Chapter-2 Data Query Language (DQL) in MS SQL Server

Data Query Language (DQL) in SQL Server is used primarily to query the database and retrieve data. The main DQL statement in MS SQL Server is **SELECT**. This command allows users to retrieve data from one or more tables based on certain criteria.

Core DQL Commands

SELECT Statement

1. The SELECT statement is used to fetch data from one or more tables in the database.
2. It can be customized with various clauses to filter, group, sort, and modify how the data is returned.

Basic SELECT Query

```
SELECT column1, column2, column3
FROM table_name;
```

- **column1, column2, column3**: The columns you want to retrieve.
- **table_name**: The name of the table from which you want to fetch data.

Example:

```
SELECT Name, Age, Department
FROM Employees;
```

This will return all rows of the Name, Age, and Department columns from the Employees table.

SELECT DISTINCT

The DISTINCT keyword is used to retrieve only unique (non-duplicate) values for the specified columns.

Syntax:

```
SELECT DISTINCT column1, column2
FROM table_name;
```

Example:

```
SELECT DISTINCT Department
FROM Employees;
```

This will return a list of unique departments from the Employees table.

WHERE Clause

The WHERE clause is used to filter records that meet specific conditions.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example:

```
SELECT Name, Age
FROM Employees
WHERE Department = 'HR';
```

This will retrieve the names and ages of employees who work in the HR department.

ORDER BY Clause

The ORDER BY clause is used to sort the result set in either ascending (ASC) or descending (DESC) order.

Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC | DESC];
```

Example:

```
SELECT Name, Age
FROM Employees
ORDER BY Age DESC;
```

This will return employee names and ages sorted by Age in descending order.

LIMIT Clause (TOP in SQL Server)

The LIMIT clause in other databases restricts the number of rows returned. In SQL Server, you use the TOP keyword to achieve this.

Syntax:

```
SELECT TOP number column1, column2
FROM table_name;
```

Example:

```
SELECT TOP 5 Name, Department
FROM Employees;
```

This will return the top 5 rows from the Employees table.

AND, OR, and NOT Operators

- **AND**: Returns records that meet multiple conditions.
- **OR**: Returns records that meet at least one condition.
- **NOT**: Reverses the result of a condition.

Example (AND):

```
SELECT Name, Department
FROM Employees
WHERE Age > 30 AND Department = 'HR';
```

This will return the names and departments of employees who are older than 30 and work in the HR department.

Example (OR):

```
SELECT Name, Department
FROM Employees
WHERE Department = 'HR' OR Department = 'IT';
```

This will return the names and departments of employees who work in either HR or IT.

Example (NOT):

```
SELECT Name, Department
FROM Employees
WHERE NOT Department = 'Sales';
```

This will return the names and departments of employees who do **not** work in the Sales department.

BETWEEN Operator

The BETWEEN operator is used to filter the result set within a certain range.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE column1 BETWEEN value1 AND value2;
```

Example:

```
SELECT Name, Age
FROM Employees
WHERE Age BETWEEN 25 AND 40;
```

This will return the names and ages of employees whose ages are between 25 and 40.

LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

- **%**: Represents zero or more characters.
- **_**: Represents a single character.

Syntax:

```
SELECT column1, column2
FROM table_name
```

```
WHERE column1 LIKE 'pattern';
```

Example:

```
SELECT Name, Department
FROM Employees
WHERE Name LIKE 'J%';
```

This will return the names and departments of employees whose names start with 'J'.

IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE column1 IN (value1, value2, value3, ...);
```

Example:

```
SELECT Name, Department
FROM Employees
WHERE Department IN ('HR', 'IT');
```

This will return the names and departments of employees who work in either HR or IT.

NULL Values

To check for NULL values (empty or undefined values), use IS NULL or IS NOT NULL.

Example (IS NULL):

```
SELECT Name, Department
FROM Employees
WHERE Department IS NULL;
```

This will return the names and departments of employees who have no department assigned.

Example (IS NOT NULL):

```
SELECT Name, Department
FROM Employees
WHERE Department IS NOT NULL;
```

This will return the names and departments of employees who have a department assigned.

Aggregating Data

SQL Server provides several aggregate functions to summarize data:

- **COUNT()**: Returns the number of rows.
- **SUM()**: Returns the sum of values in a column.
- **AVG()**: Returns the average value.
- **MIN()**: Returns the minimum value.
- **MAX()**: Returns the maximum value.

Example:

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department;
```

This will return the number of employees in each department.

Example:

```
SELECT Department, AVG(Salary) AS AverageSalary
FROM Employees
GROUP BY Department;
```

This will return the average salary in each department.

HAVING Clause

The HAVING clause is used to filter the result set based on aggregate functions (similar to the WHERE clause but for grouped data).

Syntax:

```
SELECT column1, COUNT(*) AS count
FROM table_name
GROUP BY column1
HAVING COUNT(*) > value;
```

Example:

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department
HAVING COUNT(*) > 5;
```

This will return the departments with more than 5 employees.

JOIN Operations:

1. DQL also involves joining tables to retrieve related data from multiple tables.
2. Types of joins:
    - **INNER JOIN**: Retrieves rows with matching values in both tables.
    - **LEFT JOIN**: Retrieves all rows from the left table and matching rows from the right table.
    - **RIGHT JOIN**: Retrieves all rows from the right table and matching rows from the left table.
    - **FULL JOIN**: Retrieves rows when there is a match in either table.

Example of an INNER JOIN:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

DISTINCT Keyword:

- Removes duplicate values from the result set.
- Syntax:

```
SELECT DISTINCT column_name
FROM table_name;
```

LIMIT or TOP:

- Restricts the number of rows returned by the query. In MS SQL Server, this is done using the TOP keyword.
- Example: Retrieve the top 5 highest-paid employees:

```
SELECT TOP 5 * FROM Employees
ORDER BY Salary DESC;
```

Advanced DQL Features in MS SQL Server

Subqueries:

- A subquery is a query within another query. It can be used in the WHERE, FROM, or SELECT clauses to retrieve intermediate results.
- Example:

```
SELECT * FROM Employees
WHERE DepartmentID = (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'HR');
```

Common Table Expressions (CTE):

A CTE provides temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.

Example:

```
WITH DepartmentCTE AS (
  SELECT DepartmentID, AVG(Salary) AS AvgSalary
  FROM Employees
  GROUP BY DepartmentID
)
SELECT * FROM DepartmentCTE;
```

Window Functions:

- Functions like ROW_NUMBER(), RANK(), DENSE_RANK(), and NTILE() allow users to perform calculations across a set of table rows related to the current row.
- Example:

```
SELECT Name, Salary,
    RANK() OVER (ORDER BY Salary DESC) AS Rank
FROM Employees;
```

Why Use DQL in MS SQL Server?

- **Efficiency**: DQL allows users to retrieve large datasets quickly and efficiently using filtering, aggregation, and sorting techniques.
- **Data Analysis**: It supports powerful analysis by allowing complex queries, joins, and aggregations on large sets of data.
- **Flexibility**: You can modify queries dynamically with clauses like WHERE, HAVING, and ORDER BY to meet different business requirements.