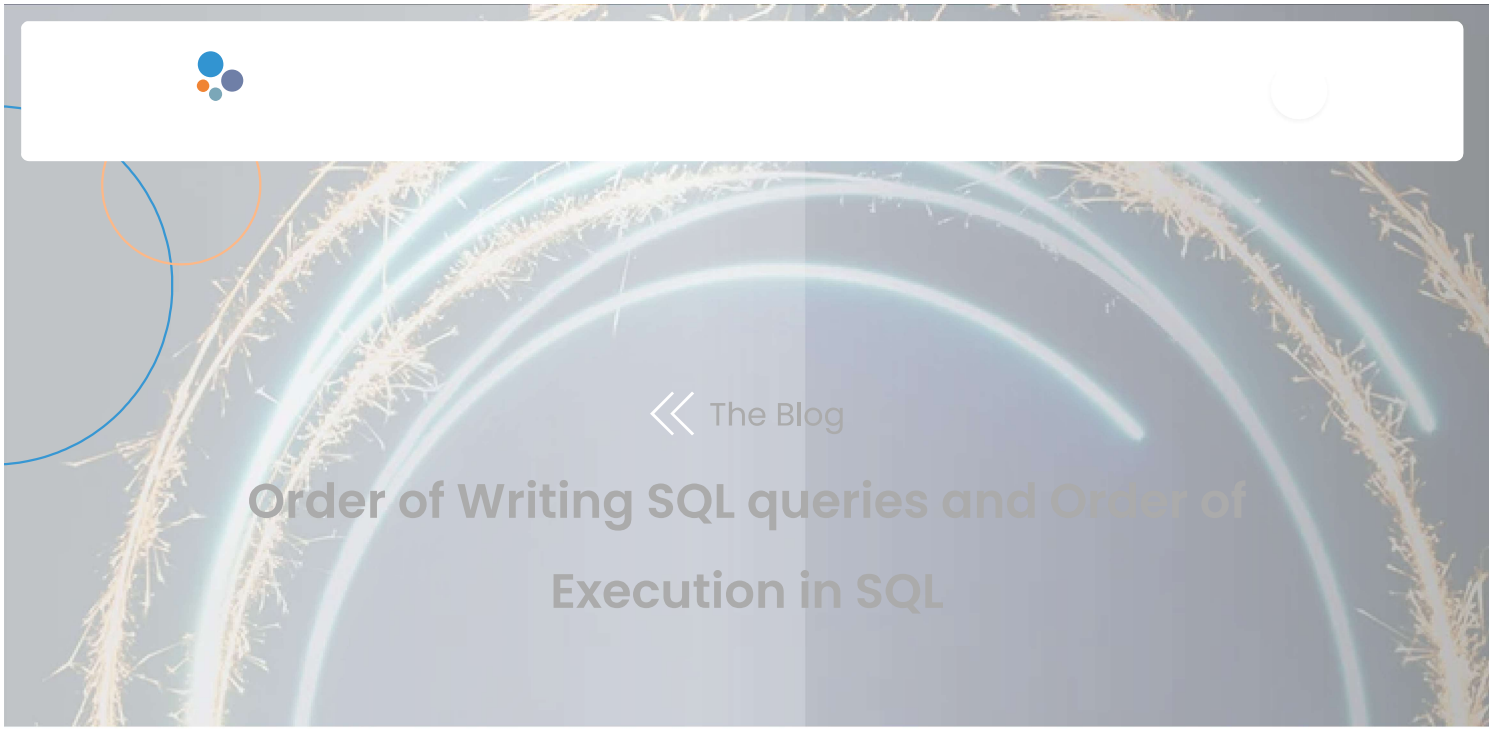

In Tableau, there is a table of the order of operations that tells the users the execution order in Tableau. The users will decide when to apply the filter based on that table. In SQL, the users usually apply some SQL clauses to write the query to return data following the order of writing SQL queries. Some common clauses are SELECT, FROM, WHERE, GROUP BY, ORDER BY.

In this blog, I am going to share:

- The order of writing SQL queries
- The order of execution in SQL

In each section, I will give examples and explain. In this blog, I use the superstore dataset from Tableau as an example dataset. Are you ready to explore how SQL queries run? If yes, let's get started!

Order of writing SQL queries

When writing a SQL query, we usually follow this order from the top to the bottom:

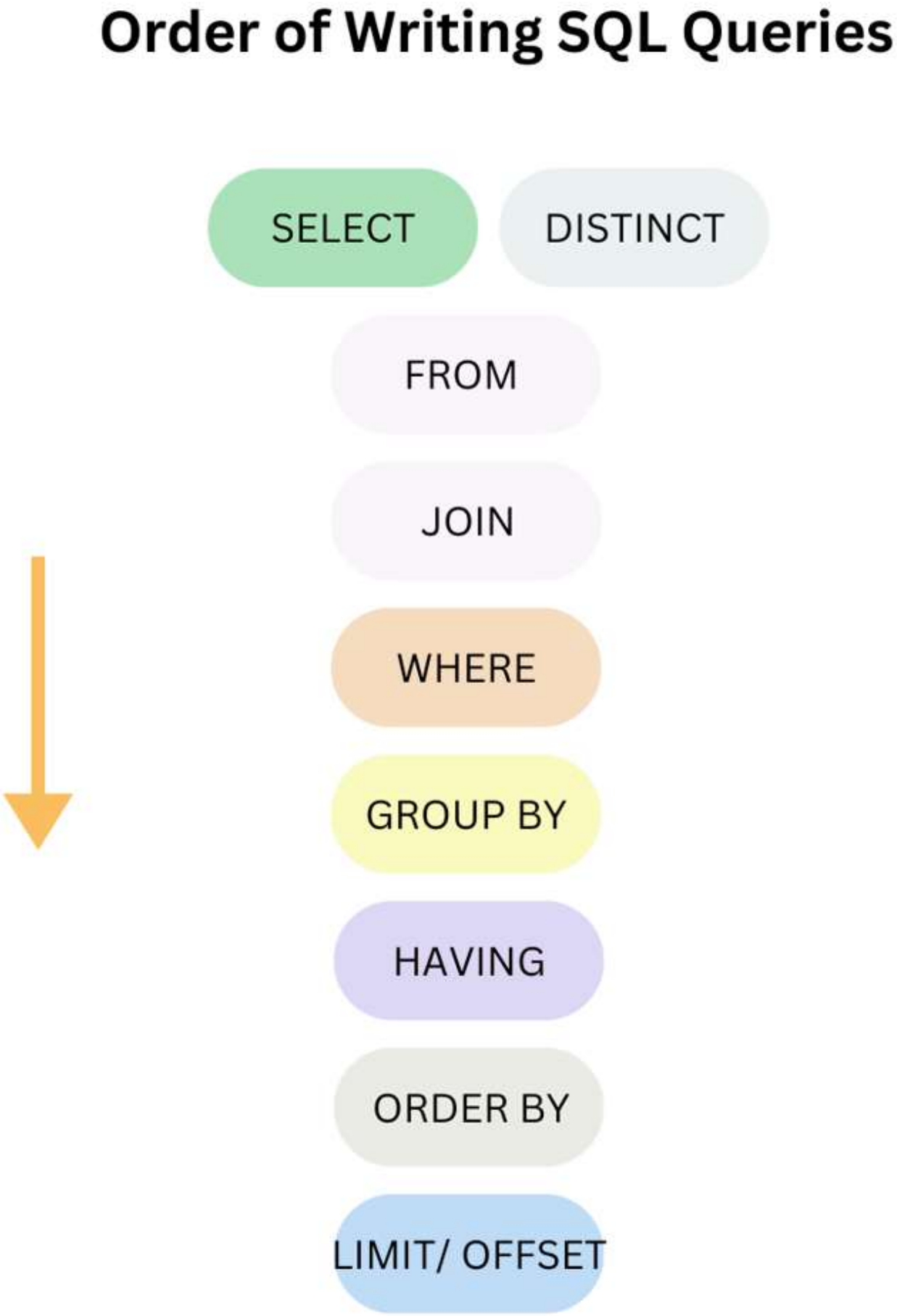

Fig. 1: Order of writing SQL queries

To explain that order in detail, I give a simple example:

Return the total sales by each month in 2021 for each category where the total sales must be greater than $25,000, and order the table by the category alphabetically and the order date in ascending.
Before writing the SQL query, let's analyze the problem first. I will list details of the problem following the SQL clause. It would be helpful to write SQL queries later and also keep tracking the requirements from the problem.

- **Return columns:** total sales, month and year of the order date, category
- **From table:** superstore
- **Condition:** in year 2021
- **Group by:** category and month year
- **Condition with aggregate function:** total sales must be greater than $25,000 (because I will use the SUM function to calculate the total sales)
- **Order:** category alphabetically and order date in ascending

After splitting the details of the problem into each category, and checking that we don't miss any details, we jump to write the SQL queries.

```
SELECT category,DATE_TRUNC('month',order_date) as month_year, SUM(sales) as total_sales
FROM superstore
WHERE YEAR(ORDER_DATE)='2021'
GROUP BY category,month_year
HAVING total_sales > 25000
ORDER BY category, month_year;
```

Following the order of writing SQL queries in Fig.1 above, I list all values that I want to see from the output with the SELECT clause at the front.

SELECT category, DATE_TRUNC('month', order_date) as month_year, SUM(sales) as total_sales

By the level of detail, I will show the category at the highest level. Then in each category, drilling down to each month and the last level will show the total sales. Verbally, I will read from the lowest level to the highest level by saying: "The total sales by each month of each category".

Therefore, in the SELECT clause, I will put the highest level at the beginning and the lowest level at the end. I used the DATE_TRUNC function to return the first day of each month and year instead of splitting one column for each month and another one for the year. For calculating the total sales value by each month and category, I need to use the "SUM" function.

Next, I need to allocate the table I will use the FROM clause with the table name "superstore".

There is a condition with the year 2021, so I use the WHERE clause with YEAR(Order Date) = '2021'.

For the total sales by each month and category, I need to group by the category and the month_year dimension.

With the condition for the aggregate function, I use the HAVING clause for the total sales greater than 25,000.

At the end, I will order the result table by category and the month_year dimension in ascending.

```
SELECT category,DATE_TRUNC('month',order_date) as month_year, SUM(sales) as total_sales
FROM superstore
WHERE YEAR(ORDER_DATE)='2021'
GROUP BY category,month_year
HAVING total_sales > 25000
ORDER BY category, month_year;
```

Results Chart

	CATEGORY	MONTH_YEAR	TOTAL_SALES
1	Furniture	2021-09-01	29028.206
2	Furniture	2021-11-01	37056.715
3	Furniture	2021-12-01	31407.4668
4	Office Supplies	2021-08-01	30059.852
5	Office Supplies	2021-09-01	31895.843
6	Office Supplies	2021-11-01	31472.337
7	Office Supplies	2021-12-01	30436.942
8	Technology	2021-03-01	33428.622
9	Technology	2021-09-01	26942.603
10	Technology	2021-10-01	32855.663
11	Technology	2021-11-01	49918.773

Fig.2: SQL query and the result table

In Fig.2, I typed in the SQL query as I explained above in Snowflake and executed the query. I got the result table at the bottom. It shows the total sales of each month_year for each category in 2021 ordered by category and month_year.

In this section, I showed you how to write the SQL query in the order of writing SQL queries following the requirements of the problem.

But does SQL run the same as that order or in a different order? Let's check it out in the next part!

Order of Execution in SQL

In the previous section, I shared how to write a SQL query following the order of writing SQL queries. But at the base, does SQL run the same as that order? Knowing the order of execution in SQL is important because it would be helpful for us to know how SQL runs in the system that we can't see and also give us some hints to find some efficient ways to write SQL queries to improve the performance.

SQL will run the query in a different order **not the same** as the order of writing SQL queries.

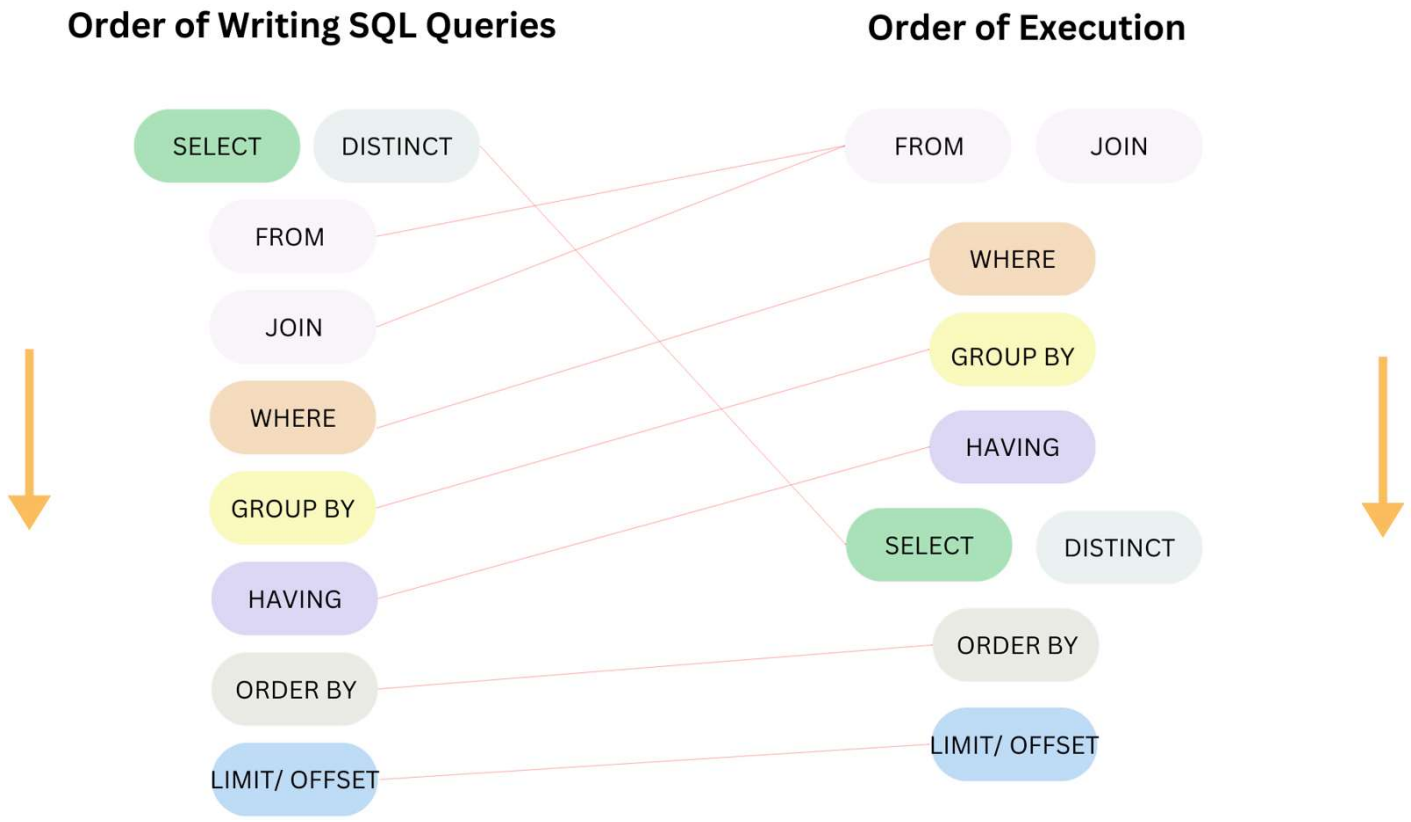

Fig. 3: The order of Writing SQL Queries and the order of execution in SQL

In Fig.3, in the right column, that is the order of execution in SQL. It will run from the top to the bottom. I will explain each stage. I use the same example above and apply it in this order of execution.

```
SELECT category,DATE_TRUNC('month',order_date) as month_year, SUM(sales) as total_sales
FROM superstore
WHERE YEAR(ORDER_DATE)='2021'
GROUP BY category,month_year
HAVING total_sales > 25000
ORDER BY category, month_year;
```

1/ FROM/ JOIN

SQL needs to know which databases and tables it will pull from. Therefore, it will run the FROM clause first. If there is a JOIN clause to join with another table, the JOIN clause also runs from this stage to decide which table to join and on which reference keys.

Example from the SQL query: In this stage, the SQL query will pull the data from the superstore table.

2/ WHERE

After deciding which table to pull data from and merging data with which tables on reference keys, the WHERE clause will be executed. At this stage, it will only keep rows in which the condition returns True. It will help to reduce some rows if the condition is satisfied.

Example from the SQL query: the SQL query will check if the year of the order date field is 2021. If the condition is True, it will only keep those rows.

3/ GROUP BY

After filtering data from the WHERE clause, the GROUP BY clause will be executed next. If there is an aggregate function in the SELECT clause, the GROUP BY clause will help to reduce some rows and keep the distinct dimension values listed here.

Example from the SQL query: I specified 2 dimensions to group which are category and month_year. Now, it will keep the unique category on the leftmost column for the highest level of detail. Then, for each unique category, it will show the unique month_year value.

The DATE_TRUNC function is executed in this stage and the aggregate function (Sum of Sales) that is also executed to group by those dimensions in the system.

4/ HAVING

When the SQL query has the GROUP BY clause, it means there is an aggregate function in the query. Similar to the WHERE clause, the HAVING clause will filter the data again but based on the aggregate function (not the dimension as the WHERE clause). It will also reduce some rows if the condition is satisfied.

Example from the SQL query: some RDBMS (Relational Database Management System) requires you to write the aggregate function the same as you write in the SELECT statement (the aliases are not allowed). However, some RDBMS allows you to use the aliases in the HAVING clause.

After the GROUP BY clause, the aggregate function (sum of sales) was also executed. In the HAVING clause, the system will check if the condition from the aggregate function is True and keep those rows.

5/ SELECT

Do you still remember where the SELECT clause stays in the order of writing SQL queries? In the first stage. But here, the SELECT clause will be executed after GROUP BY and HAVING. The SELECT clause will return the columns we specified and do some calculations or execute aggregate functions (if listed).

The DISTINCT clause is executed after the SELECT clause to remove the duplicated values and returns the unique value.

Example from the SQL query: in this stage, we specify columns to output including the category, month_year fields, and the aggregate function for the sum of sales.

If I delete the aggregate function in the SELECT statement, the query is still working well. The aggregate function for the sum of sales was executed before. (Fig. 4)

```sql
SELECT category,DATE_TRUNC('month',order_date) as month_year
FROM superstore
WHERE YEAR(ORDER_DATE)='2021'
GROUP BY category,month_year
HAVING SUM(sales) > 25000
ORDER BY category, month_year;
```

Results Chart

	CATEGORY	MONTH_YEAR
1	Furniture	2021-09-01
2	Furniture	2021-11-01
3	Furniture	2021-12-01
4	Office Supplies	2021-08-01
5	Office Supplies	2021-09-01
6	Office Supplies	2021-11-01
7	Office Supplies	2021-12-01
8	Technology	2021-03-01
9	Technology	2021-09-01
10	Technology	2021-10-01
11	Technology	2021-11-01

Fig. 4: If no aggregate function in SELECT, but exists in HAVING

6/ ORDER BY

After the SELECT clause is executed, the ORDER BY clause will be executed next. It will order the dimensions we specified in ascending or descending order. It will order the dimensions from the left to the right (If specified). The ORDER BY clause could affect the performance if the query is complicated and many dimensions to order.

Example from the SQL query: after filtering data, and grouping by dimension, now the query will order by the dimensions that I specified. I didn't put the ASC (ascending) or DESC (descending). By default, SQL will set it as ascending.

7/ LIMIT/ OFFSET

Lastly, the SQL query will only return the number of rows if we specify the LIMIT or OFFSET clause. It can reduce some rows before returning the data.

In this blog, I shared how the SQL query works in the system in 2 different views. The first view is about the order of writing SQL queries that the user usually obeys the rules from SQL (SELECT -> FROM -> WHERE ...). The second view is the order of execution that we can't see.

I hope after this blog, you know the difference between 2 of those views and how the SQL query executes each clause in the system. Based on the order of execution in SQL, I can get some tips to share on how to improve the

performance in SQL. I will write it for the next blog.

I hope you enjoy my blog and hope to see you soon!

Author:
Le Luu

VIEW PROFILE

ABOUT

APPLY

LOCATIONS

THE TEAM

COACHES

BLOG

EVENTS

PRIVACY POLICY

TERMS OF USE

COOKIE POLICY

Subscribe to our Newsletter

Get the lastest news about The Data School and application tips

Your email address

SUBSCRIBE NOW