
TOPIC : 1 SQL ASSIGNMENT OPERATORS

```sql
/*==========================================================
 FILE NAME : Assignment_Operator.sql
 TOPIC     : Assignment Operator & Compound Assignment Operators
 DB        : SQL Server
 PURPOSE   :
   - Understand assignment operator (=)
   - Learn variable assignment
   - Learn column alias assignment
   - Understand compound assignment operators
   - Practice interview-oriented examples
==========================================================*/


/*==========================================================
 1) ASSIGNMENT OPERATOR (=)
==========================================================*/
/*
 The assignment operator (=) is used to assign a value
 to a variable in SQL Server.

 NOTE:
 - SQL Server supports only ONE assignment operator (=)
*/


/*==========================================================
 EXAMPLE 1: SIMPLE VARIABLE ASSIGNMENT
==========================================================*/

DECLARE @MyCounter INT;
SET @MyCounter = 1;

SELECT @MyCounter AS CounterValue;
GO

/*
 OUTPUT:
 ----------
 CounterValue
 ----------
 1
*/


/*==========================================================
 2) ASSIGNMENT OPERATOR WITH COLUMN ALIAS
==========================================================*/
/*
 The assignment operator can also be used to assign
 expressions to column headings (aliases).
*/


/*-- Sample Employee Table for Demo --*/
DROP TABLE IF EXISTS Employee;
GO

CREATE TABLE Employee
(
    ID   INT,
    Name NVARCHAR(50)
);
GO

INSERT INTO Employee VALUES (1, 'Arun');
INSERT INTO Employee VALUES (2, 'Bala');
INSERT INTO Employee VALUES (3, 'Charan');
GO


/*==========================================================
 EXAMPLE 2: COLUMN HEADING ASSIGNMENT
==========================================================*/

SELECT
    FirstColumn  = 'abcd',
    SecondColumn = ID,
    ThirdColumn = 'Thillai'
FROM Employee;
GO

/*
 OUTPUT:
 --------------------------
 FirstColumn | SecondColumn
 --------------------------
 abcd        | 1
 abcd        | 2
 abcd        | 3
*/


/*==========================================================
 3) COMPOUND ASSIGNMENT OPERATORS
==========================================================*/
/*
 Introduced in SQL Server 2008

 Compound Assignment Operators:
 - Perform operation and assignment in ONE statement
 - Shorter and cleaner syntax
 - Improves readability
```

```sql
*/


/*==========================================================
 EXAMPLE 3: WITHOUT COMPOUND ASSIGNMENT
==========================================================*/

DECLARE @MyVariable INT;
SET @MyVariable = 10;
SET @MyVariable = @MyVariable * 5;

SELECT @MyVariable AS MyResult;
GO

/*
 OUTPUT:
 --------
 MyResult
 --------
 50
*/


/*==========================================================
 EXAMPLE 4: USING COMPOUND ASSIGNMENT
==========================================================*/

DECLARE @MyVariable INT;
SET @MyVariable = 10;
SET @MyVariable *= 5;

SELECT @MyVariable AS MyResult;
GO

/*
 OUTPUT:
 --------
 MyResult
 --------
 50
*/


/*==========================================================
 4) LIST OF COMPOUND ASSIGNMENT OPERATORS
==========================================================*/

/*
 +=   Add and assign
 -=   Subtract and assign
 *=   Multiply and assign
 /=   Divide and assign
 %=   Modulo and assign
```

```sql
*/


/*==========================================================
 EXAMPLE 5: += OPERATOR
==========================================================*/

DECLARE @Value INT = 10;
SET @Value += 5;

SELECT @Value AS Result;
GO
-- Output: 15


/*==========================================================
 EXAMPLE 6: -= OPERATOR
==========================================================*/

DECLARE @Value INT = 20;
SET @Value -= 8;

SELECT @Value AS Result;
GO
-- Output: 12


/*==========================================================
 EXAMPLE 7: *= OPERATOR
==========================================================*/

DECLARE @Value INT = 6;
SET @Value *= 4;

SELECT @Value AS Result;
GO
-- Output: 24


/*==========================================================
 EXAMPLE 8: /= OPERATOR
==========================================================*/

DECLARE @Value INT = 20;
SET @Value /= 4;

SELECT @Value AS Result;
GO
-- Output: 5


/*==========================================================
 EXAMPLE 9: %= OPERATOR (MODULO)
```

```sql
===========================================================*/

DECLARE @Value INT = 23;
SET @Value %= 5;

SELECT @Value AS Result;
GO
-- Output: 3


/*==========================================================
 INTERVIEW KEY POINTS
===========================================================*/

-- 1) SQL Server supports only (=) as assignment operator
-- 2) Assignment operator works with variables and column aliases
-- 3) Compound assignment operators introduced in SQL Server 2008
-- 4) Compound operators reduce code length and improve clarity
-- 5) Used only with variables (not table columns directly)


/*==========================================================
 END OF FILE
===========================================================*/
```

TOPIC : 6 IN BETWEEN LIKE ALL ANY EXISTS

```sql
/*==========================================================
 FILE NAME : IN_BETWEEN_LIKE_ALL_ANY_EXISTS.sql
 TOPIC     :
   - IN Operator
   - BETWEEN Operator
   - LIKE Operator
   - ALL Operator
   - ANY Operator
   - SOME Operator
   - EXISTS Operator
 DB        : SQL Server
 PURPOSE   :
   - Learn filtering operators
   - Understand subquery-based operators
   - Prepare for interviews with examples
===========================================================*/


/*==========================================================
 1) SAMPLE TABLE SETUP
===========================================================*/

DROP TABLE IF EXISTS Employees;
DROP TABLE IF EXISTS Departments;
```

```sql
GO

CREATE TABLE Departments
(
    DeptId   INT,
    DeptName NVARCHAR(20)
);
GO

CREATE TABLE Employees
(
    EmpId   INT,
    EmpName NVARCHAR(50),
    Salary  INT,
    DeptId  INT,
    City    NVARCHAR(20)
);
GO

INSERT INTO Departments VALUES
(1, 'IT'),
(2, 'HR'),
(3, 'Admin');
GO

INSERT INTO Employees VALUES
(1, 'Arun',   30000, 1, 'Chennai'),
(2, 'Bala',   18000, 2, 'Madurai'),
(3, 'Charan', 25000, 1, 'Chennai'),
(4, 'Deepak', 15000, 3, 'Trichy'),
(5, 'Ezhil',  22000, 2, 'Chennai');
GO


/*=========================================================
 2) IN OPERATOR
=========================================================*/
/*
 IN is used to match multiple values
 Alternative to multiple OR conditions
*/

SELECT *
FROM Employees
WHERE DeptId IN (1, 2);
GO

/*
 Equivalent to:
 DeptId = 1 OR DeptId = 2
*/
```

```sql
/*==========================================================
 3) BETWEEN OPERATOR
==========================================================*/
/*
 BETWEEN is inclusive (includes start and end)
*/

SELECT *
FROM Employees
WHERE Salary BETWEEN 20000 AND 30000;
GO

/*
 Includes:
 Salary = 20000
 Salary = 30000
*/


/*==========================================================
 4) LIKE OPERATOR
==========================================================*/
/*
 Used for pattern matching
 %  → any number of characters
 _  → exactly one character
*/

-- Names starting with 'A'
SELECT *
FROM Employees
WHERE EmpName LIKE 'A%';
GO

-- Names ending with 'n'
SELECT *
FROM Employees
WHERE EmpName LIKE '%n';
GO

-- Names with exactly 5 characters
SELECT *
FROM Employees
WHERE EmpName LIKE '_____';
GO


/*==========================================================
 5) ALL OPERATOR
==========================================================*/
/*
 ALL compares a value with ALL values returned
 by a subquery
```

```sql
*/

-- Employees earning more than ALL HR salaries
SELECT *
FROM Employees
WHERE Salary > ALL
(
    SELECT Salary
    FROM Employees
    WHERE DeptId = 2
);
GO

/*
 Meaning:
 Salary > every salary in HR department
*/


/*==========================================================
 6) ANY OPERATOR
==========================================================*/
/*
 ANY compares a value with ANY ONE value
 returned by a subquery
*/

-- Employees earning more than ANY HR salary
SELECT *
FROM Employees
WHERE Salary > ANY
(
    SELECT Salary
    FROM Employees
    WHERE DeptId = 2
);
GO

/*
 Meaning:
 Salary > at least one HR salary
*/


/*==========================================================
 7) SOME OPERATOR
==========================================================*/
/*
 SOME is exactly same as ANY in SQL Server
*/

SELECT *
FROM Employees
```

```
WHERE Salary > SOME
(
    SELECT Salary
    FROM Employees
    WHERE DeptId = 2
);
GO

/*
 NOTE:
 SOME = ANY (no difference)
*/


/*==========================================================
 8) EXISTS OPERATOR
==========================================================*/
/*
 EXISTS checks whether subquery returns rows
 Returns TRUE or FALSE
 Stops checking after first match (fast)
*/

-- Employees whose department exists
SELECT *
FROM Employees E
WHERE EXISTS
(
    SELECT 1
    FROM Departments D
    WHERE D.DeptId = E.DeptId
);
GO


/*==========================================================
 9) NOT EXISTS EXAMPLE
==========================================================*/

-- Employees without a valid department
SELECT *
FROM Employees E
WHERE NOT EXISTS
(
    SELECT 1
    FROM Departments D
    WHERE D.DeptId = E.DeptId
);
GO


/*==========================================================
 10) INTERVIEW COMPARISON SUMMARY
```

```
==========================================================*/

-- IN       → Match multiple values
-- BETWEEN  → Range filtering (inclusive)
-- LIKE     → Pattern matching
-- ALL      → Compare with all values in subquery
-- ANY      → Compare with any one value
-- SOME     → Same as ANY
-- EXISTS   → Checks existence (returns TRUE/FALSE)


/*==========================================================
 11) PERFORMANCE NOTES (INTERVIEW GOLD)
==========================================================*/

-- EXISTS is usually faster than IN for large datasets
-- BETWEEN is inclusive (common interview trap)
-- SOME and ANY are identical in SQL Server
-- ALL fails if subquery returns NULL (important)


/*==========================================================
 END OF FILE
==========================================================*/
```

TOPIC : 7 UNION EXCEPT INTERSECT

```
/*==========================================================
 FILE NAME : UNION_EXCEPT_INTERSECT.sql
 TOPIC     :
   - UNION
   - UNION ALL
   - EXCEPT
   - INTERSECT
 DB        : SQL Server
 PURPOSE   :
   - Combine result sets
   - Understand set-based operators
   - Learn differences, rules, and performance
   - Prepare for interview questions
==========================================================*/


/*==========================================================
 1) SAMPLE TABLE SETUP
==========================================================*/

DROP TABLE IF EXISTS IT_Employees;
DROP TABLE IF EXISTS HR_Employees;
GO
```

```
CREATE TABLE IT_Employees
(
    EmpId   INT,
    EmpName NVARCHAR(50)
);
GO

CREATE TABLE HR_Employees
(
    EmpId   INT,
    EmpName NVARCHAR(50)
);
GO

INSERT INTO IT_Employees VALUES
(1, 'Arun'),
(2, 'Bala'),
(3, 'Charan'),
(4, 'Deepak');
GO

INSERT INTO HR_Employees VALUES
(3, 'Charan'),
(4, 'Deepak'),
(5, 'Ezhil'),
(6, 'Farooq');
GO


/*==========================================================
 VIEW DATA
==========================================================*/
SELECT * FROM IT_Employees;
SELECT * FROM HR_Employees;
GO


/*==========================================================
 IMPORTANT RULES FOR SET OPERATORS
==========================================================*/
/*
 1) Number of columns must be SAME
 2) Data types must be COMPATIBLE
 3) Column order must be SAME
 4) ORDER BY allowed only at the END
*/


/*==========================================================
 2) UNION OPERATOR
==========================================================*/
/*
 UNION:
```

```sql
 - Combines result sets
 - Removes duplicate rows
 - Uses DISTINCT internally
*/

SELECT EmpId, EmpName
FROM IT_Employees

UNION

SELECT EmpId, EmpName
FROM HR_Employees;
GO

/*
 RESULT:
 - All unique employees from both tables
 - Duplicate rows removed
*/


/*==========================================================
 3) UNION ALL OPERATOR
==========================================================*/
/*
 UNION ALL:
 - Combines result sets
 - DOES NOT remove duplicates
 - Faster than UNION
*/

SELECT EmpId, EmpName
FROM IT_Employees

UNION ALL

SELECT EmpId, EmpName
FROM HR_Employees;
GO

/*
 RESULT:
 - All rows from both tables
 - Duplicate rows included
*/


/*==========================================================
 4) UNION vs UNION ALL (INTERVIEW FAVORITE)
==========================================================*/
/*
 UNION      → Removes duplicates (slower)
 UNION ALL  → Keeps duplicates (faster)
 Best Practice:
 Use UNION ALL when duplicates are acceptable
*/


/*==========================================================
 5) EXCEPT OPERATOR
==========================================================*/
/*
 EXCEPT:
 - Returns rows from first query
 - That are NOT present in second query
 - Removes duplicates
*/

SELECT EmpId, EmpName
FROM IT_Employees

EXCEPT

SELECT EmpId, EmpName
FROM HR_Employees;
GO

/*
 RESULT:
 - Employees only in IT
 - Common employees removed
*/


/*==========================================================
 6) EXCEPT (REVERSE ORDER)
==========================================================*/

SELECT EmpId, EmpName
FROM HR_Employees

EXCEPT

SELECT EmpId, EmpName
FROM IT_Employees;
GO

/*
 RESULT:
 - Employees only in HR
*/


/*==========================================================
 7) INTERSECT OPERATOR
```

```
==========================================================*/
/*
 INTERSECT:
 - Returns only COMMON rows
 - Removes duplicates
*/

SELECT EmpId, EmpName
FROM IT_Employees

INTERSECT

SELECT EmpId, EmpName
FROM HR_Employees;
GO

/*
 RESULT:
 - Employees present in BOTH tables
*/


/*==========================================================
 8) REAL-TIME USE CASES
==========================================================*/
/*
 UNION ALL    → Log tables, history tables
 UNION        → Master reports
 EXCEPT       → Mismatch / audit reports
 INTERSECT    → Common users, permissions
*/


/*==========================================================
 9) PERFORMANCE NOTES (INTERVIEW GOLD)
==========================================================*/

-- UNION uses DISTINCT → slower
-- UNION ALL is fastest
-- EXCEPT and INTERSECT remove duplicates
-- Indexes improve performance


/*==========================================================
 10) QUICK INTERVIEW QUESTIONS
==========================================================*/

-- Q1: Which is faster UNION or UNION ALL?
-- A : UNION ALL

-- Q2: Does INTERSECT remove duplicates?
-- A : Yes
-- Q3: Can ORDER BY be used in UNION queries?
-- A : Yes, only at the END


/*==========================================================
 11) FINAL SUMMARY
==========================================================*/

-- UNION       → Combine + remove duplicates
-- UNION ALL   → Combine + keep duplicates
-- EXCEPT      → First result minus second
-- INTERSECT   → Common rows only


/*==========================================================
 END OF FILE
==========================================================*/
```

TOPIC : 2.ARITHMETIC OPERATORS

```
/*==========================================================
 FILE NAME : Arithmetic_Operators.sql
 TOPIC     : Arithmetic Operators in SQL Server
 DB        : SQL Server
 PURPOSE   :
   - Understand arithmetic operators
   - Learn usage with variables and columns
   - Practice real-time and interview examples
==========================================================*/


/*==========================================================
 1) WHAT ARE ARITHMETIC OPERATORS?
==========================================================*/
/*
 Arithmetic operators are used to perform
 mathematical calculations on numeric values.

 SQL Server supports the following arithmetic operators:
 +   Addition
 -   Subtraction
 *   Multiplication
 /   Division
 %   Modulo (Remainder)
*/


/*==========================================================
 2) ADDITION OPERATOR (+)
==========================================================*/
```

```sql
DECLARE @A INT = 10, @B INT = 5;
SELECT @A + @B AS AdditionResult;
GO

/*
 OUTPUT:
 ---------------
 AdditionResult
 ---------------
 15
*/


/*==========================================================
 3) SUBTRACTION OPERATOR (-)
==========================================================*/

DECLARE @A INT = 20, @B INT = 8;
SELECT @A - @B AS SubtractionResult;
GO

/*
 OUTPUT:
 -------------------
 SubtractionResult
 -------------------
 12
*/


/*==========================================================
 4) MULTIPLICATION OPERATOR (*)
==========================================================*/

DECLARE @A INT = 6, @B INT = 7;
SELECT @A * @B AS MultiplicationResult;
GO

/*
 OUTPUT:
 ------------------------
 MultiplicationResult
 ------------------------
 42
*/


/*==========================================================
 5) DIVISION OPERATOR (/)
==========================================================*/
/*
 IMPORTANT:
 - Integer / Integer = Integer
 - Decimal / Decimal = Decimal
*/

DECLARE @A INT = 20, @B INT = 3;
SELECT @A / @B AS IntegerDivision;
GO

/*
 OUTPUT:
 ----------------
 IntegerDivision
 ----------------
 6   (Decimal part is truncated)
*/


/*==========================================================
 DIVISION WITH DECIMAL
==========================================================*/

DECLARE @A DECIMAL(10,2) = 20, @B DECIMAL(10,2) = 3;
SELECT @A / @B AS DecimalDivision;
GO

/*
 OUTPUT:
 -----------------
 DecimalDivision
 -----------------
 6.66
*/


/*==========================================================
 6) MODULO OPERATOR (%)
==========================================================*/
/*
 Returns remainder after division
*/

DECLARE @A INT = 23, @B INT = 5;
SELECT @A % @B AS ModuloResult;
GO

/*
 OUTPUT:
 -------------
 ModuloResult
 -------------
 3
*/
```

```sql
/*==========================================================
 7) ARITHMETIC OPERATORS WITH TABLE DATA
==========================================================*/

DROP TABLE IF EXISTS ProductSales;
GO

CREATE TABLE ProductSales
(
    ProductName NVARCHAR(50),
    Quantity    INT,
    Price       INT
);
GO

INSERT INTO ProductSales VALUES ('Mouse',    2, 500);
INSERT INTO ProductSales VALUES ('Keyboard', 1, 800);
INSERT INTO ProductSales VALUES ('Monitor',  3, 700);
GO


/*==========================================================
 CALCULATE TOTAL PRICE (Quantity * Price)
==========================================================*/

SELECT
    ProductName,
    Quantity,
    Price,
    Quantity * Price AS TotalPrice
FROM ProductSales;
GO

/*
 REAL-TIME USE CASE:
 - Billing
 - Invoice calculation
 - Shopping cart
*/


/*==========================================================
 8) ARITHMETIC OPERATOR PRECEDENCE
==========================================================*/
/*
 Operator precedence:
 1) *  /  %
 2) +  -
*/

SELECT 10 + 5 * 2 AS ResultWithoutBrackets;
GO
-- Output: 20
```

```sql
SELECT (10 + 5) * 2 AS ResultWithBrackets;
GO
-- Output: 30


/*==========================================================
 9) COMMON INTERVIEW QUESTIONS
==========================================================*/

-- Q1: What is the result of 10 / 3?
-- A : 3 (Integer division)

-- Q2: How to get decimal result?
-- A : Use DECIMAL or CAST

SELECT CAST(10 AS DECIMAL(5,2)) / 3 AS DecimalResult;
GO

-- Q3: What does % operator do?
-- A : Returns remainder


/*==========================================================
 KEY POINTS SUMMARY
==========================================================*/

-- +  Addition
-- -  Subtraction
-- *  Multiplication
-- /  Division
-- %  Modulo

-- Integer division truncates decimal values
-- Use DECIMAL to preserve precision
-- Widely used in financial and calculation queries


/*==========================================================
 END OF FILE
==========================================================*/
```

TOPIC : 3.COMPARISON OPERATORS

```sql
/*==========================================================
 FILE NAME : Comparison_Operators.sql
 TOPIC     : Comparison Operators in SQL Server
 DB        : SQL Server
 PURPOSE   :
   - Understand comparison operators
   - Learn usage with variables and table data
   - Practice interview-oriented examples
==========================================================*/


/*==========================================================
 1) WHAT ARE COMPARISON OPERATORS?
==========================================================*/
/*
 Comparison operators are used to compare two values.
 They always return TRUE or FALSE logically
 (in result set: row is returned or not).

 SQL Server Comparison Operators:
 =    Equal to
 <>   Not equal to
 !=   Not equal to
 >    Greater than
 <    Less than
 >=   Greater than or equal to
 <=   Less than or equal to
*/


/*==========================================================
 2) CREATE SAMPLE TABLE
==========================================================*/

DROP TABLE IF EXISTS Employees;
GO

CREATE TABLE Employees
(
    EmpId     INT,
    EmpName   NVARCHAR(50),
    Salary    INT,
    Dept      NVARCHAR(20)
);
GO

INSERT INTO Employees VALUES (1, 'Arun',   25000, 'IT');
INSERT INTO Employees VALUES (2, 'Bala',   18000, 'HR');
INSERT INTO Employees VALUES (3, 'Charan', 30000, 'IT');
INSERT INTO Employees VALUES (4, 'Deepak', 15000, 'Admin');
INSERT INTO Employees VALUES (5, 'Ezhil',  22000, 'HR');
GO


/*==========================================================
 VIEW TABLE DATA
==========================================================*/
SELECT * FROM Employees;
GO
```

```sql
/*==========================================================
 3) EQUAL TO OPERATOR (=)
==========================================================*/

SELECT *
FROM Employees
WHERE Dept = 'IT';
GO

/*
 OUTPUT:
 Employees working in IT department
*/


/*==========================================================
 4) NOT EQUAL TO OPERATOR (<>)
==========================================================*/

SELECT *
FROM Employees
WHERE Dept <> 'HR';
GO

/*
 Returns all employees except HR department
*/


/*==========================================================
 5) NOT EQUAL TO OPERATOR (!=)
==========================================================*/

SELECT *
FROM Employees
WHERE Salary != 25000;
GO

/*
 Same as <> operator
*/


/*==========================================================
 6) GREATER THAN OPERATOR (>)
==========================================================*/

SELECT *
FROM Employees
WHERE Salary > 20000;
GO
```

```
/*
 Employees earning more than 20000
*/


/*==========================================================
 7) LESS THAN OPERATOR (<)
==========================================================*/

SELECT *
FROM Employees
WHERE Salary < 20000;
GO

/*
 Employees earning less than 20000
*/


/*==========================================================
 8) GREATER THAN OR EQUAL TO (>=)
==========================================================*/

SELECT *
FROM Employees
WHERE Salary >= 22000;
GO

/*
 Salary 22000 and above
*/


/*==========================================================
 9) LESS THAN OR EQUAL TO (<=)
==========================================================*/

SELECT *
FROM Employees
WHERE Salary <= 18000;
GO

/*
 Salary 18000 and below
*/


/*==========================================================
 10) COMPARISON OPERATORS WITH VARIABLES
==========================================================*/

DECLARE @MinSalary INT = 20000;
```

```sql
SELECT *
FROM Employees
WHERE Salary >= @MinSalary;
GO


/*==========================================================
 11) REAL-TIME USE CASES
==========================================================*/
/*
 - Salary filtering
 - Age eligibility
 - Date range comparison
 - Stock availability
 - Access control rules
*/


/*==========================================================
 12) INTERVIEW TRICK QUESTIONS
==========================================================*/

-- Q1: Difference between <> and != ?
-- A : No difference in SQL Server (both mean NOT EQUAL)

-- Q2: Can comparison operators be used with WHERE?
-- A : Yes, very commonly

-- Q3: Can comparison operators be used with HAVING?
-- A : Yes, with aggregate functions

-- Example:
SELECT Dept, COUNT(*) AS EmpCount
FROM Employees
GROUP BY Dept
HAVING COUNT(*) > 1;
GO


/*==========================================================
 KEY POINTS SUMMARY
==========================================================*/

-- =   Equal
-- <>  Not Equal
-- !=  Not Equal
-- >   Greater Than
-- <   Less Than
-- >=  Greater Than or Equal
-- <=  Less Than or Equal

-- Used mainly in WHERE and HAVING clauses
-- Core concept for filtering data
```

```
/*==========================================================
 END OF FILE
==========================================================*/
```

TOPIC : 4.OPERATOR PRECEDENCE

```
/*==========================================================
 FILE NAME : Operator_Precedence.sql
 TOPIC     : Operator Precedence in SQL Server
 DB        : SQL Server
 PURPOSE   :
   - Understand operator precedence (execution order)
   - Avoid logical bugs in calculations and filters
   - Learn how parentheses change results
==========================================================*/


/*==========================================================
 1) WHAT IS OPERATOR PRECEDENCE?
==========================================================*/
/*
 Operator precedence defines the order in which SQL Server
 evaluates operators in an expression.

 If multiple operators exist in one expression:
 - SQL Server evaluates higher-precedence operators first
 - Parentheses () override default precedence
*/


/*==========================================================
 2) OPERATOR PRECEDENCE ORDER (HIGH → LOW)
==========================================================*/
/*
 1) ( )               Parentheses
 2) *, /, %           Multiplication, Division, Modulo
 3) +, -              Addition, Subtraction
 4) Comparison        =, <>, !=, >, <, >=, <=
 5) NOT
 6) AND
 7) OR
*/


/*==========================================================
 3) ARITHMETIC PRECEDENCE (NO PARENTHESES)
==========================================================*/

SELECT 10 + 5 * 2 AS Result;
```

```
GO

/*
 Evaluation:
 5 * 2 = 10
 10 + 10 = 20

 OUTPUT:
 -------
 20
*/


/*==========================================================
 4) ARITHMETIC PRECEDENCE WITH PARENTHESES
==========================================================*/

SELECT (10 + 5) * 2 AS Result;
GO

/*
 Evaluation:
 (10 + 5) = 15
 15 * 2 = 30

 OUTPUT:
 -------
 30
*/


/*==========================================================
 5) MULTIPLE OPERATORS TOGETHER
==========================================================*/

SELECT 100 - 20 / 5 * 2 AS Result;
GO

/*
 Evaluation:
 20 / 5 = 4
 4 * 2 = 8
 100 - 8 = 92

 OUTPUT:
 -------
 92
*/


/*==========================================================
 6) USING VARIABLES
==========================================================*/
```

```sql
DECLARE @A INT = 10, @B INT = 5, @C INT = 2;

SELECT @A + @B * @C AS WithoutBrackets;
SELECT (@A + @B) * @C AS WithBrackets;
GO

/*
 OUTPUT:
 -----------------
 WithoutBrackets | WithBrackets
 -----------------
 20              | 30
*/


/*==========================================================
 7) COMPARISON + LOGICAL PRECEDENCE
==========================================================*/

DROP TABLE IF EXISTS Employees;
GO

CREATE TABLE Employees
(
    EmpId   INT,
    Name    NVARCHAR(50),
    Salary  INT,
    Dept    NVARCHAR(20)
);
GO

INSERT INTO Employees VALUES
(1, 'Arun',   30000, 'IT'),
(2, 'Bala',   18000, 'HR'),
(3, 'Charan', 25000, 'IT'),
(4, 'Deepak', 15000, 'Admin'),
(5, 'Ezhil',  22000, 'HR');
GO


/*==========================================================
 8) AND vs OR (WITHOUT PARENTHESES)
==========================================================*/

SELECT *
FROM Employees
WHERE Dept = 'IT' OR Dept = 'HR' AND Salary > 20000;
GO

/*
 Evaluation:
 AND has higher precedence than OR
 Equivalent to:
 Dept = 'IT'
 OR (Dept = 'HR' AND Salary > 20000)

 RESULT:
 - All IT employees
 - HR employees with Salary > 20000
*/


/*==========================================================
 9) AND vs OR (WITH PARENTHESES)
==========================================================*/

SELECT *
FROM Employees
WHERE (Dept = 'IT' OR Dept = 'HR')
  AND Salary > 20000;
GO

/*
 Evaluation:
 Parentheses executed first

 RESULT:
 - IT and HR employees
 - Only if Salary > 20000
*/


/*==========================================================
 10) NOT OPERATOR PRECEDENCE
==========================================================*/

SELECT *
FROM Employees
WHERE NOT Salary > 20000;
GO

/*
 Equivalent to:
 WHERE Salary <= 20000
*/


/*==========================================================
 11) REAL-TIME BUG EXAMPLE (INTERVIEW FAVORITE)
==========================================================*/

--  BUGGY QUERY
SELECT *
FROM Employees
```

```sql
WHERE Dept = 'IT' OR Dept = 'HR' AND Salary >= 25000;
GO

--  CORRECT QUERY
SELECT *
FROM Employees
WHERE (Dept = 'IT' OR Dept = 'HR')
  AND Salary >= 25000;
GO


/*==========================================================
 12) BEST PRACTICES
==========================================================*/

-- 1) Always use parentheses for clarity
-- 2) Never rely on default precedence in business logic
-- 3) Parentheses improve readability and avoid bugs
-- 4) Critical in WHERE and HAVING clauses


/*==========================================================
 13) QUICK INTERVIEW QUESTIONS
==========================================================*/

-- Q1: Which has higher precedence AND or OR?
-- A : AND

-- Q2: Which executes first?
--     10 + 5 * 2  OR  (10 + 5) * 2
-- A : 10 + (5 * 2)

-- Q3: Best practice?
-- A : Always use parentheses


/*==========================================================
 END OF FILE
==========================================================*/
```

TOPIC : 5.LOGICAL OPERATORS

```sql
/*==========================================================
 FILE NAME : Logical_Operators.sql
 TOPIC     : Logical Operators in SQL Server
 DB        : SQL Server
 PURPOSE   :
   - Understand AND, OR, NOT operators
   - Learn real-time filtering logic
   - Avoid common logical mistakes
   - Prepare for interview questions
```

```sql
==========================================================*/


/*==========================================================
 1) WHAT ARE LOGICAL OPERATORS?
==========================================================*/
/*
 Logical operators are used to combine or negate
 multiple conditions in SQL.

 SQL Server Logical Operators:
 - AND
 - OR
 - NOT
*/


/*==========================================================
 2) CREATE SAMPLE TABLE
==========================================================*/

DROP TABLE IF EXISTS Employees;
GO

CREATE TABLE Employees
(
    EmpId   INT,
    EmpName NVARCHAR(50),
    Salary  INT,
    Dept    NVARCHAR(20),
    City    NVARCHAR(20)
);
GO

INSERT INTO Employees VALUES
(1, 'Arun',   30000, 'IT',    'Chennai'),
(2, 'Bala',   18000, 'HR',    'Madurai'),
(3, 'Charan', 25000, 'IT',    'Chennai'),
(4, 'Deepak', 15000, 'Admin', 'Trichy'),
(5, 'Ezhil',  22000, 'HR',    'Chennai');
GO


/*==========================================================
 VIEW DATA
==========================================================*/
SELECT * FROM Employees;
GO


/*==========================================================
 3) AND OPERATOR
==========================================================*/
```

```sql
/*
 AND returns TRUE only if ALL conditions are TRUE
*/

SELECT *
FROM Employees
WHERE Dept = 'IT'
  AND Salary > 25000;
GO

/*
 RESULT:
 - Employees in IT department
 - AND Salary greater than 25000
*/


/*==========================================================
 4) OR OPERATOR
==========================================================*/
/*
 OR returns TRUE if ANY one condition is TRUE
*/

SELECT *
FROM Employees
WHERE Dept = 'HR'
   OR Dept = 'Admin';
GO

/*
 RESULT:
 - HR employees
 - Admin employees
*/


/*==========================================================
 5) NOT OPERATOR
==========================================================*/
/*
 NOT reverses the condition
*/

SELECT *
FROM Employees
WHERE NOT Dept = 'IT';
GO

/*
 RESULT:
 - All employees except IT department
*/
```

```sql
/*==========================================================
 6) COMBINING AND + OR (WITHOUT PARENTHESES)
==========================================================*/

SELECT *
FROM Employees
WHERE Dept = 'IT'
   OR Dept = 'HR'
  AND Salary > 20000;
GO

/*
 IMPORTANT:
 AND has higher precedence than OR

 Equivalent to:
 Dept = 'IT'
 OR (Dept = 'HR' AND Salary > 20000)
*/


/*==========================================================
 7) COMBINING AND + OR (WITH PARENTHESES)
==========================================================*/

SELECT *
FROM Employees
WHERE (Dept = 'IT' OR Dept = 'HR')
  AND Salary > 20000;
GO

/*
 RESULT:
 - IT and HR employees
 - Only if Salary > 20000
*/


/*==========================================================
 8) NOT WITH AND
==========================================================*/

SELECT *
FROM Employees
WHERE NOT (Dept = 'HR' AND City = 'Chennai');
GO

/*
 RESULT:
 - Excludes HR employees from Chennai
*/
```

```sql
/*==========================================================
 9) REAL-TIME USE CASE EXAMPLES
==========================================================*/
/*
 - Employee eligibility rules
 - Access control
 - Report filtering
 - Business conditions
*/

-- Employees eligible for bonus
SELECT *
FROM Employees
WHERE Salary >= 20000
  AND Dept <> 'Admin';
GO


/*==========================================================
 10) COMMON INTERVIEW TRICK QUESTIONS
==========================================================*/

-- Q1: Which has higher precedence AND or OR?
-- A : AND

-- Q2: How to change precedence?
-- A : Use parentheses ()

-- Q3: What does NOT do?
-- A : Negates a condition


/*==========================================================
 11) BEST PRACTICES
==========================================================*/

-- Always use parentheses for complex conditions
-- Avoid relying on default precedence
-- Write readable and maintainable WHERE clauses


/*==========================================================
 12) LOGICAL OPERATOR SUMMARY
==========================================================*/

-- AND  → All conditions must be TRUE
-- OR   → Any one condition must be TRUE
-- NOT  → Reverses the condition

-- Used mainly in WHERE and HAVING clauses
```

```
/***********************************************************
 END OF FILE
***********************************************************/
```