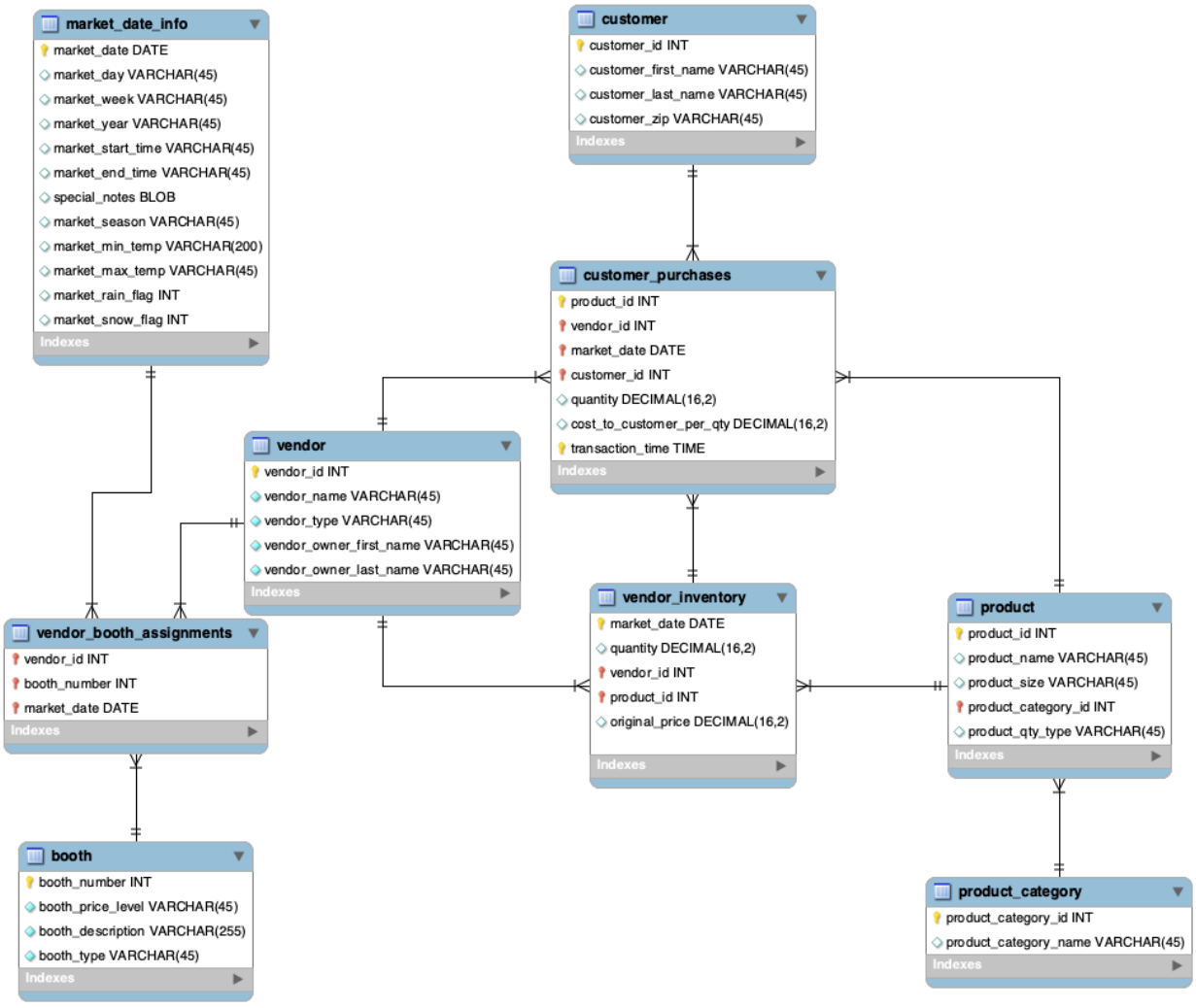
Group By and Aggregation

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 result.

They are commonly used with the SELECT statement to perform calculations on groups of rows or to **summarize 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'll again use the **customer_purchases** table here.

SUM

We first need to multiply the column **quantity** with **cost_to_customer_per_qty** and then we can use 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 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';
```

How count () works

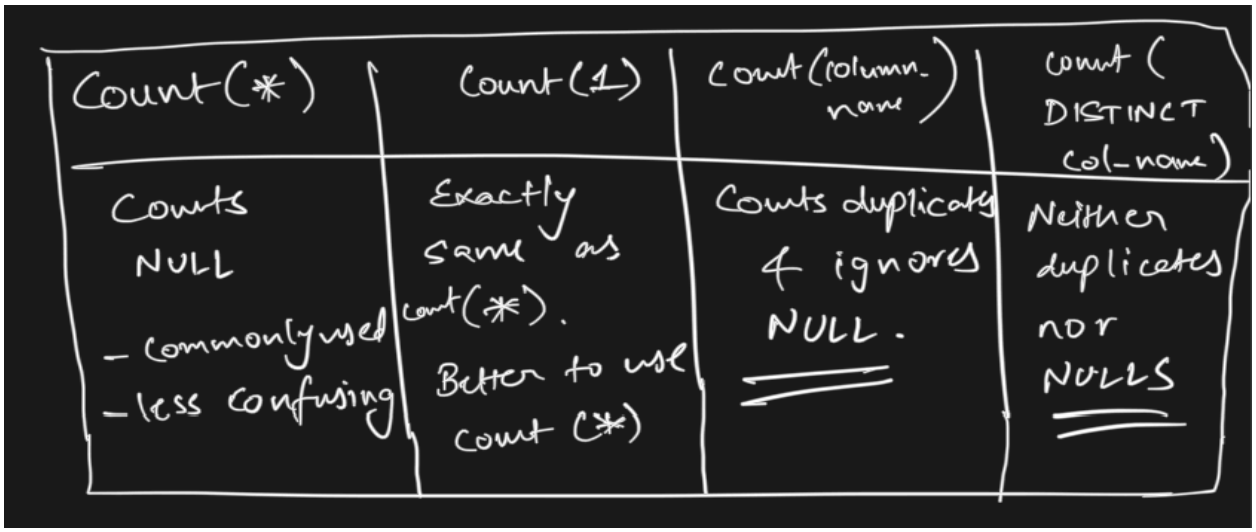
	Orders	value
▸	A	10
	A	15
	C	10
	D	NULL
	NULL	NULL

```
1 •  SELECT
2        COUNT(*),
3        COUNT(1),
4        COUNT(2),
5        COUNT(999),
6        COUNT(Orders),
7        COUNT(value),
8        COUNT(DISTINCT Orders),
9        COUNT(DISTINCT value)
10   FROM
11       temp_testing_1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(*)	COUNT(1)	COUNT(2)	COUNT(999)	COUNT(Orders)	COUNT(value)	COUNT(DISTINCT Orders)	COUNT(DISTINCT value)
5	5	5	5	4	3	3	2

Difference between COUNT(*) vs COUNT(1) vs COUNT(col_name) vs COUNT(DISTINCT col_name)



Now that you all have seen how these aggregate functions work, let's move onto how we can use them on a group of records.

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.

Illustration:

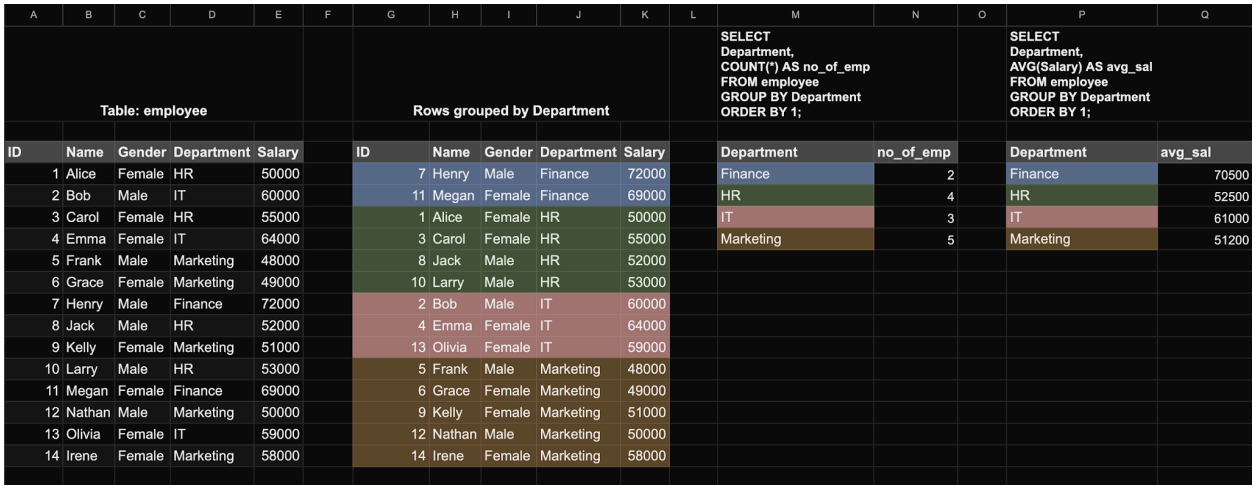


Table: employee

ID	Name	Gender	Department	Salary
1	Alice	Female	HR	50000
2	Bob	Male	IT	60000
3	Carol	Female	HR	55000
4	Emma	Female	IT	64000
5	Frank	Male	Marketing	48000
6	Grace	Female	Marketing	49000
7	Henry	Male	Finance	72000
8	Jack	Male	HR	52000
9	Kelly	Female	Marketing	51000
10	Larry	Male	HR	53000
11	Megan	Female	Finance	69000
12	Nathan	Male	Marketing	50000
13	Olivia	Female	IT	59000
14	Irene	Female	Marketing	58000

Rows grouped by Department

ID	Name	Gender	Department	Salary
7	Henry	Male	Finance	72000
11	Megan	Female	Finance	69000
1	Alice	Female	HR	50000
3	Carol	Female	HR	55000
8	Jack	Male	HR	52000
10	Larry	Male	HR	53000
2	Bob	Male	IT	60000
4	Emma	Female	IT	64000
13	Olivia	Female	IT	59000
5	Frank	Male	Marketing	48000
6	Grace	Female	Marketing	49000
9	Kelly	Female	Marketing	51000
12	Nathan	Male	Marketing	50000
14	Irene	Female	Marketing	58000

```
SELECT
Department,
COUNT(*) AS no_of_emp
FROM employee
GROUP BY Department
ORDER BY 1;
```

Department	no_of_emp
Finance	2
HR	4
IT	3
Marketing	5

```
SELECT
Department,
AVG(Salary) AS avg_sal
FROM employee
GROUP BY Department
ORDER BY 1;
```

Department	avg_sal
Finance	70500
HR	52500
IT	61000
Marketing	51200

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 keywords are followed by a comma-separated list of column names that indicate how you want to summarize the query results.

Order of Execution :

In a SQL query, the Group By clause comes right after the WHERE clause, followed by the Aggregate functions.

- **FROM** - The database gets the data from tables in 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.
- **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.

Question: Get a list of the customers who made purchases on each market date.

If I write this...

```
SELECT
    market_date,
    customer_id
 FROM farmers_market.customer_purchases
 ORDER BY market_date, customer_id;
```

- One row per item each customer purchased, **displaying duplicates in the output**, because you're querying the customer_purchases table with no grouping specified.

Query:

```
SELECT
    market_date,
    customer_id
 FROM farmers_market.customer_purchases
 GROUP BY market_date, customer_id
 ORDER BY market_date, customer_id;
```

- You can also accomplish the same result by using SELECT DISTINCT to remove duplicates,
- We are using GROUP BY to add summary columns to the output.

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 **granularity** of the table data -

- The granularity of the **customer_purchases** table is such that if a customer was 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 was 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.

Slightly complex question: How many different kinds of products were purchased by each customer on each market date?

Query:

```
SELECT
    market_date,
    customer_id,
    COUNT(DISTINCT product_id) AS
different_products_purchased
FROM farmers_market.customer_purchases
GROUP BY market_date, customer_id
ORDER BY market_date;
```

- So instead of counting how many rows there were in the customer_purchases table per customer per market date, like we did with COUNT(*),
- or adding up the quantities, like we did with SUM(quantity),
- we're identifying **how many unique product_id** values exist across those rows in the group.

Add the qty summary to the same query:

```
SELECT
    market_date,
    customer_id,
    SUM(quantity) AS total_qty_purchased,
    COUNT(DISTINCT product_id) AS
different_products_purchased
FROM farmers_market.customer_purchases
GROUP BY market_date, customer_id
ORDER BY market_date, customer_id;
```
