Database Management Systems
SQL Query Clause Order

Vikas Thammanna Gowda

02/11/2025

When writing an SQL query, the order of the clauses is important. Here's the typical order for the clauses:

1. `SELECT`
2. `FROM`
3. `WHERE`
4. `GROUP BY`
5. `HAVING`
6. `ORDER BY`
7. `LIMIT`

Each of these clauses plays a role in forming the final result set.

1 Sample Sales Table

Imagine you have a table named `Sales` with the following data:

SaleID	**CustomerID**	**Product**	**Amount**	**Date**	**Region**
1	C001	Widget	100	2024-01-01	North
2	C002	Gadget	200	2024-01-02	South
3	C001	Widget	150	2024-01-03	North
4	C003	Gadget	300	2024-01-04	East
5	C002	Widget	250	2024-01-05	South
6	C001	Gadget	180	2024-01-06	North
7	C004	Widget	220	2024-01-07	West
8	C003	Widget	130	2024-01-08	East
9	C004	Gadget	260	2024-01-09	West
10	C002	Gadget	300	2024-01-10	South

Table 1: Sample Sales Table

2 Clause-by-Clause Explanation with Examples

1. SELECT & DISTINCT

SELECT: This clause specifies which columns you want to see in your result set.

DISTINCT: Adding `DISTINCT` after `SELECT` removes duplicate rows from the result.

Example: To get a list of all unique customers who made a purchase

```
SELECT DISTINCT CustomerID
FROM Sales;
```

Result:

CustomerID
C001
C002
C003
C004

2. FROM

FROM: Specifies the table (or tables) from which to retrieve the data.

3. WHERE

WHERE: Filters rows before any grouping or ordering occurs. You can use conditions to restrict which rows are included.

IN: The `IN` operator allows you to specify multiple values in a `WHERE` clause. It checks if a column's value matches any value in a given list.

Example: To get sales transactions from either the North or East region

```
SELECT *
FROM Sales
WHERE Region IN ('North', 'East');
```

Result:

SaleID	CustomerID	Product	Amount	Date	Region
1	C001	Widget	100	2024-01-01	North
3	C001	Widget	150	2024-01-03	North
4	C003	Gadget	300	2024-01-04	East
6	C001	Gadget	180	2024-01-06	North
8	C003	Widget	130	2024-01-08	East

BETWEEN: The `BETWEEN` operator checks whether a column's value falls within a specified range (inclusive).

Example: To select sales transactions where the `Amount` is between 150 and 250 (inclusive)

```
SELECT *
FROM Sales
WHERE Amount BETWEEN 150 AND 250;
```

Result:

SaleID	CustomerID	Product	Amount	Date	Region
2	C002	Gadget	200	2024-01-02	South
3	C001	Widget	150	2024-01-03	North
5	C002	Widget	250	2024-01-05	South
6	C001	Gadget	180	2024-01-06	North
7	C004	Widget	220	2024-01-07	West

4. GROUP BY

GROUP BY: Groups rows that have the same value in one or more columns. When you use `GROUP BY`, SQL collects rows that share the same value(s) in the specified column(s) and treats them as a single group. This is often used with aggregate functions (like SUM(), AVG(), COUNT(), MIN(), MAX()) to produce summary results.

Example: Calculate the total sales per customer

```
SELECT CustomerID, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY CustomerID;
```

# This groups the rows by `CustomerID` and sums the `Amount` for each group.

What Happens in the Query?

1. Grouping

SQL groups the rows that have the same `CustomerID`:

- For C001, rows 1, 3, and 6 are grouped together.
- For C002, rows 2, 5, and 10 are grouped together.
- For C003, rows 4 and 8 are grouped together.
- For C004, rows 7 and 9 are grouped together.

2. Aggregation

Within each group, `SUM(Amount)` calculates the total sales:

- C001: $100 + 150 + 180 = 430$
- C002: $200 + 250 + 300 = 750$
- C003: $300 + 130 = 430$
- C004: $220 + 260 = 480$

Result: After executing the above query, you get:

CustomerID	TotalSales
C001	430
C002	750
C003	430
C004	480

5. HAVING

HAVING: Similar to WHERE, but it filters groups after the grouping has been done. It is used when you need to filter on aggregate values.

Example: In the previous query, suppose you only want to see customers whose total sales exceed 450

```
SELECT CustomerID, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY CustomerID
HAVING SUM(Amount) > 450;
```

Result:

CustomerID	**TotalSales**
C002	750
C004	480

6. ORDER BY

ORDER BY: Sorts the final result set by one or more columns. You can sort in ascending (ASC) or descending (DESC) order.

Example: Continuing with our grouped data, sort the results by TotalSales in descending order so that the highest totals come first:

Example: Calculate the total sales per customer and sort the total sales in descending order

```
SELECT CustomerID, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY CustomerID
ORDER BY TotalSales DESC;
```

Result: After executing the above query, you get:

CustomerID	**TotalSales**
C002	750
C004	480
C001	430
C003	430

7. LIMIT

LIMIT: Restricts the number of rows returned by the query. This is particularly useful for large data sets or when you only need a subset (such as the top result, top 3 results).

Example: To get only the top customer by total sales

```
SELECT CustomerID, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY CustomerID
ORDER BY TotalSales DESC
LIMIT 1;
```

Result:

CustomerID	**TotalSales**
C002	750