SQL - 02 Queries

Question: In the customer purchases, we have quantity and cost per qty separate, query the total amount that the customer has paid along with the date, customer id, vendor_id, qty, cost per qty, and the total amt.?

```
SELECT
        market_date,
   customer_id,
   product_id,
   quantity,
   cost_to_customer_per_qty,
   ROUND(quantity * cost_to_customer_per_qty, 2) AS total_amt
FROM customer_purchases;
```

ROUND Function

```
SELECT ROUND(505.50, -3)
```

FLOOR Function

```
SELECT FLOOR(34.76)
```

Question: We want to merge each customer's name into a single column that contains the first name, then a space, and then the last name.

- 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.

```
SELECT

customer_id,

CONCAT(customer_first_name, " ", customer_last_name) AS customer_name

FROM farmers_market.customer

LIMIT 5
```

How to Capitalise the string? - Using SUBSTRING()

Question: We need only the first letter of the first name and last name to be capitalized.

```
SELECT

    customer_id,

    CONCAT( UPPER(SUBSTR(customer_first_name, 1, 1)), SUBSTR(customer_first_name, 2), " ", UPPER(SUBSTR(customer_last_name, 1, 1)), SUBSTR(customer_last_name, 2 )) AS capitalised_name

 FROM farmers_market.customer

 ORDER BY customer_last_name, customer_first_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.

The WHERE clause goes after the FROM statement and before any GROUP BY, ORDER BY, or LIMIT statements in the SELECT query

Let's get those product names that are in category 1:

```
SELECT
product_id, product_name, product_category_id
FROM farmers_market.product
WHERE product_category_id = 1
LIMIT 5
```

Question: Print a report of everything ***customer_id 4*** has ever purchased at the farmer's market, sorted by market date, vendor ID, and product ID.

```
SELECT
        market_date, customer_id,
        vendor_id, product_id,
        quantity,
        quantity * cost_to_customer_per_qty AS price
 FROM farmers_market.customer_purchases
 WHERE customer_id = 4
```

Question: Get all the product info for products with id between 3 and 8 (not inclusive) and of product with id 10.

```
SELECT
    product_id,
    product_name
 FROM farmers_market.product
 WHERE
    product_id = 10
    OR (product_id > 3
    AND product_id < 8)
```

Question: Find the booth assignments for vendor 7 for any market date that occurred between April 3, 2019, and May 16, 2019, including either of the two dates.

```
SELECT *

 FROM farmers_market.vendor_booth_assignments

 WHERE

        vendor_id = 7

      AND market_date BETWEEN '2019-04-03' and '2019-05-16'

 ORDER BY market_date
```

IN - Keyword

Question: Return a list of customers with selected last names - [Diaz, Edwards and Wilson].

Approach 1: 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'
```

Approach 2: An alternative way to do the same thing, which may come in handy if you had a long list of names, is to use the IN keyword and provide a comma-separated list of values to compare against.

The IN keyword will return TRUE for any row with a customer_last_name that is in the provided list.

```
SELECT
    customer_id,
    customer_first_name,
    customer_last_name
 FROM farmers_market.customer
 WHERE
    customer_last_name IN ('Diaz' , 'Edwards', 'Wilson')
```

Question: You want to get data about a customer you knew as “Jerry,” but you aren’t sure if he was listed in the database as “Jerry” or “Jeremy” or “Jeremiah.”

All you knew for sure was that the first three letters were “Jer.”

In SQL, instead of listing every variation you can think of, 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.

% wildcard

The wildcard character % (percent sign) can serve as a stand-in for any number of characters (including none).

So the comparison LIKE ‘Jer%’ will search for strings that start with “Jer” and have any (or no) additional characters after the “r”:

```
SELECT
    customer_id,
    customer_first_name,
    customer_last_name
 FROM farmers_market.customer
 WHERE
    customer_first_name LIKE 'Jer%'
```

Question: Find all of the products from the product table without sizes.

```
SELECT *
FROM product
WHERE product_size IS NULL        OR TRIM(product_size) = " %";
```

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

```
SELECT *
FROM customer_purchases
WHERE market_dates IN
(
        SELECT market_date
        FROM market_date_info
        WHERE market_rain_flag = 1
)
```

Question: List down all the Product details where product_category contains 'Fresh'.

```
SELECT *
FROM product
WHERE product_category_id IN
(
        SELECT product_category_id
        FROM product_category
        WHERE product_category_name LIKE "%fresh%"
)
```