Filtering and Subqueries

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

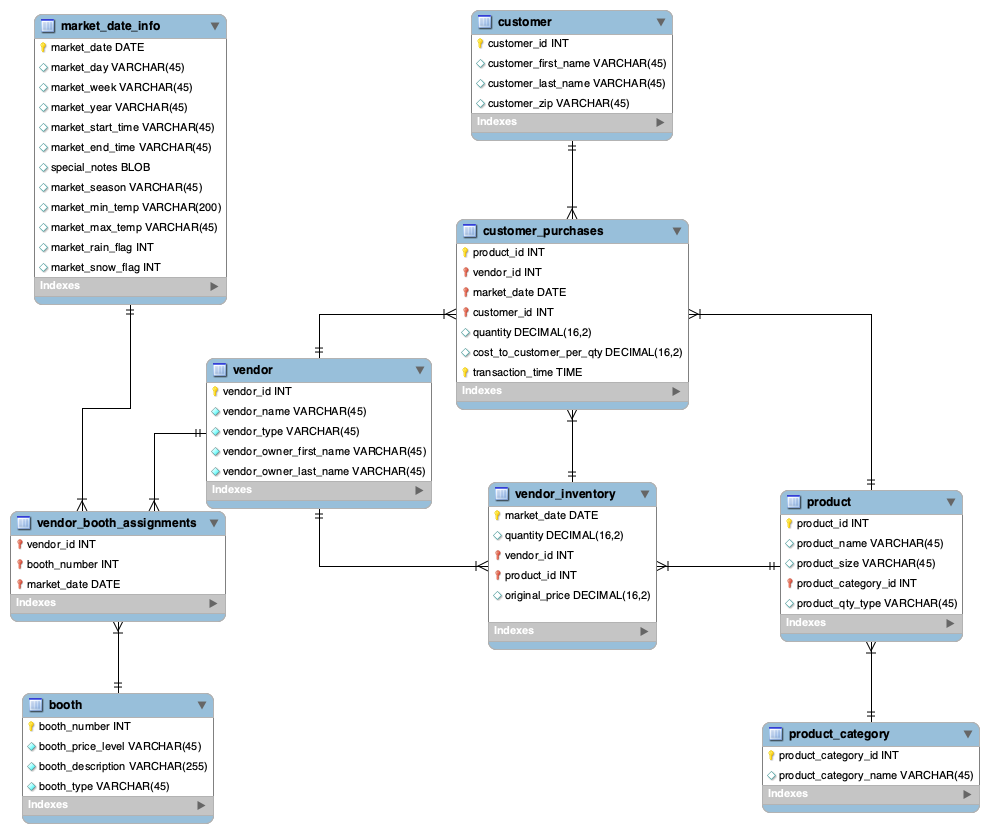
**Please note that any topics that are not covered in today's lecture will be covered in the next lecture.**

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

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



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

**Transition** -> Up until now, we have manipulated just numbers, what about strings?

## How to work with strings in SQL?

There are also **inline functions** that can be used to modify string values in SQL, as well.

**Question:**The management wants to create promotional tags for vendors and their booths. Your task is to generate a report with the following details using **only** the vendor and booth tables:

1. **Vendor Tag:**
   * Convert the vendor\_name to **uppercase** and concatenate it with the **first 3 characters** of the vendor\_type (in lowercase), separated by an underscore (\_).
   * Example: FRESH FARMS\_veg for vendor\_name = 'Fresh Farms' and vendor\_type = 'Vegetable'.
2. **Booth Label:**
   * Create a booth label by concatenating the **last 2 characters** of the booth\_type with the **booth\_number**, separated by a hyphen (-).
   * Example: th-12 for booth\_type = 'North' and booth\_number = 12.
3. **Custom Identifier:**
   * Generate a custom identifier by combining the **first 4 characters** of the vendor\_owner\_first\_name (in uppercase) with the **last 3 characters** of the vendor\_owner\_last\_name (in lowercase).
   * Example: JOHNson for vendor\_owner\_first\_name = 'Johnathan' and vendor\_owner\_last\_name = 'Henderson'.



SELECT

CONCAT(UPPER(vendor\_name), '\_', LOWER(SUBSTR(vendor\_type, 1, 3))) AS Vendor\_Tag,

CONCAT(SUBSTR(booth\_type, -2), '-', booth\_number) AS Booth\_Label,

CONCAT(UPPER(SUBSTR(vendor\_owner\_first\_name, 1, 4)), LOWER(SUBSTR(vendor\_owner\_last\_name, -3))) AS Custom\_Identifier

FROM

`farmers\_market.vendor`, `farmers\_market.booth`;

**CONCAT()**

* We can accomplish that by using the CONCAT() function.
* The list of string values you want to merge together are entered into the CONCAT() function as parameters.
* A space can be included by surrounding it with quotes.

**UPPER()** is a function that capitalizes string values.

* It’s also possible to nest functions inside other functions, which the SQL interpreter executes **from the “inside” to the “outside.”**
* We can enclose the UPPER() function inside the CONCAT() function to uppercase the full name.

Similar to the UPPER() function, we also have a **LOWER()** function in case we want to turn some string value into lowercase.

Now, we can use a function called **SUBSTR()**.

**Syntax:** SUBSTR(value, position, length)

* **position** tells us which character to start from.
* **length** tells us up to which character we should include.

In BigQuery, we also have the **INITCAP()** function that we can use.

It takes a STRING and returns it with the first character in each word in uppercase and all other characters in lowercase.

SELECT

CONCAT(INITCAP("john")," ", INITCAP("doe")) AS full\_name;

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

Question: Extract all the product names that are part of product category 1

Filtering data - The **WHERE** Clause

* The WHERE clause is the part of the SELECT statement in which you list conditions that are used to determine which rows in the table should be included in the results set.
* In other words, the WHERE clause is used for filtering of records.

SELECT

product\_id, product\_name, product\_category\_id

FROM farmers\_market.product

WHERE product\_category\_id = 1

LIMIT 5

Similarly, what if we want all the product names that are part of any other product category except 1?

We can do this using the != or <> (Not equal to)operator.

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

Question: Find the booth assignments for vendor\_id 7 for all dates between April 3, 2019 and May 16, 2019, including the 2 dates.

## Filtering on multiple conditions - using operators - **“AND”**, **“OR”**, **“NOT”**

* Conditions with OR between them will jointly evaluate to TRUE, meaning the row will be returned, if any of the clauses are TRUE.
* Conditions with AND between them will only evaluate to TRUE in combination if all of the clauses evaluate to TRUE. Otherwise, the row will not be returned.
* **Remember that NOT flips the following boolean value to its opposite (TRUE becomes FALSE, and vice versa).**

SELECT \*

FROM `farmers\_market.vba`

WHERE vendor\_id = 7 AND

(market\_date >= "2019-04-03" AND market\_date <= "2019-05-18")

Whenever you have such range-based questions, you can use the **BETWEEN** keyword.

SELECT \*

FROM `farmers\_market.vba`

WHERE vendor\_id = 7 AND

(market\_date BETWEEN "2019-04-03" AND "2019-05-18")

**NOTE:**

Between is inclusive of the upper & lower limit of the range specified.

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

Question: Return a list of customers with the following last names: [Diaz, Edwards, Wilson]

To solve this question. we could use a long list of **OR** comparisons.

SELECT

customer\_id,

customer\_first\_name,

customer\_last\_name

FROM farmers\_market.customer

WHERE

customer\_last\_name = 'Diaz'

OR customer\_last\_name = 'Edwards'

OR customer\_last\_name = 'Wilson'

**Alternative approach -**

Whenever we have a list of values to filter from, we can use the **IN** keyword and provide a comma-separated list of values to compare against.

SELECT

customer\_id,

customer\_first\_name,

customer\_last\_name

FROM farmers\_market.customer

WHERE

customer\_last\_name IN ('Diaz' , 'Edwards', 'Wilson')

In this case, the **IN** keyword will return TRUE for any row with a customer\_last\_name that is in the provided list.

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

Question: What if we want a list of all customers except those with the last names: [Diaz, Edwards, Wilson]

In such a case, we can use the **NOT** operator along with the IN keyword to exclude the customers having last names as 'Diaz' , 'Edwards' or 'Wilson'.

SELECT

customer\_id,

customer\_first\_name,

customer\_last\_name

FROM farmers\_market.customer

WHERE

customer\_last\_name NOT IN ('Diaz' , 'Edwards', 'Wilson')

Please note that the last names inside the IN clause are case sensitive.

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

Question: You want to get data about a customer you knew as Jerry but you are not sure if they are listed as Jeremy or Jeremiah or Jerry. Get all customers whose name starts with “jer”.

In SQL, you can search for **partially matched strings** using a **comparison operator** called **LIKE**, and **wildcard characters**, which serve as a placeholder for unknown characters in a string.

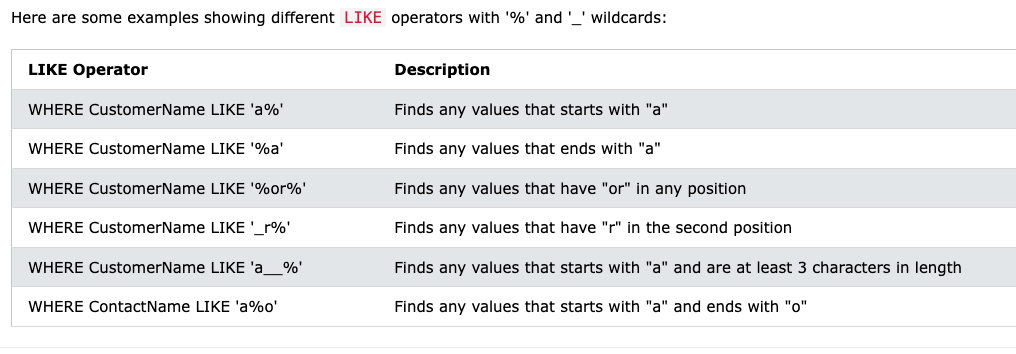
**Wildcards :**

* **%** - stand-in for 0 or more characters.
* **\_** (underscore) - stand-in for one and only one character

SELECT \*

FROM `farmers\_market.customer`

WHERE lower(customer\_first\_name) LIKE "jer%"



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

### Question: Your manager wants to see all the **unique** customer IDs present in the `customer\_purchases` table. How would you get this data?

There can be situations when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only the unique ones.

In that case, you can use the **DISTINCT** keyword in conjunction with the SELECT statement to **eliminate all the duplicate rows**.

**Syntax:**

SELECT DISTINCT col1, col2, …

FROM DB.table

Even if you want to fetch the **unique** values from a single column, you can do so using the DISTINCT keyword.

SELECT

DISTINCT customer\_id

FROM farmers\_market.customer\_purchases

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

**Order of Execution** of a SQL query :

* **FROM** - The database gets the data from tables in the FROM clause.
* **WHERE** - The data is filtered based on the conditions specified in the WHERE clause. Rows that do not meet the criteria are excluded.
* **SELECT** - After filtering is done, the SELECT statement determines which columns to include in the final result set.
* **DISTINCT** - The DISTINCT keyword is applied within the SELECT clause to ensure that only unique values are returned for the specified columns.
* **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: Find all of the products from the `product` table that don’t have their sizes mentioned.

The absence of any value in a cell is represented by the **NULL** keyword.

You can use the **IS NULL** operators in a WHERE clause to filter rows that have missing or null values in a specific column.

SELECT \*

FROM farmers\_market.product

WHERE product\_size IS NULL

NOTE: Keep in mind that **“blank”** and **NULL** are not the same thing in database terms.

* If someone asked you to find all products that didn’t have product sizes, you might also want to check for blank strings, which would equal “”, or rows where someone entered a space or any number of spaces into that field.
* The **TRIM()** function **removes excess spaces from the beginning or end of a string value**, so if you use a combination of the TRIM() function and blank string comparison, you can find any row that is blank or contains only spaces.

SELECT \*

FROM farmers\_market.product

WHERE

product\_size IS NULL

OR TRIM(product\_size) = “”

* You might wonder why the comparison operator IS NULL is used instead of equals NULL just like numbers.
* **NULL is not actually a value, it’s the absence of a value, so it can’t be compared to any existing value.**
* If you wanted to return all records that don’t have NULL values in a field, you could use the condition “[field name] **IS NOT NULL**” in the WHERE clause.

SELECT \*

FROM farmers\_market.product

WHERE

product\_size IS NOT NULL

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

Question: What if you’re asked to fetch all the product IDs from the `products` table and set a default value “medium” in rows where the product size is NULL?

We can use the **IFNULL()** function to check for null values in a column and if found, replace them with some other value.

SELECT

product\_id,

IFNULL(product\_size, “medium”)

FROM farmers\_market.product

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

**Subqueries**

* A Subquery is a SQL query embedded within the WHERE clause of another SQL query.
* A subquery is also called an **inner/child** query, while the statement containing a subquery is also called an **outer/parent** query.
* A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
* Note that the **inner query executes first before its parent query** so that the results of an inner query can be passed to the outer query.

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

Question: Analyze purchases made at the market on days when it rained.

There is a value in the *market\_date\_info* **table** called *market\_rain\_flag* that has a value of 0 if it didn’t rain while the market was open and a value of 1 if it did.

* 0 - it didn’t rain
* 1 - it did

First, let’s write a query that gets a list of market dates when it rained, using this query:

SELECT market\_date, market\_rain\_flag

FROM farmers\_market.market\_date\_info

WHERE market\_rain\_flag = 1

Now let’s use the list of dates generated by that query to return purchases made on those dates.

SELECT \*

FROM `farmers\_market.customer\_purchases`

WHERE market\_date IN (SELECT market\_date

FROM `farmers\_market.market\_date\_info`

WHERE market\_rain\_flag = 1 )

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

**CASE statement**

What if you want to take some action based on a certain condition.

For example -

“If [one condition] is true, then [take this action].

Otherwise, [take this other action].”

That’s where CASE statements come into play.

**Syntax:**

SELECT *cols*,

CASE

WHEN [first conditional statement]

THEN [value or calculation]

WHEN [second conditional statement]

THEN [value or calculation]

ELSE [value or calculation]

END AS *alias*

FROM *table*

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

**IF() function**

**Syntax:**

IF(condition, true\_value, false\_value)

**Parameters used:**

* **condition** – It is used to specify the condition to be evaluated.
* **true\_value** – It is an optional parameter that is used to specify the value to be returned if the condition evaluates to be true.
* **false\_value** – It is an optional parameter that is used to specify the value to be returned if the condition evaluates to be false.

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

**Question:**

The management wants to analyse customer purchase behaviours. Your task is to create a report from the **customer\_purchases** table that classifies purchases based on the following criteria

1. **Purchase Volume Category:**
   * If the quantity is **greater than 5**, label it as **'Bulk Purchase'**.
   * If the quantity is **between 3 and 5** (inclusive), label it as **'Moderate Purchase'**.
   * If the quantity is **less than 3**, label it as **'Small Purchase'**.
2. **Transaction Timing:**
   * If the transaction\_time is **before 12:00:00**, label it as **'Morning Purchase'**.
   * If the transaction\_time is **between 12:00:00 and 17:59:59**, label it as **'Afternoon Purchase'**.
   * If the transaction\_time is **after 18:00:00**, label it as **'Evening Purchase'**.

SELECT \*,

-- Purchase Volume Classification

CASE

WHEN quantity > 5 THEN 'Bulk Purchase'

WHEN quantity BETWEEN 3 AND 5 THEN 'Moderate Purchase'

ELSE 'Small Purchase'

END AS Purchase\_Volume,

-- Transaction Timing Classification

CASE

WHEN transaction\_time < '12:00:00' THEN 'Morning Purchase'

WHEN transaction\_time BETWEEN '12:00:00' AND '17:59:59' THEN 'Afternoon Purchase'

ELSE 'Evening Purchase'

END AS Transaction\_Timing

FROM `farmers\_marketcustomer\_purchases`

ORDER BY customer\_id;

We can do the same using the **IF() function** as well -

SELECT \*,

-- Purchase Volume Classification

IF(quantity > 5, 'Bulk Purchase',

IF(quantity BETWEEN 3 AND 5, 'Moderate Purchase', 'Small Purchase')

) AS Purchase\_Volume,

-- Transaction Timing Classification

IF(transaction\_time < '12:00:00', 'Morning Purchase',

IF(transaction\_time BETWEEN '12:00:00' AND '17:59:59', 'Afternoon Purchase', 'Evening Purchase')

) AS Transaction\_Timing

FROM `farmers\_marketcustomer\_purchases`

ORDER BY customer\_id;

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