# 8. Filtering: WHERE, Predicates, and SARGability (MS SQL)

Filtering is where SQL stops being a polite set of rows and becomes a strategic puzzle. A predicate decides which rows survive. A SARGable predicate decides whether the optimizer can zoom in with an index seek rather than lumbering through a scan.

---

## WHERE — the gatekeeper

The `WHERE` clause evaluates each row **before** grouping and projections (in logical order). Only rows that satisfy the predicate make it past the gate.

Basic example:

```
SELECT *
FROM dbo.Orders
WHERE OrderDate >= '2024-01-01';
```

This comparison is simple, deterministic, and SARGable — three virtues that make the query engine happy.

A predicate is just a condition that yields TRUE or FALSE (or UNKNOWN if NULL sneaks in). SQL's three-valued logic means `UNKNOWN` behaves like FALSE for filtering.

---

## Predicates — the logic that picks survivors

Predicates can combine several conditions using `AND`, `OR`, and `NOT`.

```
SELECT *
FROM dbo.Customers
WHERE Country = 'India'
  AND (IsActive = 1 OR LoyaltyPoints > 5000);
```

Compound predicates can be efficient or disastrous depending on how they line up with indexes.

**Beware of OR** — it can force scans if each side targets different columns.

```
WHERE LastName = 'Singh'
   OR Email LIKE 'a%'
```

The optimizer may abandon index seeks and choose a scan. Sometimes splitting into two queries and `UNION ALL` works better.

---

## SARGability — your queries' gym membership

A predicate is SARGable (Search ARGument ABLE) if SQL Server can use an index to jump directly to matching rows. Think of it as giving the optimizer coordinates instead of vague directions.

### SARGable pattern

```
WHERE OrderDate >= '2024-01-01' AND OrderDate < '2025-01-01'
```

This lets the optimizer perform an index seek on `OrderDate`.

### Non-SARGable patterns that sabotage performance

These expressions break the column into a computed value, forcing SQL Server to evaluate each row.

#### Functions on the column

```
WHERE YEAR(OrderDate) = 2024        -- Non-SARGable
```

The engine must compute YEAR(OrderDate) for every row.

#### Conversions on the column

```
WHERE CONVERT(varchar(10), OrderDate, 23) = '2024-01-01';
```

This wraps the column in a function — index lost.

#### Leading wildcards

```
WHERE CustomerName LIKE '%son'      -- Non-SARGable
```

SQL Server has no idea where to start.

### Rewrite to be SARGable

Instead of `YEAR(OrderDate) = 2024`:

```
WHERE OrderDate >= '2024-01-01'
  AND OrderDate <  '2025-01-01';
```

Instead of `LIKE '%son'` ... well, this one is doomed unless you use full-text search.

---

## IN, BETWEEN, and LIKE — the supporting cast

`IN` behaves like multiple `=` checks and is usually SARGable:

```
WHERE Status IN ('Open', 'Pending', 'Closed');
```

`BETWEEN` is inclusive on both ends, which can surprise you with datetime ranges. Use explicit ranges for dates.

```
WHERE OrderDate BETWEEN '2024-01-01' AND '2024-01-31';  -- includes midnight of Jan 31
```

`LIKE 'abc%'` is SARGable; the pattern starts at the leftmost anchor.

```
WHERE ProductCode LIKE 'ABC%';
```

---

## NULLs and filtering — the silent saboteur

A predicate comparing a column to a value will never match NULL rows.

```
WHERE Discount > 0   -- NULL rows fail the test
```

If you want to keep NULLs, you must say so:

```
WHERE Discount > 0 OR Discount IS NULL;
```

---

## Execution plan whispers

When filtering aligns with indexes, the plan shows *Index Seek* — the optimizer's equivalent of graceful ballet.

When predicates are non-SARGable, the plan shows *Index Scan* or *Table Scan* — a chore of checking everything.

You’ll recognize the pattern quickly once you start reading execution plans with **Actual Execution Plan** enabled.

---

Filtering is where intent meets performance. The next stop is joins, where multiple tables start interacting like characters in a large ensemble novel.