In SQL, there is a major difference between the **Lexical Order** (how you write the code) and the **Logical Execution Order** (how the database engine actually processes the data).

Understanding the execution order is crucial for debugging why an alias might not work in a `WHERE` clause or how a `HAVING` clause filters data.

---

1. The Lexical Order (Writing Order)

This is the standard syntax you follow when writing a query:

1. `SELECT`
2. `DISTINCT`
3. `FROM`
4. `JOIN` (on `ON` conditions)
5. `WHERE`
6. `GROUP BY`
7. `HAVING`
8. `ORDER BY`
9. `LIMIT / OFFSET`

---

2. The Logical Execution Order

The database engine doesn't start at the top. It starts by finding the data source and ends by limiting the results.

Step	Clause	What happens?
1	**FROM / JOIN**	The engine identifies the tables and performs joins to create a working dataset.
2	**WHERE**	Filters the raw rows based on specific conditions.
3	**GROUP BY**	Groups the filtered rows into buckets based on column values.
4	**HAVING**	Filters the *groups* (not individual rows) created in step 3.
5	**SELECT**	The engine finally determines which columns to return and calculates expressions/aliases.
6	**DISTINCT**	Removes duplicate rows from the selected columns.
7	**ORDER BY**	Sorts the final result set.

Step	Clause	What happens?
8	**LIMIT / OFFSET**	Discards rows outside the specified range.

---

3. Why This Matters (The "Alias" Trap)

A common mistake in SQL is trying to use an alias defined in the `SELECT` clause inside a `WHERE` clause.

Incorrect:
SQL
```
SELECT price * quantity AS total_cost
FROM orders
WHERE total_cost > 100; -- ERROR!
```

Why?

Looking at the execution order: WHERE happens at Step 2, but SELECT (where total_cost is defined) doesn't happen until Step 5. The database doesn't know what total_cost is yet.

Correct:

You must repeat the expression in the WHERE clause:

SQL
```
SELECT price * quantity AS total_cost
FROM orders
WHERE (price * quantity) > 100;
```

---

4. Where to use HAVING vs WHERE?

- **WHERE** is for filtering individual rows **before** they are grouped.
- **HAVING** is for filtering groups **after** `GROUP BY` has been performed (usually used with aggregate functions like `SUM`, `COUNT`, or `AVG`).

The Most Popular Mnemonic for remembering the sequence/order of execution

The most common way to remember this is by using a silly sentence where each word starts with the first letter of the SQL clause:

Fred **W**ants **G**reat **H**amburgers; **S**ome **D**on’t **O**ffer **L**ettuce.

- **F**red: **FROM** (and JOINs)
- **W**ants: **WHERE**
- **G**reat: **GROUP BY**
- **H**amburgers: **HAVING**
- **S**ome: **SELECT**
- **D**on't: **DISTINCT**
- **O**ffer: **ORDER BY**
- **L**ettuce: **LIMIT** / OFFSET