SQL-04 | Joins Aggregations/Window Functions

Lecture Queries

Question: Write a query that returns a list of all customers who did not purchase on March 2, 2019.

Question: Write a query that returns a list of all customers who did not purchase on March 2, 2019.

```
SELECT c.*, cp.market_date
 FROM customer AS c
 LEFT JOIN customer_purchases AS cp
     ON c.customer_id = cp.customer_id
 WHERE cp.market_date <>
'2019-03-02'  # the < > operator - greater
than or less than but not that value.
```

```
SELECT c.*, cp.market_date
 FROM customer AS c
 LEFT JOIN customer_purchases AS cp
     ON c.customer_id = cp.customer_id
 WHERE (cp.market_date <> '2019-03-02' OR cp.market_date IS NULL)
```

Question: Let’s say we want details about all farmer’s market booths, as well as every vendor booth assignment for every market date.

List all the customers and their associated purchases?

Question: Let’s say we want details about all farmer’s market booths, as well as every vendor booth assignment for every market date.

```
SELECT
    b.booth_number,
    b.booth_type,
    vba.market_date,
    v.vendor_id,
    v.vendor_name,
    v.vendor_type
 FROM booth AS b
    LEFT JOIN vendor_booth_assignments AS vba ON b.booth_number = vba.
 booth_number
    LEFT JOIN vendor AS v ON v.vendor_id = vba.vendor_id
 ORDER BY b.booth_number, vba.market_date
```

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

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

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

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

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

```
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
```

Question: Calculate the total quantity that each customer bought per market date.

Question: Calculate the total quantity that each customer bought per market date.

```
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
```

Question: how many different kinds of products were purchased by each customer on each market date:

Question: how many different kinds of products were purchased by each customer on each market date:

```
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, customer_id
```

Question: Calculate the total price paid by customer_id 3 per market_date.

Question: Calculate the total price paid by customer_id 3 per market_date.

```
SELECT
        customer_id,
        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
```

Question: Let’s add some customer details and vendor details to these results. Customer details are in the customer table and vendor details are in the vendor table.

Question: Let’s add some customer details and vendor details to these results. Customer details are in the customer table and vendor details are in the vendor table.

```
SELECT
    c.customer_first_name,
    c.customer_last_name,
    cp.customer_id,
    v.vendor_id,
    v.vendor_name,
    SUM(quantity * cost_to_customer_per_qty) AS total_price
FROM customer AS c
LEFT JOIN customer_purchases AS cp
    ON c.customer_id = cp.customer_id
LEFT JOIN vendor AS v
    ON cp.vendor_id = v.vendor_id
GROUP BY
    cp.customer_id,
    v.vendor_id,
```

Question: We want to get the most and least expensive items per product category, considering the fact that each vendor sets their own prices and can adjust prices per customer.

Question: We want to get the most and least expensive items per product category, considering the fact that each vendor sets their own prices and can adjust prices per customer.

```
SELECT
    pc.product_category_name,
    p.product_category_id,
    MIN(vi.original_price) AS minimum_price,
    MAX(vi.original_price) AS maximum_price
 FROM farmers_market.vendor_inventory AS vi
    INNER JOIN farmers_market.product AS p
        ON vi.product_id = p.product_id
    INNER JOIN farmers_market.product_category AS pc
        ON p.product_category_id = pc.product_category_id
 GROUP BY pc.product_category_name, p.product_category_id
```

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

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

```
SELECT
     market_date,
     COUNT(product_id) AS
product_count
  FROM
farmers_market.vendor_inventory
  GROUP BY market_date
  ORDER BY market_date
```

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

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

```
SELECT
    vendor_id,
    COUNT(DISTINCT product_id) AS different_products_offered,
    AVG(original_price) AS average_product_price
 FROM farmers_market.vendor_inventory
 GROUP BY vendor_id
 ORDER BY vendor_id
```

```
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_item_price
 FROM
farmers_market.vendor_inventory
 GROUP BY vendor_id
 ORDER BY vendor_id
```

Question: Filter out vendors who brought at least 10 items to the farmer’s market over the time period - 2019-03-02 and 2019-03-16

Question: filter out vendors who brought at least 10 items to the farmer’s market over the time period - 2019-03-02 and 2019-03-16

```
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,
    SUM(quantity * original_price) / SUM(quantity) AS average_item_price
 FROM farmers_market.vendor_inventory
 WHERE market_date BETWEEN '2019-03-02' AND '2019-03-16'
 GROUP BY vendor_id
 HAVING inventory_item_count >= 100
 ORDER BY vendor_id
```

WINDOW Functions

Window Functions

Window fns give the ability to put the values from one row of data into context compared to a group of rows, or partition.

We can answer questions like

- If the dataset were sorted, where would this row land in the results?
- How does a value in this row compare to a value in the prior row?
- How does a value in the current row compare to the average value for its group?

So, window functions **return group aggregate calculations alongside individual row-level** information for items in that group, or partition.

New Question: Extract the most expensive items and the **product_id** they are associated with per vendor.

```
SELECT
            vendor_id,
            market_date,
            product_id,
            original_price,
        ROW_NUMBER() OVER (PARTITION BY vendor_id ORDER BY original_price DESC)
AS price_rank
FROM farmers_market.vendor_inventory
```

RANK()

The RANK function numbers the results just like ROW_NUMBER does, but gives rows with the same value the same ranking.

```
SELECT
      vendor_id,
      market_date,
      product_id,
      original_price,
      RANK() OVER (PARTITION BY vendor_id ORDER BY
original_price DESC) AS
  price_rank
      FROM farmers_market.vendor_inventory
```

DENSE_RANK()

If you don’t want to skip rank numbers for tied values like in case of RANK, use the DENSE_RANK function.

```
SELECT
    vendor_id,
    market_date,
    product_id,
    original_price,
    DENSE_RANK() OVER (PARTITION BY vendor_id ORDER
BY original_price DESC) AS
 price_rank
    FROM farmers_market.vendor_inventory
```

Return the “**top tenth**” of the inventory, when sorted by price?

The dynamic solution is to use the NTILE function.

```
SELECT
	vendor_id,
	market_date,
	product_id,
	original_price,
	NTILE(10) OVER (ORDER BY original_price DESC) AS price_ntile
FROM farmers_market.vendor_inventory
ORDER BY original_price DESC
```

Question: As a farmer, you want to figure out which of your products were above the average price per product on each market date?

Question: As a farmer, you want to figure out which of your products were above the average price per product on each market date?

```
SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
        AVG(original_price) OVER (PARTITION BY market_date ORDER BY
       market_date) AS average_cost_product_by_market_date
 FROM farmers_market.vendor_inventory
```

Extract the farmer’s products that have prices above the market date’s average product cost.

- Using a **subquery**, we can filter the results to a single vendor, with **vendor_id 8**, and
- only **display products that have prices above the market date’s average product** cost.

Extract the farmer’s products that have prices above the market date’s average product cost.

- Using a **subquery**, we can filter the results to a single vendor, with **vendor_id 8**, and
- only **display products that have prices above the market date’s average product** cost.

```
SELECT * FROM
	(
		SELECT
			vendor_id,
			market_date,
			product_id,
			original_price,
			ROUND(AVG(original_price) OVER (PARTITION BY market_date ORDER
	BY market_date), 2) AS average_cost_product_by_market_date
FROM farmers_market.vendor_inventory )x
WHERE x.vendor_id = 8
			AND x.original_price > x.average_cost_product_by_market_date
```

Question: Count how many different products each vendor brought to market on each date, and displays that count on each row.

Question: Count how many different products each vendor brought to market on each date, and displays that count on each row.

```
SELECT
    vendor_id,
    market_date,
    product_id,
    original_price,
    COUNT(product_id) OVER (PARTITION BY market_date, vendor_id)
 vendor_product_count_per_market_date
    FROM farmers_market.vendor_inventory
 ORDER BY vendor_id, market_date, original_price DESC
```

Question: Using the **vendor_booth_assignments** table in the Farmer’s Market database, display each vendor’s booth assignment for each ***market_date*** alongside their previous booth assignments

Question: Using the **vendor_booth_assignments** table in the Farmer’s Market database, display each vendor’s booth assignment for each ***market_date*** alongside their previous booth assignments

```
SELECT
        market_date,
        vendor_id,
        booth_number,
        LAG(booth_number,1) OVER (PARTITION BY vendor_id ORDER BY market_date,
vendor_id) AS previous_booth_number
FROM farmers_market.vendor_booth_assignments
ORDER BY market_date, vendor_id, booth_number
```

Question: The Market manager may want to filter these query results to a specific market date to determine which vendors are new or changing booths that day, so we can contact them and ensure setup goes smoothly.

Question: The Market manager may want to filter these query results to a specific market date to determine which vendors are new or changing booths that day, so we can contact them and ensure setup goes smoothly.

```
SELECT * FROM
  (
     SELECT
         market_date,
          vendor_id,
          booth_number,
          LAG(booth_number,1) OVER (PARTITION BY vendor_id ORDER BY market_
          date, vendor_id) AS previous_booth_number
FROM farmers_market.vendor_booth_assignments
ORDER BY market_date, vendor_id, booth_number
  ) x
WHERE x.market_date = '2019-04-10'
         AND (x.booth_number <> x.previous_booth_number OR x.previous_
  booth_number IS NULL)
```

Question: Let’s say you want to find out if the total sales on each market date are higher or lower than they were on the previous market date.

Question: Let’s say you want to find out if the total sales on each market date are higher or lower than they were on the previous market date.

```
SELECT
    market_date,
    SUM(quantity * cost_to_customer_per_qty) AS market_date_total_sales,
    LAG(SUM(quantity * cost_to_customer_per_qty), 1) OVER (ORDER BY
 market_date) AS previous_market_date_total_sales
 FROM farmers_market.customer_purchases
 GROUP BY market_date
```