LESSON 3: Manipulating Data

You have just been hired by a medium-sized company and asked to provide employee record lists to the newly hired vice president of Human Resources. In order to manipulate the data to fit the vice president's needs, you must create a variety of different lists using Transact-SQL statements.

Although the title of this lesson refers to manipulating data for query purposes, it's important to note that querying is itself considered a form of data modification because it involves changing query statements to obtain a desired output. No matter the name or descriptor, the object of any database is for the user to be able to extract data from it. In fact, the vast majority of SQL statements are designed to retrieve user-requested information from a database through the use of queries. In this lesson, we'll look at some of the most important query statements you must understand to further your understanding of Transact-SQL.

Using Queries to Select Data

THE BOTTOM LINE
In this section, you'll learn how to utilize the SELECT query to retrieve or extract data from a table, how to retrieve or extract data using joins, and how to combine results using
UNION and INTERSECT.

The SQL command for retrieving any data from a database is SELECT. Much like any other SQL command, SELECT is similar to an English statement. Composing a SELECT statement is akin to filling in the blanks, as shown in the following example:

CERTIFICATION READY
What command is used to display records from a table?
3.1

```
SELECT id, name //columns

FROM sysobjects // tables

WHERE type = “jones” //conditions you wish to produce results from
```

This simple example provides a basic understanding of what the SELECT statement does. You will always follow the same pattern each time you issue a SELECT statement to a database. Moreover, there are only three things you need to be sure to identify in your statement to form a proper SELECT query:

- Columns to retrieve
- Tables to retrieve the columns from
- Conditions, if any, that the data must satisfy

The previous constructs are considered your framework for creating SELECT query statements using the SQL text editor window. Let's say, for example, that you want to give your boss a list of employees whose salary is above $50,000 per year. You are interested in retrieving only those employees who fit that criteria. Here's how you could do this in SQL:

```
first_name                 last_name          salary

________________           _______________    _________

John                       Allan               52,000

Sylvia                     Goddard             51,200

Julia                      Smith               55,000

David                      Thompson            62,900

(4 row(s) affected)
```

If you want to select only a single column for your query, you can identify the name of the column by typing it between *select* and *from* in the query statement. To identify more than one column to include in your query (as in the above example), simply type each column name and separate the names with a comma. The reason for using a comma instead of a space is that SQL treats a space as an identifier, or match word, such as "value" or "select." Thus, if you need to use a space in your statement, you need to enclose the words in square brackets or double quotes—for example, [select] or "value."

If you wish to choose all columns from within a table, you can do so by typing an asterisk (*) in the place where the column name(s) would otherwise be given.

TAKE NOTE*
Do you see the line stating the number of rows "affected"? You can turn this line off and on using the SET NOCOUNT statement.
This statement is set to off by default.

The only required component of the SQL SELECT query is the SELECT … FROM clause, meaning that you could select all available fields from one table simply by issuing the following command:

```
SELECT *

FROM employees
```

This could produce a large result or a small result, depending on the number of employees within a company. Let’s use an example of a company with only six employees. If we entered the above command, our result would look similar to the following:

```
first_name      last_name     employee_id   phone           gender

____________     __________     ___________    _____________    _______

Jim              Alexander      610001         574-555-0001     M

Frances          Drake          610002         574-555-0346     F

David            Thompson       610003         574-555-0985     M

Alexandria       Link           610004         574-555-9087     F

Peter            Link           610005         574-555-7863     M

Antoin           Drake          610006         574-555-2597     M

(6 row(s) affected)
```

You now have a basic understanding of what the `SELECT` statement is designed to do. But what if you want to retrieve only specific types of data from the tables you identified with your original `SELECT` … `FROM` query statement? This is when the `WHERE` clause comes in handy. For instance, the `WHERE` clause could be added to a query to find only those employees who work in the company’s shipping department, as shown here:

```
first_name              last_name

_______________         _____________

Jim                      Alexander

Frances                  Drake

David                    Thompson

(3 row(s) affected)
```

Combining Conditions

Perhaps you require more from a query than simply one set of parameters. In such cases, you can combine several conditions in one query statement. For instance, in the previous example, we ran a query that returned only those employees who work in the shipping department. Say you now want that result *and* you want to determine which of these employees are female. You could obtain this desired result using the following SQL statement:

```
SELECT first_name, last_name
FROM employees
WHERE department = ‘shipping’ AND gender = ‘F’ AND hired >=
‘2000-JAN-01’
```

The expected output shows the following results:

```
first_name             last_name
____________           ____________
Frances                Drake
(1 row(s) affected)
```

Thus, this query uses the AND conjunction to yield the names of all employees who are in the shipping department *and* who are female.

You can also use the OR conjunction to return a result that meets *either* of two conditions, as shown below:

```
SELECT first_name, last_name
FROM employees
WHERE department = ‘shipping’ OR employee_id <= 610007
```

This would return the following result:

```
first_name             last_name
______________         _____________
James                  Alexander
David                  Thompson
Frances                Drake
Alexandria             Link
Peter                  Link
Antoin                 Drake
(6 row(s) affected)
```

In this example, notice that the query yields not just the three employees in the shipping department (thereby meeting the first condition), but also three new rows of data that satisfy *only* the second condition of the query (i.e., an employee ID number less than 610007). This means that Alexandria Link, Peter Link, and Antoin Drake meet the second condition in the `WHERE` query statement, but not the first.

Using the BETWEEN Clause

In some situations, you may need to retrieve records that satisfy a range condition and also contain a value within a range of another specified value. For instance, perhaps you need to retrieve a list of employees who were hired between 1990 and 2000. One way you can obtain this result is to join the two conditions using the `AND` conjunction:

```
SELECT first_name, last_name, hire_date

FROM employees

WHERE hire_date >= '1-Jan-1990' AND hire_date <= '1-Jan-2000'
```

This query would produce the following results:

```
first_name         last_name       hire_date

____________       ___________     ___________

James              Alexander       1990-12-10

Frances            Drake           1998-03-04

Peter              Link            1997-07-08

Antoin             Drake           1999-12-31

(4 row(s) affected)
```

You may be wondering what this query is really doing based on the two conditions and why the syntax looks awkward. To help resolve the awkwardness in the `AND` clause in this query statement, try replacing it with a `BETWEEN` clause. This allows you to specify the range to be used in a "between x and y" query format, yielding a much cleaner statement. Let's rewrite the previous statement to reflect the *between* condition instead:

```
SELECT first_name, last_name, hire_date

FROM employees

WHERE hire_date BETWEEN '1-Jan-1990' AND '1-Jan-2000'
```

With this statement, you will receive exactly the same output as with the previous query statement.

Using the NOT Clause

In some instances, it is simpler to write your query in terms of what you *don't* want in your output. Transact-SQL provides you with the NOT keyword for precisely such situations. For example, say you want a list of all employees who don't work in the shipping department. You could obtain this list using the following query:

```
SELECT first_name, last_name

FROM employees

WHERE NOT department = 'shipping'
```

The use of operators, as shown in this and several earlier examples, can help in achieving the same results in many instances. Thus, you could write a query using a < (greater than) and a > (less than) operator in place of an equal sign. Doing so would yield the following query statement:

```
SELECT first_name, last_name

FROM employees

WHERE department <> 'shipping'
```

No matter which way you write the syntax for the query statement, it will produce the same results.

Using the UNION Clause

The UNION clause allows you to combine the results of two or more queries into a resulting single set that includes all the rows belonging to the query in that union. The UNION clause is entirely different from the JOIN statements, which combine columns from two different tables. You must remember a couple of basic rules when combining the results of two queries via the UNION clause:

- The number and order of the columns must be the same in each of the queries in the clause.
- The data types you use must be compatible.

For instance, you could use the UNION clause as follows to create a query that returns a list of all employees in the shipping department who were hired between January 1, 1990, and January 1, 2000:

```
SELECT first_name, last_name

FROM employees

WHERE department = 'shipping'

UNION

SELECT first_name, last_name
```

```
FROM employees

WHERE hire_date BETWEEN ‘1-Jan-1990’ AND ‘1-Jan-2000’
```

Using the EXCEPT and INTERSECT Clauses

Both the EXCEPT and the INTERSECT statements are designed to return distinct values by comparing the results of two queries. In particular, the EXCEPT clause gives you the final result set where data exists in the first query and not in the second dataset. The INTERSECT gives you the final result set where values in both of the queries match by the query on both the left and right sides of the operand.

The same two basic rules apply to use of the EXCEPT and INTERSECT clauses as apply to use of the UNION clause:

- The number and order of the columns must be the same in all queries.
- The data types must be compatible.

For example, say you worked in a factory setting, and you wanted to retrieve one list showing products with work orders and another list showing products without any work orders. You could structure the query as follows, first using the INTERSECT clause:

```
SELECT ProductID

FROM Production.Product // The database name is Production and

    the table name is Product

INTERSECT

SELECT ProductID

FROM Production.WorkOrder;

——Result: 238 Rows (products that have work orders)
```

Here’s the same query, but using the EXCEPT clause:

```
SELECT ProductID

FROM Production.Product

EXCEPT

SELECT ProductID

FROM Production.WorkOrder;

——Result: 266 Rows (products without work orders)
```

Using the JOIN Clause

The `JOIN` clause allows you to combine related data from multiple table sources. `JOIN` statements are similar in application to both `EXCEPT` and `INTERSECT` in that they return values from two separate table sources. Using this knowledge, let's see what data can be extracted through the use of `JOIN` statements.

`JOIN` statements can be specified in either the `FROM` or the `WHERE` clause, but it is recommended that you specify them in the `FROM` clause.

There are three types of `JOIN` statements you should be aware of:

- Inner joins allow you to match related records taken from different source tables.
- Outer joins can include records from one or both tables you are querying that do not have any corresponding record(s) in the other table. There are three types of outer joins: `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, and `FULL OUTER JOIN`
- Cross joins return all rows from one table along with all rows from the other table. `WHERE` conditions should always be included.

For instance, you could use the most common of the `JOIN` statements, `INNER JOIN`, if you wanted to retrieve a list of employees by their ID numbers and match each employee with the ID of his or her current department supervisor. For this type of query, you will have to identify the matching column in each of the tables you wish to write the query against and obtain the desired output from. In this example, the foreign key in Table 3-1 is identified in the column "department_id," and in Table 3-2, the foreign key is identified as the "department" column match: In other words, the Department table's Department ID is linked to the department column in the Employee table

Table 3-1 Employee table

first_name	last_name	employee_id	department
James	Alexander	610001	1
David	Thompson	620002	1
Frances	Drake	610003	1
Alexandria	Link	610004	2

first_name	last_name	employee_id	department
Peter	Link	620005	2
David	Cruze	610007	NULL

Table 3-2 Department table

department_id	first_name	last_name
1	Jane	Horton
2	Mitch	Simmons
3	Paul	Franklin

Trying to combine data between tables can be very cumbersome, especially if you are creating specific lists from thousands of rows of data. Using a SELECT statement query lets you produce individual lists, but the result may be that you get all the information you need but in an individual list format.

The INNER JOIN keyword simplifies this data retrieval by not only using the information from the two tables from which you require output, but using the INNER JOIN keyword to specify the required conditions for which records will appear For example, from the two example tables, you may wish to create a list showing which employees work for each of the different department supervisors. You would write the SQL query statement as follows:

```
SELECT employee.first_name, employee.last_name,

    department.first_name, department.last_name

FROM employee INNER JOIN department

ON employee.department = department.department_id
```

The resulting output is shown below:

```
first_name      last_name       first_name      last_name

___________     __________      ___________     __________

James           Alexander       JaneHorton

David           Thompson        JaneHorton

Frances         Drake           JaneHorton
```

```
Alexandria      Link Mitch      Simmons

Peter           Link Mitch      Simmons

Antoin          Drake           PaulFranklin

(6 row(s) affected)
```

Did you notice that David Cruze does not appear in the output list of employees matched with department supervisors? In the department column, his name is not identified as being in any department, although he is an employee. This could happen for a variety of reasons; perhaps he is a new hire and has not officially started working for any department.

Perhaps your employer would like a list of records from the second table that do not actually match any of your previous conditions. Any of the `OUTER JOIN` statements, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, or `FULL OUTER JOIN`, can yield the query output you desire. The `OUTER JOIN` statements begin where the results of `INNER JOIN` finish and include all records in the left table along with the matching records of the right table AND any non-matching records.

A sample `LEFT OUTER JOIN` statement includes the statement from the `INNER JOIN` shown previously and also includes the non-matching clause:

```
SELECT employee.first_name, employee.last_name,

    department.first_name, department.last_name

FROM employee LEFT OUTER JOIN department

ON employee.department = department.department_id
```

The resulting output would be as follows:

```
first_name      last_name      first_name     last_name

-----------     ----------     -----------    ----------

James           Alexander      Jane           Horton

David           Thompson       Jane           Horton

Frances         Drake          Jane           Horton

Alexandria      Link           Mitch          Simmons

Peter           Link           Mitch          Simmons

Antoin          Drake          Paul           Franklin

David           Cruze          NULL           NULL

(7 row(s) affected)
```

Notice that the only difference between our `INNER JOIN` and `OUTER JOIN` statements is the inclusion of David Cruze. As mentioned previously, David Cruze is not assigned to any department supervisor and thus his name shows a `NULL` value in the list where the columns are identified by each supervisor's first and last name.

In some cases, you may wish to have a table join with itself, say if you want to compare records from within the same table. This is called a self-join. These types of tables are generally found when creating an output list of organizational hierarchies. For example, you may want to find out how many authors live in the same city, so as to provide a list to a publishing house. You could get this output using the following self-join statement:

```
USE pubs

SELECT author1.first_name, author1.last_name, author2.first_name,

    author2.last_name

FROM author1 INNER JOIN author2

ON author1.zip = author2.zip

WHERE author1.city = 'Pittsburgh'

ORDER BY author1.first_name ASC, author1.last_name ASC
```

The resulting output would be as follows:

```
first_name      last_name      first_name      last_name

___________     __________     ___________     __________

David           Jones          David           Jones

David           Jones          Alex            Starr

David           Jones          Linda           Arrow

Alex            Starr          David           Jones

Alex            Starr          Alex            Starr

Alex            Starr          Linda           Arrow

Linda           Arrow          David           Jones

Linda           Arrow          Alex            Starr

Linda           Arrow          Linda           Arrow

Delinda         Burris         Delinda         Burris

Jules           Allan          Jules           Allan

(11 row(s) affected)
```

If you want to eliminate those rows in which the same author is repeatedly matched, you could make the following change to the self-join query statement:

```
USE pubs

SELECT author1.first_name, author1.last_name, author2.first_name,

    author2.last_name

FROM author1 INNER JOIN author2

ON author1.zip = author2.zip

WHERE author1.city = ‘Pittsburgh’

    AND author1.state = ‘PA’

    AND author1.author_id < author2.author_id

ORDER BY author1.first_name ASC, author1.last_name ASC
```

The resulting output would be:

```
first_name        last_name      first_name     last_name

———————————       ——————————     ———————————    ——————————

David             Jones          Alex           Starr

David             Jones          Linda          Arrow

Alex              Starr          Linda          Arrow

(3 row(s) affected)
```

From the results obtained from the query statement, you can confirm that David Jones, Alex Starr, and Linda Arrow all live in Pittsburgh, PA, and have the same ZIP code.

Using Queries to Insert Data

THE BOTTOM LINE

In this section, you’ll develop an understanding of how data is inserted into a database and how you can use `INSERT` statements.

Microsoft SQL Server gives you a number of different ways to insert new data into your databases. Different insertion tools are available to achieve the end goal of joining data together.

If you want to insert small quantities of data by adding a few new rows into your database, for instance, you can accomplish this in two different ways. The first method uses the graphical interface too (SSMS), and the second uses the `INSERT` statement. Either way accomplishes the same goal.

Inserting Data

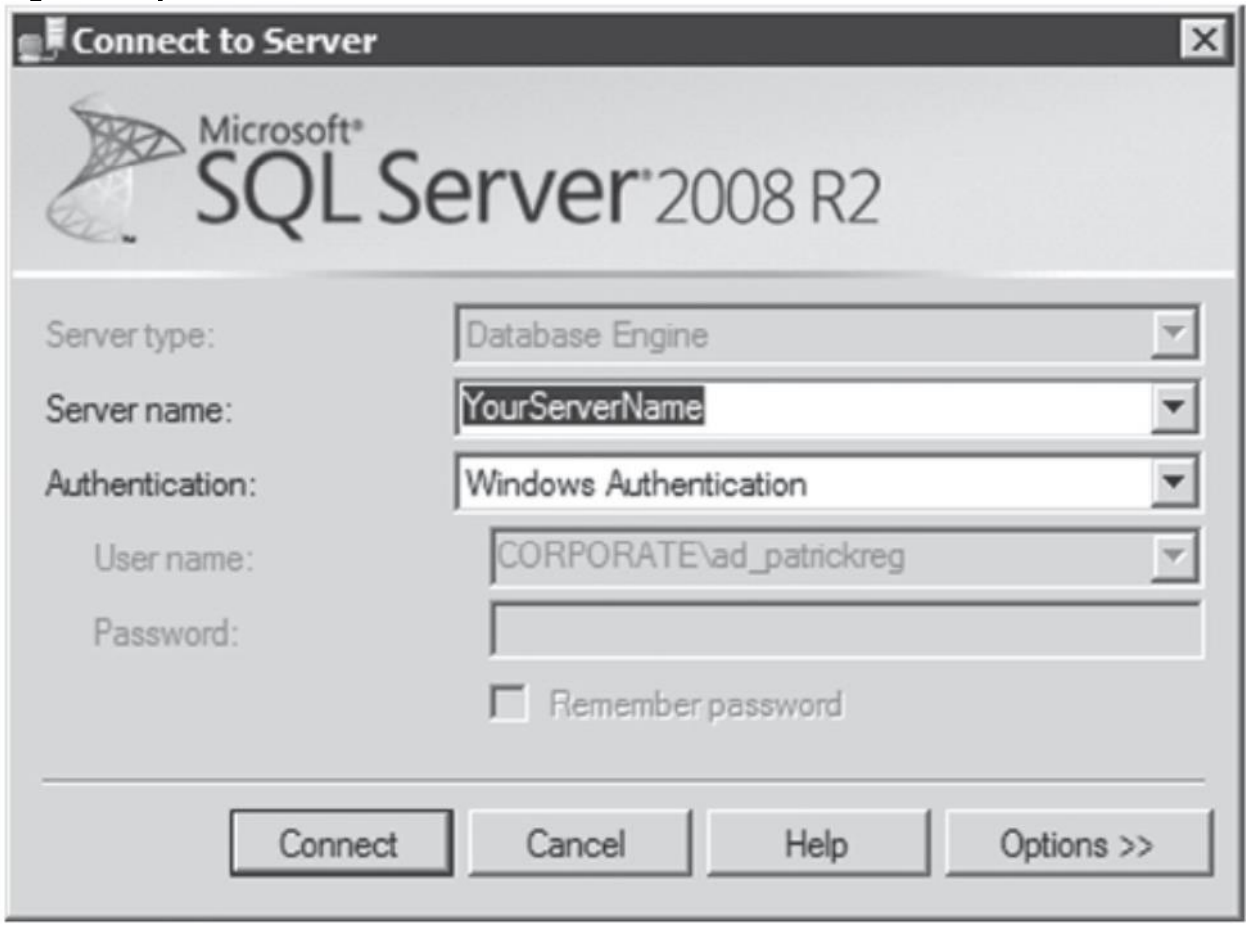
CERTIFICATION READY
What command is used to add data to a table?
3.2

Let's first learn how to insert data into a table using SSMS before we move on to the syntax method.

INSERT DATA USING SQL SERVER MANAGEMENT STUDIO

GET READY. Before you begin, be sure to launch the SSMS application and connect to the database you wish to work with. Then, follow these steps:

1. Check that you have connected to the database you want to work with (see Figure 3-1).



Server type: Database Engine
Server name: YourServerName
Authentication: Windows Authentication
User name: CORPORATE\ad_patrickreg
Password:
Remember password

Figure 3-1 Connecting to the desired database

2. Expand the Databases folder by clicking the plus (+) icon beside the word "Databases."
3. Expand the folder of the database you want to modify.

4. Expand the Tables folder by clicking on the plus sign next to the word "Tables."
5. Right-click the table name and chose Edit Top 200 Rows (see Figure 3-2).

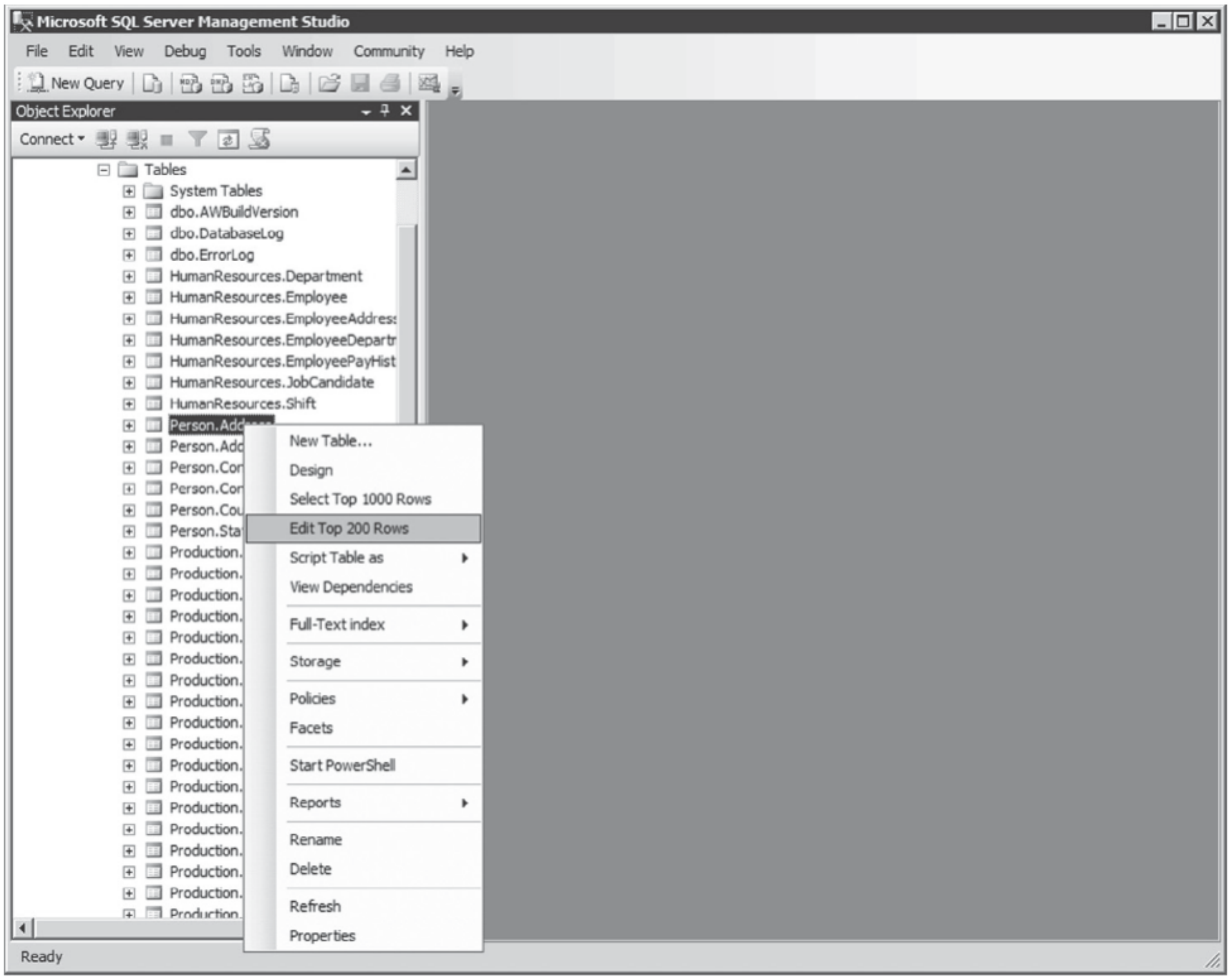
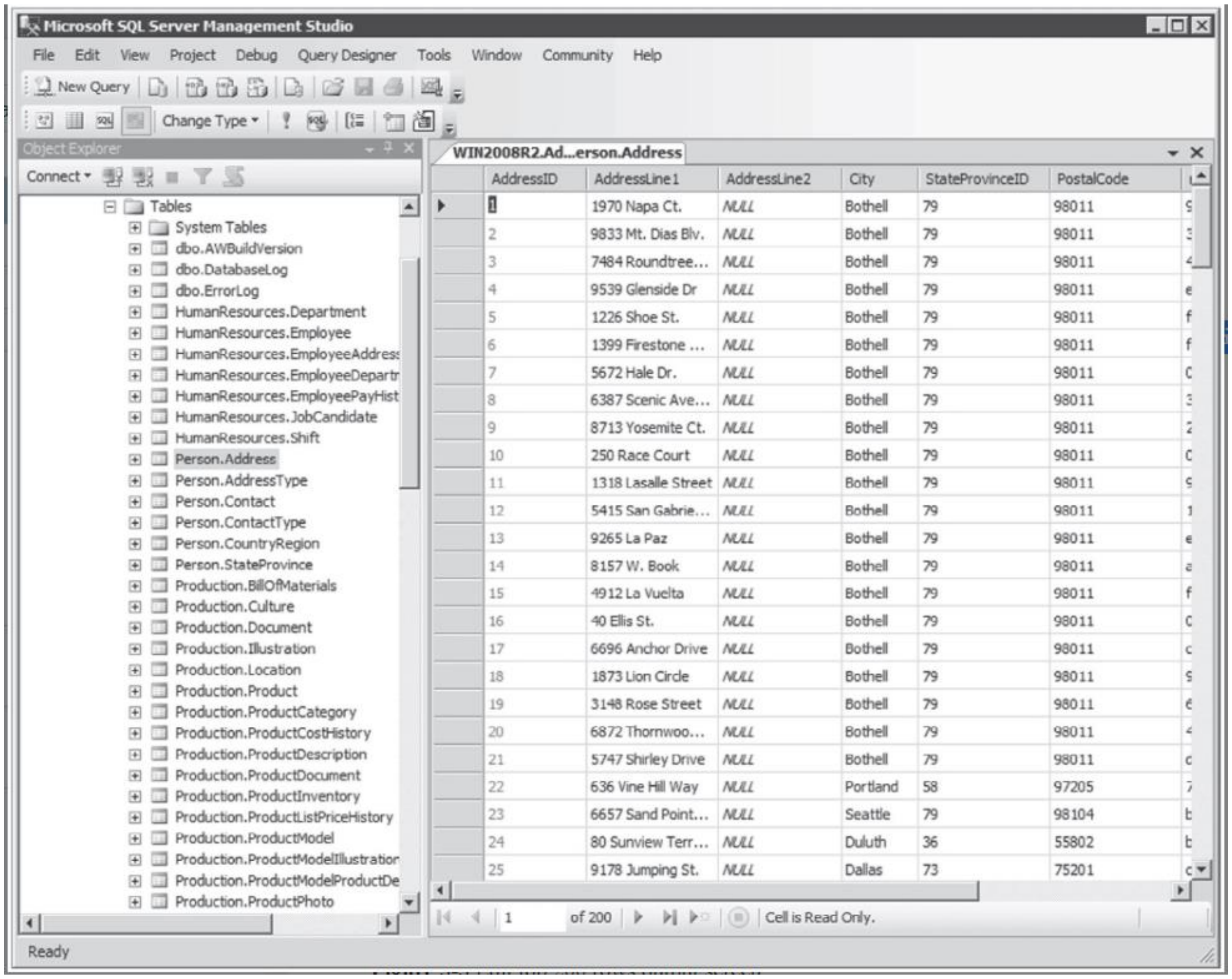


Figure 3-2 Edit top 200 rows

Figure 3-3 shows the output screen. If you have a default value like the Int Identity field, you will not need to provide the field in your query statement.


WIN2008R2.Ad...erson.Address

AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode
1	1970 Napa Ct.	NULL	Bothell	79	98011
2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011
3	7484 Roundtree...	NULL	Bothell	79	98011
4	9539 Glenside Dr	NULL	Bothell	79	98011
5	1226 Shoe St.	NULL	Bothell	79	98011
6	1399 Firestone ...	NULL	Bothell	79	98011
7	5672 Hale Dr.	NULL	Bothell	79	98011
8	6387 Scenic Ave...	NULL	Bothell	79	98011
9	8713 Yosemite Ct.	NULL	Bothell	79	98011
10	250 Race Court	NULL	Bothell	79	98011
11	1318 Lasalle Street	NULL	Bothell	79	98011
12	5415 San Gabrie...	NULL	Bothell	79	98011
13	9265 La Paz	NULL	Bothell	79	98011
14	8157 W. Book	NULL	Bothell	79	98011
15	4912 La Vuelta	NULL	Bothell	79	98011
16	40 Ellis St.	NULL	Bothell	79	98011
17	6696 Anchor Drive	NULL	Bothell	79	98011
18	1873 Lion Circle	NULL	Bothell	79	98011
19	3148 Rose Street	NULL	Bothell	79	98011
20	6872 Thornwoo...	NULL	Bothell	79	98011
21	5747 Shirley Drive	NULL	Bothell	79	98011
22	636 Vine Hill Way	NULL	Portland	58	97205
23	6657 Sand Point...	NULL	Seattle	79	98104
24	80 Sunview Terr...	NULL	Duluth	36	55802
25	9178 Jumping St.	NULL	Dallas	73	75201

1 of 200 Cell is Read Only.

Microsoft SQL Server Management Studio
File Edit View Project Debug Query Designer Tools Window Community Help
New Query
Change Type
Object Explorer
Connect
Tables
System Tables
dbo.AWBuildVersion
dbo.DatabaseLog
dbo.ErrorLog
HumanResources.Department
HumanResources.Employee
HumanResources.EmployeeAddress
HumanResources.EmployeeDepartr
HumanResources.EmployeePayHist
HumanResources.JobCandidate
HumanResources.Shift
Person.Address
Person.AddressType
Person.Contact
Person.ContactType
Person.CountryRegion
Person.StateProvince
Production.BillOfMaterials
Production.Culture
Production.Document
Production.Illustration
Production.Location
Production.Product
Production.ProductCategory
Production.ProductCostHistory
Production.ProductDescription
Production.ProductDocument
Production.ProductInventory
Production.ProductListPriceHistory
Production.ProductModel
Production.ProductModelIllustration
Production.ProductModelProductDe
Production.ProductPhoto
Ready

Figure 3-3 Edit top 200 rows output screen

6. Enter your data into the last row of the table for that data to be considered new (or *inserted*) data. The last row of the table will have the value NULL in each of the columns.

PAUSE. Leave the SSMS interface open for the next exercise.

The other method of using the `INSERT` clause is by writing a SQL statement in the text editor window. With the following syntax, this will provide the same result as using the graphical interface:

```
INSERT INTO <table_name> (<columns>)

VALUES (<values>)
```

The <`columns`> clause would contain your comma-separated list of the column names in the table you wish to include, and the <`values`> clause would contain the values you would like to insert.

You are not limited to inserting only one row at a time with the `INSERT` statement; instead, you can indicate multiple rows using a comma to separate them. This is similar to the commas in Excel.CSV files that separate columns of information imported for use in another database program, such as Access. But within an actual `INSERT` statement, each of the rows identified by commas will be enclosed within parentheses. Thus, an `INSERT` statement that adds two new employees to our employee table would appear as follows:

```
INSERT INTO employee (first_name, last_name, employee_id, department)

VALUES (‘David’, ‘Clark’, 610008, ‘shipping’),

    (‘Arnold’, ‘Davis’, 610009, ‘accounting’)
```

TAKE NOTE*
Although the columns list in the `INSERT` statement is entirely optional, it is recommended that you specify what columns you wish to use, as SQL will automatically assume your list of values includes all columns in the correct order.

This will output the following result:

```
(2 row(s) affected)
```

It really is as simple as that to harness the power of database modification and administration. Now, let’s explore some other types of data modification.

Updating Data and Databases

THE BOTTOM LINE
As a database administrator, you must understand how data is updated in a database, how to write update data to a database using appropriate `UPDATE` statements, and how to update a database using a table.

The function of the `UPDATE` statement is to change data in a table or view. Much like any of the data manipulation or modification clauses and statements within SQL, you can use this statement in either SSMS or a text editor window.

Using the UPDATE Statement

CERTIFICATION READY
What command is used to change existing data in a table?
3.3

The UPDATE clause allows you to modify data stored in tables using data attributes such as the following:

```
UPDATE <table_name>

SET <attribute> = <value>

WHERE <conditions>
```

As you've seen from the beginning of this lesson, you can read this type of SQL statement much as you would any sentence. Say you want to update a table in which you want a certain column identifier to reflect a certain value. Perhaps you want to have an attribute of a new supervisor (think of our employee example), Doug Able, being assigned to new employees for training purposes. That supervisor could have the attribute set for him or her as (looking back at our department table) an ID of 4, and the WHERE clause would be satisfied by having it match the NULL condition for our employees without a supervisor. Let's write that scenario UPDATE statement to update the previous example.

The first step would be to add a record in the department table with our new supervisor's name and department ID information using the INSERT statement:

```
INSERT INTO department (first_name, last_name, department_id)

VALUES (‘Doug’, ‘Able’, 4)
```

The output response would be as follows:

```
(1 row(s) affected)
```

Now, we need to update our employee table to reflect any employees who do not have an assigned department supervisor. Here, our UPDATE statement would look as follows:

```
UPDATE employee

SET department = 4

WHERE department IS NULL
```

The result is shown in Table 3-4.

TAKE NOTE*
If you don't specify what records to update, all records or rows will be updated with the new value. This is potentially harmful.

Table 3-4 NULL values in the department column

92	2010-04-03....	Dbo	CREATE_VIEW	dbo	vDVMPrep	CREATE VIEW [...	<EVENT_INSTA...
92	2010-04-03....	Dbo	CREATE_VIEW	dbo	vTimeSeries	CREATE VIEW [...	<EVENT_INSTA...
92	2010-04-03....	dbo	CREATE_VIEW	dbo	vTargetMail	CREATE VIEW [...	<EVENT_INSTA...
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Referring back to Table 3-1, notice that only one employee, David Cruze, did not have a department ID number assigned to him, meaning his value was NULL in the department column. Because we added a new supervisor to the department table in our `INSERT INTO` statement above, David Cruze now has an identified department number and has Doug Able as his assigned supervisor.

In comparison, to update a table using the SSMS graphical interface, you simply need to follow these steps:

- Open the SSMS interface.
- Open the table in which you wish to update data.
- Locate the row in which you wish to update the records within the Open Table view.

Deleting Data

THE BOTTOM LINE
All database administrators must understand how to delete data from single or multiple tables, as well how to ensure data and referential integrity by using transactions.

There are several different ways to remove rows from a table or view. You can identify and delete individual rows from the database using the `DELETE` syntax, delete all the rows using a `truncate table` statement, or remove the entire table using the `drop table` statement. Which method you choose depends entirely on your needs or the amount of data you need to remove.

CERTIFICATION READY
What commands are used to delete data from a database?
3.4

Using the DELETE Statement

You can use the DELETE statement to remove one or more rows in a table or view. This statement is structured as follows:

```
DELETE FROM <table_name>

WHERE <conditions>
```

You can use the DELETE statement in a variety of situations. For instance, you could delete all accounting department employees from a company's employee table if, because of a corporate takeover, they are no longer no longer employed by the company. You could delete this information using the following command:

TAKE NOTE*
Rows are not actually deleted from the table source (table_source) identified in the FROM clause, only from the table named in the DELETE clause (table_ or_view).

```
DELETE FROM employee

WHERE department = ‘accounting’
```

This would return the following value:

```
(1 row(s) affected)
```

This result shows that the one employee who worked in the accounting department has now been removed from the employee table.

Truncating a Table with TRUNCATE TABLE

Perhaps you would like to delete all the rows from a particular table. You could use the TRUNCATE TABLE statement to do this, although you may also be tempted to use DELETE and the where condition. This latter technique would produce the same output, but it could take a great deal of time if you are deleting rows from very large databases. Thus, the TRUNCATE TABLE statement would be the better option. The syntax of the statement looks like this:

```
TRUNCATE TABLE <table_name>
```

Each successfully executed result from SQL will appear as follows:

```
Command(s) completed successfully.
```

The `TRUNCATE TABLE` statement removes the actual data from within the table, but it leaves the table structure in place for future use.

Deleting a Table with DROP TABLE

Perhaps you want to delete an entire table because it is obsolete, because it contains too much data to move around, or for some other reason. Removing an entire table involves use of the DROP TABLE statement, which looks like this:

```
DROP TABLE <table_name>
```

Using Referential Integrity

One of the most important steps in planning a database, creating tables, manipulating data, and so forth is setting up a proper security model, of which referential integrity forms a part. A larger problem in database manipulation and maintenance is that sometimes SQL Server data is lost and must be recovered. If proper safeguards are not in place as part of the backup and recovery process, recovering lost data could prove a very harrowing ordeal.

One safeguard measure that can be taken with regard to database tables is the use of referential integrity practice methods. One of the most common mistakes in database manipulation is the accidental loss of entire tables. The best way to avoid this type of situation in the first place is to ensure that your database uses ***referential integrity***. Referential integrity does not allow deletion of tables unless all of the related tables are deleted using a cascading delete.

This brings us to another best-practice method: using ***transactions*** when updating data. Data is most commonly deleted, truncated, or accidentally updated during regular maintenance tasks, and one of the best ways to keep this from occurring is to use transactions when updating data. Inserting a simple `begin tran` before an actual SQL statement is a good place to start, and if you have executed everything correctly, a `COMMIT` statement is issued from SQL. If there is an error in any of the statements, a `ROLLBACK` is issued from SQL.

A sample transaction statement might appear as follows:

```
BEGIN TRAN

DELETE FROM <table_name>
```

What happens at this juncture is that you would verify that what you did really did happen and then issue a `COMMIT` statement to save those changes, or else issue a `ROLLBACK` to undo them. Many times mistakes occur through simple errors, and if you use the `BEGIN TRAN` and a `COMMIT` or `ROLLBACK` while performing maintenance tasks, you will catch most accidents before they happen.

SKILL SUMMARY

IN THIS LESSON, YOU LEARNED THE FOLLOWING:

- The SQL command for retrieving any data from a database is SELECT.
- There are only three things you need to identify in your statement in order to form a proper SELECT query: what columns to retrieve, what tables to retrieve them from, and what conditions, if any, the data must satisfy.
- A BETWEEN clause allows you to specify the range to be used in a "between x and y" query format.
- The NOT keyword is used to search data in terms of what you don't want in your output.
- The UNION clause allows you to combine the results of any two or more queries into a resulting single set that will include all the rows belonging to the query in that union.
- The EXCEPT clause returns any distinct values from the left query that are not also found on the right query, whereas the INTERSECT clause returns any distinct values not found on both the left and right sides of this operand.
- The JOIN clause allows you to combine related data from multiple table sources.
- To insert data, you can use SSMS or the INSERT statement.
- The function of the UPDATE statement is to change data in a table or a view.
- The DELETE statement removes rows from a table or a view.
- The TRUNCATE TABLE statement removes data from within a table but leaves the table structure in place for future use.
- An entire table can be removed with the DROP TABLE command. The best way to avoid the accidental deletion of entire tables is to use referential integrity. Referential integrity does not allow deletion of tables unless all of the related tables are deleted using a cascading delete.