Applied Analytics Using SQL

/*We will be working with the following tables throughout this course:

1. employees contains information about the employees of Northwind.

2. customers contains information about Northwind's customers.

3. products contains information about the products sold.

4. categories contains information about product categories.

5. suppliers contains information about the companies that supply products.

6. orders contains general information about orders made by Northwind's customers.

7. order_items contains information about the individual items in each customer order.

*/

1. For each product, display its name (product_name), the name of the category it belongs to (category_name), quantity per unit (quantity_per_unit), the unit price (unit_price), and the number of units in stock (units_in_stock). Order the results by unit price.

```
SELECT p.product_name,c.category_name,
p.quantity_per_unit, p.unit_price, p.units_in_stock
FROM PRODUCTS p INNER JOIN CATEGORIES c
ON p.category_id = c.category_id
order by p.unit_price
;
```

2. We like to see information about all the suppliers who provide the store four or more different products. Show the following columns: supplier_id, company_name, and products_count (the number of products supplied).

```
SELECT s.supplier_id, s.company_name,
count(product_id) AS products_count
FROM SUPPLIERS s INNER JOIN PRODUCTS p
ON s.supplier_id = p.supplier_id
group by 1,2
having count(product_id) >= 4
;
```

3. Display the list of products purchased in the order with ID equal to 10250. Show the following information: product name (product_name), the quantity of the product ordered (quantity), the unit price (unit_price from the order_items table), the discount (discount), and the order_date. Order the items by product name.

```
Select p.product_name, oi.quantity,

    oi.unit_price, oi.discount, o.order_date

from orders o

INNER JOIN order_items oi

on o.order_id = oi.order_id

INNER JOIN products p

on oi.product_id = p.product_id

where o.order_id = '10250'

order by 1

;
```

4. Show the following information related to all items with order_id = 10248: the product name, the unit price (taken from the order_items table), the quantity, and the name of the suppliers company (as supplier_name).

```
Select p.product_name,

            oi.unit_price,

    oi.quantity,

    s.company_name as supplier_name

From orders o

INNER JOIN order_items oi

on o.order_id = oi.order_id

INNER JOIN products p

on oi.product_id = p.product_id

INNER JOIN suppliers s

on s.supplier_id = p.supplier_id

where o.order_id = '10248';
```

5. -- Which language appears to be more popular?

```
WITH base_table AS (
SELECT DISTINCT title,
case when title LIKE '%python%' then 'python'
    when title LIKE '%sql%' then 'sql'
    when title LIKE '%javascript%' then 'javascript'
    when title LIKE '%java%' then 'java'
    ELSE 'Other_Case'
end as language
FROM `jrjames83-1171.sampledata.top_questions`
)
SELECT language, COUNT(*)
FROM base_table
group by 1
order by 2 desc
;
```

6. -- For each of these categories, how often is the string within the tag occurs as well?

```
WITH base_table AS (
SELECT DISTINCT id, title, ARRAY_TO_STRING(ARRAY_AGG(DISTINCT tag)," ") as tag_content
FROM `jrjames83-1171.sampledata.top_questions`
GROUP BY 1,2
),
language_table as (
SELECT  id,
case when title LIKE '%python%' AND tag_content LIKE '%python%' then 'python_in_both'
    when title LIKE '%python%' AND tag_content NOT LIKE '%python%' then 'python_title_only'
    when title NOT LIKE '%python%' AND tag_content LIKE '%python%' then 'python_only_tag'
    when title LIKE '%sql%' AND tag_content LIKE '%sql%' then 'sql_in_both'
    when title LIKE '%sql%' AND tag_content NOT LIKE '%sql%' then 'sql_title_only'
    when title NOT LIKE '%sql%' AND tag_content LIKE '%sql%' then 'sql_only_tag'
    when title LIKE '%javascript%' AND tag_content LIKE '%javascript%' then 'javascript_in_both'
    when title LIKE '%javascript%' AND tag_content NOT LIKE '%javascript%' then 'javascript_title_only'
    when title NOT LIKE '%javascript%' AND tag_content LIKE '%javascript%' then 'javascript_only_tag'
  ELSE NULL
end as language
FROM base_table)
SELECT
COALESCE(language, "no match") as match_description, COUNT(*) as number_of_questions
 from language_table
GROUP BY 1
ORDER BY 1,2 DESC
;
```

7. /*Which language has seen the greatest increase in views YoY?*/

```
WITH BASE_TABLE AS (
SELECT case when title LIKE '%python%' THEN 'python'
        when title LIKE '%sql%' then 'sql'
        when title LIKE '%javascript%' then 'javascript'
        when title LIKE '%java%' then 'java'
     else 'other language' end as language,
     EXTRACT(year from quarter) as Year,
     SUM(quarter_views) as Total_Views
from `jrjames83-1171.sampledata.top_questions`
GROUP BY 1,2
ORDER BY 1,2,3 desc)
```

```
SELECT language,
       Year,
       Total_Views,
        LEAD(Total_Views, 1) OVER (
                        PARTITION by language
                        ORDER BY Year
            ) AS next_year_views,
ROUND(((LEAD(Total_Views, 1) OVER (
                        PARTITION by language
                        ORDER BY Year
            ) - Total_Views)/Total_Views) * 100,2) as YoY_Growth,
round((Total_Views/LAG(Total_Views) OVER (PARTITION BY language ORDER BY Year)-1)*100,2)  as
YoY_Growth2   // Correct One
FROM BASE_TABLE ;
```

```
-- Generating array of numbers
SELECT num from UNNEST(GENERATE_ARRAY(5,10)) as num;
```

```
-- Getting percentage of orders by order's status
SELECT order_status, count(*) /sum(count(*)) OVER() as pct_of_orders -- This will get the over all
orders
from `jrjames83-1171.sampledata.orders`
group by 1
;
```

```
SELECT DATE_TRUNC(DATE(order_purchase_timestamp), YEAR) as year,
       EXTRACT(month FROM order_purchase_timestamp) as month,
count(*) as number_orders
from `jrjames83-1171.sampledata.orders`
```

```
where order_status IN ('shipped','created')

group by 1,2

order by 1,2

;


-- What customer city is responsible for the most orders

SELECT   c.customer_city,

        count(distinct o.order_id) as number_orders

from `jrjames83-1171.sampledata.orders` o

join `jrjames83-1171.sampledata.customers` c

ON o.customer_id = c.customer_id

group by 1

order by 2 desc

;

-- Customer with more than 1 orders?

WITH base_table AS (

SELECT   c.customer_unique_id,

        o.order_purchase_timestamp,

        row_number() over

        (partition by c.customer_unique_id order by o.order_purchase_timestamp)

        customer_order_number

from `jrjames83-1171.sampledata.orders` o

join `jrjames83-1171.sampledata.customers` c

ON o.customer_id = c.customer_id

order by 3 desc ),

exclude_theme AS (

SELECT customer_unique_id, max(customer_order_number)

FROM base_table

GROUP BY 1
```

```
HAVING max(customer_order_number) = 1

)

SELECT * FROM

base_table where customer_unique_id NOT IN

(SELECT customer_unique_id FROM exclude_theme)

ORDER BY 1,3

;

-- We want to find order the instances of people purchasing a product by time. E.g. for some product,
when was the nth time it was purchased?

SELECT oi.product_id,

      o.order_purchase_timestamp,

      ROW_NUMBER() OVER(PARTITION BY oi.product_id

      ORDER BY o.order_purchase_timestamp

      ) as order_nth_occurence

from `jrjames83-1171.sampledata.order_items` oi

JOIN `jrjames83-1171.sampledata.orders` o

on o.order_id = oi.order_id;


-- Find the time between in-between orders of the product

WITH base_table AS (

SELECT oi.product_id,

      p.product_category_name,

      o.order_purchase_timestamp,

      ROW_NUMBER() OVER(PARTITION BY oi.product_id

      ORDER BY o.order_purchase_timestamp

      ) as order_nth_occurence,

  LAG(o.order_purchase_timestamp)

  OVER (PARTITION BY oi.product_id  ORDER BY o.order_purchase_timestamp) as
prev_order_timestamp
```

```sql
from `jrjames83-1171.sampledata.order_items` oi

JOIN `jrjames83-1171.sampledata.orders` o on o.order_id = oi.order_id

JOIN `jrjames83-1171.sampledata.products` p on oi.product_id = p.product_id

order by 1,3)

SELECT bt.*, DATE_DIFF(bt.order_purchase_timestamp, bt.prev_order_timestamp, day) as
days_between_order_purchase_date

 from base_table bt;


-- Find the avg days between orders of any product

WITH base_table AS (

SELECT oi.product_id,

       p.product_category_name,

       o.order_purchase_timestamp,

       ROW_NUMBER() OVER(PARTITION BY oi.product_id

       ORDER BY o.order_purchase_timestamp

       ) as order_nth_occurence,

  LAG(o.order_purchase_timestamp)

  OVER (PARTITION BY oi.product_id  ORDER BY o.order_purchase_timestamp) as
prev_order_timestamp

from `jrjames83-1171.sampledata.order_items` oi

JOIN `jrjames83-1171.sampledata.orders` o on o.order_id = oi.order_id

JOIN `jrjames83-1171.sampledata.products` p on oi.product_id = p.product_id

order by 1,3),

diffs_table AS (

SELECT bt.*, DATE_DIFF(bt.order_purchase_timestamp, bt.prev_order_timestamp, day) as
days_between_order_purchase_date

 from base_table bt)

 SELECT

 product_id,AVG(days_between_order_purchase_date) as mean_days_between,

 count(*) as times_ordered
```

```
  from diffs_table where days_between_order_purchase_date is not null
 group by 1 ;

WITH base_table AS (
SELECT   c.customer_unique_id,
        o.order_purchase_timestamp,
        row_number() over
        (partition by c.customer_unique_id order by o.order_purchase_timestamp)
        customer_order_number
from `jrjames83-1171.sampledata.orders` o
join `jrjames83-1171.sampledata.customers` c
ON o.customer_id = c.customer_id
order by 3 desc ),
exclude_theme AS (
SELECT customer_unique_id, max(customer_order_number)
FROM base_table
GROUP BY 1
HAVING max(customer_order_number) = 1
),
prev_date_table as (
SELECT bt.*,
LAG(order_purchase_timestamp) OVER (PARTITION BY customer_unique_id ORDER BY
order_purchase_timestamp) as prev_order
 FROM
base_table bt
where customer_unique_id NOT IN
(SELECT customer_unique_id FROM exclude_theme)
ORDER BY 1,3)
SELECT customer_order_number,
AVG(DATE_DIFF(order_purchase_timestamp, prev_order, DAY)) as mean_days_between_orders,

count(distinct customer_unique_id ) as count_unique_customers

from prev_date_table pd

group by 1

;


-- Revenue Trends by Hour

-- The Site updates its database at 2 AM and slows down for 20 mins, are we risking revenue?

SELECT EXTRACT(hour FROM o.order_purchase_timestamp) as hour, ROUND(sum(payment_value),2)
as sales,

ROUND(SUM(SUM(payment_value)) OVER(),2) as total_sales

FROM `jrjames83-1171.sampledata.orders` o

JOIN `jrjames83-1171.sampledata.order_payments` op

ON op.order_id = o.order_id

group by 1

order by 1;


-- Revenue Trends by Hour

-- The Site updates its database at 2 AM and slows down for 20 mins, are we risking revenue?

-- Revenue by time of day, morning, afternoon, evening,

WITH base_table as (

SELECT EXTRACT(hour FROM o.order_purchase_timestamp) as hour,

    ROUND(sum(payment_value),2) as sales


FROM `jrjames83-1171.sampledata.orders` o

JOIN `jrjames83-1171.sampledata.order_payments` op

ON op.order_id = o.order_id

group by 1

order by 1)
```

```
Select
CASE
    WHEN hour BETWEEN 0 and 5 or hour = 23 THEN 'overnight'
    WHEN hour BETWEEN 6 and 11 THEN 'morning'
    WHEN hour BETWEEN 12 and 16 THEN 'afternoon'
    WHEN hour BETWEEN 17 and 22 THEN 'evening'
    ELSE 'check my logic' END AS datpart,
    SUM(sales) as sales
FROM base_table bt
group by 1;


-- Tricks Using CASE statement
WITH base_table as (
SELECT EXTRACT(hour FROM o.order_purchase_timestamp) as hour,
      ROUND(sum(payment_value),2) as sales


FROM `jrjames83-1171.sampledata.orders` o
JOIN `jrjames83-1171.sampledata.order_payments` op
ON op.order_id = o.order_id
group by 1
order by 1)
-- conditional outputs using small case statements, creates columns
Select
SUM (CASE  WHEN hour BETWEEN 0 and 5 or hour = 23 THEN sales ELSE 0 END ) as overnight_sales,
SUM (CASE  WHEN hour BETWEEN 6 and 11 THEN sales ELSE 0 END) as morning_sales,
SUM (CASE  WHEN hour BETWEEN 12 and 16 THEN sales ELSE 0 END) as afternoon_sales,
SUM (CASE  WHEN hour BETWEEN 17 and 22 THEN sales ELSE 0 END) as evening_sales
FROM base_table bt;
```

```
-- Unnest, Running Totals & Correlated subquery
WITH base_table as (
SELECT dollars,index, SUM(dollars) OVER() as global_sum
from
UNNEST(GENERATE_ARRAY(1,5)) dollars WITH OFFSET AS index)
-- correlated subquery
SELECT bt.*,
(SELECT sum(bt2.dollars) from base_table bt2
where bt2.index <= bt.index
) as running_dollars_total
FROM base_table bt
;
```

```
-- Moving Average
with base_table as (
SELECT dollars, index
FROM UNNEST(GENERATE_ARRAY(1,10)) dollars
WITH OFFSET as index
)
SELECT *, AVG(dollars) OVER (ORDER BY index ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) as
Three_Day_MA
FROM base_table;
```

```
-- 1) Customers, Lifetime Value
-- [customerid, first order date, total revenue, 1st order revenue]
WITH Customer_first_order as (
SELECT * FROM (
SELECT customer_id,
    amount,
    payment_date ,
    ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY payment_date) as order_nth
FROM `jrjames83-1171.sampledata.payments`
order by 1)
WHERE order_nth = 1)
SELECT p.customer_id,
    c.amount as first_order_amount,
    min(p.payment_date) as first_payment,
    sum(p.amount) as total_revenue,
    c.amount/sum(p.amount) as first_as_pct_total_rev
FROM `jrjames83-1171.sampledata.payments` p
   INNER JOIN Customer_first_order c ON p.customer_id = c.customer_id
GROUP BY 1,2
;

-- 2) Customers, orders within the first 30, 60, 90 days of purchase

WITH Customer_first_order as (
SELECT * FROM (
SELECT customer_id,
    amount,
    payment_date ,
    ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY payment_date) as order_nth
FROM `jrjames83-1171.sampledata.payments`
order by 1)
WHERE order_nth = 1),
summary_so_far as (
SELECT p.customer_id,
    c.amount as first_order_amount,
    min(p.payment_date) as first_payment,
    sum(p.amount) as total_revenue,
    c.amount/sum(p.amount) as first_as_pct_total_rev
FROM `jrjames83-1171.sampledata.payments` p
   INNER JOIN Customer_first_order c ON p.customer_id = c.customer_id
GROUP BY 1,2)
Select sf.*,
(SELECT
sum(p2.amount)
from `jrjames83-1171.sampledata.payments`p2
where p2.customer_id = sf.customer_id
AND DATE(p2.payment_date) BETWEEN DATE(sf.first_payment) AND DATE(sf.first_payment) + 30
) as customer_TV_First_30days,
(SELECT
sum(p2.amount)
from `jrjames83-1171.sampledata.payments`p2
where p2.customer_id = sf.customer_id
AND DATE(p2.payment_date) BETWEEN DATE(sf.first_payment) AND DATE(sf.first_payment) + 60
) as customer_TV_First_60days
, DATE_ADD( DATE(sf.first_payment), INTERVAL 30 DAY) as thirty_days_from_first_order
from summary_so_far sf
;


-- Moving averages
-- [50 day, 200 day, is the 50>200 day?]
-- What are these columns?
-- Buy and hold vs trade on the x-over
--  AVG(s.Close) OVER(ORDER BY s.DATE RANGE BETWEEN 50 PRECEDING AND CURRENT ROW)
```

```
WITH base_table AS (

Select s.*, ROW_NUMBER() OVER(ORDER BY Date) as index

FROM `jrjames83-1171.sampledata.stock_prices` s

ORDER BY 1),

averages as (

SELECT bt.*,

AVG(bt.close) OVER(ORDER BY bt.index RANGE BETWEEN 49 PRECEDING AND CURRENT ROW) as
MA_50days,

AVG(bt.close) OVER(ORDER BY bt.index RANGE BETWEEN 199 PRECEDING AND CURRENT ROW) as
MA_200days

from base_table bt),

-- Filter rows where the signal is changing (buy to sell or sell to buy)

signals as (

SELECT a.*, IF(a.MA_50days > a.MA_200days, 'Buy', 'Sell') as signal

FROM averages a

order by 1,2)

SELECT * FROM (

Select s.*, LAG(s.signal) OVER(ORDER BY s.DATE) <> s.signal as changed_signal

from signals s

Order BY 1)

WHERE changed_signal

;


-- More complex JOINs and Partitioning

-- For each customer, their top 2 movie ratings by rental revenue

WITH base_table as (

SELECT p.payment_id , p.amount, p.customer_id,p.rental_id, i.film_id, f.title, f.rating

from `jrjames83-1171.sampledata.payments` p
```

```
INNER JOIN `jrjames83-1171.sampledata.rental` r ON r.rental_id = p.rental_id
INNER JOIN `jrjames83-1171.sampledata.inventory` i ON i.inventory_id = r.inventory_id
INNER JOIN `jrjames83-1171.sampledata.film` f ON i.film_id  = f.film_id
),
customer_rating_revenue as (
SELECT bt.customer_id,bt.rating, SUM(bt.amount) as rental_revenue
FROM base_table bt
group by 1,2
ORDER BY 1)
SELECT * FROM (
SELECT c.*,
ROW_NUMBER() OVER (PARTITION BY customer_id  ORDER by rental_revenue desc ) rn
from customer_rating_revenue c
ORDER BY 1)
WHERE rn = 1;

-- First, Last touch channels, nbr conversions
-- number of conversions by path length
-- pattern of nested data or semi-structred data is common

-- Logic
-- Which is the first touch, index = 0
-- Which is the last touch, index = nbr_in_path - 1
WITH base_table AS
(
-- Eliminating any paths consisting of 1 touch
SELECT split(path, '>') as path, conversions
FROM
`jrjames83-1171.sampledata.conversion_paths`
```

```
WHERE ARRAY_LENGTH(split(path, '>')) > 1
), path_row_table AS (
SELECT distinct lower(trim(path_row)) as row, index,
conversions,
array_length(path) as nbr_in_path
FROM base_table, UNNEST(path) as path_row with offset as index
), summary_table as (
SELECT *,
CASE WHEN index = 0 THEN 'first_touch'
    WHEN nbr_in_path - 1 = index THEN 'last_touch'
    ELSE 'middle_touch'
    END
    AS touch_type
FROM path_row_table )
SELECT row,touch_type, SUM(conversions) as conversions FROM summary_table
WHERE touch_type != 'middle_touch'
group by 1,2
order by 1,2;
```