Chapter 4

Database Development and Management with SQL

Objectives

- Get started with SQL
- Create database objects with SQL
- Manage database objects with SQL
- Control database object privileges with SQL

4.1 Introduction

In this chapter, SQL, a commonly used database development tool, will be introduced. You will learn some often-used SQL commands and use them to create and manage database objects. You will use SQL to create database objects such as tables, constraints, users, and roles. You will learn how to modify the database structure and data content with SQL commands.

4.2 Structured Query Language

As shown in the previous chapter, you can use Windows Azure SQL Database to create database objects with a graphical tool. Like Microsoft, some other DBMS products also provide graphical tools for database development and management. It is relatively easy to learn how to use graphical tools. Especially for beginners, graphical database development tools can help avoid many mistakes due to lack of experience.

On the other hand, experienced database developers prefer a more popular database development tool called SQL. There are some advantages to using SQL instead of graphical database development tools. For example, a DBMS product does not share its graphical tools with another DBMS product. Also, a graphical database development tool requires interactivity with users.

Therefore, it is not good for bulk operations. On the other hand, SQL code is portable and can be executed by bulk operations.

With SQL, database developers can create database objects such as tables and views. They can also use SQL to perform operations such as inserting, updating, and deleting records in a table. SQL provides functions and procedures to help you implement database security measures and performance tuning. SQL commands can be used to manage databases such as defining constraints to maintain database integrity. Since the early 1970s, SQL has been a primary tool in database development.

As the most important database development tool, SQL is supported by various DBMS products. It has been modified in many ways to serve the special needs of each individual DBMS product. To set up a common ground, the American National Standards Institute (ANSI) published the standard for SQL. In 1986, the ANSI standardized the database querying language and named it ANSI SQL-86. Since then, the SQL standard has been updated several times. SQL has been formally adopted as an International Standard by the International Organization for Standardization (ISO), the International Electrotechnical Commission (IEC), and it has also been adopted as a Federal Information Processing Standard (FIPS) for the U.S. federal government. There are important new features added to this latest version such as the support for XML and additional collection data types.

Even with the added features, ANSI-based SQL is still a very basic query language. DBMS vendors have added their own extensions on top of the basic ANSI-based SQL. The extended SQL language used by Microsoft SQL Server is called Transact-SQL, which supports ANSI SQL. In addition, Transact-SQL has added some commands for procedural programming. In this chapter, let us first study the basic commands for creating and managing database objects.

There are three major types of commands for creating and managing database objects:

- **Data Definition Language (DDL)**: Statements in DDL can be used to create, modify, and delete database objects such as databases, tables, columns, and constraints. The commonly used DDL statements are
  CREATE: Used to create database objects
  ALTER: Used to modify database objects
  DROP: Used to delete database objects
- **Data Control Language (DCL)**: Statements in DCL can be used to grant or revoke permissions on database objects. The commonly used DCL statements are
  GRANT: Used to grant permissions for a user to use a database object or execute some SQL commands
  REVOKE: Used to remove previously granted permissions
- **Data Manipulation Language (DML)**: Statements in DML can be used to select, insert, update, and delete data in a database object. The commonly used DML statements are
  SELECT: Used to query information from a database
  INSERT: Used to insert new rows of data into database tables
  UPDATE: Used to modify data values in database tables
  DELETE: Used to delete existing rows in database tables

SQL statements by various DBMS vendors may be different. Often, the difference is minor. SQL code conversion from one DBMS to another DBMS does not take much effort. This makes SQL almost portable across DBMS products from different database vendors. Some companies

may own multiple DBMS products. In such a case, learning SQL is particularly important for those who will work with different DBMS products.

To work with SQL statements, Windows Azure SQL Database provides tools for you to edit, debug, and execute Transact-SQL statements. In the following sections, you will use SQL statements to create database objects such as tables and constraints with DDL. You will also learn how to modify the structures of database objects. You will use DML to insert, update, and delete data for database objects created by you. To accomplish some database management tasks, you will use DCL to grant and revoke permissions.

4.3 Creating Database Objects

In this section, as a case study, a data model called Computer_Store will be converted to a relational database with SQL. The Computer_Store data model is given in Figure 4.1.

4.3.1 Defining Data Types

When creating a table, you need to specify the data type for each column. Let us first take a look at the data types before creating tables. Some commonly used data types are listed below. Data types were introduced in the previous chapter. Here, we review some of them and focus on how to define the data types with SQL statements. For each data type, there is an example to demonstrate how to specify the data type for a column.

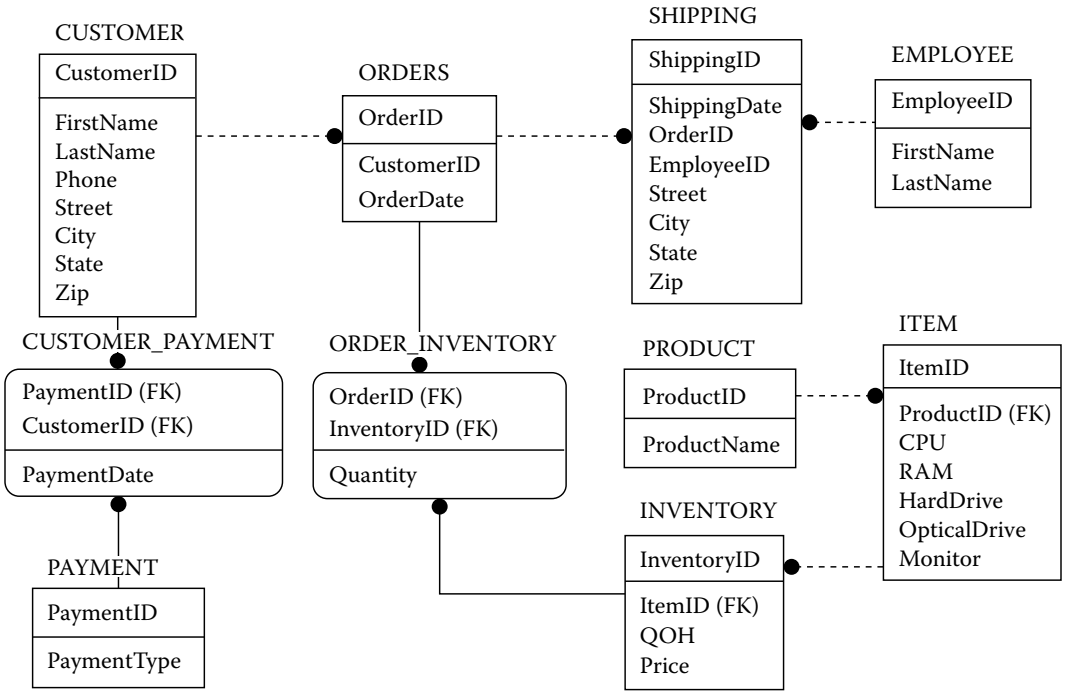

Figure 4.1 Computer_Store data model.

1. Character data types: Character data consist of any combination of letters, symbols, and numeric characters.
   a. **CHAR**: It is a fixed-length character data type.

      Example: To specify the column, Street, and to have the CHAR data type with length 30, use

      Street CHAR(30)
   b. **VARCHAR**: It is a variable-length character data type.
      Example: To specify the column, Description, and to have the VARCHAR data type with the maximum length of 300, use

      Description VARCHAR(300)
   c. **VARCHAR(MAX)**: This data type allows you to define a character data type with a length up to 2^{31} bytes of data.
      The difference between the data type CHAR and the data type VARCHAR is that a column specified by CHAR may have many blank spaces if the data values do not fill up the column width. The advantage of CHAR is that it runs faster when processing data stored in a fixed-length column.
      Corresponding to CHAR, VARCHAR, and VARCHAR(MAX), there are NCHAR, NVARCHAR, and NVARCHAR(MAX) data types for Unicode characters. Unicode characters are used for some special characters in languages such as Chinese and Japanese. Each Unicode character takes two bytes. Therefore, the lengths of NCHAR, NVARCHAR, and NVARCHAR(MAX) are half of CHAR, VARCHAR, and VARCHAR(MAX).

2. Number data types: Number data types include integers and decimals.
   a. **INT**: The INT data type consists of positive, zero, and negative whole numbers. The range of INT is from –2,147,483,648 to 2,147,483,647.
      Example: Integers can be used as the data type for a key column. In the following, we specify the primary key column ClassID with the INT data type:

      ClassID INT PRIMARY KEY

   b. **DECIMAL (or NUMERIC)**: The DECIMAL data type includes positive and negative decimal numbers. The values of DECIMAL data are from – 10^38 + 1 through 10^38 – 1.
      Example: The following statement specifies the column GPA with the DECIMAL data type:

      GPA DECIMAL

3. Date and Time data type: It represents date and time values.
   a. **DATETIME**: This data type specifies a column to contain date and time values from 1-1-1753 to 12-31-9999.
      Example: To define the column ExpirationDate to have date and time values, use

      ExpirationDate DATETIME

4. Monetary data type: It represents amounts of money.
   a. **MONEY**: This data type specifies a column to contain money values.
      Example: To define the money data type for the column Price, use

      Price MONEY

More data types are supported by Windows Azure SQL Database.

4.3.2 Constraints

When creating a table, you also need to specify constraints for some of the columns. Constraints can be used to keep the integrity of your database, define the range of values to be entered into a column, and specify the properties of a column. Let us take a look at the commonly used constraints.

1. *Referential Constraints*: Referential constraints are used to keep the relationship between tables. The commonly used foreign key constraint and primary key constraint belong to referential constraints.
   a. **Primary Key Constraint**: This constraint is used to specify a column or a group of columns as a primary key. There are two ways to define a primary key. If the primary key is defined on a single column, you can specify the column as a primary key without the keyword CONSTRAINT. For example, to make the column ProductID the primary key, use the SQL statement:

      ```
      ProductID INT PRIMARY KEY
      ```

      A constraint enforced on a single column is classified as a column constraint.
      If a primary key is defined on multiple columns, it is called a combination primary key and you need to use the keyword CONSTRAINT to define it. The syntax for defining a primary key by using the keyword CONSTRAINT is

      ```
      CONSTRAINT constraint_name PRIMARY KEY (multiple_column_names)
      ```

      To illustrate the use of the keyword CONSTRAINT, consider the example that defines the combination primary key on the columns OrderID and InventoryID:

      ```
      CONSTRAINT OrderID_InventoryID_pk PRIMARY KEY (OrderID, InventoryID)
      ```

      If a constraint is enforced on multiple columns, it is called a table constraint. When more than one column is included in a constraint, you should use the table constraint.
   b. **Foreign Key Constraint**: A foreign key constraint can be defined on a single column or multiple columns. When defining it on a single column, you can use either an unnamed or a named constraint. For an unnamed foreign key constraint, consider the following example that makes the column ProductID the foreign key:

      ```
      FOREIGN KEY (ProductID) REFERENCES PRODUCT (ProductID)
      ```

      The word ProductID after the keyword phrase FOREIGN KEY is the foreign key column name in the child table. The parent table name and the primary key column in the parent table are placed behind the keyword REFERENCES. You can also use a named foreign key constraint by using the keyword CONSTRAINT. When a foreign key is defined on multiple columns, you should use a named foreign key constraint. The syntax for the named foreign key constraint is

      ```
      CONSTRAINT constraint_name FOREIGN KEY (fk_column_name)
      REFERENCES parent_table_name (primary_key_column)
      ```

      The constraint name is a character string. To make the name meaningful, you can name the constraint as TableName_ColumnName_fk. The keyword phrase FOREIGN KEY

indicates that the constraint is a type of foreign key. The keyword REFERENCES specifies the name of the parent table whose primary key column is used as the foreign key column in the child table. For example, if you want to specify the column ItemID as a foreign key, use the SQL statement below:

```
CONSTRAINT Inventory_ItemId_fk FOREIGN KEY (ItemID)
REFERENCES ITEM (ItemID)
```

As indicated in the above SQL statement, the constraint name is Inventory_ItemId_fk, which should be unique in the database. The name of the foreign key column is ItemID. The parent table is ITEM, and the primary key column in the parent table is ItemID. As you can see in the above SQL statement, two tables are connected by a foreign key constraint. If the primary key column in the parent table is referenced by a foreign key in the child table, its content values cannot be deleted or changed. If you really need to delete or change a value of the primary key referenced by a foreign key, you have to delete the foreign key constraint first and then make a change.

2. Domain Constraints: A domain constraint determines the values to be entered into a column. Constraints such as CHECK, NOT NULL, and UNIQUE are domain constraints.
   a. **CHECK Constraint**: A CHECK constraint can be used to limit the values entered into a column to enforce domain integrity. The syntax of the constraint is shown below:

      ```
      CONSTRAINT constraint_name CHECK(constraint_conditions)
      ```

      Logical operators such as AND or OR can be used to enforce multiple conditions. As an example, let us include only the values that are greater or equal to zero in the column Quantity. The SQL statement is given below:

      ```
      CHECK(Quantity >= 0)
      ```

      Or, we can use the named CHECK constraint shown below:

      ```
      CONSTRAINT Quantity_ck CHECK(Quantity >= 0)
      ```

   b. **UNIQUE Constraint**: A UNIQUE constraint can be used to enforce uniqueness on nonprimary key columns. For example, we can make the values in the column ProductName unique with the following SQL statement:

      ```
      CONSTRAINT ProductName_unique UNIQUE(ProductName)
      ```

      You can also use an unnamed unique constraint by using the keyword UNIQUE.
   c. **DEFAULT Constraint**: This constraint specifies the default value for a column. For example, suppose that you want the default value for the column Quantity be 0; you can do this by using the following SQL statement:

      ```
      Quantity INT DEFAULT 0
      ```

   d. **NOT NULL Constraint**: This constraint can be used to prevent null values in a column. If you want to make sure that the column has no null value, use the following SQL statement:

      ```
      OrderDate DATETIME NOT NULL
      ```

e. **Uniqueidentifier**: This constraint can be used to create a surrogate key, which is a column with automatically generated unique integers. Suppose you want to let the machine generate a unique number for the column ProductID every time a new product is added to the table PRODUCT. You can use the following SQL statement to accomplish this task:

   ProductID uniqueidentifier NOT NULL

Now that you have learned about data types and constraints, you can create tables. You will create the database and tables corresponding to the entities displayed in Figure 4.1.

4.3.3 Creating Database

The first task is to create database objects such as tables. In Windows Azure SQL Database, the database can be created in the Windows Azure Management Portal. Later in this chapter, we will demonstrate how to create databases Computer_Store with the Windows Azure Management Portal.

4.3.4 Creating Tables

For each entity in Figure 4.1, we will first define a corresponding table.

Then, we will create the table with SQL statements based on the table definition. Let us start with the tables that do not have foreign key columns. The tables CUSTOMER, EMPLOYEE, PAYMENT, and PRODUCT belong to this type. The column definitions for the table CUSTOMER are given in Table 4.1.

The SQL statement to create the table is

```
CREATE TABLE CUSTOMER
(
       CustomerID INT PRIMARY KEY,
       FirstName VARCHAR(50),
       LastName VARCHAR(50) NOT NULL,
       Phone CHAR(12),
       Street VARCHAR(50),
       City VARCHAR(50),
       State VARCHAR(50),
       Zip CHAR(5)
)
```

The keyword phrase PRIMARY KEY indicates that a column is defined as a primary key. In the above SQL statement, the keywords are capitalized only for clarity. In fact, keywords in an SQL statement are not case sensitive.

For the table EMPLOYEE, the definitions of the columns are listed in Table 4.2.

The SQL statement for creating the table EMPLOYEE is given below:

```
CREATE TABLE EMPLOYEE
(
       EmployeeID INT PRIMARY KEY,
       FirstName VARCHAR(50),
       LastName VARCHAR(50) NOT NULL
)
```

Table 4.1 Column Descriptions for CUSTOMER Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
CustomerID	INT	PK	
FirstName	VARCHAR(50)		
LastName	VARCHAR(50)		
Phone	CHAR(12)		
Street	VARCHAR(50)		
City	VARCHAR(50)		
State	VARCHAR(50)		
Zip	CHAR(5)		

Table 4.2 Column Descriptions for EMPLOYEE Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
EmployeeID	INT	PK	
FirstName	VARCHAR(50)		
LastName	VARCHAR(50)		

Table 4.3 gives the definition for the table PAYMENT.

The following is the SQL statement for creating the table PAYMENT:

```
CREATE TABLE PAYMENT
(
      PaymentID INT PRIMARY KEY,
      PaymentType VARCHAR(50) NOT NULL
)
```

For the table PRODUCT, you have the table definition in Table 4.4.

Table 4.3 Column Descriptions for PAYMENT Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
PaymentId	INT	PK	
PaymentType	VARCHAR(50)		

Table 4.4 Column Descriptions for PRODUCT Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
ProductID	INT	PK	
ProductName	VARCHAR(50)		

The SQL statement for creating the table PRODUCT is listed below:

```
CREATE TABLE PRODUCT
(
             ProductID INT PRIMARY KEY,
             ProductName VARCHAR(50),
)
```

For the tables ITEM, INVENTORY, SHIPPING, and ORDERS, the table definition contains foreign keys. To implement a foreign key, you need to specify a foreign key constraint in the SQL statement.

Starting with the table ITEM, we will define a foreign key. The ITEM table definition is given in Table 4.5.

Using the following SQL statement, you can create the table ITEM:

```
CREATE TABLE ITEM
(
       ItemID INT PRIMARY KEY,
       ProductID INT,
       CPU VARCHAR(50),
       RAM VARCHAR(50),
       HardDrive VARCHAR(50),
       OpticalDrive VARCHAR(50),
       Monitor VARCHAR(50),
       CONSTRAINT Item_ProductId_fk
                    FOREIGN KEY (ProductID)
                    REFERENCES PRODUCT (ProductID)
)
```

As indicated in the above SQL statement, the foreign key constraint name is Item_ProductId_fk, which should be unique. The name of the foreign key column is ProductID. The entry after the REFERENCES is the name of the parent table. The parent table is PRODUCT, and the primary key in the parent table is ProductID.

After the table ITEM is created, you can create the table INVENTORY that has a foreign key column ItemID. The table definition is given in Table 4.6.

Table 4.5 Column Descriptions for ITEM Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
ItemID	INT	PK	
ProductID	INT	FK	PRODUCT
CPU	VARCHAR(50)		
RAM	VARCHAR(50)		
HardDrive	VARCHAR(50)		
OpticalDrive	VARCHAR(50)		
Monitor	VARCHAR(50)		

Table 4.6 Column Descriptions for INVENTORY Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
InventoryID	INT	PK	
ItemID	INT	FK	ITEM
QOH	INT		
Price	Money		

The following code can be used to create the table INVENTORY. In the SQL statement, you need to add the code for the foreign key constraint.

```
CREATE TABLE INVENTORY
(
       InventoryID INT PRIMARY KEY,
       ItemID INT,
       QOH INT DEFAULT 0 CHECK(QOH >= 0) NOT NULL,
       Price MONEY DEFAULT 0 CHECK (Price >= 0) NOT NULL,
       CONSTRAINT Inventory_ItemId_fk
                    FOREIGN KEY (ItemID)
                    REFERENCES ITEM (ItemID)
)
```

The table definition of ORDERS is given in Table 4.7.

To create the table ORDERS, use the following SQL statement:

```
CREATE TABLE ORDERS
(
       OrderID INT PRIMARY KEY,
       CustomerID INT NOT NULL,
       OrderDate DATETIME NOT NULL,
       CONSTRAINT Orders_CustomerId_fk
              FOREIGN KEY (CustomerID)
              REFERENCES CUSTOMER (CustomerID)
)
```

The table definition for SHIPPING is shown in Table 4.8.

In Table 4.8, there are two foreign key columns. To implement these two foreign keys, you need to specify two foreign key constraints. The SQL statement used to create the table SHIPPING is given below:

Table 4.7 Column Descriptions for ORDERS Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
OrderID	INT	PK	
CustomerID	INT	FK	CUSTOMER
OrderDate	DATETIME		

Table 4.8 Column Descriptions for SHIPPING Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
ShippingID	INT	PK	
OrderID	INT	FK	ORDERS
EmployeeID	INT	FK	EMPLOYEE
ShippingDate	DATETIME		
Street	VARCHAR(50)		
City	VARCHAR(50)		
State	VARCHAR(50)		
Zip	CHAR(5)		

```
CREATE TABLE SHIPPING
(
       ShippingID INT PRIMARY KEY,
       OrderID INT NOT NULL,
       EmployeeID INT NOT NULL,
       ShippingDate DATETIME NOT NULL,
       Street VARCHAR(50) NOT NULL,
       City VARCHAR(50) NOT NULL,
       State VARCHAR(50) NOT NULL,
       Zip CHAR(5) NOT NULL,
       CONSTRAINT Shipping_OrderId_fk
                    FOREIGN KEY (OrderID)
                    REFERENCES ORDERS (OrderID),
       CONSTRAINT Shipping_EmployeeId_fk
                    FOREIGN KEY (EmployeeID)
                    REFERENCES EMPLOYEE (EmployeeID)
)
```

There are two intersection tables in the Computer_Store database, ORDER_INVENTORY and CUSTOMER_PAYMENT. These two tables contain multiple foreign key columns that are also used in a combination primary key. In Table 4.9, we have the definition of the intersection table ORDER_INVENTORY.

In the SQL statement, you will have three constraints, two for the foreign keys and one for the combination primary key. To create the table ORDER_INVENTORY with these constraints, consider the SQL statement below:

Table 4.9 Column Descriptions for ORDER_INVENTORY Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
OrderID	INT	PK, FK	ORDERS
InventoryID	INT	PK, FK	INVENTORY
Quantity	INT		

Table 4.10 Column Descriptions for CUSTOMER_PAYMENT Table

Column Name	*Data Type*	*PK or FK*	*Reference Table*
CustomerID	INT	PK, FK	CUSTOMER
PaymentID	INT	PK, FK	PAYMENT
PaymentDate	DATETIME		

```
CREATE TABLE ORDER_INVENTORY
(
       OrderID INT,
       InventoryID INT,
       Quantity INT DEFAULT 0 CHECK(Quantity >= 0) NOT NULL,
       CONSTRAINT OInventory_OrderID_fk
                           FOREIGN KEY (OrderID)
                           REFERENCES ORDERS (OrderID),
       CONSTRAINT OInventory_InventoryID_fk
                           FOREIGN KEY (InventoryID)
                           REFERENCES INVENTORY (InventoryID),
       CONSTRAINT OrderID_InventoryID_pk
                           PRIMARY KEY (OrderID, InventoryID)
)
```

For the table CUSTOMER_PAYMENT, you have the definition in Table 4.10.

Similar to the SQL statement used to create the table ORDER_INVENTORY, the keyword CONSTRAINT is used to define a combination primary key shown in the following code:

```
CREATE TABLE CUSTOMER_PAYMENT
(
             CustomerID INT,
             PaymentID INT,
             PaymentDate DATETIME NOT NULL,
             CONSTRAINT CustPayment_CustomerID_fk
                           FOREIGN KEY (CustomerID)
                           REFERENCES CUSTOMER (CustomerID),
             CONSTRAINT CustPaymant_PaymentID_fk
                           FOREIGN KEY (PaymentID)
                           REFERENCES PAYMENT (PaymentID),
             CONSTRAINT CustomerID_PaymentID_pk
                           PRIMARY KEY (CustomerID, PaymentID)
)
```

Now, all the tables are defined. With the SQL statements given in this section, you should be able to create the tables in Windows Azure SQL Database. Next, you will enter and execute the SQL statements in Windows Azure SQL Database.

ACTIVITY 4.1 CREATING DATABASE OBJECTS WITH SQL IN WINDOWS AZURE SQL DATABASE

Windows Azure SQL Database provides a query tool to run SQL statements. In the following, we will use the query tool to execute SQL statements:

1. In **Internet Explorer**, enter the following URL to the Windows Azure Management Portal: https://windows.azure.com
2. Click the **SQL DATABASES** link on the left-hand side of your screen. Then, click **NEW**.
3. Select SQL DATABASE and CUSTOM CREATE.
4. After the **Specify database settings** dialog is opened, enter the database name **Computer_Store** and select your database server as shown in Figure 4.2. Then, click the check mark to create the database.
5. On the sql databases page, select the newly created database **Computer_Store** and click **MANAGE**.
6. Enter your user name and password as shown in Figure 4.3. Then, click the **Log on** arrow to log on to the SQL Database portal.
7. Once the SQL Database portal is opened, on the toolbar, click **New Query**. As shown in Figure 4.4, the SQL Query pane is open and is ready for you to enter the SQL statements.

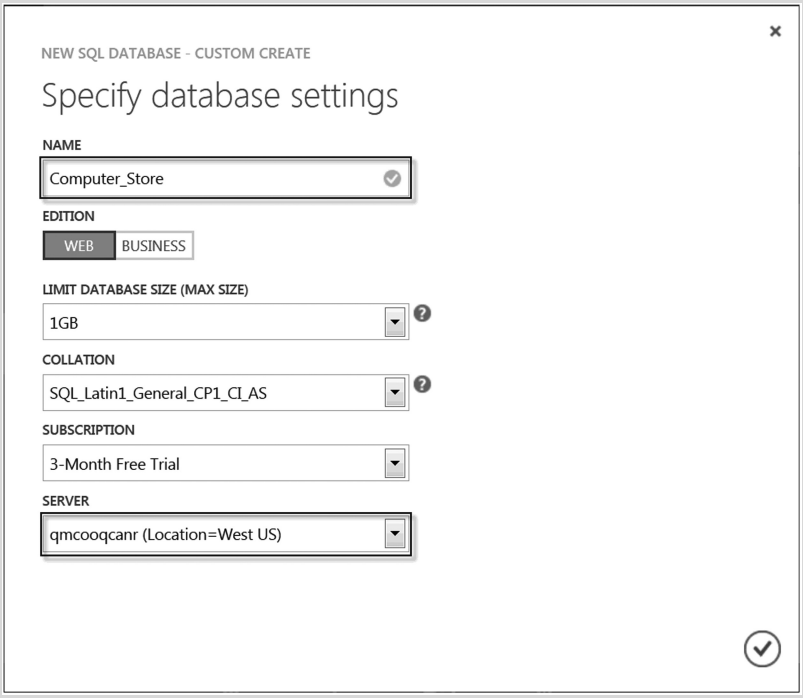

Figure 4.2 Create Computer_Store database.

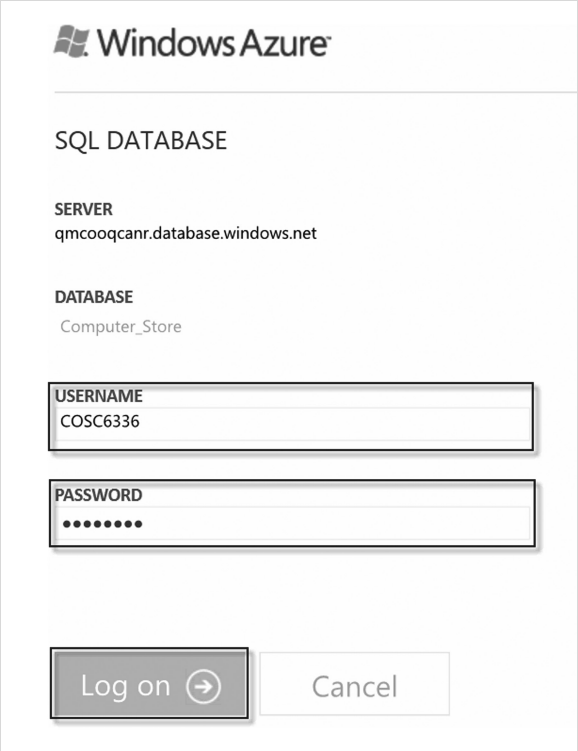

Figure 4.3 Log on to SQL database portal.

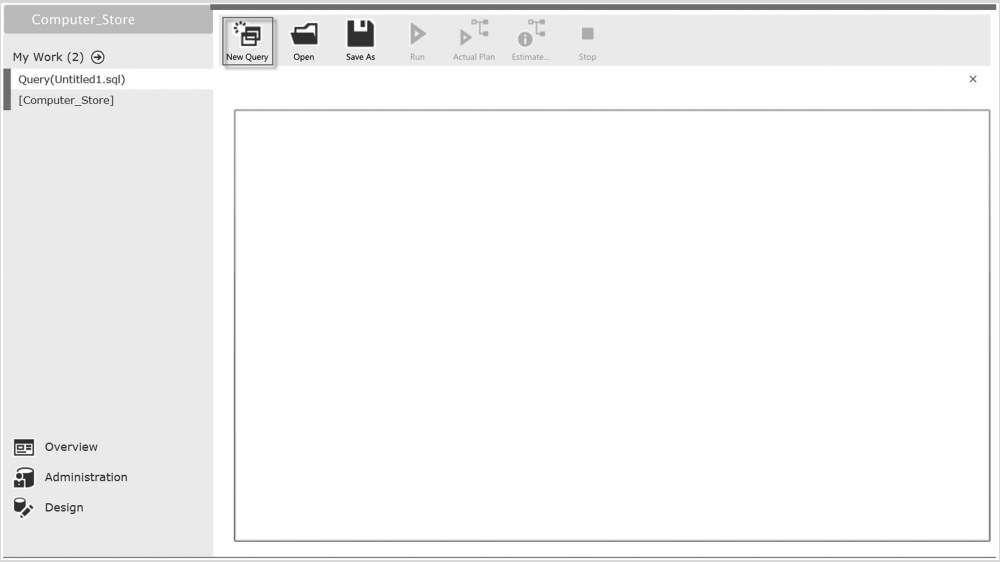

Figure 4.4 SQL query editor.

8. Let us enter the following SQL statements to create tables. After the SQL statements are entered, click **Run**.

```
— -Create tables
CREATE TABLE CUSTOMER
(
       CustomerID INT PRIMARY KEY,
       FirstName VARCHAR(30),
       LastName VARCHAR(30) NOT NULL,
       Phone CHAR(12),
       Street VARCHAR(30),
       City VARCHAR(30),
       State VARCHAR(30),
       Zip CHAR(5)
)
GO

CREATE TABLE EMPLOYEE
(
       EmployeeID INT PRIMARY KEY,
       FirstName VARCHAR(30),
       LastName VARCHAR(30) NOT NULL
)
GO

CREATE TABLE PAYMENT
(
       PaymentID INT PRIMARY KEY,
       PaymentType VARCHAR(30) NOT NULL
)
GO

CREATE TABLE PRODUCT
(
       ProductID INT PRIMARY KEY,
       ProductName VARCHAR(30),
)
GO

CREATE TABLE ITEM
(
       ItemID INT PRIMARY KEY,
       ProductID INT,
       CPU VARCHAR(30),
       RAM VARCHAR(30),
       HardDrive VARCHAR(30),
       OpticalDrive VARCHAR(30),
       Monitor VARCHAR(30),
       CONSTRAINT Item_ProductId_fk
                     FOREIGN KEY (ProductID)
                     REFERENCES PRODUCT (ProductID)
```

```
)
GO

CREATE TABLE INVENTORY
(
       InventoryID INT PRIMARY KEY,
       ItemID INT,
       Qoh INT DEFAULT 0 CHECK(Qoh >= 0) NOT NULL,
       Price MONEY DEFAULT 0 CHECK (Price >= 0) NOT NULL,
       CONSTRAINT Inventory_ItemId_fk
                    FOREIGN KEY (ItemID)
                    REFERENCES ITEM (ItemID)
)
GO

CREATE TABLE ORDERS
(
       OrderID INT PRIMARY KEY,
       CustomerID INT NOT NULL,
       OrderDate DATETIME NOT NULL,
       CONSTRAINT Orders_CustomerId_fk
              FOREIGN KEY (CustomerID)
              REFERENCES CUSTOMER (CustomerID)
)
GO

CREATE TABLE SHIPPING
(
       ShippingID INT PRIMARY KEY,
       OrderID INT NOT NULL,
       EmployeeID INT NOT NULL,
       ShippingDate DATETIME NOT NULL,
       Street VARCHAR(30) NOT NULL,
       City VARCHAR(30) NOT NULL,
       State VARCHAR(30) NOT NULL,
       Zip CHAR(5) NOT NULL,
       CONSTRAINT Shipping_OrderId_fk
                    FOREIGN KEY (OrderID)
                    REFERENCES ORDERS (OrderID),
       CONSTRAINT Shipping_EmployeeId_fk
                    FOREIGN KEY (EmployeeID)
                    REFERENCES EMPLOYEE (EmployeeID)
)
GO

CREATE TABLE ORDER_INVENTORY
(
       OrderID INT,
       InventoryID INT,
       Quantity INT DEFAULT 0 CHECK(Quantity >= 0) NOT NULL,
       CONSTRAINT OInventory_OrderID_fk
```

```
                              FOREIGN KEY (OrderID)
                              REFERENCES ORDERS (OrderID),
        CONSTRAINT OInventory_InventoryID_fk
                              FOREIGN KEY (InventoryID)
                              REFERENCES INVENTORY (InventoryID),
        CONSTRAINT OrderID_InventoryID_pk
                              PRIMARY KEY (OrderID, InventoryID)
)
GO

CREATE TABLE CUSTOMER_PAYMENT
(
        CustomerID INT,
        PaymentID INT,
        PaymentDate DATETIME NOT NULL,
        CONSTRAINT CustPayment_CustomerID_fk
                              FOREIGN KEY (CustomerID)
                              REFERENCES CUSTOMER (CustomerID),
        CONSTRAINT CustPaymant_PaymentID_fk
                              FOREIGN KEY (PaymentID)
                              REFERENCES PAYMENT (PaymentID),
        CONSTRAINT CustomerID_PaymentID_pk
                              PRIMARY KEY (CustomerID, PaymentID)
)
```

9. As shown in Figure 4.5. If everything works properly, you should get the message "Command(s) completed successfully."

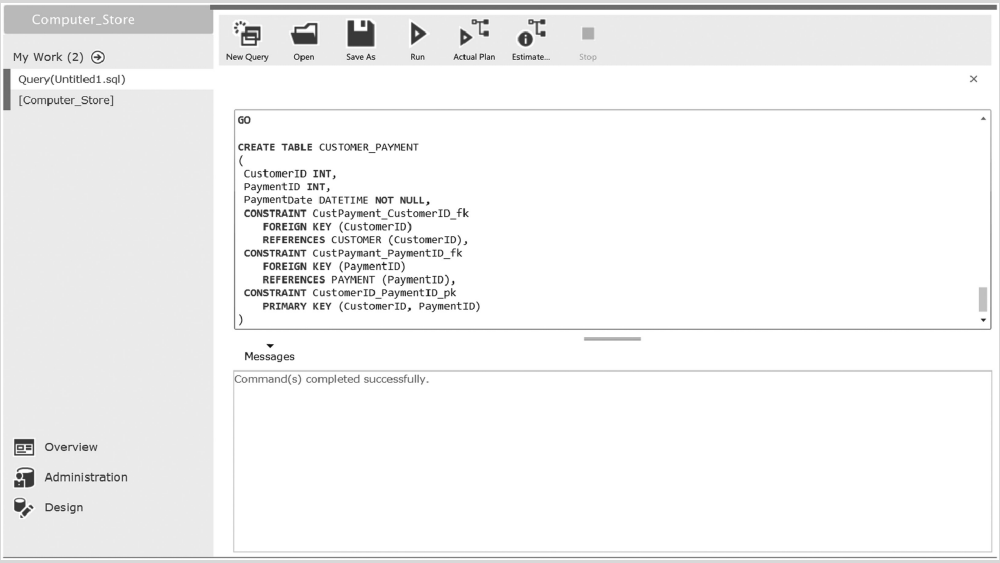

Figure 4.5 Create tables.

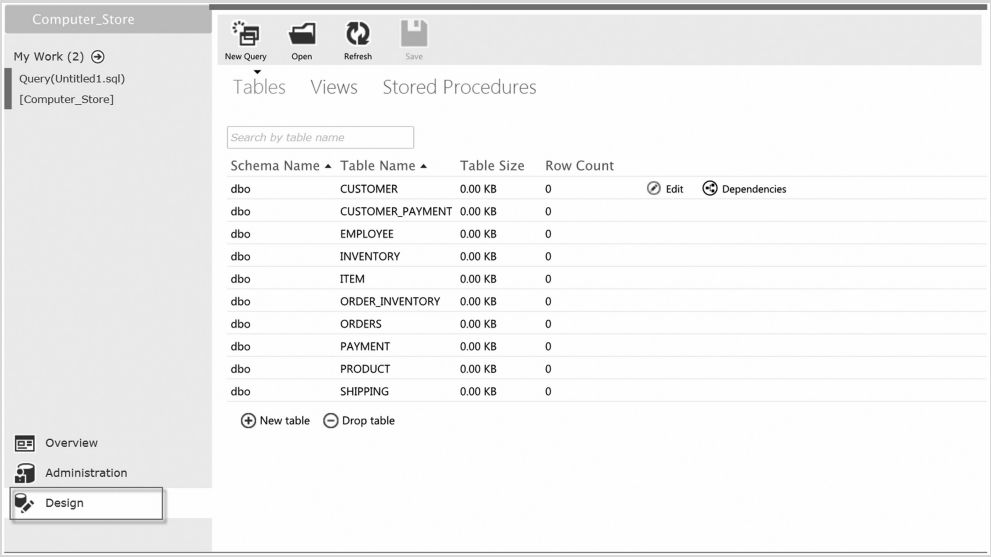

Figure 4.6 Tables created by SQL statements.

10. To verify that the tables are indeed created in the database **Computer_Store**, click the **Design** link. The created tables should be listed as shown in Figure 4.6.
11. In Figure 4.7, each table name is attached to the word **dbo** that stands for database owner and is the default schema name. A schema is a container that may hold several database objects that are owned by the same schema. It can be used to manage database object ownerships and security. In Windows Azure SQL Database, each schema may have several owners and each owner may own several schemas. You can grant a set of permissions to a schema and assign a group of users as the owners of the schema. In this way, you can avoid assigning permissions to each individual user, which is time consuming. In the next section, you will learn how to grant permissions to schemas.
12. To check the dependencies among tables, click the icon **Dependencies**. As shown in Figure 4.7, the table CUSTOMER is related to the tables ORDERS and CUSTOMER_ PAYMENT.
13. Click **Design** to switch back to the design window. Click the **Edit** icon next to the table CUSTOMER as shown in Figure 4.8.
14. Figure 4.9 shows the definitions of the columns for the CUSTOMER table. You can modify the column definitions here.

After the tables are created, SQL statements can be used to enter data in those tables. The data entry process will be covered later in this chapter.

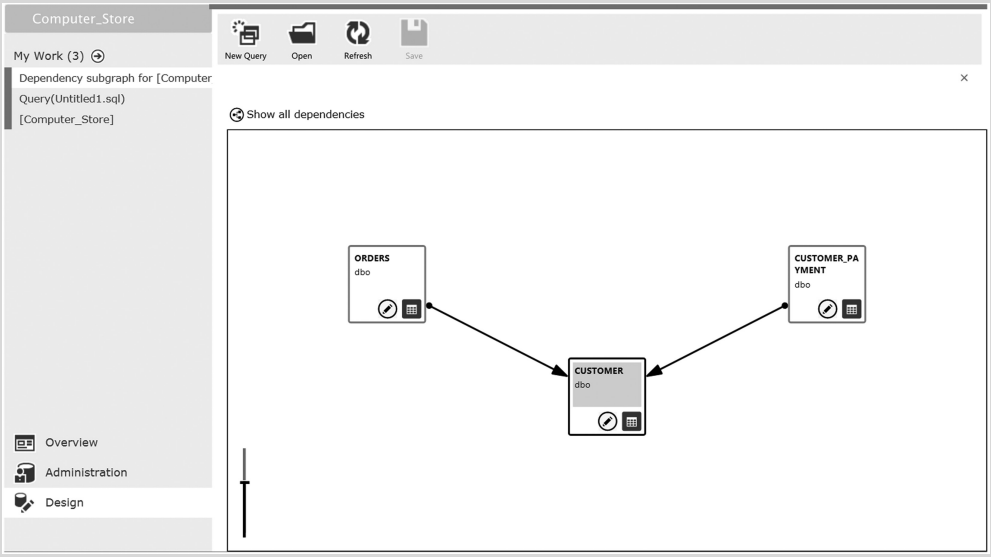

Figure 4.7 Dependencies on CUSTOMER table.

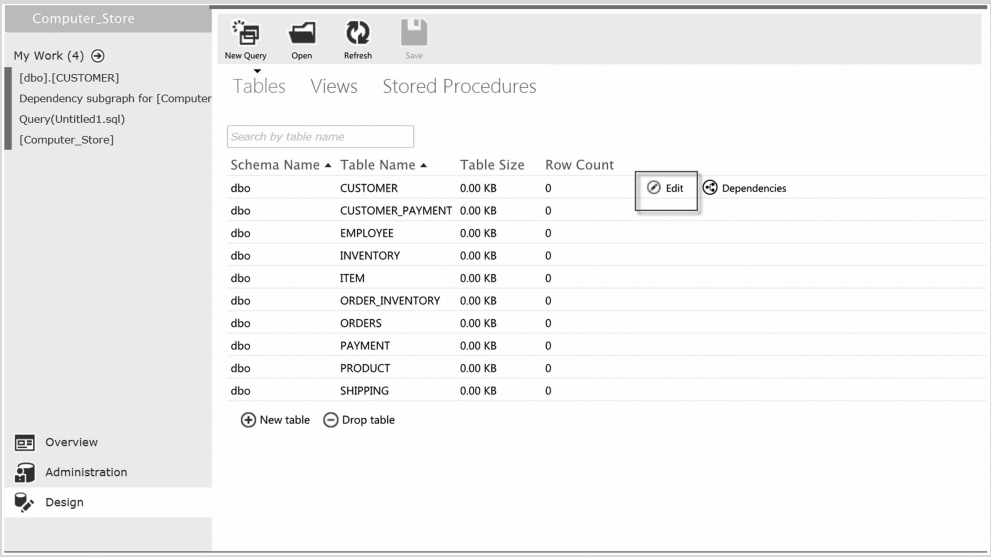

Figure 4.8 Edit tables.

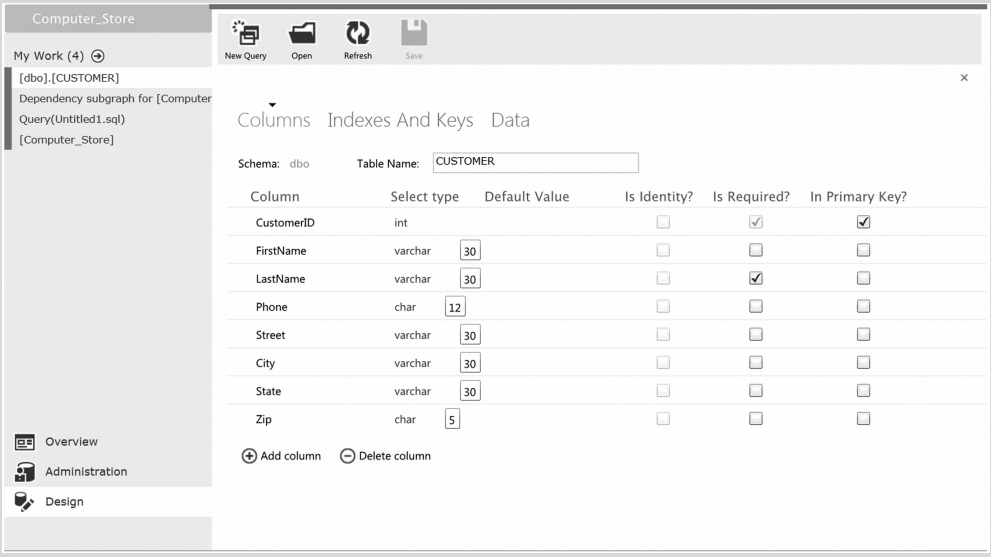

Figure 4.9 Column definitions.

4.4 Managing Database Objects

After the tables have been created, the table structures can be viewed and modified with SQL statements or the built-in stored procedures.

4.4.1 Viewing Table Structures

To view the table properties, you can use the following stored procedures and commands provided by Windows Azure SQL Database:

- sp_help: a built-in stored procedure used to view table definitions
- sp_depends: a built-in stored procedure used to view table dependencies

When running the stored procedure sp_help to view the table definition for the table PRODUCT, you should be able to see the detailed table definition as shown in Figure 4.10.

4.4.2 Modifying Table Structures

SQL statements can be used to modify the structure of database objects. You can change the data type of a column, rename a column or a table, modify constraints, add new columns, or assign permissions to schemas.

The ALTER TABLE statement is used to change the table structure. For example, if you want to add a foreign key constraint to the column EmployeeID in the table SHIPPING, you can use the following SQL statement:

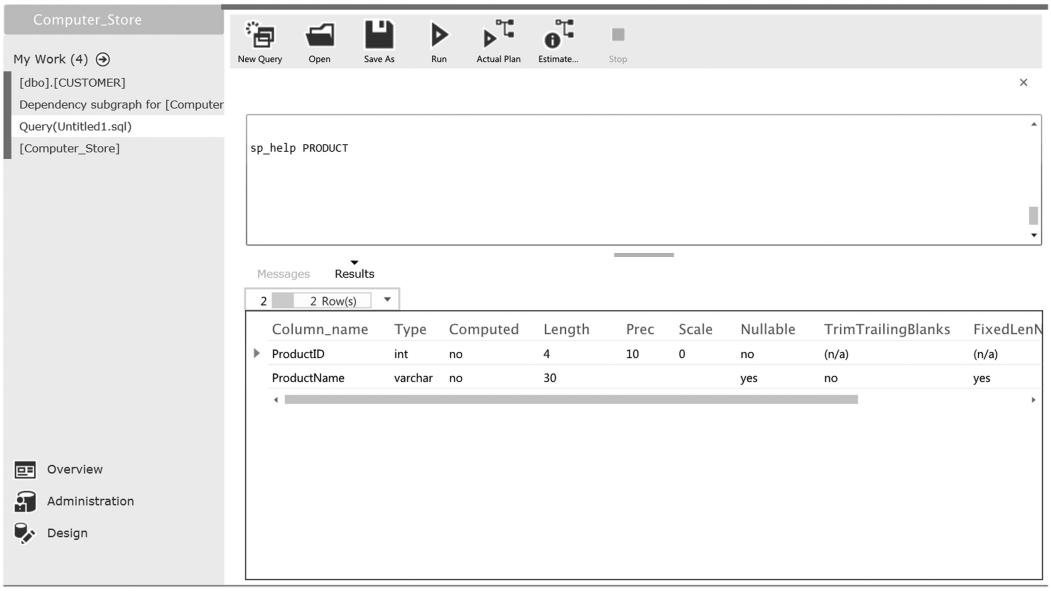

Figure 4.10 Use stored procedure to obtain table definition.

```
ALTER TABLE SHIPPING ADD
CONSTRAINT Shipping_EmployeeID_fk
       FOREIGN KEY (EmployeeID)
       REFERENCES EMPLOYEE (EmployeeID)
```

To add a check constraint to the column Qoh in the INVENTORY table, use

```
ALTER TABLE INVENTORY ADD
       CONSTRAINT Inventory_ck CHECK(Qoh >= 0)
```

With the ALTER TABLE constraint, you can also add and delete columns in a table. For example, if you want to add a column ShelfID to the table INVENTORY, use the following code:

```
ALTER TABLE INVENTORY ADD ShelfID INT
```

You can also delete the column ShelfID from the table INVENTORY with the following code:

```
ALTER TABLE INVENTORY Drop COLUMN ShelfID
```

The ALTER TABLE keyword phrase can also be used to change the data type of the column. For example, if you want to change the data type of the column Qoh from INT to BIGINT, use the following SQL statement:

```
ALTER TABLE INVENTORY ALTER COLUMN Qoh BIGINT
```

When changing the column properties, make sure that the column is not used by constraints and referenced by other columns. For example, the column Qoh in the above code has a domain

constraint Inventory_ck built on it. Simply running the above ALTER TABLE SQL statement will cause an error.

You can also modify the length of a column. As the length of a column is reduced, the data in that column will be truncated to fit in that column. This is also true for changing data types. The DROP keyword can be used to drop a table. The following SQL statement drops the table INVENTORY:

```
DROP TABLE INVENTORY
```

If a table is referenced by a foreign key column from another table, deleting the rows in the table or dropping the table will cause errors. Updating the rows in the referenced table can also cause similar problems. Windows Azure SQL Database prevents you from dropping a table referenced by another table. You must drop the referencing table first before you can drop the referenced table. Windows Azure SQL Database allows you to view the dependency of the tables. As an example, Figure 4.7 shows the dependencies on the CUSTOMER table.

4.4.3 Controlling Database Object Privileges

The database objects created by one Windows Azure SQL Database user cannot be accessed by another Windows Azure SQL Database user without permission. The database object owner can grant permissions to other users by using DCL commands. For example, to allow the public to update the table PRODUCT, use the following SQL statement:

```
GRANT UPDATE
ON PRODUCT
TO PUBLIC
```

To remove a previously granted privilege on a database object, use the DCL command REVOKE. Since it is not a good idea to grant the update permission to the public on the PRODUCT table, it needs to be removed. To do that, run the following SQL statement:

```
REVOKE UPDATE
ON PRODUCT
TO PUBLIC
```

If you log on to a database as a database administrator, you have the privilege to grant and revoke database object access privileges to other users.

ACTIVITY 4.2 DATABASE MANAGEMENT WITH SQL

In Chapter 3, you created the Class_Registration database. In this activity, you will create relationships among the tables created for the Class_Registration database according to Figure 4.11. The ALTER TABLE statement will be used to carry out the task.

1. Assume that you have logged on to the Windows Azure Management Portal. Select the database **Class_Registration** and click **MANAGE**. Then, log on to the SQL Database portal with your user name and password.

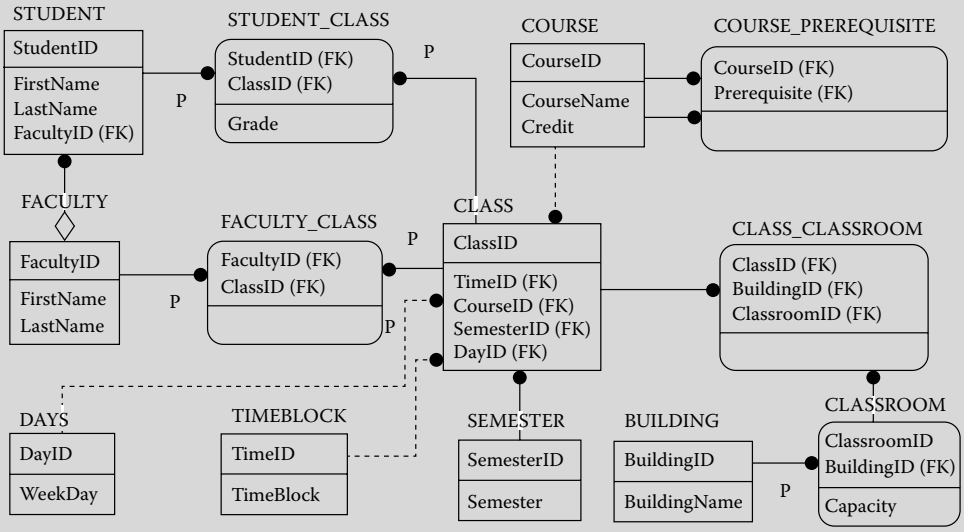

Figure 4.11 Class_Registration data model.

2. Click **New Query** to open the query tool. To create a foreign key constraint on the table STUDENT, enter the following SQL statement. Then click **Run** as shown in Figure 4.12.

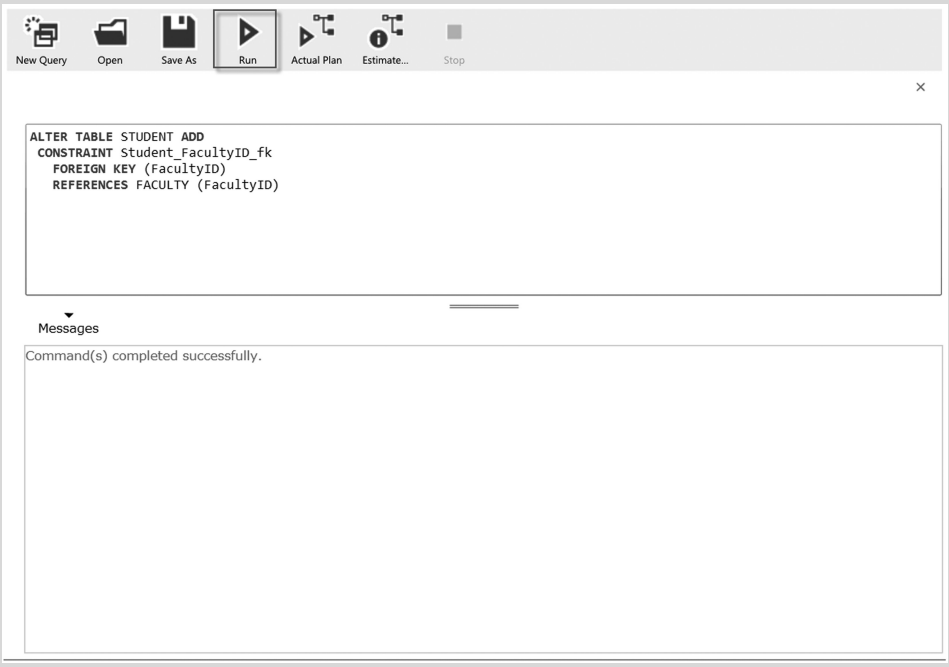

Figure 4.12 Create foreign key on STUDENT table.

```
ALTER TABLE STUDENT ADD
CONSTRAINT Student_FacultyID_fk
       FOREIGN KEY (FacultyID)
       REFERENCES FACULTY (FacultyID)
```

3. Similarly, you can enter the following SQL statements to create the rest of the foreign key constraints. Then, highlight each SQL Statement and click **Run**.

```
ALTER TABLE STUDENT_CLASS ADD CONSTRAINT SC_StudentID_fk
  FOREIGN KEY (StudentID) REFERENCES STUDENT (StudentID)

ALTER TABLE STUDENT_CLASS ADD CONSTRAINT SC_ClassID_fk
  FOREIGN KEY (ClassID) REFERENCES CLASS (ClassID)

ALTER TABLE COURSE_PREREQUISITE ADD CONSTRAINT CP_CourseID_fk
  FOREIGN KEY (CourseID) REFERENCES COURSE (CourseID)

ALTER TABLE COURSE_PREREQUISITE
  ADD CONSTRAINT CP_Prerequisite_fk
  FOREIGN KEY (Prerequisite) REFERENCES COURSE (CourseID)

ALTER TABLE FACULTY_CLASS ADD CONSTRAINT FC_FacultyID_fk
  FOREIGN KEY (FacultyID) REFERENCES FACULTY (FacultyID)

ALTER TABLE FACULTY_CLASS ADD CONSTRAINT FC_ClassID_fk
  FOREIGN KEY (ClassID) REFERENCES CLASS (ClassID)

ALTER TABLE CLASS ADD CONSTRAINT CLASS_CourseID_fk
  FOREIGN KEY (CourseID) REFERENCES COURSE (CourseID)

ALTER TABLE CLASS ADD CONSTRAINT CLASS_DayID_fk
  FOREIGN KEY (DayID) REFERENCES DAYS (DayID)

ALTER TABLE CLASS ADD CONSTRAINT CLASS_TimeID_fk
  FOREIGN KEY (TimeID) REFERENCES TIMEBLOCK (TimeID)

ALTER TABLE CLASS ADD CONSTRAINT CLASS_SemesterID_fk
  FOREIGN KEY (SemesterID) REFERENCES SEMESTER (SemesterID)

ALTER TABLE CLASSROOM ADD CONSTRAINT CLASS_BuildingID_fk
  FOREIGN KEY (BuildingID) REFERENCES BUILDING (BuildingID)

ALTER TABLE CLASS_CLASSROOM ADD CONSTRAINT CC_ClassID_fk
  FOREIGN KEY (ClassID) REFERENCES CLASS (ClassID)

ALTER TABLE CLASS_CLASSROOM ADD CONSTRAINT CC_ClassroomID_fk
  FOREIGN KEY (ClassroomID, BuildingID)
  REFERENCES CLASSROOM (ClassroomID, BuildingID))
```

4. To view a newly created constraint, click **Design** on the left-hand side of the screen. Select the table **CLASS_CLASSROOM** and then click **Edit**. Click **Indexes And Keys**, and you should be able to see the foreign keys as shown in Figure 4.13.

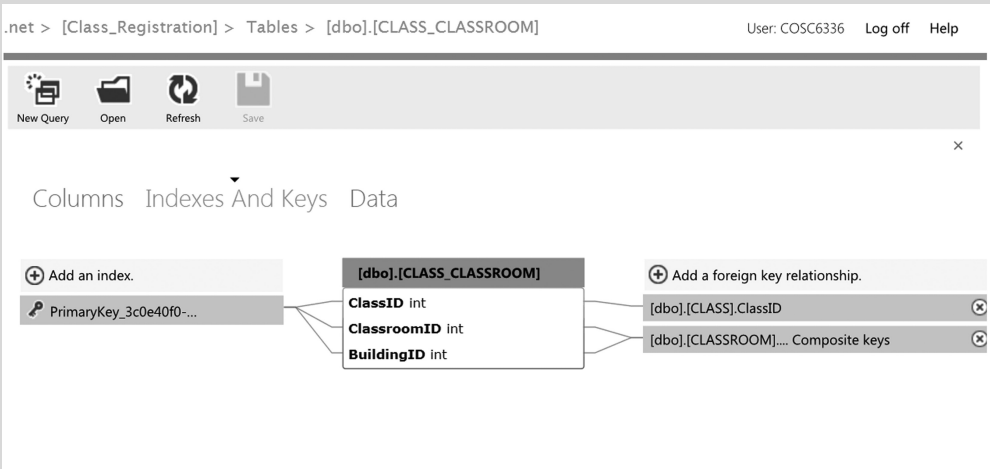

Figure 4.13 Foreign key relationships.

4.5 Manipulating Data in Tables

You can manipulate the data in an existing table with DML commands. You can insert a row of data with the INSERT command, modify data with the UPDATE command, delete data with the DELETE command, and query data with the SELECT command. The following examples show you how to use these commands to manipulate data in a table.

4.5.1 Inserting Data

To insert a new row of data in the table EMPLOYEE, use the following SQL statement:

```
INSERT INTO EMPLOYEE
VALUES(1, 'Liz', 'Chen')
```

As seen from the above SQL statement, the order and data types of the data values listed in the VALUES (. . .) statement exactly match the column definitions. For example, the data value of the first column is an integer; the first name and last name data values are character strings that should be quoted with single quotes.

4.5.2 Modifying Data

In this example, let us run the following SQL statement that will change the last name Chen to Dean:

```
UPDATE EMPLOYEE
SET LastName = 'Dean'
WHERE LastName = 'Chen'
```

In the above code, WHERE specifies the value to be updated. Without the WHERE clause, all the values in the same column will be updated. The SET keyword is used to set a new value.

4.5.3 Querying Data

The following example shows how to query the information about the employees with the last name Smith:

```
SELECT FirstName, LastName
FROM EMPLOYEE
WHERE LastName = 'Smith'
```

4.5.4 Deleting Data

In the following example, you will delete the row where the employee's last name is Chen:

```
DELETE FROM EMPLOYEE
WHERE LastName = 'Chen'
```

Again, the WHERE clause is necessary to specify which row to be deleted. Otherwise, all rows will be deleted.

ACTIVITY 4.3 POPULATING DATABASE TABLES WITH DATA

After the tables are created, it is time to populate the tables with data. In this section, we will insert data into the tables in the database Computer_Store as well as the tables in the database Class_Registration.

When entering data into the tables, you should first enter the data into the tables without foreign key columns. After the data are entered into the tables without foreign keys, you can populate the tables with foreign key columns. In such a way, the data values entered into foreign key columns can be verified by the foreign key constraint to see if there is a match with the primary keys in the parent tables. If there is no match, you will get an error message.

TASK 1: POPULATE COMPUTER_STORE DATABASE

Follow the steps below to populate the Computer_Store database with SQL statements in Windows Azure SQL Database:

1. Assume that you have logged on to the Windows Azure Management Portal. Select **SQL DATABASES** and the database **Computer_Store**. Click **MANAGE** to log on to the SQL Database portal shown in Figure 4.14.
2. Once the SQL Database portal is opened, click **New Query**. Enter the following SQL statements in the New Query pane. Then, click **Run**.

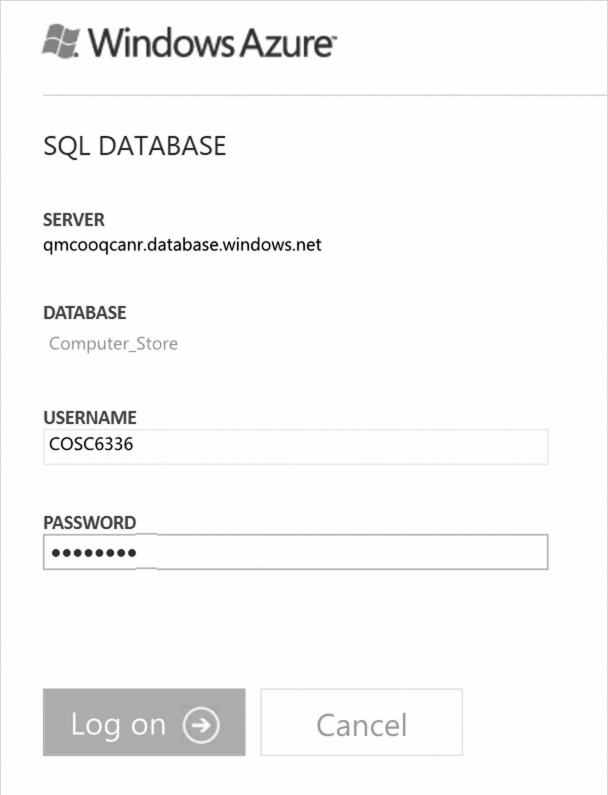

Figure 4.14 Log on to SQL database portal to manage Computer_Store.

```
— - inserting records into table CUSTOMER
INSERT INTO CUSTOMER
VALUES(1,'John','Burge','972-456-5555','214
Nelson','Dallas','TX','75201')

INSERT INTO CUSTOMER
VALUES(2,'Keren','Wu','281-333-1111','345
Bagby','Houston','TX','77002')

INSERT INTO CUSTOMER
VALUES(3,'Dan','Parr','361-111-2222','324
Miori','Victoria','TX','77903')

INSERT INTO CUSTOMER
VALUES(4,'Lisa','Garcia','806-333-1111','874 Wilson','Amorillo',
'TX','79105')

INSERT INTO CUSTOMER
VALUES(5,'Susan','Holly','512-111-2222','4848 Lakeside','Austin',
'TX','78767')
```

```
INSERT INTO CUSTOMER
VALUES(6,'Al','Dean','972-111-3333','1398 International
Rd','Dallas','TX','75201')

INSERT INTO CUSTOMER
VALUES(7,'David','Jouns','281-000-1111','267 College','Houston',
'TX','77487')

INSERT INTO CUSTOMER
VALUES(8,'Bud','Fry','512-234-0000','2396 Anderson','Austin',
'TX','78767')

INSERT INTO CUSTOMER
VALUES(9,'Jane','Young','212-111-4444','1563 Madison','New
York','NY','10159')

INSERT INTO CUSTOMER
VALUES(10,'Robert','Smith','334-555-1111','2995 Post','Montgamery',
'AL','35824')

INSERT INTO CUSTOMER
VALUES(11,'Roy','Munoz','303-333-0000','201 Bellaire','Denver',
'CO','80222')

INSERT INTO CUSTOMER
VALUES(12,'Paul','Sitka','650-555-3333','1240 Villa','Mountain
View','CA','94041')

INSERT INTO CUSTOMER
VALUES(13,'Gary','Sherman','508-333-7777','145 Corporate',
'Cambridge','MA','02142')

INSERT INTO CUSTOMER
VALUES(14,'Larry','Hursh','281-000-1234','1248 Katy
Frwy','Houston','TX','77429')

INSERT INTO CUSTOMER
VALUES(15,'David','Reed','281-444-9999','579 S.
Mason','Houston','TX','77290')

INSERT INTO CUSTOMER
VALUES(16,'John','Shum','281-123-2345','389
Brooks','Houston','TX','76385')

INSERT INTO CUSTOMER
VALUES(17,'Joe','Brand','281-987-1111','2970 Hwy
6','Houston','TX','77638')

INSERT INTO CUSTOMER
VALUES(18,'Eric','Revis','281-567-7890','9108
Travis','Houston','TX','76325')

INSERT INTO CUSTOMER
VALUES(19,'Lilla','White','281-999-1234','1590 Memorial','Houston',
'TX','77389')

INSERT INTO CUSTOMER
VALUES(20,'Frank','Dees','281-888-2345','337 First
St.','Houston','TX','78330')
```

```
INSERT INTO CUSTOMER
VALUES(21,'Aian','Ramos','281-777-1111','466
Sugar','Houston','TX','77449')

INSERT INTO CUSTOMER
VALUES(22,'Mike','Garcia','281-000-2222','1400 Richmond','Houston',
'TX','77562')

INSERT INTO CUSTOMER
VALUES(23,'Dana','Kersh','281-666-7777','8879 Telephone
Rd.','Houston','TX','77541')

INSERT INTO CUSTOMER
VALUES(24,'Jeff','Rose','281-555-1234','735 Tomball','Houston',
'TX','77339')

INSERT INTO CUSTOMER
VALUES(25,'Sam','Parker','281-444-1234','6839 Antoine','Houston',
'TX','77456')

INSERT INTO CUSTOMER
VALUES(26,'Helen','Wood','281-411-1111','179
Greens','Houston','TX','77894')

INSERT INTO CUSTOMER
VALUES(27,'Wanda','Burton','281-012-0123','4221 Gulf
Frwy','Houston','TX','77903')

INSERT INTO CUSTOMER
VALUES(28,'Steve','Perry','281-456-0000','827 Louetta','Houston',
'TX','76450')

INSERT INTO CUSTOMER
VALUES(29,'Jay','Jackson','281-000-1000','1849 Westheimer',
'Houston','TX','76889')

INSERT INTO CUSTOMER
VALUES(30,'Toya','Vela','281-777-2222','163
Fugua','Houston','TX','77398')


— - inserting records into table EMPLOYEE
INSERT INTO EMPLOYEE
VALUES(1, 'Liz', 'Chen')

INSERT INTO EMPLOYEE
VALUES(2, 'Jan','Dean')

INSERT INTO EMPLOYEE
VALUES(3, 'Don','Fry')

INSERT INTO EMPLOYEE
VALUES(4, 'Linda', 'Green')

INSERT INTO EMPLOYEE
VALUES(5, 'Mark','Smith')
```

```
INSERT INTO EMPLOYEE
VALUES(6, 'Mary','Smith')

INSERT INTO EMPLOYEE
VALUES(7, 'Joe','Wyer')

INSERT INTO EMPLOYEE
VALUES(8, 'Bruce','Young')

— - inserting records into table PAYMENT
INSERT INTO PAYMENT
VALUES(1,'Check')

INSERT INTO PAYMENT
VALUES(2,'Master')

INSERT INTO PAYMENT
VALUES(3,'VISA')

INSERT INTO PAYMENT
VALUES(4,'American Express')

INSERT INTO PAYMENT
VALUES(5,'Discover')

— - inserting records into table PRODUCT
INSERT INTO PRODUCT
VALUES(1,'DeskTop')

INSERT INTO PRODUCT
VALUES(2,'Notebook')

INSERT INTO PRODUCT
VALUES(3,'Tablet')

INSERT INTO PRODUCT
VALUES(4,'PDA')

— - inserting records into table ITEM
INSERT INTO ITEM
VALUES(1,3,'PM 2.2 GHz','512MB','60GB','DVD-ROM/CD-RW','14.1”
swivel')

INSERT INTO ITEM
VALUES(2,2,'CM 2.0 GHz','256MB','40GB','DVD-ROM/CD-RW','15” XGA TFT')

INSERT INTO ITEM
VALUES(3,2,'PM 3.0 GHz','1024MB','100GB','DVD-RAM','15.4” WXGA TFT')

INSERT INTO ITEM
VALUES(4,2,'PM 3.6 GHz','512MB','40GB','DVD SuperMulti','7.2” WXGA
TFT')

INSERT INTO ITEM
VALUES(5,2,'PM 3.6GHZ','512MB','60GB','DVD SuperMulti','12.1” XGA')

INSERT INTO ITEM
VALUES(6,2,'SD 1.5 GHz','512MB','80GB','Bluetooth 2.0','12.1” XGA')
```

```
INSERT INTO ITEM
VALUES(7,1,'SD 1.25 GHz','256MB','40GB','DVD Combo','')

INSERT INTO ITEM
VALUES(8,2,'SD 1.5 GHz','512MB','80GB','Bluetooth 2.0','15" XGA')

INSERT INTO ITEM
VALUES(9,1,'SD Dual 2.3 GHz','512MB','250GB','16x SuperDrive DVD','')

INSERT INTO ITEM
VALUES(10,2,'AA 2.2 GHz','512MB','80GB','DVD+-RW','15.4" WXGA')

INSERT INTO ITEM
VALUES(11,1,'AA 3.2 GHz','1GB','200GB','DVD+-RW','')

INSERT INTO ITEM
VALUES(12,1,'CD 2.0 GHz','512MB','100GB','DVD-ROM/CD-RW','')

INSERT INTO ITEM
VALUES(13,4,'SS 300 MHz','56MB','','','3.5" Transflective TFT')

INSERT INTO ITEM
VALUES(14,4,'SS 400 MHz','152MB','','','3.5" Transflective TFT')

INSERT INTO ITEM
VALUES(15,1,'AA 2.6 GHz','512MB','160GB','DVD+-RW','')

INSERT INTO ITEM
VALUES(16,1,'CD 3.4 GHz','512MB','100GB','DVD-ROM/CD-RW','')

INSERT INTO ITEM
VALUES(17,1,'AA 3.2 GHz','1GB','250GB','DVD+-RW','')

INSERT INTO ITEM
VALUES(18,2,'CM 1.40 GHz','256MB','40GB','CD-RW','14.1" XGA TFT')

INSERT INTO ITEM
VALUES(19,2,'PM 1.6 GHz','256MB','40GB','DVD-ROM/CD-RW','14.1" XGA
TFT')

INSERT INTO ITEM
VALUES(20,2,'PM 1.50 GHz','512MB','40GB','DVD-ROM/CD-RW','12.1" XGA
TFT')

— - inserting records into table INVENTORY
INSERT INTO INVENTORY
VALUES(1,1,5,1500)

INSERT INTO INVENTORY
VALUES(2,1,500,1400)

INSERT INTO INVENTORY
VALUES(3,2,2,799)

INSERT INTO INVENTORY
VALUES(4,2,454,749)

INSERT INTO INVENTORY
VALUES(5,3,12,1799)
```

```
INSERT INTO INVENTORY
VALUES(6,3,200,1699)

INSERT INTO INVENTORY
VALUES(7,4,58,1824)

INSERT INTO INVENTORY
VALUES(8,5,56,1999)

INSERT INTO INVENTORY
VALUES(9,6,23,1699)

INSERT INTO INVENTORY
VALUES(10,7,455,499)

INSERT INTO INVENTORY
VALUES(11,8,79,1999)

INSERT INTO INVENTORY
VALUES(12,9,51,2499)

INSERT INTO INVENTORY
VALUES(13,10,222,1040)

INSERT INTO INVENTORY
VALUES(14,11,344,649)

INSERT INTO INVENTORY
VALUES(15,12,450,449)

INSERT INTO INVENTORY
VALUES(16,13,899,299)

INSERT INTO INVENTORY
VALUES(17,14,682,449)

INSERT INTO INVENTORY
VALUES(18,15,739,649)

INSERT INTO INVENTORY
VALUES(19,16,145,409)

INSERT INTO INVENTORY
VALUES(20,17,500,654)

INSERT INTO INVENTORY
VALUES(21,18,130,599)

INSERT INTO INVENTORY
VALUES(22,19,5,1350)

INSERT INTO INVENTORY
VALUES(23,19,100,1300)


— - inserting records into ORDERS
INSERT INTO ORDERS
VALUES(1,1,'2012-02-01')
```

```
INSERT INTO ORDERS
VALUES(2,2,'2012-05-01')

INSERT INTO ORDERS
VALUES(3,3,'2011-02-01')

INSERT INTO ORDERS
VALUES(4,4,'2011-03-01')

INSERT INTO ORDERS
VALUES(5,5,'2011-03-01')

INSERT INTO ORDERS
VALUES(6,6,'2011-04-01')

INSERT INTO ORDERS
VALUES(7,7,'2011-04-01')

INSERT INTO ORDERS
VALUES(8,8,'2012-05-01')

INSERT INTO ORDERS
VALUES(9,9,'2011-05-01')

INSERT INTO ORDERS
VALUES(10,10,'2011-05-01')

INSERT INTO ORDERS
VALUES(11,11,'2011-06-01')

INSERT INTO ORDERS
VALUES(12,12,'2011-06-01')

INSERT INTO ORDERS
VALUES(13,13,'2012-07-01')

INSERT INTO ORDERS
VALUES(14,14,'2012-07-01')

INSERT INTO ORDERS
VALUES(15,15,'2013-08-01')

INSERT INTO ORDERS
VALUES(16,16,'2012-08-01')

INSERT INTO ORDERS
VALUES(17,17,'2011-08-01')

INSERT INTO ORDERS
VALUES(18,18,'2011-09-01')

INSERT INTO ORDERS
VALUES(19,19,'2011-09-01')

INSERT INTO ORDERS
VALUES(20,20,'2012-09-01')

INSERT INTO ORDERS
VALUES(21,21,'2012-10-01')
```

```
INSERT INTO ORDERS
VALUES(22,22,'2011-10-01')

INSERT INTO ORDERS
VALUES(23,23,'2010-11-01')

INSERT INTO ORDERS
VALUES(24,24,'2010-11-01')

INSERT INTO ORDERS
VALUES(25,25,'2010-12-01')

INSERT INTO ORDERS
VALUES(26,26,'2012-12-01')

INSERT INTO ORDERS
VALUES(27,27,'2013-09-11')

INSERT INTO ORDERS
VALUES(28,28,'2009-09-21')

INSERT INTO ORDERS
VALUES(29,29,'2008-10-01')

INSERT INTO ORDERS
VALUES(30,30,'2008-10-06')

INSERT INTO ORDERS
VALUES(31,1,'2008-10-31')

INSERT INTO ORDERS
VALUES(32,2,'2008-11-01')

INSERT INTO ORDERS
VALUES(33,11,'2008-11-07')

INSERT INTO ORDERS
VALUES(34,12,'2008-11-21')

INSERT INTO ORDERS
VALUES(35,13,'2008-12-17')

INSERT INTO ORDERS
VALUES(36,14,'2008-12-21')

— - inserting records into SHIPPING
INSERT INTO SHIPPING
VALUES(1,1,1,'2008-02-01','214 Nelson','Dallas','TX','75201')

INSERT INTO SHIPPING
VALUES(2,2,2,'2008-05-01','345 Bagby','Houston','TX','77002')

INSERT INTO SHIPPING
VALUES(3,3,3,'2008-02-12','324 Miori','Victoria','TX','77903')

INSERT INTO SHIPPING
VALUES(4,4,4,'2008-03-01','874 Wilson','Amorillo','TX','79105')
```

```
INSERT INTO SHIPPING
VALUES(5,5,5,'2008-03-11','4848 Lakeside','Austin','TX','78767')

INSERT INTO SHIPPING
VALUES(6,6,6,'2008-04-01','1398 International
Rd','Dallas','TX','75201')

INSERT INTO SHIPPING
VALUES(7,7,7,'2008-04-11','267 College','Houston','TX','77487')

INSERT INTO SHIPPING
VALUES(8,8,8,'2008-04-15','2396 Anderson','Austin','TX','78767')

INSERT INTO SHIPPING
VALUES(9,9,1,'2008-05-01','1563 Madison','New York','NY','10159')

INSERT INTO SHIPPING
VALUES(10,10,2,'2008-05-11','2995 Post','Huntsville','AL','35824')

INSERT INTO SHIPPING
VALUES(11,11,3,'2008-05-21','201 Bellaire','Denver','CO','80222')

INSERT INTO SHIPPING
VALUES(12,12,4,'2008-05-31','1240 Villa','Mountain
View','CA','94041')

INSERT INTO SHIPPING
VALUES(13,13,5,'2008-06-01','145 Corporate','Cambridge','MA','02142')

INSERT INTO SHIPPING
VALUES(14,14,6,'2008-06-11','1248 Katy Frwy','Houston','TX','77429')

INSERT INTO SHIPPING
VALUES(15,15,7,'2008-06-23','579 S. Mason','Houston','TX','77290')

INSERT INTO SHIPPING
VALUES(16,16,8,'2008-06-25','389 Brooks','Houston','TX','76385')

INSERT INTO SHIPPING
VALUES(17,17,1,'2008-07-01','2970 Hwy 6','Houston','TX','77638')

INSERT INTO SHIPPING
VALUES(18,18,2,'2008-07-11','9108 Travis','Houston','TX','76325')

INSERT INTO SHIPPING
VALUES(19,19,3,'2008-07-21','1590 Memorial','Houston','TX','77389')

INSERT INTO SHIPPING
VALUES(20,20,4,'2008-07-31','337 First St.','Houston','TX','78330')

INSERT INTO SHIPPING
VALUES(21,21,5,'2008-08-01','466 Sugar','Houston','TX','77449')

INSERT INTO SHIPPING
VALUES(22,22,6,'2008-08-05','1400 Richmond','Houston','TX','77562')

INSERT INTO SHIPPING
VALUES(23,23,7,'2008-08-09','8879 Telephone
Rd.','Houston','TX','77541')
```

```
INSERT INTO SHIPPING
VALUES(24,24,8,'2008-08-21','735 Tomball','Houston','TX','77339')

INSERT INTO SHIPPING
VALUES(25,25,1,'2008-09-02','6839 Antoine','Houston','TX','77456')

INSERT INTO SHIPPING
VALUES(26,26,2,'2008-09-08','179 Greens','Houston','TX','77894')

INSERT INTO SHIPPING
VALUES(27,27,3,'2008-09-11','4221 Gulf Frwy','Houston','TX','77903')

INSERT INTO SHIPPING
VALUES(28,28,4,'2009-09-21','827 Louetta','Houston','TX','76450')

INSERT INTO SHIPPING
VALUES(29,29,5,'2008-10-01','1849 Westheimer','Houston','TX','76889')

INSERT INTO SHIPPING
VALUES(30,30,6,'2008-10-06','163 Fugua','Houston','TX','77398')

INSERT INTO SHIPPING
VALUES(31,31,7,'2008-10-31','2995 Post','Huntsville','AL','35824')

INSERT INTO SHIPPING
VALUES(32,32,8,'2008-11-01','201 Bellaire','Denver','CO','80222')

INSERT INTO SHIPPING
VALUES(33,33,1,'2008-11-07','1240 Villa','Mountain
View','CA','94041')

INSERT INTO SHIPPING
VALUES(34,34,2,'2008-11-21','145 Corporate','Cambridge','MA','02142')

INSERT INTO SHIPPING
VALUES(35,35,3,'2008-12-17','145 Corporate','Cambridge','MA','02142')

INSERT INTO SHIPPING
VALUES(36,36,4,'2008-12-21','1248 Katy Frwy','Houston','TX','77429')

INSERT INTO SHIPPING
VALUES(37,33,1,'2008-11-07','1240 Villa','Mountain
View','CA','94041')

INSERT INTO SHIPPING
VALUES(38,34,2,'2008-11-21','145 Corporate','Cambridge','MA','02142')

INSERT INTO SHIPPING
VALUES(39,35,3,'2008-12-17','145 Corporate','Cambridge','MA','02142')

INSERT INTO SHIPPING
VALUES(40,36,4,'2008-12-21','1248 Katy Frwy','Houston','TX','77429')


— - inserting records into ORDER_INVENTORY
INSERT INTO ORDER_INVENTORY
VALUES(1,1,2)
```

```
INSERT INTO ORDER_INVENTORY
VALUES(2,2,2)

INSERT INTO ORDER_INVENTORY
VALUES(3,3,1)

INSERT INTO ORDER_INVENTORY
VALUES(4,4,1)

INSERT INTO ORDER_INVENTORY
VALUES(5,5,1)

INSERT INTO ORDER_INVENTORY
VALUES(6,6,1)

INSERT INTO ORDER_INVENTORY
VALUES(7,7,1)

INSERT INTO ORDER_INVENTORY
VALUES(8,8,1)

INSERT INTO ORDER_INVENTORY
VALUES(9,9,2)

INSERT INTO ORDER_INVENTORY
VALUES(10,10,1)

INSERT INTO ORDER_INVENTORY
VALUES(11,11,2)

INSERT INTO ORDER_INVENTORY
VALUES(12,12,2)

INSERT INTO ORDER_INVENTORY
VALUES(13,13,2)

INSERT INTO ORDER_INVENTORY
VALUES(14,14,2)

INSERT INTO ORDER_INVENTORY
VALUES(15,15,1)

INSERT INTO ORDER_INVENTORY
VALUES(16,16,1)

INSERT INTO ORDER_INVENTORY
VALUES(17,17,1)

INSERT INTO ORDER_INVENTORY
VALUES(18,18,2)

INSERT INTO ORDER_INVENTORY
VALUES(19,19,1)

INSERT INTO ORDER_INVENTORY
VALUES(20,20,1)
```

```
INSERT INTO ORDER_INVENTORY
VALUES(21,21,3)

INSERT INTO ORDER_INVENTORY
VALUES(22,22,1)

INSERT INTO ORDER_INVENTORY
VALUES(23,23,1)

INSERT INTO ORDER_INVENTORY
VALUES(24,1,1)

INSERT INTO ORDER_INVENTORY
VALUES(25,2,1)

INSERT INTO ORDER_INVENTORY
VALUES(26,6,1)

INSERT INTO ORDER_INVENTORY
VALUES(27,7,1)

INSERT INTO ORDER_INVENTORY
VALUES(28,8,1)

INSERT INTO ORDER_INVENTORY
VALUES(29,9,2)

INSERT INTO ORDER_INVENTORY
VALUES(30,10,1)

INSERT INTO ORDER_INVENTORY
VALUES(30,3,1)

INSERT INTO ORDER_INVENTORY
VALUES(31,2,1)

INSERT INTO ORDER_INVENTORY
VALUES(32,4,1)

INSERT INTO ORDER_INVENTORY
VALUES(32,11,1)

INSERT INTO ORDER_INVENTORY
VALUES(33,12,1)

INSERT INTO ORDER_INVENTORY
VALUES(34,12,1)

INSERT INTO ORDER_INVENTORY
VALUES(33,13,1)

INSERT INTO ORDER_INVENTORY
VALUES(35,8,1)

INSERT INTO ORDER_INVENTORY
VALUES(11,14,2)
```

```
INSERT INTO ORDER_INVENTORY
VALUES(12,11,1)

INSERT INTO ORDER_INVENTORY
VALUES(13,12,2)

INSERT INTO ORDER_INVENTORY
VALUES(14,13,1)

INSERT INTO ORDER_INVENTORY
VALUES(36,14,1)


— - inserting records into CUSTOMER_PAYMENT
INSERT INTO CUSTOMER_PAYMENT
VALUES(1,1,'2012-02-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(2,2,'2012-05-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(3,3,'2011-02-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(4,4,'2011-03-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(5,5,'2011-03-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(6,1,'2011-04-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(7,2,'2011-04-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(8,3,'2012-04-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(9,4,'2012-05-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(10,2,'2012-05-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(11,3,'2011-05-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(12,2,'2011-05-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(13,3,'2011-06-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(14,4,'2011-06-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(15,5,'2011-06-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(16,2,'2012-06-01')
```

```
INSERT INTO CUSTOMER_PAYMENT
VALUES(17,3,'2012-07-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(18,2,'2012-07-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(19,3,'2012-07-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(20,2,'2012-07-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(21,2,'2013-08-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(22,2,'2012-08-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(23,2,'2011-08-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(24,2,'2011-08-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(25,2,'2011-09-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(26,2,'2011-09-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(27,2,'2012-09-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(28,2,'2012-09-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(29,2,'2012-10-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(30,3,'2012-10-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(1,3,'2011-10-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(2,3,'2010-11-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(3,2,'2010-11-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(4,3,'2010-11-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(5,3,'2010-12-01')

INSERT INTO CUSTOMER_PAYMENT
VALUES(6,3,'2010-12-01')
```

3. As shown in Figure 4.15, rows are added to the tables in the Computer_Store database.

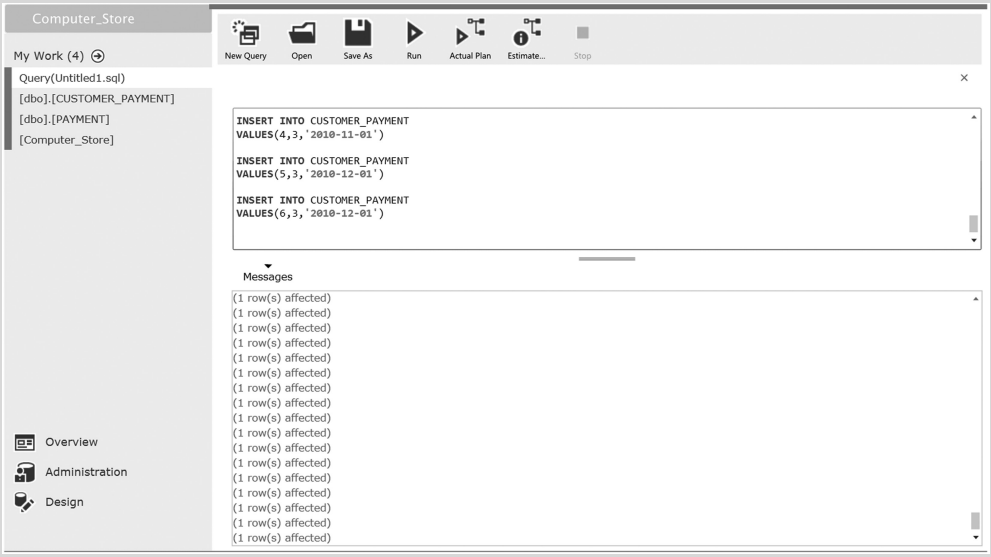

Figure 4.15 Populate Computer_Store database with SQL statements.

TASK 2: POPULATE CLASS_REGISTRATION DATABASE

Follow the steps below to populate the Class_Registration database with SQL statements in Windows Azure SQL Database:

1. Assume that you have logged on to the Windows Azure Management Portal. Select **SQL DATABASES** and the **Class_Registration** database. Click **MANAGE** to log on to the SQL Database portal as shown in Figure 4.16.
2. Once the SQL Database portal is opened, click **New Query**. Enter the following SQL statements in the New Query pane. Then, click **Run**.

```
— - inserting records into BUILDING
INSERT INTO BUILDING (BuildingId, BuildingName)
VALUES(1, 'East')

INSERT INTO BUILDING (BuildingId, BuildingName)
VALUES(2, 'West')

INSERT INTO BUILDING (BuildingId, BuildingName)
VALUES(3, 'South')

INSERT INTO BUILDING (BuildingId, BuildingName)
VALUES(4, 'North')

— - inserting records into FACULTY
INSERT INTO FACULTY (FacultyID, FirstName, LastName)
VALUES(1, 'Fred', 'Smith')

INSERT INTO FACULTY (FacultyID, FirstName, LastName)
VALUES(2, 'Chris', 'Lee')
```

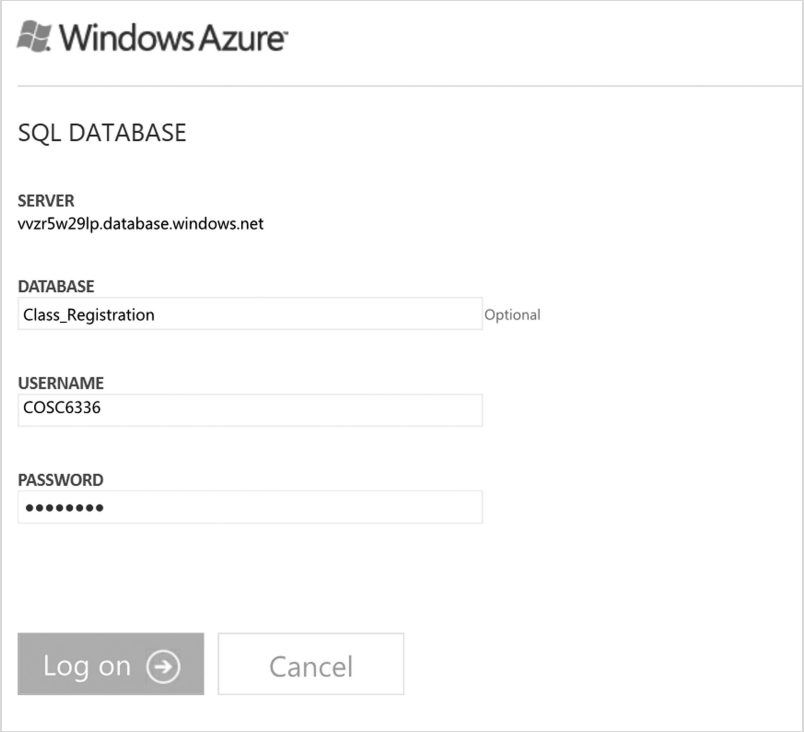

Figure 4.16 Log on to SQL database portal to manage Class_Registration.

```
INSERT INTO FACULTY (FacultyID, FirstName, LastName)
VALUES(3, 'Mary', 'Fry')

INSERT INTO FACULTY (FacultyID, FirstName, LastName)
VALUES(4, 'Jen', 'Garza')

— - inserting records into STUDENT
INSERT INTO STUDENT (StudentID, FirstName, LastName, FacultyID)
VALUES(10, 'Liz', 'Cox', 1)

INSERT INTO STUDENT (StudentID, FirstName, LastName, FacultyID)
VALUES(11, 'Joe', 'Cole', 2)

INSERT INTO STUDENT (StudentID, FirstName, LastName, FacultyID)
VALUES(12, 'Linda', 'Diaz', 1)

INSERT INTO STUDENT (StudentID, FirstName, LastName, FacultyID)
VALUES(13, 'Don', 'Ford', 3)

INSERT INTO STUDENT (StudentID, FirstName, LastName, FacultyID)
VALUES(14, 'Jen', 'Brooks', 4)

INSERT INTO STUDENT (StudentID, FirstName, LastName, FacultyID)
VALUES(16, 'Bruce', 'Cox', 3)
```

```
— - inserting records into COURSE
INSERT INTO COURSE (CourseID, CourseName)
VALUES('ISC2301', 'VB')

INSERT INTO COURSE (CourseID, CourseName)
VALUES('ISC3311', 'Database')

INSERT INTO COURSE (CourseID, CourseName)
VALUES('ISC4301', 'E-Commerce')

INSERT INTO COURSE (CourseID, CourseName)
VALUES('ISC3321', 'Info-Systems')

— - inserting records into TIMEBLOCK
INSERT INTO TIMEBLOCK (TimeID, TimeBlock)
VALUES(1, '9am-12pm')

INSERT INTO TIMEBLOCK (TimeID, TimeBlock)
VALUES(2, '1pm-4pm')

INSERT INTO TIMEBLOCK (TimeID, TimeBlock)
VALUES(3, '7pm-10pm')

— - inserting records into SEMESTER
INSERT INTO SEMESTER(SemesterID, Semester)
VALUES(1, 'Fall')

INSERT INTO SEMESTER(SemesterID, Semester)
VALUES(2, 'Spring')

INSERT INTO SEMESTER(SemesterID, Semester)
VALUES(3, 'Summer')

— - inserting records into DAYS
INSERT INTO DAYS(DayID, WeekDay)
VALUES(1, 'Monday')

INSERT INTO DAYS(DayID, WeekDay)
VALUES(2, 'Tuesday')

INSERT INTO DAYS(DayID, WeekDay)
VALUES(3, 'Wednesday')

INSERT INTO DAYS(DayID, WeekDay)
VALUES(4, 'Thursday')

INSERT INTO DAYS(DayID, WeekDay)
VALUES(5, 'Friday')

INSERT INTO DAYS(DayID, WeekDay)
VALUES(6, 'Saturday')

INSERT INTO DAYS(DayID, WeekDay)
VALUES(7, 'Sunday')

— - inserting records into CLASS
```

```
INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1000, 'ISC2301', 4, 1, 1, 1)

INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1001, 'ISC3311', 4, 2, 1, 2)

INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1002, 'ISC4301', 3, 3, 2, 1)

INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1003, 'ISC2301', 4, 1, 2, 2)

INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1004, 'ISC3311', 4, 2, 3, 2)

INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1005, 'ISC4301', 3, 3, 3, 1)

INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(1006, 'ISC3321', 3, 1, 4, 1)


— - inserting records into CLASSROOM
INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(103, 2, 30)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(105, 1, 25)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(206, 3, 20)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(107, 3, 25)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(305, 2, 20)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(215, 1, 20)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(105, 4, 20)

INSERT INTO CLASSROOM (ClassroomID, BuildingID, Capacity)
VALUES(121, 2, 20)


— - inserting records into STUDENT_CLASS
INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(10, 1000, 'A')
```

```
INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(11, 1000, 'C')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(12, 1002, 'B')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(13, 1000, 'C')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(14, 1001, 'B')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(10, 1001, 'A')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(16, 1000, 'B')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(13, 1001, 'A')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(10, 1002, 'A')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(11, 1006, NULL)

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(12, 1005, NULL)

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(13, 1002, 'C')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(14, 1002, 'B')

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(10, 1005, NULL)

INSERT INTO STUDENT_CLASS (StudentID, ClassID, Grade)
VALUES(16, 1001, NULL)


— - inserting records into FACULTY_CLASS
INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(1, 1000)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(2, 1001)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(1, 1002)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(3, 1000)
```

```
INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(4, 1001)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(1, 1003)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(3, 1004)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(2, 1006)

INSERT INTO FACULTY_CLASS (FacultyID, ClassID)
VALUES(3, 1005)

— - inserting records into COURSE_PREREQUISITE
INSERT INTO COURSE_PREREQUISITE (CourseId, Prerequisite)
VALUES('ISC3311', 'ISC2301')

INSERT INTO COURSE_PREREQUISITE (CourseId, Prerequisite)
VALUES('ISC3321', 'ISC3311')

INSERT INTO COURSE_PREREQUISITE (CourseId, Prerequisite)
VALUES('ISC4301', 'ISC3311')

INSERT INTO COURSE_PREREQUISITE (CourseId, Prerequisite)
VALUES('ISC4301', 'ISC3321')

— - inserting records into CLASS_CLASSROOM
INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1000, 103, 2)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1001, 105, 1)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1002, 206, 3)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1003, 107, 3)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1004, 305, 2)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1000, 121, 2)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1001, 107, 3)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1006, 215, 1)

INSERT INTO CLASS_CLASSROOM (ClassID, ClassroomID, BuildingID)
VALUES(1005, 105, 4)
```

```
INSERT INTO CLASS(ClassID, CourseID, Credit, TimeID, DayID,
SemesterID)
VALUES(2001, 'ISC3321', 3, 2, 4, 2)
```

3. As shown in Figure 4.17, rows are added to the tables in the database Class_Registration.

Now, you have both the Computer_Store and Class_Registration databases ready for operation on Windows Azure. In later chapters, you will learn how to retrieve information from these databases and develop database applications.

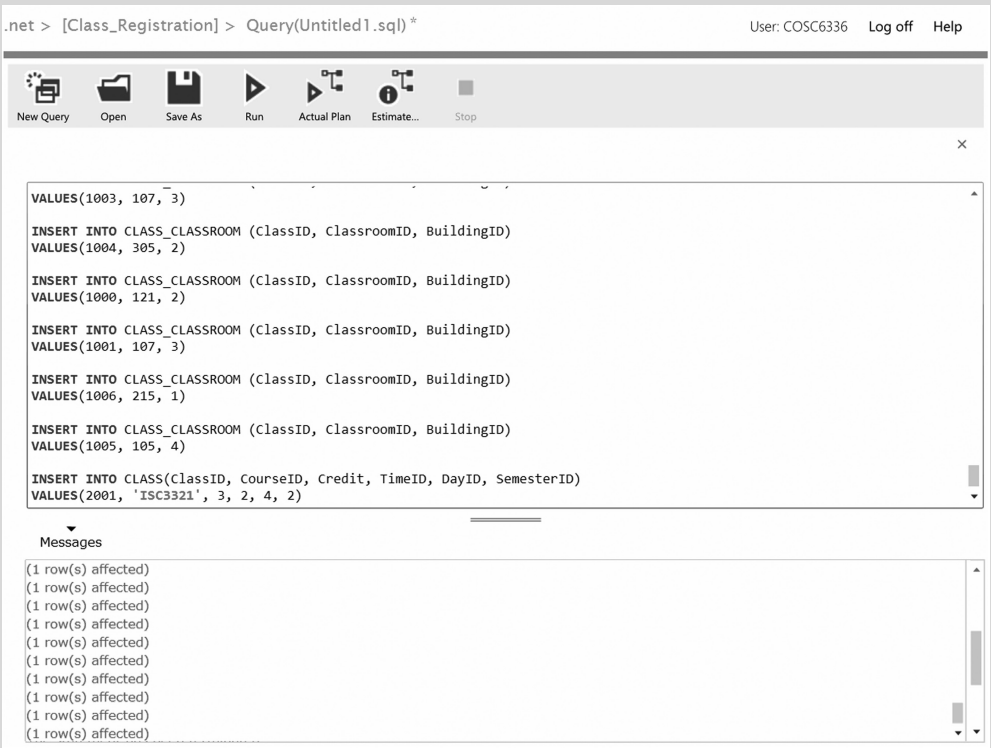

Figure 4.17 Populate Class_Registration database with SQL statements.

4.6 Summary

In this chapter, SQL has been used to implement relational databases. You have learned how to use Data Definition Language (DDL) to create, alter, and delete database objects such as databases, tables, and schemas. You also learned to use Data Control Language (DCL) and Data Manipulation Language (DML) statements to manage database objects. After tables are created, you can enter data into the tables with SQL statements.

Review Questions

1. Explain why SQL is important to database development.
2. What is the SQL extension used in SQL Server?
3. What are the tasks that can be done by DDL?
4. What are the commonly used DDL commands?
5. What are the tasks that can be done by DCL?
6. What are the commonly used DCL commands?
7. What are the tasks that can be carried out by DML?
8. What are the commonly used DML commands?
9. Create the table ORDER(OrderId, OrderDate, ShippingCharge, Total).
10. Write an SQL statement to create a table called COMPUTER with the definition COMPUTER(ComputerId, Maker, Model, Price).
11. Write an SQL statement to create a table with the following information: HARDDRIVE(SerialNumber, Maker, Model, Price).
12. Write an SQL statement to create a table with the following information: ORDER_HARDDRIVE(*SerialNumber*, *OrderId*)
13. Write an SQL statement to create a table with the following information: ORDER_COMPUTER(*OrderId*, *ComputerId*).
14. Alter the table HARDDRIVE by adding a new column Description with CHAR(100) data type.
15. Use the INSERT statement to insert the following information into the table HARDDRIVE:

11111	KEM	TI20GB	168
12293	QVT	SL40GB	243
23412	TTP	FK100GB	405

16. For the table HARDDRIVE, update the price for the hard drive with the serial number 12293 to $220.
17. Use the GRANT statement to grant the insert privilege to public on table HARDDRIVE, and then use the REVOKE statement to remove the insert privilege to public.
18. Insert the following data into the table ORDER created for Question 9:

10	7/9/04	90	1290
11	7/11/04	56	1456
12	7/15/04	78	2078
13	8/1/04	103	1903
14	8/3/04	50	1650
15	8/7/04	90	1896

Insert the following data into the table COMPUTER created for Question 10:

D0001	CQ	DT4135	1200
N0001	DP	NS2190	1400
D0002	CQ	DT6135	1800
N0002	GN	HK4900	1600
N0003	BN	CT1289	2000

Insert the following data into the table ORDER_HARDDRIVE created in Question 11:

10	12293
12	11111
15	23412

Insert the following data into the table ORDER_COMPUTER created in Question 13:

10	D0001
11	N0001
12	N0003
13	D0002
14	N0002
15	D0002