Group By and Aggregation

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

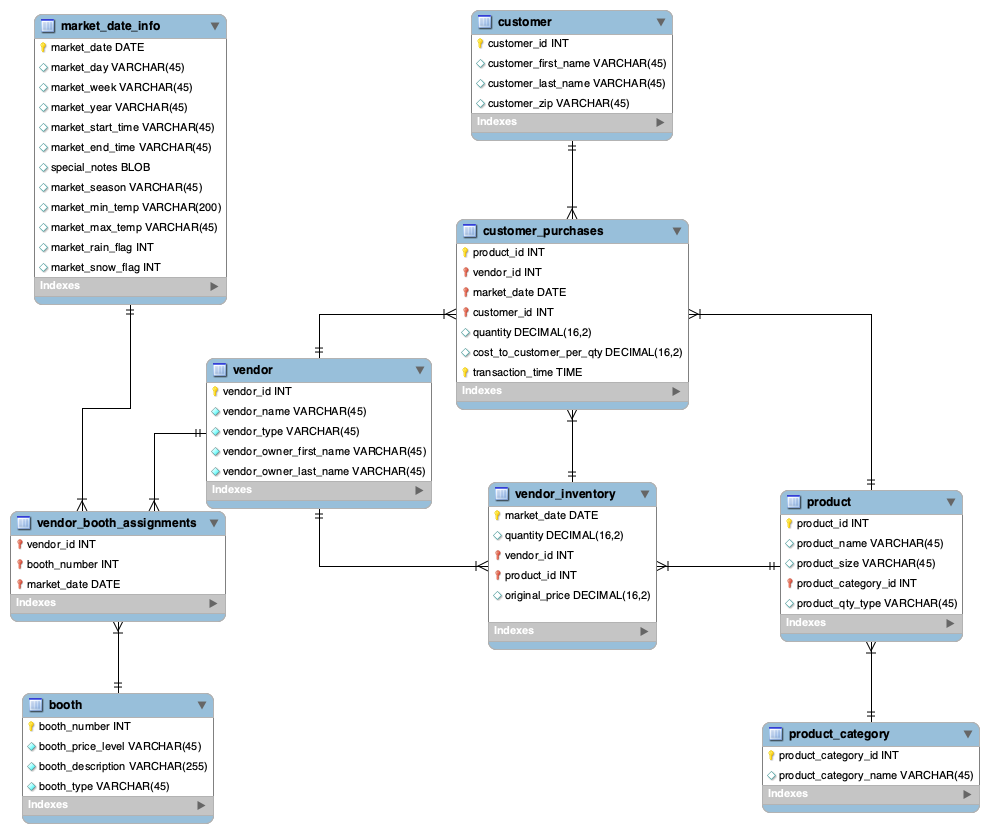
**Please note that any topics that are not covered in today's lecture will be covered in the next lecture.**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement:

You are a Data Analyst at Amazon Fresh. You have been tasked to study the Farmer’s Market.

Dataset: Farmer’s Market database



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Aggregate Functions**

SQL starts becoming more powerful when you use it to aggregate data.

Here we have a bunch of aggregate functions that take all the values present in a column as input and return a single value as a result.

They are commonly used with the SELECT statement to perform calculations on groups of rows or to **summarise data**.

Later on, we’ll talk about groups also but for now, let’s just stick to performing aggregations on a column.

1. **MIN()** - Returns the minimum value of a column.
2. **MAX()** - Returns the maximum value of a column.
3. **COUNT()** - Returns the number of rows in a column
4. **SUM()** - Returns the sum of values in a column
5. **AVG()** - Returns the average of values in a column

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: ​​We want to get the most and least expensive items available in the vendor’s inventory.

We will use the **vendor\_inventory** table, which has a field for the original price the vendors set for each item they bring to market on each market date.

First, let’s look at all of the available fields in the vendor\_inventory table by using the select \* query.

SELECT \*

FROM farmers\_market.vendor\_inventory

ORDER BY original\_price

LIMIT 10;

**MIN and MAX**

We can get the least and most expensive item prices in the entire table by using the MIN() and MAX() aggregate functions.

**Query:**

SELECT

MIN(original\_price) AS min\_price,

MAX(original\_price) AS max\_price

FROM farmers\_market.vendor\_inventory;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question: ​​We want to calculate how much revenue has been generated in total by purchases made by our customers.

We will use the **customer\_purchases** table that has the details of the quantity of items purchased by the customers and the per-quantity

cost of that item.

**SUM**

We first need to multiply the column **quantity** with **cost\_to\_customer\_per\_qty** and then we canuse the SUM() aggregate function to get the total revenue generated.

**Query:**

SELECT

SUM(quantity \* cost\_to\_customer\_per\_quantity) AS total\_revenue

FROM farmers\_market.customer\_purchases;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question: ​​We want to find out the average quantity of products purchased by the customers on the date ‘2019-05-01’.

We will use the **customer\_purchases** table that has the details of the quantity of items purchased by the customers on specific dates.

Let us also have a look at the data present in the customer\_purchased table using the same select \* query.

SELECT \*

FROM farmers\_market.vendor\_inventory

ORDER BY original\_price

LIMIT 10;

## **AVG**

We can use the AVG() aggregate function to get the average quantity of items purchased by the customers.

**Query:**

SELECT

AVG(quantity) AS avg\_qty

FROM farmers\_market.customer\_purchases

WHERE market\_date = ‘2019-05-01’;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: We want to know the total number of purchases that happened during the second quarter of the year 2019.

**COUNT**

We use the COUNT(\*) function to count the number of records in the **customer\_purchases** table that meet the specified criteria.

**Query:**

SELECT

COUNT(\*) AS total\_purchases

FROM farmers\_market.customer\_purchases

WHERE

market\_date >= '2019-04-01'

AND market\_date <= '2019-06-30';

## 

Follow-up Question: What if we ask you to get the number of unique customers who made any purchases during the second quarter of the year 2019?

## 

## For this, you can use the **COUNT()** function along with the **DISTINCT** keyword to count only the unique customer IDs from the **customer\_purchases** table for the specified period.

## 

## **Query:**

SELECT

COUNT(DISTINCT customer\_id)

FROM farmers\_market.customer\_purchases

WHERE

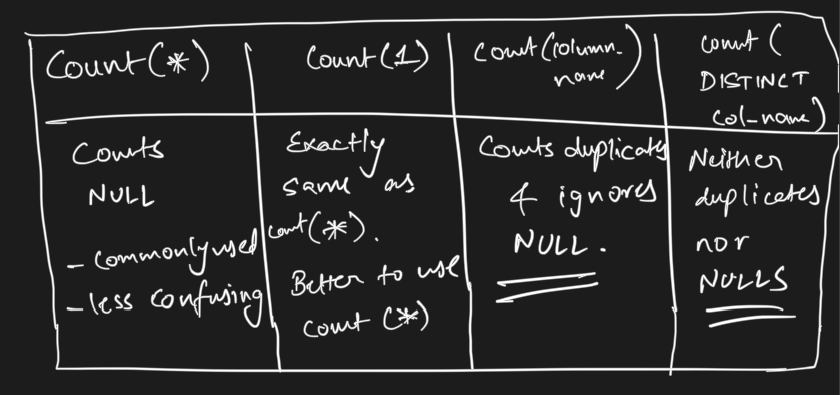
market\_date >= '2019-04-01'

AND market\_date <= '2019-06-30’;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 

## **Difference between COUNT(\*) vs COUNT(1) vs COUNT(col\_name) vs COUNT(DISTINCT col\_name)**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Group By**

Using the GROUP BY statement, you can specify the level of summarization and then use aggregate functions to summarize values for the records in each group.

## **Syntax:**

SELECT [columns to return]

FROM [table]

WHERE [conditional filter statements]

GROUP BY [columns to group on]

HAVING [conditional filter statements that are run after grouping]

ORDER BY [columns to sort on]

LIMIT [first x number of rows to be selected];

The GROUP BY keyword is followed by a comma-separated list of column names that indicate how you want to summarize the query results.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: Count the number of purchases each customer made per market date.

**Query:**

SELECT

market\_date,

customer\_id,

COUNT(\*) AS num\_purchases

FROM farmers\_market.customer\_purchases

GROUP BY market\_date, customer\_id

ORDER BY market\_date, customer\_id

LIMIT 10;

Keep an eye on the **granularity** of the table data -

* The granularity of the **customer\_purchases** table is such that if a customer were to buy three identical items, such as tomatoes, at once from a vendor, that would show up as 1 in the ***num\_purchases*** column of this query’s output, since the item purchase is recorded in one row in the table, with a quantity value of 3.
* If the customer were to buy three tomatoes, walk away from the stand, and then purchase another three tomatoes, that would be counted as two by the preceding query since the new separate purchase would generate a new line in the database.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Alternate Question: Calculate the total quantity purchased by each customer per market\_date.

Here, we’ll use the SUM() function.

**Query:**

SELECT

market\_date,

customer\_id,

SUM(quantity) AS total\_qty\_purchased

FROM farmers\_market.customer\_purchases

GROUP BY market\_date, customer\_id

ORDER BY market\_date, customer\_id

LIMIT 10;

* **Note**: It’s important to **understand the granularity and structure** of the underlying table to ensure that your result means what you think it does.
* Thus, write the query without aggregation first to see the values you will be summarizing before grouping the results.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question: Calculate the total price paid by customer\_id 3 per market\_date.

## We can perform calculations inside the Aggregation Functions

**Query:**

SELECT

market\_date,

SUM(quantity \* cost\_to\_customer\_per\_qty) AS total\_spent

FROM farmers\_market.customer\_purchases

WHERE

customer\_id = 3

GROUP BY market\_date

ORDER BY market\_date;

* The price will be calculated per row of the table, and then the results will be summed up per group.

**Important:** ​​

* Notice that **vendor\_id** **has been removed from the list of columns** to be displayed and from the ORDER BY clause.
* That’s because if we want the aggregation level of one row per customer per date, we **can’t also include** **vendor\_id** in the output, **because the customer can purchase from multiple vendors on a single date**, so the results wouldn’t be aggregated at the level we wanted.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question: Count how many products were for sale on each market date, or how many different products each vendor offered.

We can determine these values using COUNT and COUNT DISTINCT.

COUNT will count up the rows within a group when used with GROUP BY, and COUNT DISTINCT will count up the unique values present in the specified field within the group.

* To determine how many products are offered for sale each market date, we can count up the rows in the **vendor\_inventory** table, grouped by date.
* This doesn’t tell us what quantity of each product was offered or sold, but counts the number of products available, because there is a row in this table for each product for each vendor for each market date.

**Query:**

SELECT

market\_date,

COUNT(product\_id) AS product\_count

FROM farmers\_market.vendor\_inventory

GROUP BY market\_date

ORDER BY market\_date;

If we wanted to know how many different products, with unique product IDs each vendor brought to **market during a date range**, we could use COUNT DISTINCT on the product\_id field, like so:

**Query:**

SELECT

vendor\_id,

COUNT(DISTINCT product\_id) AS different\_products\_offered

FROM farmers\_market.vendor\_inventory

WHERE

market\_date BETWEEN '2019-04-03' AND '2019-05-16'

GROUP BY vendor\_id

ORDER BY vendor\_id;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Question: In addition to the count of different products per vendor, we also want the average original price of a product per vendor.

We can add a line to the preceding query, and use the **AVG()** function.

**Query:**

SELECT

vendor\_id,

COUNT(product\_id) AS different\_products\_offered,

AVG(original\_price) AS average\_product\_price

FROM farmers\_market.vendor\_inventory

WHERE market\_date BETWEEN '2019-04-03' AND '2019-05-16'

GROUP BY vendor\_id, product\_id

ORDER BY vendor\_id;

**But is this the average product price?**

* Is it fair to call it “average product price” when the underlying table has one row per type of product?
* If the vendor brought 100 tomatoes to market, those would all be in one line of the underlying vendor inventory table, so the price of a tomato would only be included in the average once.
* If you calculated the “average product price” for the vendor this way, you would just get the average price of one tomato and one bouquet.

How to calculate the price per item?

To get an actual average price of items in each vendor’s inventory between the specified dates, it might make more sense to multiply the quantity of each type of item by the price of that item, which is a calculation that would occur per row, then sum that up and divide by the total quantity of items, which is a calculation that would occur per vendor.

Let’s try a calculation that includes these two summary values.

**Query:**

SELECT

vendor\_id,

COUNT(DISTINCT product\_id) AS different\_products\_offered,

SUM(quantity \* original\_price) AS value\_of\_inventory,

SUM(quantity) AS inventory\_item\_count,

ROUND(SUM(quantity \* original\_price) / SUM(quantity), 2) AS

average\_inventory\_price

FROM farmers\_market.vendor\_inventory

WHERE

market\_date BETWEEN '2019-04-03' AND '2019-05-16'

GROUP BY vendor\_id

ORDER BY vendor\_id;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Filtering with Having**

Filtering is another thing that can be done in the query after summarization occurs.

Using the **HAVING** clause allows you to filter the results of a query after the aggregate functions are applied, to grouped data.

This filters the groups based on the summary values.

**Syntax:**

SELECT aggregate\_function(col)

FROM table

GROUP BY col

HAVING condition;

Recall the **Order of Execution** of a SQL query (as discussed earlier):

* **FROM** - The database gets the data from tables in the FROM clause and if necessary, performs the JOINs.
* **WHERE** - The data is filtered based on the conditions specified in the WHERE clause. Rows that do not meet the criteria are excluded.
* **GROUP** **BY** - After filtering the rows using the WHERE clause, the rows that remain are grouped together based on the columns specified in the GROUP BY clause.
* **Aggregate** **functions** - The aggregate functions are applied to the groups created in the GROUP BY clause.
* **HAVING** - The HAVING clause filters the groups of rows based on aggregate functions applied to the grouped data.
* **SELECT** - After grouping and filtering, the SELECT clause specifies which columns and aggregate functions should be included in the result set.
* **ORDER BY** - It allows you to sort the result set based on one or more columns, either in ascending or descending order.
* **OFFSET** - The specified number of rows are skipped from the beginning of the result set.
* **LIMIT** - After skipping the rows, the LIMIT clause is applied to restrict the number of rows returned.

### The HAVING clause is executed after the WHERE and Group By clauses.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Question: Find the average amount spent on each market day. We want to consider only those days where the number of purchases were more than 3 and every single transaction amount must be greater than 5.

**Query:**

SELECT

market\_date,

ROUND(AVG(quantity \* cost\_to\_customer\_per\_qty), 2) AS avg\_spent

FROM farmers\_market.customer\_purchases

WHERE quantity \* cost\_to\_customer\_per\_qty > 5

GROUP BY market\_date

HAVING COUNT(\*) > 3

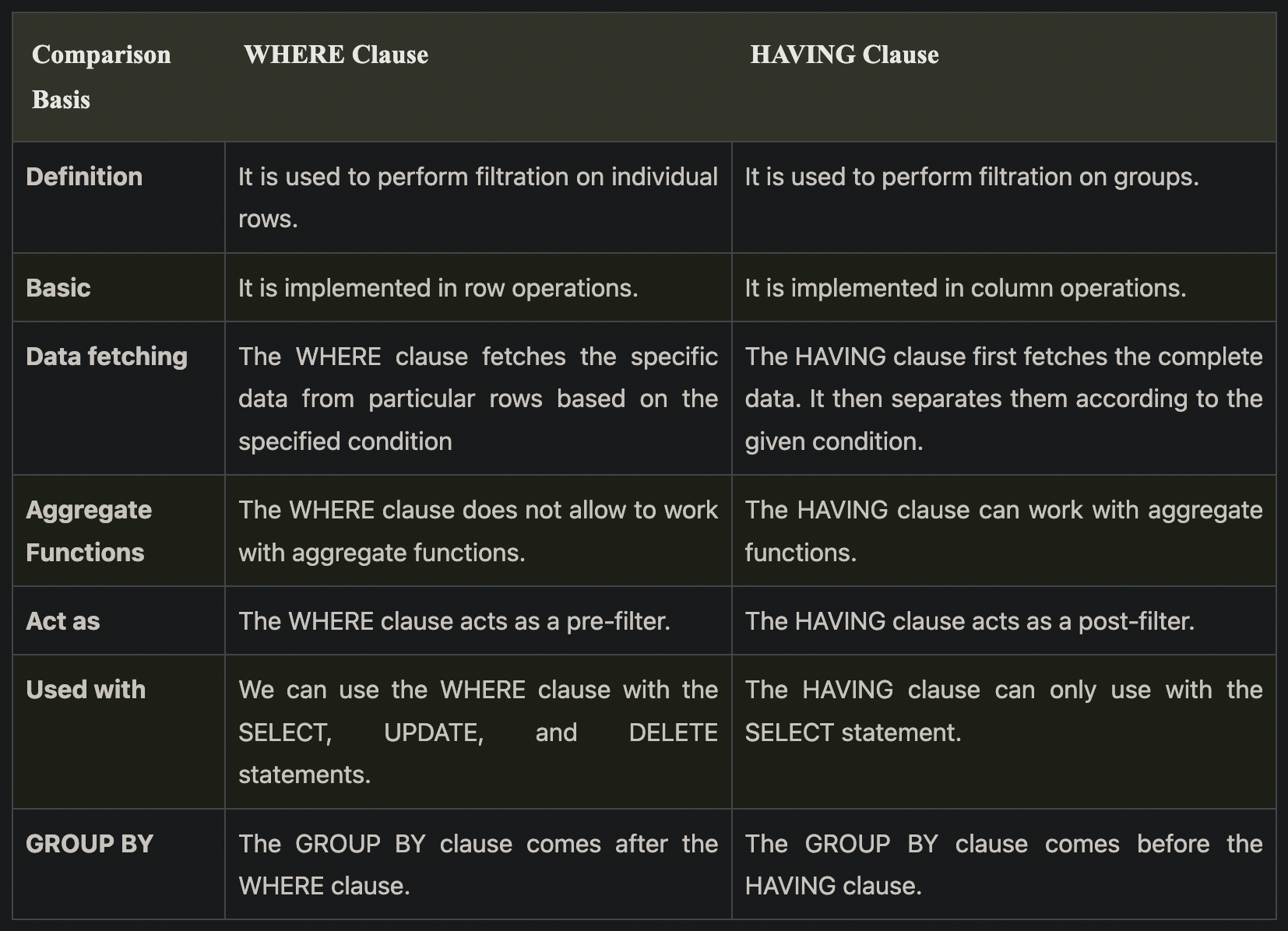
ORDER BY market\_date;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **HAVING and WHERE**

The main difference between the WHERE & HAVING clause is that

* the WHERE clause is used to specify a condition for filtering records before any groupings are made,
* while the HAVING clause is used to specify a condition for filtering values from a group.

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_