Module 7: Using DML to Modify Data

Contents:

Module Overview

Lesson 1: Adding Data to Tables

Lesson 2: Modifying and Removing Data

Lesson 3: Generating Automatic Column Values

Lab: Using DML to Modify Data

Module Review and Takeaways

Module Overview

Transact-SQL (T-SQL) data manipulation language (DML) is the subset of the SQL Language that contains commands to add and modify data column values, within rows, within tables. In this module, you will learn the basics of using INSERT to add column values to rows within tables, using UPDATE to make changes to column values to rows within tables, and using DELETE to remove complete rows from tables. You can also use the TRUNCATE command to delete all rows within a table quickly, without incurring an overhead that protects accidental deletion of rows when using the DELETE statement.

You will also learn how to generate sequences of numbers using the IDENTITY property of a column, in addition to the sequence object, which is a stand-alone object that can be applied to many columns—in the same or different tables—to gain consistency between identities within different tables.

You can use the MERGE command to change existing columns within rows of a destination table, based on the values stored within a source table, and comparisons between the source and destination table contents.

Objectives

After completing this module, you will be able to:

- Write T-SQL statements that insert column values into rows within the tables.
- Write T-SQL statements that modify values in columns, within rows, within tables.
- Write T-SQL statements that remove existing rows from tables.
- Appreciate the importance of the WHERE clause when using data modification language (DML).
- Appreciate T-SQL statements that automatically generate values for columns and see how this affects you when using DML.
- Understand the use of the MERGE statement to compare and contrast two tables and direct different DML statements, based on their content comparisons.

Lesson 1: Adding Data to Tables

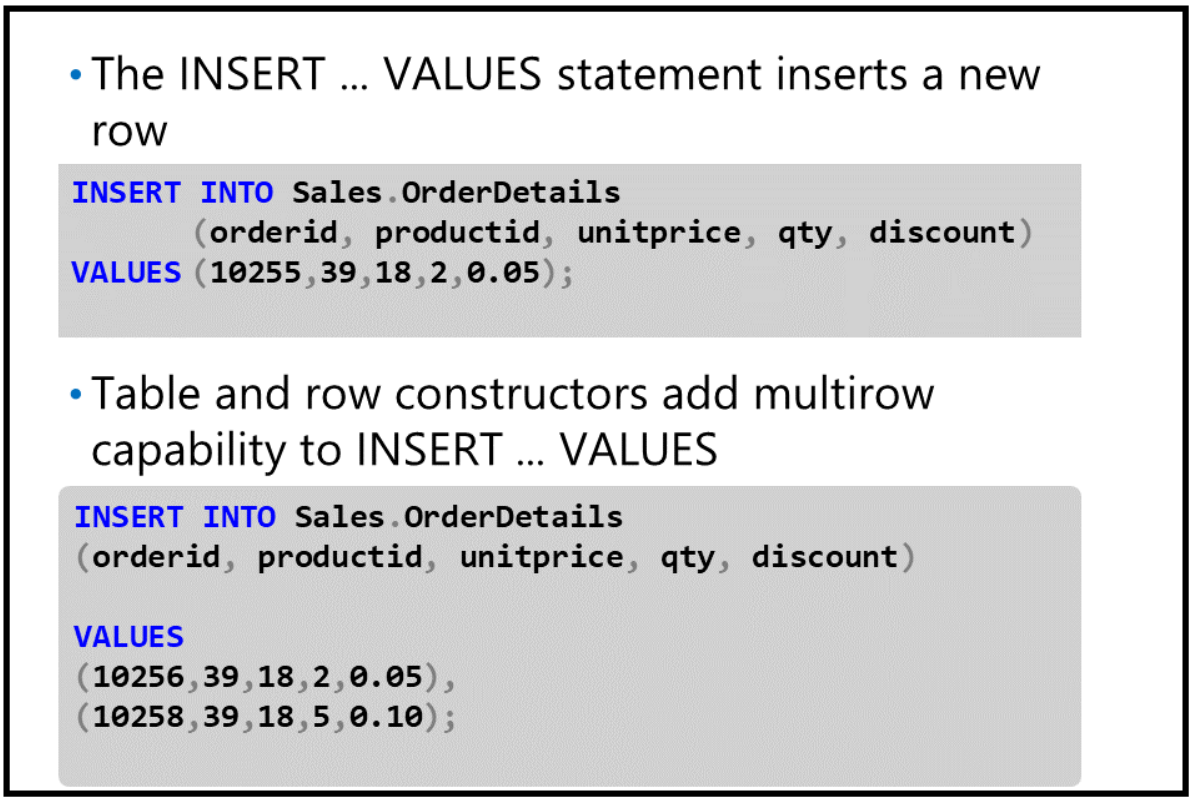
In this lesson, you will learn how to write queries that add new rows with column values to tables.

Lesson Objectives

After completing this lesson, you will be able to:

- Write queries that use the INSERT statement to add data to tables.
- Use the INSERT statement with SELECT and EXEC clauses.
- Use SELECT INTO to create and populate tables without resort to data definition language (DDL).
- Describe the behavior of default constraints when rows are inserted into a table.

Using INSERT to Add Data



In SQL, the INSERT statement is used to add one or more rows to a table. There are several forms of the statement.

INSERT Syntax

```
INSERT [INTO] <Table or View> [(column_list)]  -- column_list is optional but the code is
safer with it
VALUES ([ColumnName or an expression or DEFAULT or NULL], .…n)
```

With this form, called INSERT VALUES, you can specify the columns that will have values placed in them and the order in which the data will be presented for each row inserted into the table. In addition, you can provide the values for those columns as a comma separated list.

When inserting values, the keyword DEFAULT means the predefined value that should be presented where a column value has not been listed, but a value is required.

When inserting values, the keyword NULL means the predefined value that should be presented where a column value has not been listed and a value is not required.

The following example shows the use of the INSERT VALUES statement:

INSERT VALUES Example

```
USE TSQL
GO

INSERT INTO Sales.OrderDetails (OrderID, ProductID, UnitPrice, Qty, Discount)
VALUES (10248, 39, 18, 2, 0.05)
```

If the column list is omitted, a column value or the keyword (DEFAULT or NULL) must be specified for each column, in the order in which they are defined in the table. If a value is not specified for a column that does not have a value automatically assigned, such as through an IDENTITY column, the INSERT statement will fail.

In addition to inserting a single row at a time, the INSERT VALUES statement can be used to insert multiple rows by providing multiple comma separated sets of values, themselves separated by commas, like this:

(1,2,3), (3,2,1), (2,2,2).

Insert Rows

```
USE TSQL
GO
INSERT INTO Sales.OrderDetails(orderid, productid, unitprice, qty, discount)
VALUES  (10249,39,18,2,0.05), (12002,39,18,5,0.10);

-- Some people prefer this alternative layout for multiple row inserts

INSERT INTO Sales.OrderDetails(orderid, productid, unitprice, qty, discount)
VALUES (10250,39,18,2,0.05)
,             (10251,39,18,5,0.10)
,             (10252,39,18,2,0.05)
,             (10254,39,18,5,0.10);
```

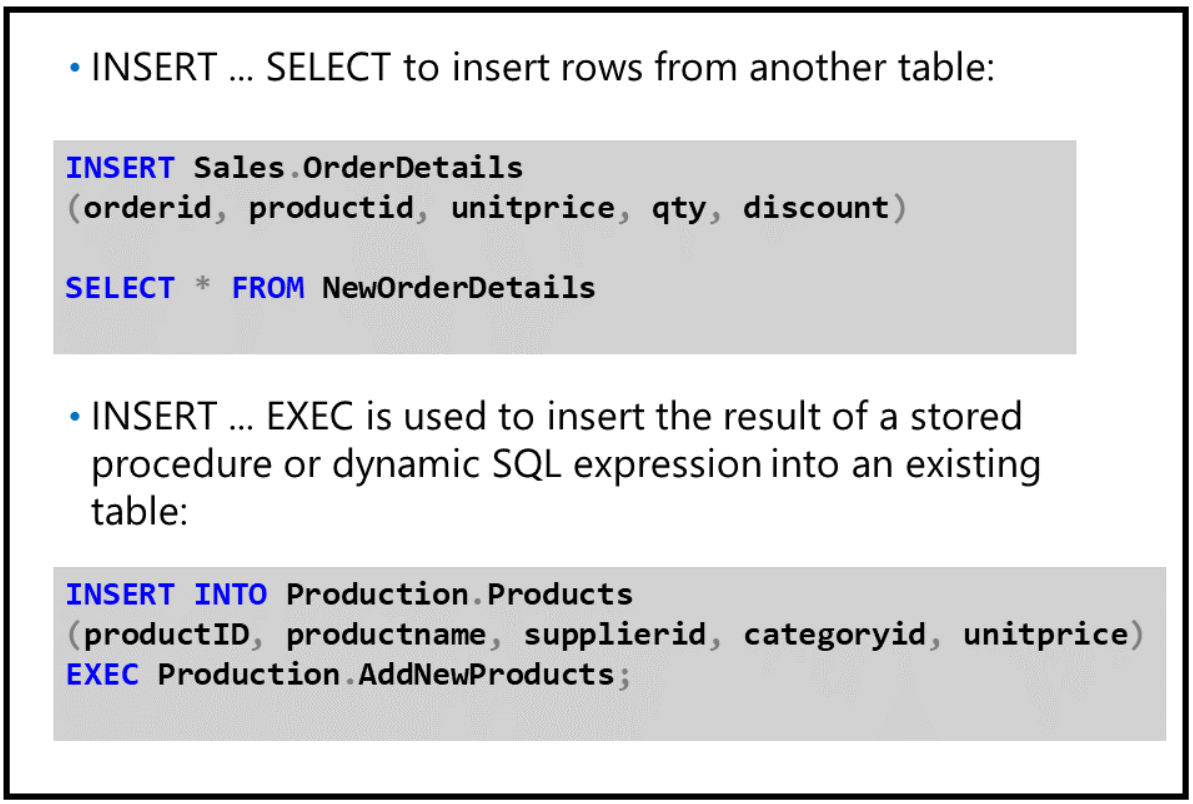
INSERT (Transact-SQL)

http://aka.ms/ifsc6i

Table Value Constructor (Transact-SQL)

http://aka.ms/rnwb93

Using INSERT with Data Providers



Beyond specifying a literal set of values in an INSERT statement, T-SQL also supports using the output of other operations to provide values for INSERT. You can pass the results of a SELECT clause or the output of a stored procedure to the INSERT clause.

To use the SELECT statement with an INSERT statement, build a SELECT clause to replace the VALUES clause. With this form, called INSERT SELECT, you can insert the set of rows returned by a SELECT query into a destination table. The use of INSERT SELECT presents the same considerations as INSERT VALUES:

- You may optionally specify a column list following the table name.
- You must provide column values or DEFAULT, or NULL, for each column.

INSERT SELECT

```
INSERT [INTO] <table or view> [(column_list)]
SELECT <column_list> FROM <table_list>...;
```

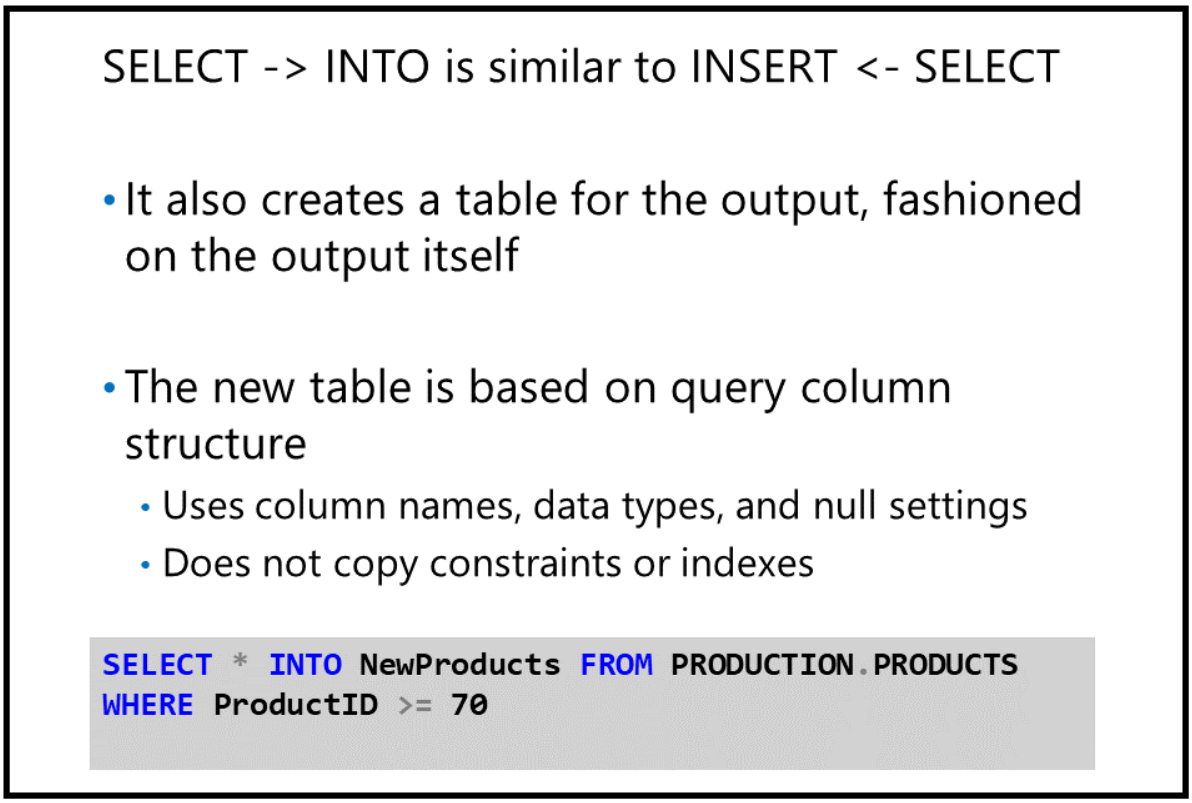
Result sets from stored procedures (or even dynamic batches) may also be used as input to an INSERT statement. This form of INSERT, called INSERT EXEC, is conceptually similar to INSERT SELECT and will present the same considerations.

Inserting Rows into a Table from a Stored Procedure

```
INSERT INTO Production.Products (productID, productname, supplierid, categoryid, unitprice)
EXEC Production.AddNewProducts;
GO
```

Note: The example above references a procedure that is not supplied with the course database. Code to create it appears in the demonstration for this module.

Using SELECT INTO



In T-SQL, you can use the SELECT INTO statement to create and populate a new table with the results of a SELECT query. SELECT INTO cannot be used to insert rows into an existing table. A new table is created, with a schema defined by the columns in the SELECT list. Each column in the new table will have the same name, data type, and nullability as the corresponding column (or expression) in the SELECT list.

To use SELECT INTO, add INTO <new_target_table_name> in the SELECT clause of the query, just before the FROM clause.

INTO clause (Transact-SQL)

http://aka.ms/qae4zn

```
SELECT column1
,            column2
 …
INTO   NewTable FROM  OldTable
```

```
SELECT  ordered
,             custid
,             empid
,             orderdate
,             shipcity
,             shipregion
,             shipcountry
INTO     Sales.OrdersExport FROM Sales.Orders
WHERE  empid = 5;
```

> **Note:** SELECT INTO creates a new table in the default FileGroup. Starting with SQL Server 2017 you can specify the FileGroup using the ON keyword.

> **Note:** The use of SELECT INTO requires permissions to create table objects in the destination database. Do not try to put this clause inside a view, because it will only work once. If a table cannot be created when the view is activated, an error will occur after the first use of the view.

Check Your Knowledge

Select the best answer

You want to populate three columns of an existing table with data from another table in the same database. Which of the following types of query should you use?

INSERT INTO <TableName> (<Columns,…>) VALUES (<Column Value> …)

INSERT INTO <DestinationTableName> SELECT <Columns> FROM <SourceTableName>

INSERT INTO <DestinationTableName> EXECUTE usp_SomeStorerdProcedure

SELECT <Columns,…> INTO DestinationTableName FROM SourceTableName

SELECT <Columns,…> INTO SourceTableName FROM DestinationTableNAme

Check answer Show solution Reset

INSERT … VALUES creates new records with data you supply in the query. INSERT … EXEC creates new records from the results of a stored procedure. SELECT INTO creates a new table from the results of a SELECT query. Because you

want to populate an existing table with data from another table, use INSERT … SELECT. The final option is to re-enforce the difference in placement of source and destination table names in a query that takes data from one table and places it in the other table.

Demonstration: Adding Data to Tables

In this demonstration, you will see how to:

- ADD data to tables using fully qualified parameters.
- ADD data to tables with partially qualified parameters.
- Understand how to use the OUTPUT clause to monitor data changes during data INSERT.
- Understand how to insert data into a table that is produced by a stored procedure.
- Use the SELECT INTO keywords to insert data into a table.

Demonstration Steps

INSERT Data into a Table

1. Start the **MT17B-WS2016-NAT**, **20761C-MIA-DC**, and **20761B-MIA-SQL** virtual machines, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod07\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. In the Command Prompt window press **y**, and then press Enter.
5. When the script has finished, press Enter.
6. Open SQL Server Management Studio, and connect to the **MIA-SQL** database engine instance using Windows authentication.
7. On the **File** menu, point to **Open**, and then click **Project/Solution**.
8. In the **Open Project** dialog box, navigate to the **D:\Demofiles\Mod07\Demo** folder, click **Demo.ssmssln**, and then click **Open**.
9. In Solution Explorer, expand **Queries**, and double-click **11 - Demonstration A.sql**.
10. Highlight the code **USE TSQL GO**, and click **Execute**.
11. First you will populate a table with some data from a stored procedure. Highlight the code under the comment that begins **-- First try the INSERT by stored procedure**:

```
INSERT INTO Production.Products
        (       productID
        ,       productname
        ,       supplierid
        ,       categoryid
```

```
        ,       unitprice)
EXEC Production.AddNewProducts;
```

Click **Execute**. You will receive a message saying that the procedure is not there.

12. Highlight the code below the comment **--Create a backup of the Products with a chosen ID**, and click **Execute**.

```
DROP TABLE IF EXISTS NewProducts
GO

SELECT * INTO NewProducts
FROM PRODUCTION.PRODUCTS WHERE ProductID >= 70
```

You are creating a new table for NewProducts where the Product ID >= 70.

13. You are also going to create a NewOrderDetails table that will contain rows for those products that have been transferred into NewProducts. To do this, highlight the code under the comment **-- Create a backup of the Order Details for the chosen productID**, up to the point shown in the code section for the next step below, and click **Execute**:

```
DROP TABLE IF EXISTS NewOrderDetails
GO

SELECT * INTO NewOrderDetails
FROM SALES.OrderDetails WHERE ProductID >= 70

-- Delete the copied data from the original tables
DELETE FROM SALES.OrderDetails
OUTPUT DELETED.*
WHERE ProductID >= 70

DELETE FROM Production.Products
OUTPUT DELETED.*
WHERE ProductID >= 70

-- check that they have been transferred safely
SELECT * FROM NewProducts
SELECT * FROM NewOrderDetails

SELECT * FROM SALES.OrderDetails
WHERE productid >= 70

SELECT * FROM Production.Products
WHERE productid >= 70
```

14. Highlight the code below the comment **Now we can put back the rows from the NewTables, using the INSERT statement**, and click **Execute**.

```
DROP PROCEDURE IF EXISTS Production.AddNewProducts
GO

CREATE PROCEDURE Production.AddNewProducts
AS
BEGIN
SELECT Productid, productname, SupplierID, CategoryID, Unitprice FROM NewProducts
END
```

When you click Execute, SQL Server creates the stored procedure that you were missing when you tried to run it at the beginning of the demo.

15. Now you need to populate the original products table with the data within the secondary table as if you were adding new rows. Highlight the code below the comment **Having created it, we can run it to feed the missing rows into the Products table**:

```
INSERT INTO Production.Products (productid, productname, supplierid, categoryid,
unitprice)
EXEC Production.AddNewProducts;

SELECT * FROM Production.Products
WHERE productid >= 70
```

Click **Execute** to transfer the rows and see that they have been transferred.

16. For the other table, you will use the SELECT INSERT statement. Highlight the code below the comment **-- The OrderDetails will be put back using INSERT .. SELECT**, and click **Execute**:

```
INSERT Sales.OrderDetails (orderid, productid, unitprice, qty, discount)
OUTPUT INSERTED.*
SELECT * FROM NewOrderDetails
```

17. Having seen various ways to add data to a new or existing table, you can clean up the database by dropping the objects used in this demo. Highlight the rest of the code below **-- Clean up the database** and click **Execute**:

```
DROP TABLE NewProducts
GO

DROP TABLE NewOrderDetails
GO
```

```
DROP PROCEDURE Production.AddNewProducts
```

18. Close SQL Server Management Studio, without saving any changes.

Lesson 2: Modifying and Removing Data

In this lesson, you will learn how to write queries that modify or remove rows from a target table. You will also learn how to perform a MERGE between source and destination tables, in which new rows are added and existing rows are modified in the same operation.

Lesson Objectives

After completing this lesson, you will be able to:

- Write queries that modify existing rows using UPDATE.
- Write queries that modify existing rows and insert new rows using MERGE.
- Write queries that remove existing rows using DELETE.
- Remove all rows from a table using TRUNCATE.

Using UPDATE to Modify Data

- UPDATE changes all rows in a table or view
- Unless rows are filtered with a WHERE clause or constrained with a JOIN clause
- Column values are changed with the SET clause

```
UPDATE Production.Products
   SET   unitprice = (unitprice * 1.04)
WHERE   categoryid =  1 AND discontinued = 0
;
```

```
UPDATE Production.Products
   SET     unitprice *= 1.04
                    -- Using compound
                    -- assignment operators
WHERE   categoryid =  1 AND discontinued = 0;
```

SQL Server provides the UPDATE statement to change existing data in a table or a view. UPDATE operates on a set of rows, either defined by a condition in a WHERE clause or defined in a join. It uses a SET clause that can perform one or more assignments, separated by commas, to allocate new values to the target. The WHERE clause in an UPDATE statement has the same structure as a WHERE clause in a SELECT statement.

> **Note:** It's important to note that an UPDATE without a corresponding WHERE clause, and/or a join, will target all rows that are not filtered out of the operation. Use the UPDATE statement with caution.

UPDATE Syntax

```
UPDATE <TableName>
SET
      <ColumnName1> = { expression | DEFAULT | NULL }
      {,…n}
```

Any column omitted from the SET clause will not be modified by the UPDATE operation.

UPDATE Example

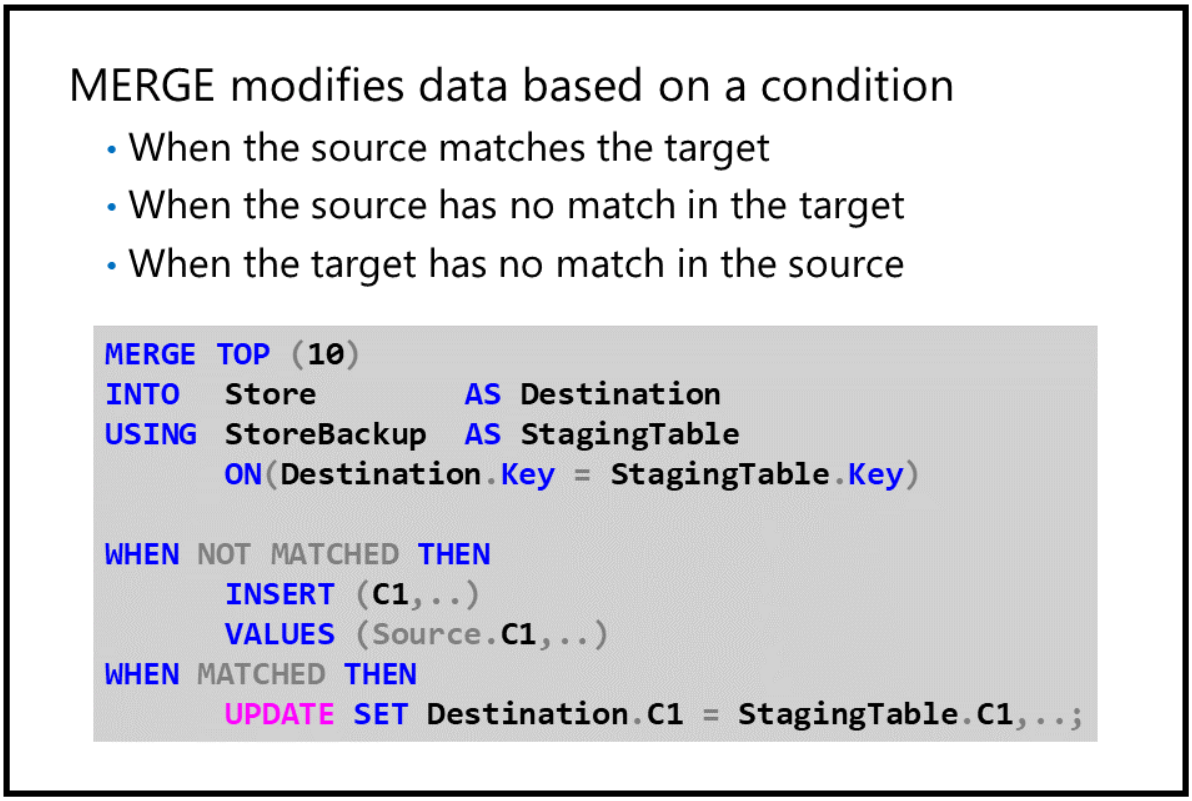
```
UPDATE Production.Products
      SET unitprice = (unitprice * 1.04)
WHERE categoryID = 1
AND     discontinued = 0;
```

> **Note:** In an earlier module, you learned that T-SQL supports compound assignment operators. These can be used when assigning values to columns using the SET statement within the update clause, as shown below:
> UPDATE Production.Products
> SET unitprice *= 1.04
> WHERE categoryID = 1
> AND discontinued = 0;

UPDATE (Transact-SQL)

http://aka.ms/sbikqm

Using MERGE to Modify Data



In database operations, there is a common need to perform a SQL MERGE operation, in which some rows within a destination table are updated or deleted and new rows are inserted from a source data table. The oldest versions of SQL Server, before support for the MERGE statement was added, required multiple operations to update and insert data into a destination table. You can use the MERGE statement to insert, update, and even delete rows from a destination table, based on a join to a source data set, all in a single statement.

MERGE modifies data, based on one or more conditions:

- When the source data matches the data in the target, it updates data.
- When the source data has no match in the target, it inserts data.
- When the target data has no match in the source, it deletes the target data.

> **Note:** Because the T-SQL implementation of MERGE supports the WHEN NOT MATCHED BY SOURCE clause, MERGE is more than just an upsert operation—because it also deletes, it is a delupsert or something similar.

The following code shows the general syntax of a MERGE statement:

The MERGE Example

```
MERGE INTO schema_name.table_name AS TargetTbl
    USING (SELECT <select_list>) AS SourceTbl
    ON (TargetTbl.col1 = SourceTbl.col1)
```

```
    WHEN MATCHED THEN
        UPDATE SET TargetTbl.col2 = SourceTbl.col2
                  WHEN NOT MATCHED THEN
    INSERT (<column_list>)
    VALUES (<value_list>);
```

The following example shows the use of a MERGE statement to update shipping information for existing orders, or to insert rows for new orders when no match is found. Note that this example is for illustration only and cannot be run using the sample database for this course.

MERGE Example

```
MERGE top (10) INTO  Store             AS Destination    -- Known in online help as Target,
which is a reserved word
        USING      StoreBackup AS StagingTable -- Known in online help as the source, which
is also a reserved word
        ON                      (Destination.BusinessEntityID =
StagingTable.BusinessEntityID)
                                                                              -- the matching
control columns
WHEN NOT MATCHED THEN
        INSERT (        BusinessEntityID
                        ,       Name
                        ,       SalesPersonID
                        ,       Demographics
                        ,       rowguid
                        ,       ModifiedDate
                        )
        VALUES (        StagingTable.BusinessEntityID
                        ,       StagingTable.Name
                        ,       StagingTable.SalesPersonID
                        ,       StagingTable.Demographics
                        ,       StagingTable.rowguid
                        ,       StagingTable.ModifiedDate
                        );
```

MERGE (Transact-SQL)

http://aka.ms/nbsfg7

Demonstration: Manipulating Data Using the UPDATE and DELETE Statements and MERGING Data Using Conditional DML

In this demonstration, you will see how to:

- UPDATE row and column intersections within tables.

- DELETE complete rows from within tables.
- Apply multiple data manipulation language (DML) operations by using the MERGE statement.
- Understand how to use the OUTPUT clause to monitor data changes during DML operations.
- Understand how to access prior and current data elements, in addition to showing the DML operation performed.

Demonstration Steps

Update and Delete Data in a Table

1. Start the **MT17B-WS2016-NAT**, **20761C-MIA-DC**, and **20761B-MIA-SQL** virtual machines, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod07\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. In the Command Prompt window press **y**, and then press Enter.
5. When the script has finished, press Enter.
6. Start SQL Server Management Studio, and connect to the **MIA-SQL** database engine instance using Windows authentication.
7. On the **File** menu, point to **Open**, and then click **Project/Solution**.
8. In the **Open Project** dialog box, navigate to the **D:\Demofiles\Mod07\Demo** folder, click **Demo.ssmssln**, and then click **Open**.
9. In Solution Explorer, open the **21 - Demonstration B.sql** script file.
10. Highlight the code **USE AdventureWorks GO**, and click **Execute**.
11. Select the code under **USE AdventureWorks GO**, and then click **Execute**.
12. Select the code under the comment **Remove the copied rows from the store table**, and then click **Execute**.
13. Select the code under the comment **Show that they have been removed**, and then click **Execute**.
14. Select the code under the comment **Use the Merge statement to put them back**, and then click **Execute**.
15. Select the code under the comment **SELECT * FROM Sales.Store where 1 = 0 -- used to extract column names for all columns, without cost of data access**, and then click **Execute**.
16. Select the code under the comment **Use the Merge statement to Change the names back**, and then click **Execute**.
17. Select the code under the comment **Ensure that the environment has been restored to the state it was in before the changes were made**, and then click **Execute**.
18. Select the code under the comment **Clean up the database**, and then click **Execute**.
19.

Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Discovery

A user cannot delete records in the Cars table by using a DELETE statement. The query was intended to remove all pool cars that have been sold. The query used was:

```
DELETE
FROM  Scheduling.Cars
WHERE Cars.DateSold <> NULL
```

What mistake did the user make?

Show solution Reset

An expression comparison operator '<>' was used instead of the IS NOT NULL clause. Because a comparison operator between the value NULL and a NULL column value will never return true, this will not lead to any rows being deleted.

Lesson 3: Generating Automatic Column Values

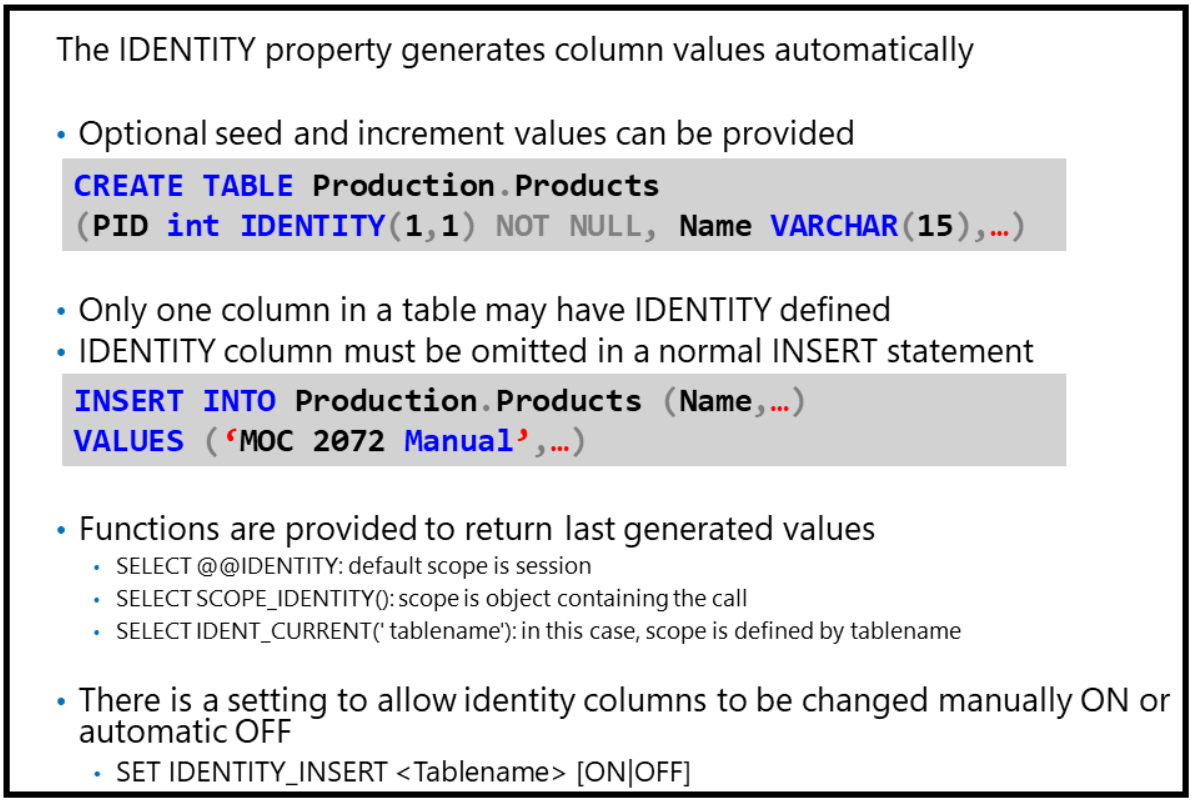
In this lesson, you will learn how to automatically generate a sequence of numbers for use as column values.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how to use the IDENTITY property of a column to generate a sequence of numbers when rows are inserted into a table.
- Describe how to use a sequence object in SQL Server to generate numbers that can be used within a column, in one or more tables.

Using IDENTITY



You may need to automatically generate sequential values for a column in a table. SQL Server provides two mechanisms for generating values:

- IDENTITY property, for all versions of SQL Server.
- Sequence object in SQL Server 2012 and later.

Each mechanism can be used to provide sequential numbers when rows are inserted into a table. With the sequence object, the number variable can be used efficiently in multiple tables.

To use the IDENTITY property, define a column using a numeric data type with a scale of 0—meaning whole numbers only—and include the IDENTITY keyword.

An optional seed (starting value), and an increment (step value) can also be specified. Leaving out the seed and increment will set them both to 1.

Only one column in a table may have the IDENTITY property set; it is customary for it to be an alternate primary key.

IDENTITY Example

```
CREATE TABLE Employee
(
    EmployeeID int IDENTITY(100, 10) NOT NULL
, …
)
```

When an IDENTITY property is defined on a column, INSERT statements against the table do not reference the IDENTITY column. SQL Server will generate a value using the next available value for the column. If a value must be explicitly assigned to an IDENTITY column, the SET IDENTITY INSERT statement must be executed to override the default behavior of the IDENTITY column.

For more information, see SET IDENTITY_INSERT (Transact-SQL) in Microsoft Docs:

SET IDENTITY_INSERT (Transact-SQL)

http://aka.ms/l4wavg

When a value is assigned to a column by the IDENTITY property, the value may be retrieved like any other value in a column. Values generated by the IDENTITY property are unique within a table. However, without a constraint on the column (such as a PRIMARY KEY or UNIQUE constraint), uniqueness is not enforced after the value has been generated.

To return the most recently assigned value within the same session and scope, such as a stored procedure, use the SCOPE_IDENTITY() function. The legacy @@IDENTITY function will return the last value generated during a session, but it does not distinguish scope. You can use SCOPE_IDENTITY() for most purposes.

To reset the IDENTITY property by assigning a new seed, use the DBCC CHECKIDENT statement.

For more information, see DBCC CHECKIDENT (Transact-SQL) in Microsoft Docs:

DBCC CHECKIDENT (Transact-SQL)

http://aka.ms/g3mejh

IDENTITY_CACHE

A new option in SQL Server 2017 allows you to enable or disable the identity cache. This is on by default, and improves the performance of INSERT statement for tables with identity columns.

Set the IDENTITY_CACHE to OFF if you want to avoid gaps in the numbering when a server restarts or fails over to another server. Note that gaps can occur in the numbering for other reasons.

IDENTITY_CACHE

```
ALTER DATABASE SCOPED CONFIGURATION SET IDENTITY_CACHE=ON;
```

For more information about IDENTITY_CACHE, see Microsoft Docs:

ALTER DATABASE SCOPED CONFIGURATION (Transact-SQL)

https://aka.ms/lmdlno

Check Your Knowledge

Select the best answer

You are using an IDENTITY column to store the sequence in which orders were placed in a given year. It is a new year and you want to start the count again from 1. Which of the following statements should you use?

OrderSequence int IDENTITY(1,1) NOT NULL

SET IDENTITY INSERT

SCOPE_IDENTITY()

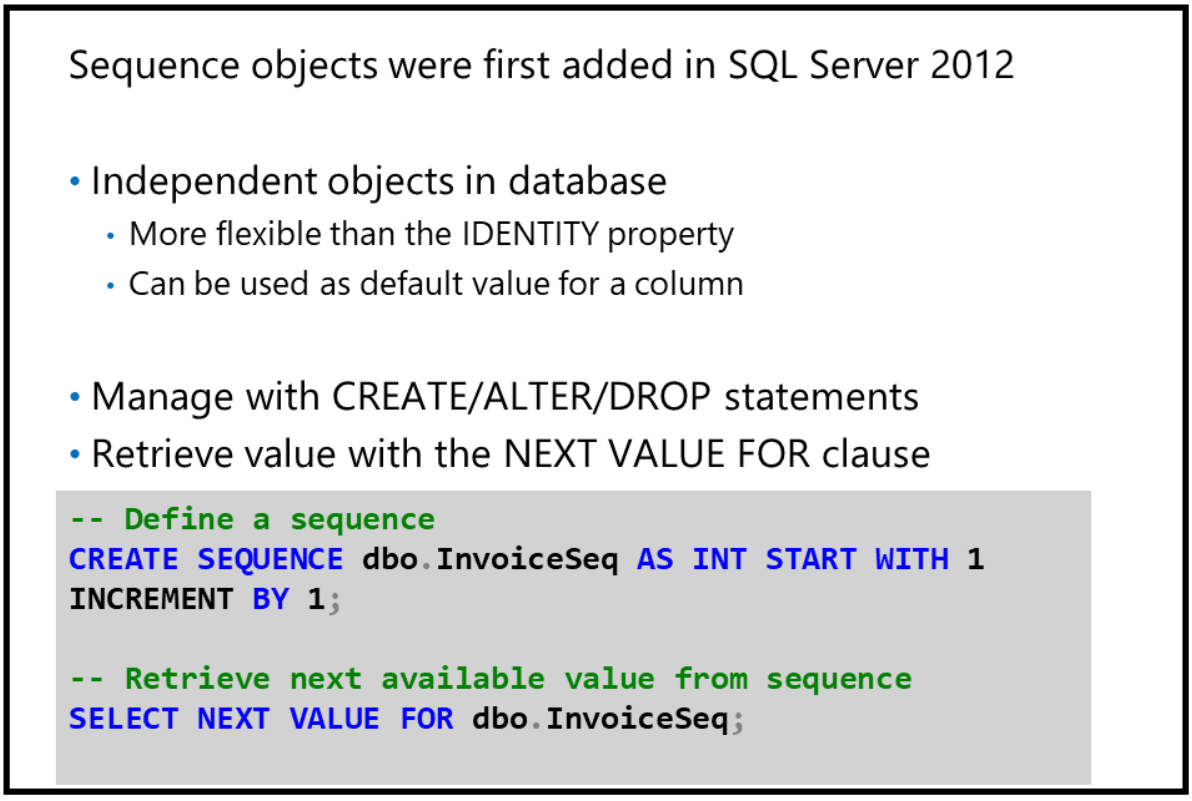
DBCC CHECKIDENT

CREATE SEQUENCE

Check answer Show solution Reset

OrderSequence int IDENTITY(1,1) NOT NULL

Using Sequences



As you have learned, the IDENTITY property is used to generate a sequence of values for a column within a table. However, the IDENTITY property is not suitable for coordinating values across multiple tables within a database. Database administrators and developers need to create tables of numbers manually to provide a pool of sequential values across tables.

SQL Server 2012 provides the new sequence object, an independent database object that is more flexible than the IDENTITY property, and can be referenced by multiple tables within a database. The sequence object is created

and managed with typical data definition language (DDL) statements such as CREATE, ALTER, and DROP. SQL Server provides a command for retrieving the next value in a sequence, such as within an INSERT statement or a default constraint in a column definition.

To define a sequence, use the CREATE SEQUENCE statement, optionally supplying the data type (must be an integer type or decimal/numeric with a scale of 0), the starting value, an increment value, a maximum value, and other options related to performance.

For more information, see CREATE SEQUENCE (Transact-SQL) in Microsoft Docs:

CREATE SEQUENCE (Transact-SQL)

http://aka.ms/lquwo6

To retrieve the next available value from a sequence, use the NEXT VALUE FOR function. To return a range of multiple sequence numbers in one step, use the system procedure sp_sequence_get_range.

SEQUENCE Example

```
CREATE SEQUENCE dbo.demoSequence
    AS INT
    START WITH 1
    INCREMENT BY 1;
GO

CREATE TABLE dbo.tblDemo
     (SeqCol int PRIMARY KEY,
      ItemName nvarchar(25) NOT NULL);
GO

INSERT
    INTO dbo.tblDemo (SeqCol,ItemName)
    VALUES                   (NEXT VALUE FOR dbo.demoSequence, 'Item');
GO
```

When you use a select statement against the table, you will see that a sequence value is inserted for the new row.

Lab: Using DML to Modify Data

Scenario

You are a database developer for Adventure Works and need to create DML statements to update data in the database to support the website development team. The team need T-SQL statements that they can use to carry out updates to data, based on actions performed on the website. You will supply template DML statements that they can modify to their specific requirements.

Objectives

After completing this lab, you will be able to:

- Insert records.
- Update and delete records.

Lab Setup

Estimated Time: 30 minutes

Virtual machine: **20761C-MIA-SQL**

User name: **ADVENTUREWORKS\STUDENT**

Password: **Pa55w.rd**

Exercise 1: Inserting Records with DML

Scenario

You need to add a new employee to the TempDB.Hr.Employee table and test the required T-SQL code. You can then pass the T-SQL code to the human resources system's web developers, who are creating a web form to simplify this task. You also want to add all potential customers to the Customers table to consolidate those records.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Insert a Row
3. Insert a Row with a SELECT Statement As the Data Provider

Detailed Steps ▲

Task 1: Prepare the Lab Environment

> **Note:** The exercises will be performed within the TempDB database so that none of the real data is affected. Two scripts are used to set up the environment for the lab—both are included in the project for the lab, along with a sample solution for each exercise.
> If you need to start again, open and execute the cleanup script, followed by the setup script; you have a clean environment and can try again.

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab07\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Insert a Row

1. Open the project file **D:\Labfiles\Lab07\Starter\Project\Project.ssmssln** and execute the **01 setup.sql** query.
2. Write an INSERT statement to add a record to the Employees table within the TempDB.HR.Employees table, with the following values:
   - Title: Sales Representative
   - Titleofcourtesy: Mr
   - FirstName: Laurence
   - Lastname: Grider
   - Hiredate: 04/04/2013
   - Birthdate: 10/25/1975
   - Address: 1234 1st Ave. S.E.
   - City: Seattle
   - Country: USA
   - Phone: (206)555-0105

Detailed Steps ▲

Task 3: Insert a Row with a SELECT Statement As the Data Provider

- Write an INSERT statement to add all the records from the PotentialCustomers table to the Customers table.

Result: After successfully completing this exercise, you will have one new employee and three new customers.

Exercise 2: Update and Delete Records Using DML

Scenario

You want to update the use of contact titles in the database to match the most commonly-used term in the company —making searches more straightforward. You also want to remove the three potential customers who have been added to the Customers table.

The main tasks for this exercise are as follows:

1. Update Rows
2. Delete Rows

Detailed Steps ▲

Task 1: Update Rows

- Write an UPDATE statement to update all the records in the Customers table that have a city of **'**Berlin' and a contacttitle of 'Sales Representative' to have a contacttitle of **'**Sales Consultant'.

Detailed Steps ▲

Task 2: Delete Rows

- Write a DELETE statement to delete all the records in the PotentialCustomers table which have the contactname of 'Taylor, Maurice', 'Mallit, Ken', or 'Tiano, Mike', as these records have now been added to the Customers table.

Result: After successfully completing this exercise, you will have updated all the records in the Customers table that have a city of Berlin and a contacttitle of Sales Representative, to now have a contacttitle of Sales Consultant. You will also have deleted the three records in the PotentialCustomers table, which have already been added to the Customers table.

Review Question(s)

Check Your Knowledge

Discovery

What attributes of the source columns are transferred to a table created with a SELECT INTO query?

Show solution Reset

Name, data type, and whether the column is NULL enabled.

Check Your Knowledge

Discovery

The presence of which constraint prevents TRUNCATE TABLE from executing successfully?

Show solution Reset

A foreign key reference to the table.

Module Review and Takeaways

In this module, you have learned how to:

- Write T-SQL statements that insert column values into rows within the tables.
- Write T-SQL statements that modify values in columns, within rows, within tables.
- Write T-SQL statements that remove existing rows from tables.
- Appreciate the importance of the WHERE clause when using data modification language (DML).
- Appreciate T-SQL statements that automatically generate values for columns and how this affects you when using DML.
- Understand the use of the MERGE statement to compare and contrast two tables and direct different DML statements, based on their content comparisons.

Common Issues and Troubleshooting Tips

Common Issue	Troubleshooting Tip
You are partway through the exercises and want to start again from the beginning. You run the setup script within the solution and receive lots of error messages. This might occur if you have tried to execute the setup script without running the cleanup script to remove any changes you might have made during the lab.	Please see Student Companion Content for this course.