



3

Inserting, Updating, and Deleting Data

Once a database is designed and constructed it is ready to be populated with data. This chapter covers populating tables and manipulating data in the database. We will be using the Teachers table from the COLLEGE database that we created in Chapter 2 to practice our code.

CHAPTER OBJECTIVES

The objectives for this chapter are as follows:

- Understand how to add data using the INSERT statement.
- Understand how to modify data using the UPDATE statement.
- Understand how to remove data using the Delete statement.
- Understand how to use comments
- Understand how to save SQL files.

Insert Statement

The INSERT statement allows you to add data to the table. There are two forms of INSERT statements to be aware of. The first form involves populating each column in a table with a value. An example of the syntax for this form is as follows:

```
INSERT INTO Teachers Values (Value 1, Value 2, Value 2....)
```

Notice in the above example the values are in parenthesis and separated by commas. Let's try executing an actual INSERT statement but before we do let's take a look at the design of the Teachers table shown in Figure 1. We can see there are three columns that cannot have NULL values. They are TeacherID, LastName, and FirstName. The rest of the columns do not require data.

We can also see that the TeacherID is the primary key and it has an identity specification causing it to be automatically incremented by SQL Server starting with the number 1 and incrementing by 1 in sequence.

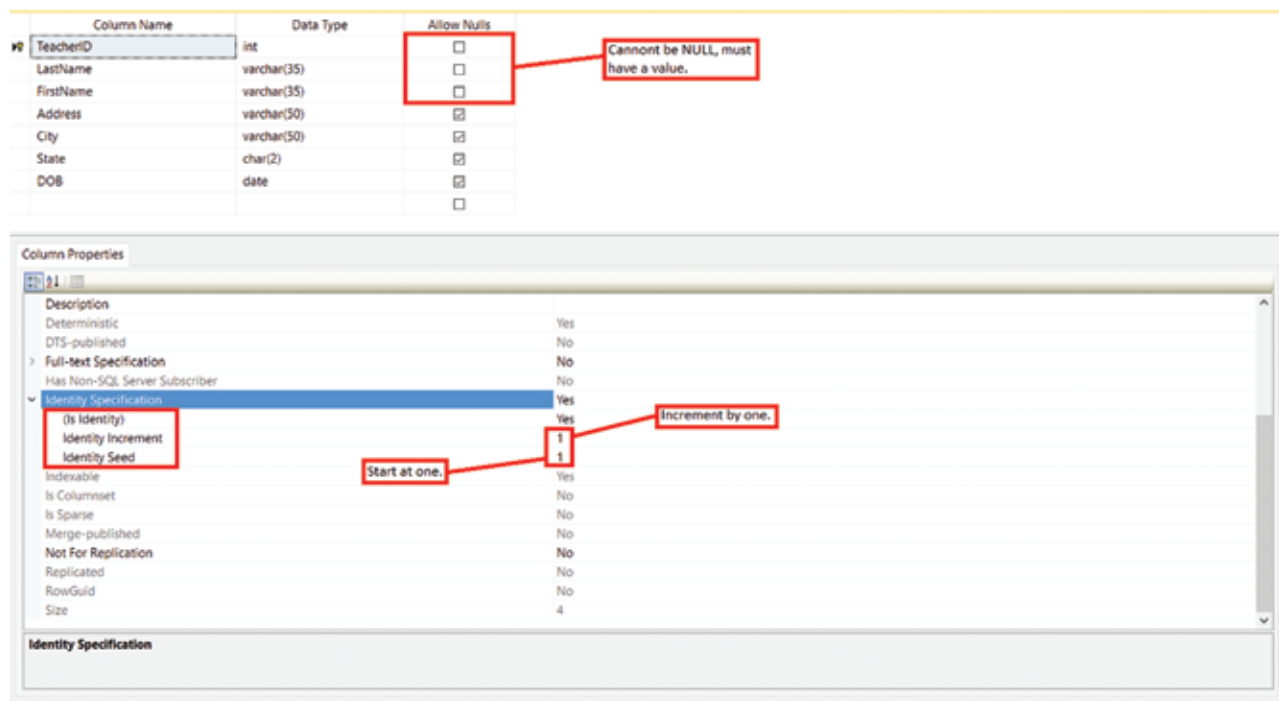


Figure 1

Let's write our first INSERT statement populating all columns in the table as shown below. Make sure you are in the correct database (COLLEGE) and that you highlight the statement and click the Execute button. See Figure 2.

```
INSERT INTO Teachers VALUES ('Smith', 'Thomas', '25 Fox Hill RD.', 'Enfield',  
'CT', '1970-12-02')
```

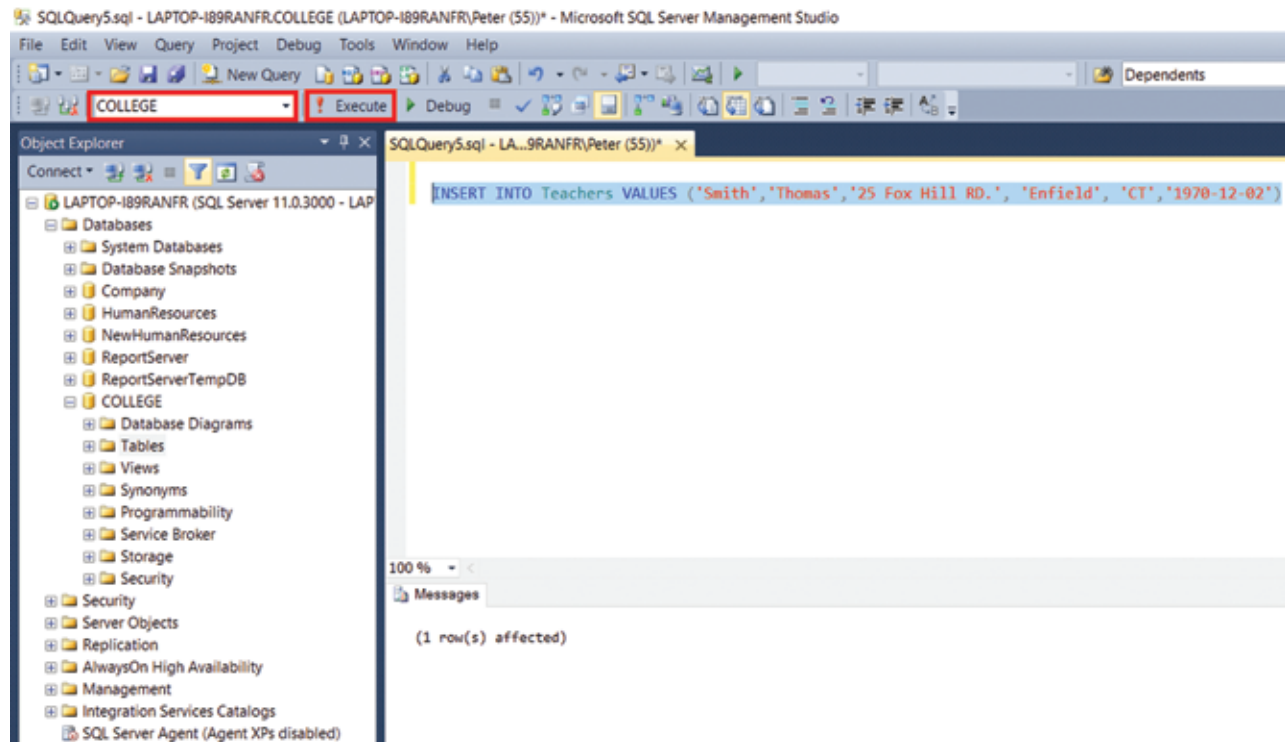


Figure 2

After you've successfully executed the INSERT statement select all columns from the table by issuing the following query:

```
SELECT * FROM Teachers
```

The result from the above statement is shown in Figure 3.

- **NOTE:** SELECT * is a very useful command and you will often find yourself using it throughout this book. SELECT * is SQL for select all. SELECT * will display every column and all the data that populates each column.

SQLQuery5.sql - LA...9RANFR\Peter (55))*

```
INSERT INTO Teachers VALUES ('Smith','Thomas','25 Fox Hill RD.', 'Enfield', 'CT','1970-12-02')
SELECT * FROM Teachers
```

100 %

Results Messages

	Teache...	LastNa...	FirstNa...	Address	City	St...	DOB
1	1	Smith	Thomas	25 Fox Hill RD.	Enfield	CT	1970-12-02

Figure 3

Notice that all the fields are populated and that TeacherID is equal to one as instructed by the identity seed in the design of the table. Let's issue another INSERT statement as follows:

```
INSERT INTO Teachers VALUES ('Jackson','Robert','589 Main ST.', 'Enfield', 'CT','1965-09-11')
```

Once you've executed the above INSERT statement execute a SELECT statement on the Teachers table again to see the results as shown in Figure 4.

- **NOTE:** Don't forget to highlight the SQL statement you want to execute before clicking the Execute button.

SQLQuery5.sql - LA...9RANFR\Peter (55))*

```
INSERT INTO Teachers VALUES ('Smith','Thomas','25 Fox Hill RD.', 'Enfield', 'CT','1970-12-02')
INSERT INTO Teachers VALUES ('Jackson','Robert','589 Main ST.', 'Enfield', 'CT','1965-09-11')
SELECT * FROM Teachers
```

100 %

Results Messages

	Teache...	LastNa...	FirstNa...	Address	City	St...	DOB
1	1	Smith	Thomas	25 Fox Hill RD.	Enfield	CT	1970-12-02
2	2	Jackson	Robert	589 Main ST.	Enfield	CT	1965-09-11

Figure 4

Notice the increment by one to the TeacherID. Each new row will receive a TeacherID one more than the last TeacherID issued to make the TeacherID unique.

Let's understand the TeacherID incremental a little bit more. Let's say we delete the row we just added by executing the following statement and shown in Figure 5.

```
DELETE FROM Teachers
WHERE TeacherID = 2
```

► **NOTE:** The DELETE statement will be covered in more detail later in this chapter.

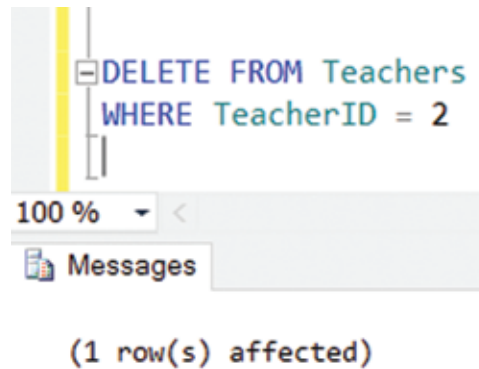


Figure 5

Next let's add another row to the Teachers table with the following INSERT statement.

```
INSERT INTO Teachers VALUES ('Thompson', 'Manuel', '79 Parsons RD.', 'Springfield',
'MA', '1979-05-21')
```

Execute a SELECT statement on the Teachers table and you should see the same results as those shown in Figure 6.

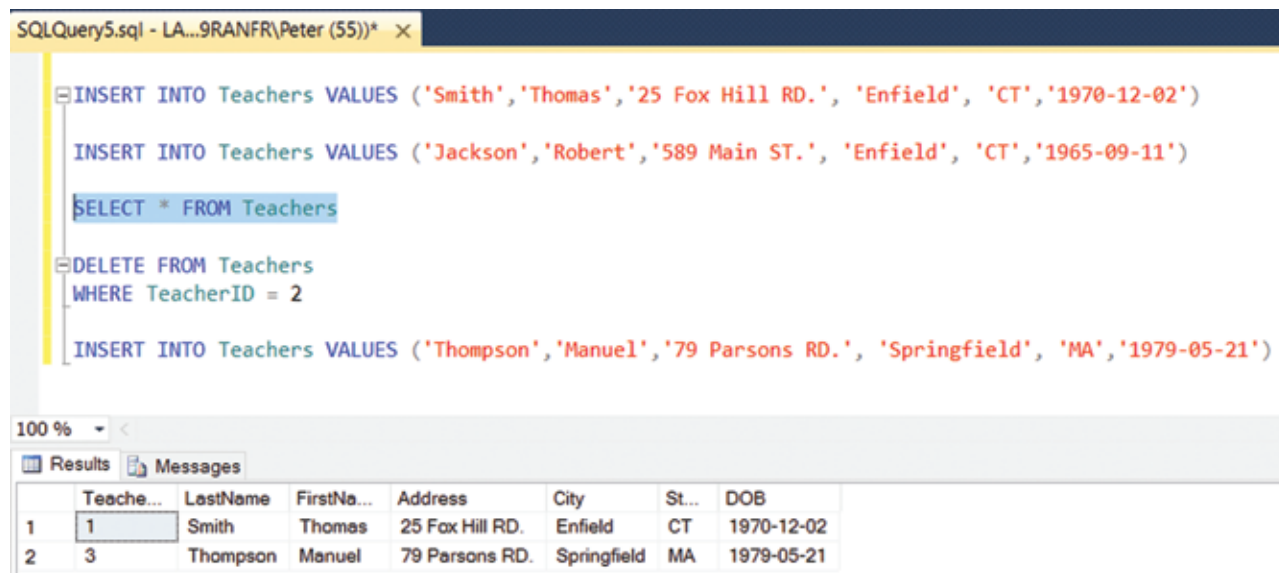


Figure 6

Notice that the new TeacherID is the number 3 rather than 2, even though we deleted the teacher who had a TeacherID of the number two. This is because SQL Server never uses the same identity again. It simply increments from where it left off.

The design of NOT reusing numbers helps to ensure true uniqueness. For example, let's say Manuel Thomson was given the TeacherID of 2 after Robert Jackson was deleted. Then let's say Robert Jackson was deleted by accident and we have to insert him back. What would we do with Robert Jackson's TeacherID if it was assigned to Manuel Thompson? SQL Server never uses the same ID and neither should we if we are ever in a position of assigning IDs manually.

The second form of INSERT statement is where all the columns are not populated. In this case we need to type out what columns we want to populate and have the values in the same order as the columns. For example, the following INSERT statement populates the last name, and first name of the Teachers table which can't be NULL, and populates the DOB which can be NULL.

```
INSERT INTO Teachers (LastName, FirstName, DOB) VALUES ('Bevans', 'Chris',
'1985-08-01')
```

Notice that the values in the above INSERT statement are in the same order as the column names. See the results in Figure 7.

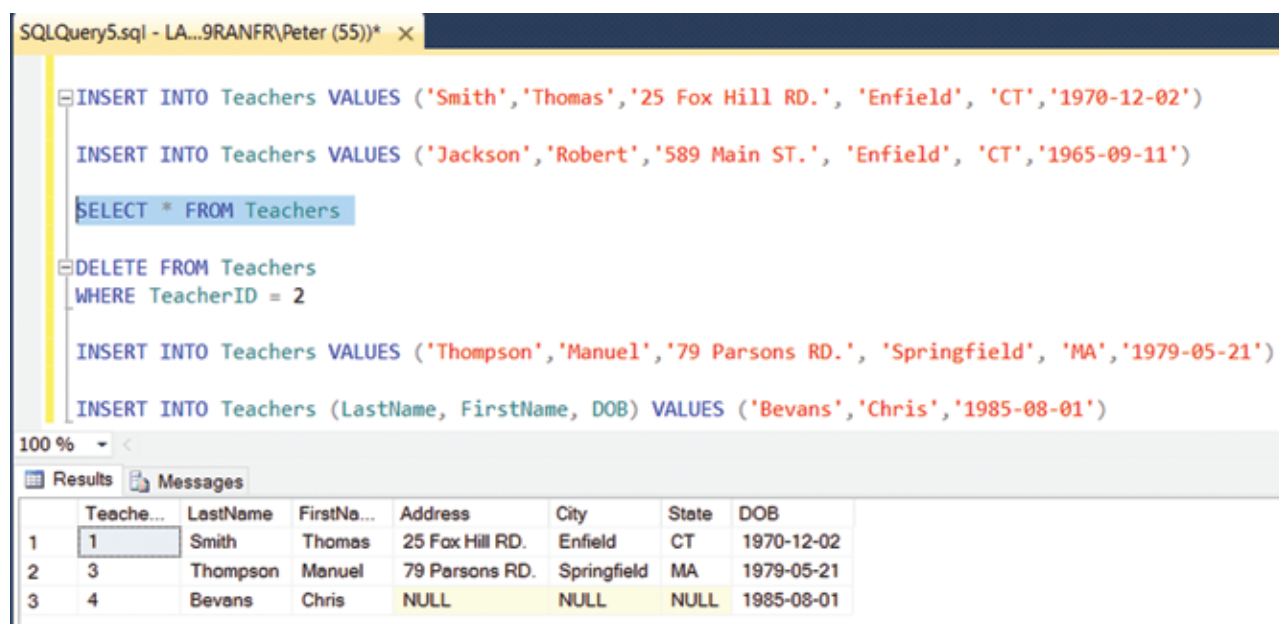


Figure 7

As you can see from the results in Figure 7 Address, City, and State have not been populated resulting in NULL values.

What if we tried executing an INSERT statement without populating all the NOT NULL columns. For example, if we didn't include the FirstName column as shown in Figure 8 we would receive an error message.

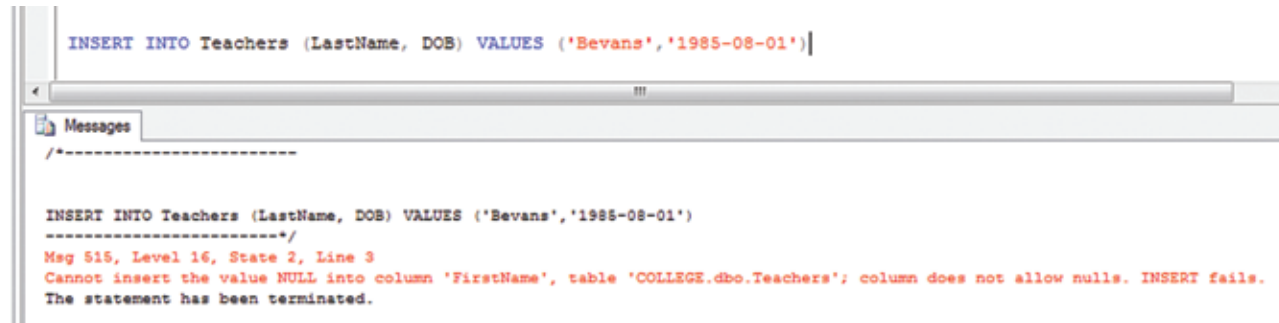
The screenshot shows a SQL query execution window. At the top, the query is: `INSERT INTO Teachers (LastName, DOB) VALUES ('Bevans', '1985-08-01')`. Below the query, there is a 'Messages' pane. The message pane shows the following text: `INSERT INTO Teachers (LastName, DOB) VALUES ('Bevans', '1985-08-01')`, followed by a separator line, then `Msg 515, Level 16, State 2, Line 3`, and finally the error message: `Cannot insert the value NULL into column 'FirstName', table 'COLLEGE.dbo.Teachers'; column does not allow nulls. INSERT fails. The statement has been terminated.`

Figure 8

So, you can see from Figure 8 that “not null” columns must contain data in the insert statement in order for the statement to execute correctly.

Before we begin with the rest of the chapter add these last two teachers to the Teachers table.

```
INSERT INTO Teachers VALUES ('Carone', 'Melvin', '789 Elm ST.', 'South Windsor',  
'CT', '1982-02-12')
```

```
INSERT INTO Teachers VALUES ('Barret', 'John', '70 Nutmeg RD.', 'Bloomfield',  
'CT', '1992-07-22')
```


Saving Your Work

We can save the SQL code we write much like a Word document only that SQL files contain ".sql" extensions rather than ".docx". To save your work you would perform a "File, save as" much like your Office applications and save it to a location of your choosing. Look at Figure 9 for an example.

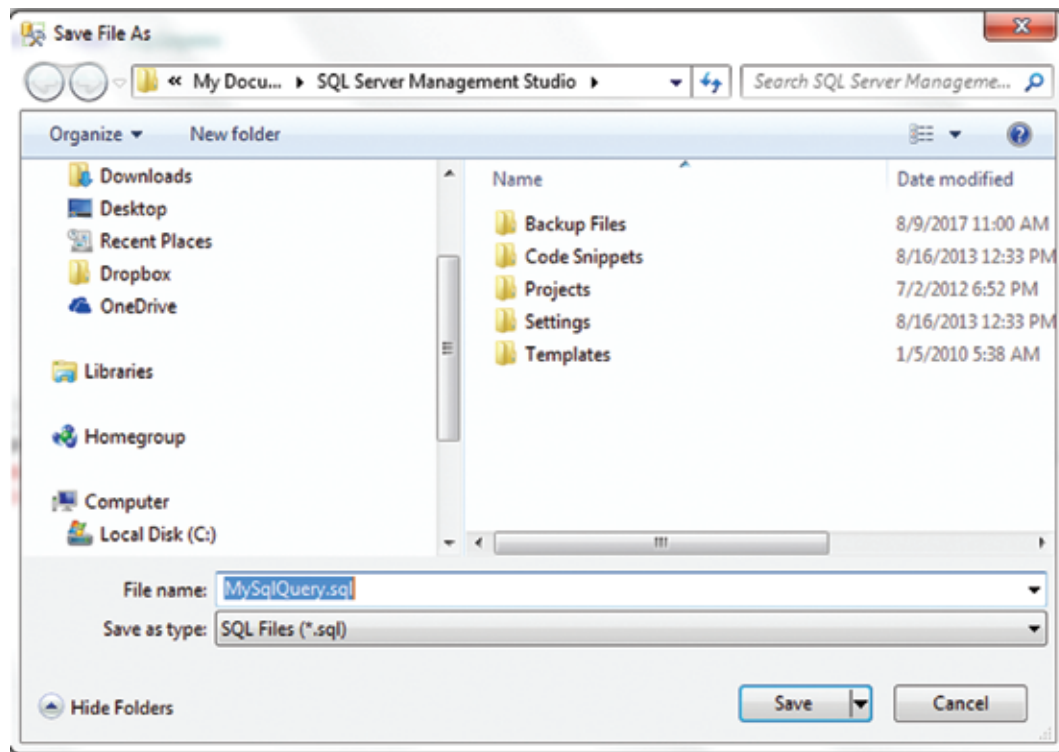


Figure 9

Comments

It is often convenient to include comments in and around your SQL Statements to help you or another developer understand the code involved. There are two ways to write comments. One is with two dashes right before the comment as shown below. This is used for single line comments.

```
-- This is a single line comment
```

The other is for multiline comments and begins with a forward slash and asterisk and ends with an ending asterisk and backward slash. See the example below as well as the example in Figure 10. Notice from Figure 10 comments are displayed in green by SQL Server.

```
/* This is a multiline comment
   where I can write as many
   lines as I want to. */
```




```

SQLQuery2.sql - ...r-PC\Owner (52)) SQL_Server_Chapt...PC\Owner (56))
INSERT INTO Teachers (LastName, FirstName, DOB) VALUES ('Bevans', 'Chris', '1985-08-01')

INSERT INTO Teachers (LastName, DOB) VALUES ('Bevans', '1985-08-01')

-- This is a single comment line. Notice comments are green

/* The forward slash and asterisk starts the beginning
of the comment section. you are able to enter as many lines as
you would like. To end the comment section you use an
asterisk and a backward slash */

```

Figure 10

UPDATE Statement

The UPDATE statement is used to modify information already stored in the database. It uses the SET operator to assign values to columns as well as the WHERE clause to designate what row will be updated. Below is an example.

```

UPDATE Teachers
SET Address = '21 Quincy Ave.'
WHERE TeacherID = 3

```

Be very careful when using the UPDATE statement without a WHERE clause because if you don't identify the rows you want update with a WHERE clause all rows will be updated.

You can also update many columns in the same statement by separating the columns and values with commas as shown below.

```

UPDATE Teachers
SET Address = '28 Town Line Road',
    City = 'South Windsor',
    State = 'CT'
WHERE TeacherID = 3

```

Let's execute the above UPDATE statement but before we do let's look at the row for TeacherID 3 first and then UPDATE and look at it again afterward so we can analyze the results.

Before:

The screenshot shows a SQL query window with the following SQL code:

```
SELECT * FROM TEACHERS WHERE TEACHERID = 3
```

The Results tab displays the following data:

TeacherID	LastName	FirstName	Address	City	State	DOB
3	Thompson	Manuel	79 Parsons RD.	Springfield	MA	1979-05-21

Below the query window, the Messages tab shows the following message:

```
UPDATE Teachers  
SET Address = '28 Town Line Road',  
City = 'South Windsor',  
State = 'CT'  
WHERE TeacherID = 3
```

The status bar indicates: (1 row(s) affected)

Figure 11

After:

The screenshot shows the same SQL query window as before, but the Results tab now displays the updated data:

TeacherID	LastName	FirstName	Address	City	State	DOB
3	Thompson	Manuel	28 Town Line Road	South Windsor	CT	1979-05-21

You can see from the results above that the UPDATE was successful.

DELETE Statement

The DELETE Statement will remove any rows of data that you specify. Like the UPDATE statement, be very careful to use the DELETE statement WITHOUT a WHERE clause because doing so will remove all rows from the table. For example, the following DELETE statement will remove all rows from the Teachers table if it was executed.

DO NOT EXECUTE.

```
DELETE  
FROM Teachers
```

THIS COMMAND WILL DELETE ALL ROWS IN THE TEACHERS TABLE

DO NOT EXECUTE IT.

Instead you should be specific about the rows you want deleted. The DELETE statement below will remove Manuel Thomson from the Teachers table because it is using his TeacherID in the WHERE clause.

```
DELETE  
FROM Teachers  
WHERE TeacherID = 3
```

After executing the above DELETE statement you should see the same results as those in Figure 12.

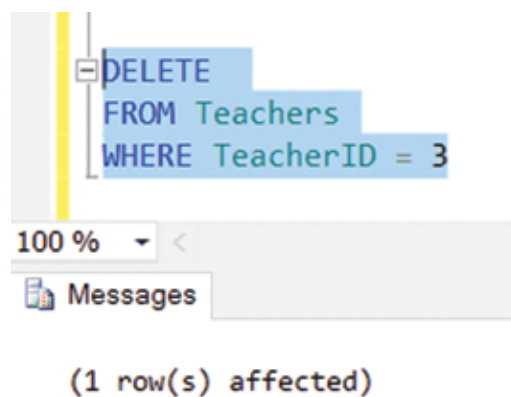


Figure 12

Summary

This chapter covered the INSERT, UPDATE, and DELETE statements to add, modify, and remove data from the database. In addition, you learned to add comments to your SQL files as well as save them to a directory of your choice.

Exercises

Perform the following functions on the patients table that you created in Chapter 2.

1. Insert five rows into the Patients table. Show your INSERT statements and resulting rows by taking screen shots and pasting them in a Word document. Enter whatever data you wish.
2. Update three of the rows. Show a screen shot of the UPDATE statements and the resulting rows.
3. Delete three rows. Show DELETE statements and resulting rows with a screen shot.