Chapter-5 Data Modification Language (DML) with MS SQL Server

Data Modification Language (DML) is a subset of SQL used for managing and manipulating data within database tables. DML operations include inserting new data, updating existing data, and deleting data. In MS SQL Server, these operations are performed using the INSERT, UPDATE, and DELETE statements.

---

1. INSERT

The INSERT statement is used to add new rows of data into a table.

Basic Syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Example:

Inserting a new employee record into the Employees table.

```
INSERT INTO Employees (EmployeeID, EmployeeName, DepartmentID, Salary)
VALUES (101, 'John Doe', 2, 50000);
```

This statement adds a new employee with EmployeeID 101, EmployeeName 'John Doe', assigned to department 2 with a salary of 50,000.

Inserting Multiple Rows:

You can insert multiple rows in one statement.

```
INSERT INTO Employees (EmployeeID, EmployeeName, DepartmentID, Salary)
VALUES
(102, 'Jane Smith', 1, 60000),
(103, 'Sam Brown', 2, 55000),
(104, 'Emily White', 3, 48000);
```

INSERT INTO SELECT:

You can also insert data into a table by selecting data from another table.

```
INSERT INTO NewEmployees (EmployeeID, EmployeeName, DepartmentID, Salary)
SELECT EmployeeID, EmployeeName, DepartmentID, Salary
FROM Employees
WHERE Salary > 50000;
```

This query copies employees with a salary greater than 50,000 into the NewEmployees table.

---

2. UPDATE

The UPDATE statement is used to modify existing records in a table. You can update one or more columns for one or more rows.

Basic Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

The WHERE clause is crucial for updating specific rows. Without it, all rows in the table will be updated.

Example:

Updating the salary of an employee.

```
UPDATE Employees
SET Salary = 55000
WHERE EmployeeID = 101;
```

This query updates the salary of the employee with EmployeeID 101 to 55,000.

Updating Multiple Columns:

You can update multiple columns in one statement.

```
UPDATE Employees
SET Salary = 60000, DepartmentID = 3
WHERE EmployeeID = 102;
```

This query updates both the salary and department of the employee with EmployeeID 102.

Using Subqueries in UPDATE:

You can also update a table using a subquery.

```
UPDATE Employees
SET Salary = (SELECT AVG(Salary) FROM Employees WHERE DepartmentID = 2)
WHERE DepartmentID = 2;
```

This query sets the salary of all employees in department 2 to the average salary of that department.

---

3. DELETE

The DELETE statement is used to remove rows from a table based on a specified condition.

Basic Syntax:

```
DELETE FROM table_name
WHERE condition;
```

The WHERE clause is important to ensure that only specific rows are deleted. If no condition is provided, all rows in the table will be deleted.

Example:

Deleting an employee record.

```
DELETE FROM Employees
WHERE EmployeeID = 101;
```

This query deletes the employee with EmployeeID 101.

Deleting All Rows in a Table:

If you want to delete all rows in a table but keep the structure intact, you can omit the WHERE clause.

```
DELETE FROM Employees;
```

This will remove all records from the Employees table.

Truncate vs. Delete:

- **TRUNCATE**: Removes all rows in a table but does not log individual row deletions. It is faster but cannot be rolled back if the transaction is committed.
- **DELETE**: Removes rows one by one and logs each row deletion, allowing it to be rolled back in case of errors.

Example of TRUNCATE:

```
TRUNCATE TABLE Employees;
```

---

4. MERGE (Upsert)

The MERGE statement is used to perform "upserts"—a combination of INSERT, UPDATE, or DELETE operations in a single statement. It allows you to synchronize two tables.

Syntax:

```
MERGE INTO target_table AS target
USING source_table AS source
ON target.column = source.column
WHEN MATCHED THEN
   UPDATE SET target.column1 = source.column1
WHEN NOT MATCHED THEN
   INSERT (column1, column2) VALUES (source.column1, source.column2);
```

Example:

Synchronizing the Employees table with new data from the NewEmployees table.

```
MERGE INTO Employees AS E
USING NewEmployees AS NE
ON E.EmployeeID = NE.EmployeeID
WHEN MATCHED THEN
   UPDATE SET E.Salary = NE.Salary
WHEN NOT MATCHED THEN
   INSERT (EmployeeID, EmployeeName, DepartmentID, Salary)
   VALUES (NE.EmployeeID, NE.EmployeeName, NE.DepartmentID, NE.Salary);
```

This query updates the salary for existing employees and inserts new employees from the NewEmployees table.

---

5. OUTPUT Clause

The OUTPUT clause allows you to return information about rows affected by an INSERT, UPDATE, or DELETE operation, such as the old and new values of a column. It can be useful for auditing, logging, or further processing the modified data.

Syntax for OUTPUT:

```
UPDATE table_name
SET column = value
OUTPUT inserted.*, deleted.*
WHERE condition;
```

Example with INSERT:

```
INSERT INTO Employees (EmployeeID, EmployeeName, DepartmentID, Salary)
OUTPUT inserted.EmployeeID, inserted.EmployeeName
VALUES (105, 'Mark Taylor', 2, 70000);
```

This query inserts a new employee and outputs the EmployeeID and EmployeeName of the newly inserted row.

---

6. Transactions in DML

DML operations can be wrapped in a **transaction** to ensure data consistency. Using transactions, you can commit or roll back changes if necessary.

Syntax for Transactions:

```
BEGIN TRANSACTION;

-- DML statements
INSERT INTO Employees (EmployeeID, EmployeeName, DepartmentID, Salary)
VALUES (106, 'Anna Lee', 4, 65000);

-- If no error occurs, commit the transaction
COMMIT;

-- If there is an error, roll back the transaction
ROLLBACK;
```

Using transactions ensures that either all DML operations succeed, or none of them do, preserving data integrity.