Module 7

Using DML to Modify Data

Lesson 1: Adding Data to Tables

- Using INSERT to Add Data
- Using INSERT with Data Providers
- Using SELECT INTO
- Demonstration: Adding Data to Tables

Using INSERT to Add Data

- The INSERT … VALUES statement inserts a new row (Implicita e auto commit)

```
INSERT INTO Sales.OrderDetails
      (orderid, productid, unitprice, qty, discount)
VALUES (10255,39,18,2,0.05);
```

- Table and row constructors add multirow capability to INSERT … VALUES

```
INSERT INTO Sales.OrderDetails
(orderid, productid, unitprice, qty, discount)

VALUES
(10256,39,18,2,0.05),
(10258,39,18,5,0.10);
```

Using INSERT with Data Providers

- INSERT … SELECT to insert rows from another table:

```
INSERT Sales.OrderDetails
(orderid, productid, unitprice, qty, discount)

SELECT * FROM NewOrderDetails
```

- INSERT … EXEC is used to insert the result of a stored procedure or dynamic SQL expression into an existing table:

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

Using SELECT INTO

SELECT -> INTO is similar to INSERT <- SELECT

- It also creates a table for the output, fashioned on the output itself
- The new table is based on query column structure
  - Uses column names, data types, and null settings
  - Does not copy constraints or indexes

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

Demonstration: Adding Data to Tables

In this demonstration, you will see how to:

- Add data to a table using the INSERT statement
- Use the OUTPUT keyword with INSERT
- Use stored procedure output to insert data into a table
- Use SELECT INTO for populating a table with data and create the table structure at the same time

Lesson 2: Modifying and Removing Data

- Using UPDATE to Modify Data
- Using MERGE to Modify Data
- Demonstration: Manipulating Data Using the UPDATE and DELETE Statements and MERGING Data Using Conditional DML

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;
```

Updating Data in One Table Based on a Join to Another

```
UPDATE Reason  -- Notice use of Alias to make reading better
       SET Name += ' ?'

FROM Production.ScrapReason AS Reason
INNER JOIN Production.WorkOrder AS WorkOrder

ON     Reason.ScrapReasonID = WorkOrder.ScrapReasonID
AND    WorkOrder.ScrappedQty > 300;
```

Using MERGE to Modify Data

MERGE modifies data based on a condition

- When the source matches the target
- When the source has no match in the target
- When the target has no match in the source

```
MERGE TOP (10)
INTO   Store         AS Destination
USING  StoreBackup   AS StagingTable
       ON(Destination.Key = StagingTable.Key)

WHEN NOT MATCHED THEN
       INSERT (C1,..)
       VALUES (Source.C1,..)
WHEN MATCHED THEN
       UPDATE SET Destination.C1 = StagingTable.C1,..;
```

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, 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

Lesson 3: Generating Automatic Column Values

- Using IDENTITY
- Using Sequences

Using IDENTITY

The IDENTITY property generates column values automatically

- Optional seed and increment values can be provided

```
CREATE TABLE Production.Products
(PID int IDENTITY(1,1) NOT NULL, Name VARCHAR(15),…)
```

- Only one column in a table may have IDENTITY defined
- IDENTITY column must be omitted in a normal INSERT statement

```
INSERT INTO Production.Products (Name,…)
VALUES ('MOC 2072 Manual',…)
```

- Functions are provided to return last generated values
  - SELECT @@IDENTITY: default scope is session
  - SELECT SCOPE_IDENTITY(): scope is object containing the call
  - SELECT IDENT_CURRENT(' tablename'): in this case, scope is defined by tablename
- There is a setting to allow identity columns to be changed manually ON or automatic OFF
  - SET IDENTITY_INSERT <Tablename> [ON|OFF]

Using Sequences

Sequence objects were first added in SQL Server 2012

- Independent objects in database
  - More flexible than the IDENTITY property
  - Can be used as default value for a column
- Manage with CREATE/ALTER/DROP statements
- Retrieve value with the NEXT VALUE FOR clause

```
-- Define a sequence
CREATE SEQUENCE dbo.InvoiceSeq AS INT START WITH 1
INCREMENT BY 1;

-- Retrieve next available value from sequence
SELECT NEXT VALUE FOR dbo.InvoiceSeq;
```

Lab: Using DML to Modify Data

- Exercise 1: Inserting Records with DML
- Exercise 2: Update and Delete Records Using DML

Logon Information
Virtual Machine: **20761C-MIA-SQL**
User Name: **ADVENTUREWORKS\STUDENT**
Password: **Pa55w.rd**

Estimated Time: 30 Minutes