1179
5	Melissa	Sharon	AZ	8600
6	Edgar	Harrison	IL	6040
7	Rose	Callahan	CA	9356
8	Ken	Dedarian	WA	9850
9	Kim	Park	NE	6850
10	Terry	Lawrence	AL	3620

The BETWEEN operator

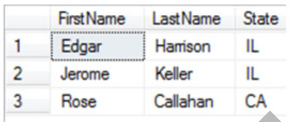
You can use the BETWEEN operator to query a table for rows that meet a condition falling between a specified range of values. The values can be numbers, text or dates. The AND keyword is used with the BETWEEN operator to define the range. The values that establish the upper and lower limits of the range are called test values. The first test value in the BETWEEN clause must be either alphabetically or numerically before the second test value. Consider the following statements and their result sets:

```
SELECT FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE CustomerID BETWEEN 1012 AND 1034
```



	CustomerID	FirstName	Last
1	1012	Helen	And
2	1022	Andrew	Willi
3	1030	Alice	Golc
4	1034	Robert	Mey

```
SELECT CustomerID, FirstName, LastName, State, ZIP
FROM CUSTOMERS
WHERE state BETWEEN 'C' AND 'J'
```



	FirstName	LastName	State
1	Edgar	Harrison	IL
2	Jerome	Keller	IL
3	Rose	Callahan	CA

The BETWEEN operator is handled differently by different DBMSs. In some databases, the BETWEEN operator is exclusive; selects only the rows that are between the test values and excludes records matching the test values themselves.

In most DBMSs, the BETWEEN operator is inclusive; selects rows that are between the test values, and includes rows matching the test values.

In still other DBMSs, the BETWEEN operator selects rows between the test values, including the first test value, but excluding the last test value.

It is important to check your database documentation to see how the BETWEEN operator is handled.

In SQL Server, the BETWEEN operator is inclusive, meaning that it selects rows that are between the test values and includes rows matching the test values.

The LIKE operator

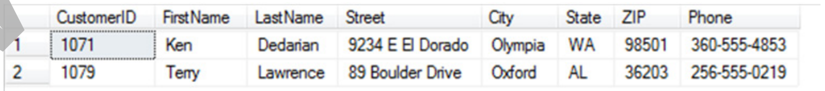
The LIKE operator is used to test for a specified pattern in a column containing character data. Wildcard characters are used with the LIKE operator. The wildcard characters are the underscore (_) and the percent sign (%).

The underscore character is used to represent individual characters. For instance, the string f_ _t would match foot, fact, fort and fast, but would not match fat or first.

The percent sign is used to represent a string of zero or more characters. For instance, the string f%t would match foot, fact, fort, fast, fat, first, furthest and the string ft, but would not match facts or fits.

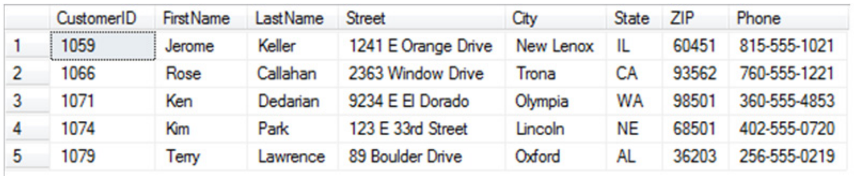
Consider the following select statements and their result sets. The first one selects records where the city begins with the letter "O." The second one selects records where the city includes the letter "O" anywhere in the city name. The third one selects records where the city does not include the letter "O" anywhere in the city name, and the fourth one selects records for cities that end in the letter "D."

```
SELECT *
FROM CUSTOMERS
WHERE City LIKE 'o%'
```



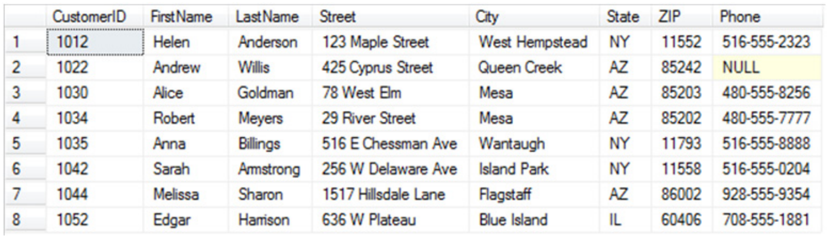
	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	1071	Ken	Dedarian	9234 E El Dorado	Olympia	WA	98501	360-555-4853
2	1079	Terry	Lawrence	89 Boulder Drive	Oxford	AL	36203	256-555-0219

```
SELECT *
FROM CUSTOMERS
WHERE City LIKE '%o%'
```



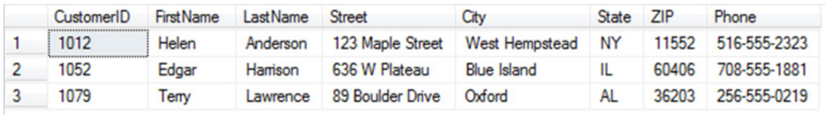
	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	1059	Jerome	Keller	1241 E Orange Drive	New Lenox	IL	60451	815-555-1021
2	1066	Rose	Callahan	2363 Window Drive	Trona	CA	93562	760-555-1221
3	1071	Ken	Dedarian	9234 E El Dorado	Olympia	WA	98501	360-555-4853
4	1074	Kim	Park	123 E 33rd Street	Lincoln	NE	68501	402-555-0720
5	1079	Terry	Lawrence	89 Boulder Drive	Oxford	AL	36203	256-555-0219

```
SELECT *
FROM CUSTOMERS
WHERE City NOT LIKE '%o%'
```



	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	1012	Helen	Anderson	123 Maple Street	West Hempstead	NY	11552	516-555-2323
2	1022	Andrew	Willis	425 Cyprus Street	Queen Creek	AZ	85242	NULL
3	1030	Alice	Goldman	78 West Elm	Mesa	AZ	85203	480-555-8256
4	1034	Robert	Meyers	29 River Street	Mesa	AZ	85202	480-555-7777
5	1035	Anna	Billings	516 E Chessman Ave	Wantaugh	NY	11793	516-555-8888
6	1042	Sarah	Armstrong	256 W Delaware Ave	Island Park	NY	11558	516-555-0204
7	1044	Melissa	Sharon	1517 Hillsdale Lane	Flagstaff	AZ	86002	928-555-9354
8	1052	Edgar	Harrison	636 W Plateau	Blue Island	IL	60406	708-555-1881

```
SELECT *
FROM CUSTOMERS
WHERE City LIKE '%d'
```



	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	1012	Helen	Anderson	123 Maple Street	West Hempstead	NY	11552	516-555-2323
2	1052	Edgar	Harrison	636 W Plateau	Blue Island	IL	60406	708-555-1881
3	1079	Terry	Lawrence	89 Boulder Drive	Oxford	AL	36203	256-555-0219

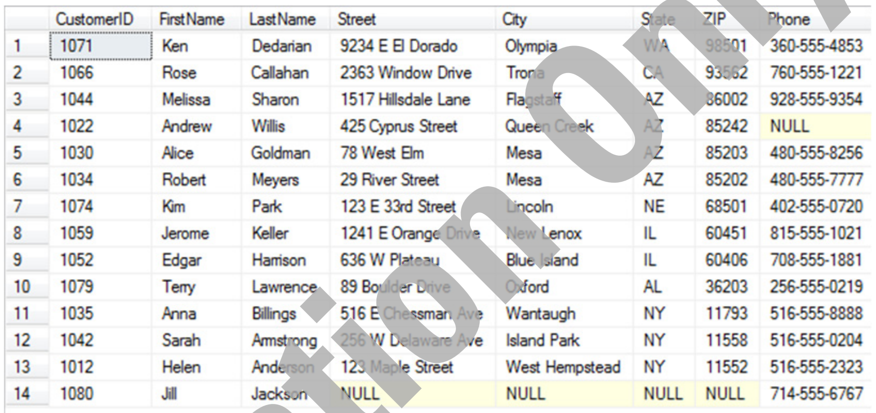
In every one of these examples, SQL is case-insensitive when comparing the data. That is, an upper case letter ‘O’ is equivalent to a lower case letter ‘o’. This behavior is controlled by the collation setting for the database. If you choose a different collation setting, then SQL could treat the letters ‘O’ and ‘o’ as nonequivalent, and sort them into different groups.

The ORDER BY clause

SQL also provides the ORDER BY clause to allow output to be sorted based on a column or group of columns. The order of output can be designated as ascending (ASC) which is the default, or descending (DESC).

Suppose you want to send out a bulk mailing to your customers and you need to sort your list by ZIP code in descending order. You could accomplish that with the following statement.

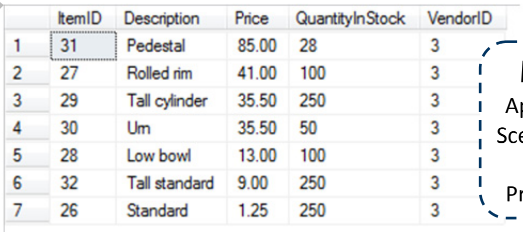
```
SELECT *
FROM CUSTOMERS
ORDER BY ZIP DESC
```



	CustomerID	FirstName	LastName	Street	City	State	ZIP	Phone
1	1071	Ken	Dedarian	9234 E El Dorado	Olympia	WA	98501	360-555-4853
2	1066	Rose	Callahan	2363 Window Drive	Trona	CA	93562	760-555-1221
3	1044	Melissa	Sharon	1517 Hillsdale Lane	Flagstaff	AZ	86002	928-555-9354
4	1022	Andrew	Willis	425 Cyprus Street	Queen Creek	AZ	85242	NULL
5	1030	Alice	Goldman	78 West Elm	Mesa	AZ	85203	480-555-8256
6	1034	Robert	Meyers	29 River Street	Mesa	AZ	85202	480-555-7777
7	1074	Kim	Park	123 E 33rd Street	Lincoln	NE	68501	402-555-0720
8	1059	Jerome	Keller	1241 E Orange Drive	New Lenox	IL	60451	815-555-1021
9	1052	Edgar	Harrison	636 W Plateau	Blue Island	IL	60406	708-555-1881
10	1079	Terry	Lawrence	89 Boulder Drive	Oxford	AL	36203	256-555-0219
11	1035	Anna	Billings	516 E Chessman Ave	Wantaugh	NY	11793	516-555-8888
12	1042	Sarah	Armstrong	256 W Delaware Ave	Island Park	NY	11558	516-555-0204
13	1012	Helen	Anderson	123 Maple Street	West Hempstead	NY	11552	516-555-2323
14	1080	Jill	Jackson	NULL	NULL	NULL	NULL	714-555-6767

The ORDER BY clause can also use multiple columns to order the output. For example, suppose you want to see a list of inventory items for a particular vendor, listed in descending order by price and quantity in stock. Consider the following statement and its result set.

```
SELECT *
FROM INVENTORYITEMS
WHERE VendorID=3
ORDER BY Price DESC, QuantityInStock DESC
```



	ItemID	Description	Price	QuantityInStock	VendorID
1	31	Pedestal	85.00	28	3
2	27	Rolled rim	41.00	100	3
3	29	Tall cylinder	35.50	250	3
4	30	Urn	35.50	50	3
5	28	Low bowl	13.00	100	3
6	32	Tall standard	9.00	250	3
7	26	Standard	1.25	250	3

MMM
Application Scenario 3-1: Making Predictions

Exercise 3-5: Selecting data

In this exercise, you will execute queries against the tables in the HomeGrown database.

1. Restore the SQL Server window.
2. Open a new Query window.
3. Type: USE HomeGrown, press ENTER to move the cursor to the next line, type GO, then press ENTER three times.
4. To view all the fields of all the records in the VENDORS table, enter the following command, then click the **Execute** button:

   ```
   SELECT * FROM VENDORS
   ```

5. To view only the Name, State and ZIP fields, delete the previous SELECT statement and type and execute the following statement:

   ```
   SELECT Name, State, ZIP
   FROM VENDORS
   ```

You can also click the drop-down arrow in the Available Databases box in the SQL Editor toolbar to specify which database to use.

6. To filter the result set to show only this information for vendors in NY state, add the following WHERE clause to the SELECT statement:

```
WHERE State = 'NY'
```

7. To view vendors whose company name begins with the letter G, change the WHERE clause to the following:

```
WHERE Name LIKE 'G%'
```

8. To list the distinct states where the vendors reside, replace the SELECT statement with the following:

```
SELECT DISTINCT State
FROM VENDORS
```

9. To select all the fields of the records for which the ZIP code begins with the number 8 or higher, replace the SELECT statement with the following:

```
SELECT *
FROM VENDORS
WHERE ZIP > '8'
```

10. Select the records with ZIP codes higher than 98000 or lower than 86000.
11. Use the NOT keyword to list all the records except the one in CA.
12. Display the records in ascending order by ZIP code.
13. Display the records that include the word "Way" in the street address.
14. Display the records for vendors in NY, CA or OR.
15. Close the Query window without saving the changes, then minimize the SQL Server window.

Case Scenario 3-2: Now Where Did I Put that Record?

Case Scenarios

Your Illuminated database should now contain data in all its tables and you can now retrieve specific records.

Write and test SELECT statements that will accomplish the following:

1. List book Title, PublisherID and MSRP in descending order by MSRP.
2. List the FirstName and LastName of all authors whose first name begins with the letter “S.”
3. List book Title and MSRP for books that cost more than $30.00 and less than $50.00.
4. List all books that have the letters “ing” somewhere in the title.
5. List all publishers that reside in a state where the ZIP code is lower than “45000.”
6. List all books published after 01/01/2011.

Updating Data

Objective 3.3

The SQL UPDATE command is used to update existing table rows with new data values. An UPDATE statement can update only one table; however, you can use a single statement to update multiple columns. In its simplest form, the syntax for the UPDATE command is:

UPDATE *table_name*
SET *column1=value, column2=value, …*
WHERE *condition*

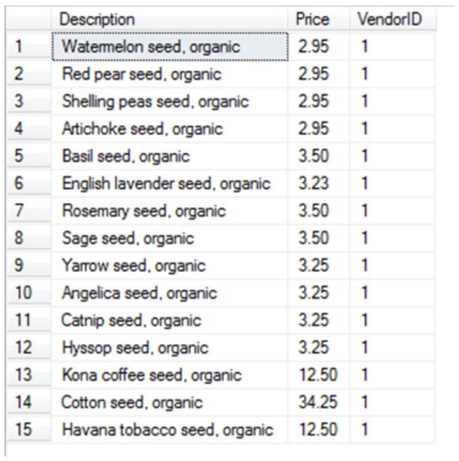
The SET clause often makes use of mathematical operators. The mathematical operators supported by SQL include:

+	addition
-	subtraction
*	multiplication
/	division
%	modulus

If a WHERE clause is not included, all rows in the table will be updated.

Consider the following select statement and result set shown for the INVENTORYITEMS table. The query selects the Description, Price and VendorID fields for all items where the VendorID=1.

```
SELECT Description, Price, VendorID
FROM INVENTORYITEMS
WHERE VendorID=1
```



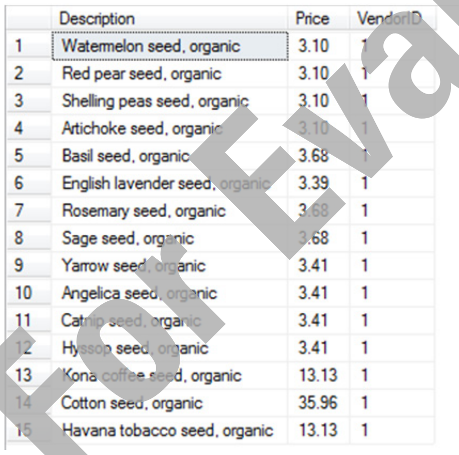
	Description	Price	VendorID
1	Watermelon seed, organic	2.95	1
2	Red pear seed, organic	2.95	1
3	Shelling peas seed, organic	2.95	1
4	Artichoke seed, organic	2.95	1
5	Basil seed, organic	3.50	1
6	English lavender seed, organic	3.23	1
7	Rosemary seed, organic	3.50	1
8	Sage seed, organic	3.50	1
9	Yarrow seed, organic	3.25	1
10	Angelica seed, organic	3.25	1
11	Catnip seed, organic	3.25	1
12	Hyssop seed, organic	3.25	1
13	Kona coffee seed, organic	12.50	1
14	Cotton seed, organic	34.25	1
15	Havana tobacco seed, organic	12.50	1

Suppose this vendor has increased the prices for their products by 5% and you need to pass that cost along to your customers. You can use the following UPDATE statement to accomplish the task.

```
UPDATE INVENTORYITEMS
SET Price = Price + (Price* 0.05)
WHERE VendorID=1
```

To check the results of an UPDATE statement, execute a SELECT statement.

After the UPDATE statement above is executed, the statement `SELECT Description, Price, VendorID FROM INVENTORYITEMS WHERE VendorID=1` returns the following result set.



	Description	Price	VendorID
1	Watermelon seed, organic	3.10	1
2	Red pear seed, organic	3.10	1
3	Shelling peas seed, organic	3.10	1
4	Artichoke seed, organic	3.10	1
5	Basil seed, organic	3.68	1
6	English lavender seed, organic	3.39	1
7	Rosemary seed, organic	3.68	1
8	Sage seed, organic	3.68	1
9	Yarrow seed, organic	3.41	1
10	Angelica seed, organic	3.41	1
11	Catnip seed, organic	3.41	1
12	Hyssop seed, organic	3.41	1
13	Kona coffee seed, organic	13.13	1
14	Cotton seed, organic	35.96	1
15	Havana tobacco seed, organic	13.13	1

You can update multiple columns simultaneously with the UPDATE command. For example, the following command will increase the price for selected inventory items by 10% and reduce the Quantity In Stock by 1:

```
UPDATE INVENTORYITEMS
SET Price = Price + (Price* 0.10), QuantityInStock = QuantityInStock-1
WHERE VendorID=2
```

If a field allows NULL values, you can use the UPDATE statement to set the value for that field to NULL.

Be careful with the UPDATE command; it is very powerful, enabling you to update hundreds or thousands of rows with a single command. Be certain that your SET and WHERE clauses are accurate and will produce the results you anticipate. It is also highly recommended that you back up your data and test the UPDATE statement on a separate copy of the database before you actually execute it on the real one.

UPDATE with CASE

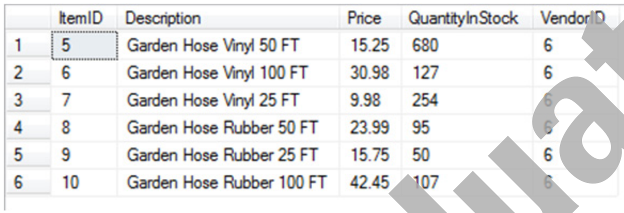
In SQL, CASE is a unique conditional clause that provides if-then-else logic for a SELECT or UPDATE command. CASE provides when-then-else functionality (as in WHEN this condition is true, THEN do this). There are two forms of the CASE clause. These are:

```
CASE
    WHEN true or false expression THEN result_1
    ELSE result_2
    END
```

```
CASE the expression
    WHEN the condition THEN result_1
    ELSE result_2
    END
```

- The CASE expression can have multiple WHEN clauses.
- The ELSE clause may be omitted.
- The value of the CASE expression is the value of the first WHEN clause that is true. If none are true, the result is the ELSE.
- If there is no ELSE clause, the result is NULL.
- All expressions must be compatible data types.

Consider the following data:

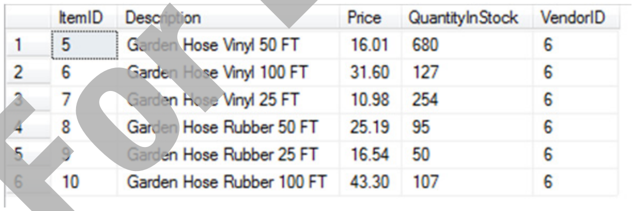


	ItemID	Description	Price	QuantityInStock	VendorID
1	5	Garden Hose Vinyl 50 FT	15.25	680	6
2	6	Garden Hose Vinyl 100 FT	30.98	127	6
3	7	Garden Hose Vinyl 25 FT	9.98	254	6
4	8	Garden Hose Rubber 50 FT	23.99	95	6
5	9	Garden Hose Rubber 25 FT	15.75	50	6
6	10	Garden Hose Rubber 100 FT	42.45	107	6

Now consider the following UPDATE with CASE statement and its result set:

```
UPDATE INVENTORYITEMS
SET Price = CASE
WHEN (Price < 10) THEN Price *1.10
WHEN (Price BETWEEN 10 AND 25) THEN Price *1.05
ELSE Price *1.02
END
WHERE VendorID=6
```

MMM
Application Scenario 3-2: State Your Case



	ItemID	Description	Price	QuantityInStock	VendorID
1	5	Garden Hose Vinyl 50 FT	16.01	680	6
2	6	Garden Hose Vinyl 100 FT	31.60	127	6
3	7	Garden Hose Vinyl 25 FT	10.98	254	6
4	8	Garden Hose Rubber 50 FT	25.19	95	6
5	9	Garden Hose Rubber 25 FT	16.54	50	6
6	10	Garden Hose Rubber 100 FT	43.30	107	6

Deleting Data

Objective
3.4

The DELETE command is used to remove records from a table. When the DELETE command is used, the entire row is removed. DELETE cannot be used to remove values from individual columns. Once a DELETE statement has been executed successfully against a table, the deleted data cannot be recovered. For this reason it is highly recommended that you back up your data and test the DELETE statement on a separate copy of the database before you actually execute it on the real one.

To remove a specific record, use the primary key for the table in the WHERE clause. For example, the following command would remove Customer 1088 from the CUSTOMERS table:

```
DELETE FROM CUSTOMERS
WHERE CustomerID = 1088
```

Notice that you cannot name fields, because all fields for the given record will be removed.

You can also remove multiple records with one statement. For example, the following statement would remove all records for customers who live in Arizona from the CUSTOMERS table:

```
DELETE FROM CUSTOMERS
WHERE State = 'AZ'
```

To delete all the data from a table, do not specify a WHERE clause. For example, following statement would remove **all** records from the CUSTOMERS table:

```
DELETE FROM CUSTOMERS
```

This command deletes all the records, but leaves the table structure intact. The TRUNCATE statement performs the same function. The syntax for the TRUNCATE command is: TRUNCATE TABLE *table_name.*

The SQL DELETE statement removes records from a single table. If you need to delete records from multiple tables, such as in a situation where you delete a parent key, a common solution is to handle the issue programmatically. However, if you have specified to cascade deletions, then SQL will delete records from multiple tables when you delete a parent key. (See Lesson 2, *Specifying actions for foreign key constraints* for information on cascading deletions.)

Exercise 3-6: Updating and deleting data

Exercise

In this exercise, you will update and delete data.

1. Restore the SQL Server window.
2. Open a new Query window.
3. Execute the following code in the Query window:
```
Use HomeGrown
GO
```
4. Check the price of all items supplied by vendor 1 with the following statement:
```
SELECT *
FROM VENDORS
WHERE VendorID=1
```
5. Increase the price by 5 % for all items supplied by vendor 1, by executing the following code in the Query window:
```
UPDATE INVENTORYITEMS
SET Price = Price + (Price* 0.05)
WHERE VendorID=1
```
6. Execute a SELECT statement to view the items again now that the updates have been completed.
7. Execute a SELECT statement to view the prices of all inventory items in descending order.
8. Execute a DELETE statement to delete the one item that costs more than $200 USD.
9. Execute a SELECT statement to view the prices of all inventory items in descending order to view the remaining items.
10. Close the Query window without saving any changes.
11. Log off SQL Server and close the window.

Lesson Summary

You are now able to:

☑ Understand the purpose and function of Data Manipulation Language (DML).

☑ Perform data entry tasks using the SSMS GUI.

☑ Create a database diagram.

☑ Insert data using the INSERT statement.

☑ Insert data using INSERT scripts.

☑ Select data using the SELECT statement.

☑ Update data using the UPDATE statement.

☑ Delete data using the DELETE statement.

Review Questions

1. Which of the following SQL statements would you use to change the data in a record?
   a. SELECT.
   b. UPDATE.
   c. INSERT.
   d. DELETE.

2. In the SSMS GUI, which tool can you use to view and check the relationships between tables in a database?
   a. The Properties window.
   b. The Object Explorer.
   c. A Database Diagram.
   d. An INSERT file.

3. When can you skip fields when inserting values with an INSERT statement?
   a. When you specify columns for the INSERT statement.
   b. When the fields accept NULL values.
   c. When the fields are text fields.
   d. When the fields have a default value assigned.

4. When you include column names in an INSERT statement and you do not list a column, what happens to that column when a record is inserted?
   a. That column is dropped from the table.
   b. That column receives a 0 if it is a numeric column.
   c. That column receives the same value as the column that precedes it in the table structure.
   d. That column receives its default value or a NULL value if the column can accept nulls.

5. When you include column names in an INSERT statement how do you handle columns that auto-increment?
   a. Name then in the statement, but do not supply any value for them.
   b. Skip auto-increment columns when naming columns in the INSERT statement.
   c. Insert a 1 into these columns, and SQL will increment them accordingly.
   d. Insert a 0 into these columns, and SQL will increment them accordingly.

6. What is an INSERT script?
    a. An INSERT script is a special piece of programming code that includes characters you cannot enter into the Query window.
    b. An INSERT script is a built-in SQL object that you must initialize before you can add records to a table.
    c. An INSERT script is a text file that includes a series of INSERT statements.
    d. An INSERT script is a built-in SQL object that you must deactivate when you want to add records to a table.

7. Which of the following is true of INSERT scripts?
    a. They must be written in PERL.
    b. They must be run outside the SQL Server GUI.
    c. They can always be executed in any order.
    d. They may need to be executed in a particular order to maintain relational (referential) integrity.

8. The records returned as the result of a query are called a(n):
    a. result set.
    b. query set.
    c. selection.
    d. datasheet.

9. To ensure that a query returns only unique values, you can use the:
    a. UNIQUE ONLY clause.
    b. SET UNIQUE ON phrase.
    c. DISTINCT keyword.
    d. NOT LIKE operator.

10. Which of the following can you use to select data that matches a list of values?
    a. The BETWEEN operator.
    b. The IN operator.
    c. The LIKE operator.
    d. The Truth table.