upGrad ENTERPRISE

Advanced Filtering in SQL Server

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

Microsoft® SQL Server®

Day-19 (04-Aug-2025)

1. Use Cases & Enforcement at Data-Entry Level

Data-entry-level enforcement ensures that only valid data is entered into the database using:

- **Constraints** (CHECK, UNIQUE, DEFAULT, etc.)
- **Triggers**
- **Data types**
- **Validation logic in stored procedures or applications**

Example: Enforcing Age Constraint

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name NVARCHAR(50),
    Age INT CHECK (Age >= 5)
);
```

Use Case: Prevents invalid age entries (like -1 or 2) when inserting data.

2. Advanced Filtering Techniques

These help filter data **efficiently** using smarter, readable queries.

BETWEEN

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31';
```

IN / NOT IN

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

SELECT * FROM Employees
WHERE Department NOT IN ('Marketing', 'Sales');
```

Pattern Matching using LIKE, %, _

- `%` → wildcard for multiple characters
- `_` → wildcard for a single character

```
-- Names starting with A
SELECT * FROM Customers WHERE Name LIKE 'A%';

-- Names with second letter 'e'
SELECT * FROM Customers WHERE Name LIKE '_e%';

-- Ends with 'n'
SELECT * FROM Customers WHERE Name LIKE '%n';
```

3. Working with NULLs

IS NULL / IS NOT NULL

```
SELECT * FROM Products WHERE Discount IS NULL;
SELECT * FROM Products WHERE Discount IS NOT NULL;
```

4. Understanding SQL's 3-Valued Logic

SQL uses **TRUE**, **FALSE**, and **UNKNOWN** (usually from NULL comparisons)

```
-- This will NOT return rows with NULL Discount
SELECT * FROM Products WHERE Discount <> 0;

-- Better:
SELECT * FROM Products WHERE Discount <> 0 OR Discount IS NULL;
```

NULL is not 0 and **NULL = NULL is UNKNOWN**

Always use `IS NULL` and `IS NOT NULL` for checks!

5. Use of Aliases in Joins

Aliases make queries shorter & more readable.

```
SELECT e.Name, d.DepartmentName
FROM Employees AS e
JOIN Departments AS d ON e.DepartmentID = d.DepartmentID;
```

TIP: Use short, meaningful aliases (like `e`, `d`) to reduce clutter.

6. Combining Joins with Subqueries

Power combo for complex filtering and data fetching!

Example: Join with subquery

```
SELECT e.Name, e.Salary
FROM Employees e
JOIN (
    SELECT DepartmentID, AVG(Salary) AS AvgSalary
    FROM Employees
    GROUP BY DepartmentID
) avgDept ON e.DepartmentID = avgDept.DepartmentID
WHERE e.Salary > avgDept.AvgSalary;
```

Use case: Find employees earning more than their department's average.

7. Writing Readable & Maintainable Queries

Tips for readability:

- Indent nested logic
- Use meaningful aliases
- Break long queries into sections
- Add **comments**

Example

```sql
-- Get high-value customers from active regions
SELECT
    c.CustomerName,
    o.TotalAmount
FROM
    Customers AS c
JOIN
    Orders AS o ON c.CustomerID = o.CustomerID
WHERE
    o.TotalAmount > 10000
    AND c.Region IN ('North', 'South');
```

Note: Use comments (`--`) to describe logic sections for teammates or your future rerference.

Real-Time Case Study

Scenario: An online store wants to:

- Filter orders from the last 30 days
- Show only customers who haven't provided an email
- Highlight high-value orders (> ₹10,000)
- Skip orders from banned regions

```sql
SELECT o.OrderID, c.CustomerName, o.OrderDate, o.TotalAmount
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE
    o.OrderDate >= DATEADD(DAY, -30, GETDATE())
    AND c.Email IS NULL
    AND o.TotalAmount > 10000
    AND c.Region NOT IN ('RestrictedArea1', 'RestrictedArea2');
```

Impact: Helps marketing team retarget high-value but unreachable customers.