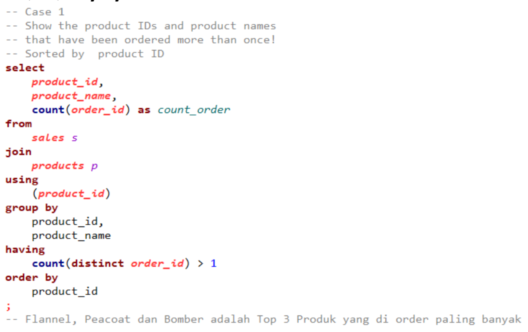
Exercise Week 4 (Mentoring 2)

SQL & Relational Database

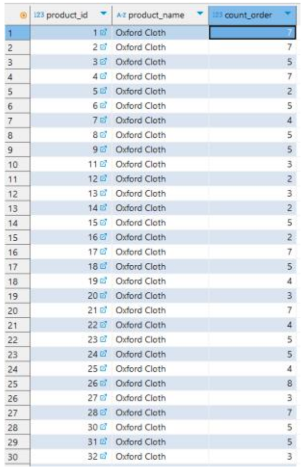
By : Oktavio Reza Putra

1. Show the product IDs and product names that have been ordered more than once! Sorted by product ID
   a. **SQL Query Syntax:**



```sql
-- Case 1
-- Show the product IDs and product names
-- that have been ordered more than once!
-- Sorted by  product ID
select
    product_id,
    product_name,
    count(order_id) as count_order
from
    sales s
join
    products p
using
    (product_id)
group by
    product_id,
    product_name
having
    count(distinct order_id) > 1
order by
    product_id
;
-- Flannel, Peacoat dan Bomber adalah Top 3 Produk yang di order paling banyak
```

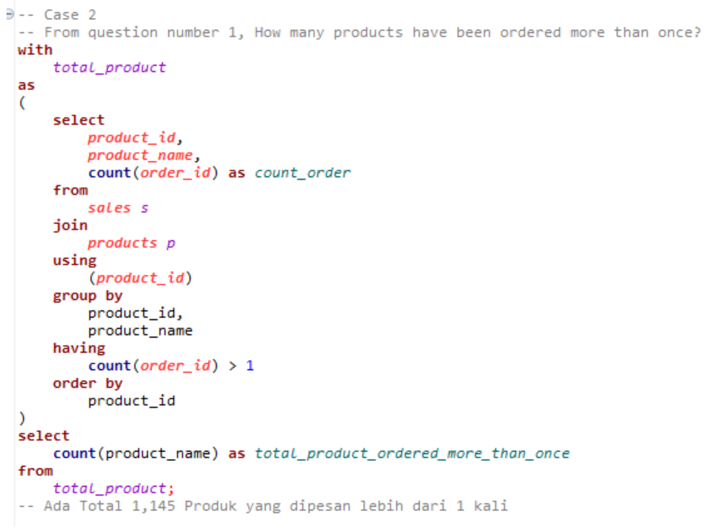
   b. **Query Result:**



	product_id	product_name	count_order
1	1	Oxford Cloth	7
2	2	Oxford Cloth	7
3	3	Oxford Cloth	5
4	4	Oxford Cloth	7
5	5	Oxford Cloth	2
6	6	Oxford Cloth	5
7	7	Oxford Cloth	4
8	8	Oxford Cloth	5
9	9	Oxford Cloth	5
10	11	Oxford Cloth	3
11	12	Oxford Cloth	2
12	13	Oxford Cloth	3
13	14	Oxford Cloth	2
14	15	Oxford Cloth	5
15	16	Oxford Cloth	2
16	17	Oxford Cloth	7
17	18	Oxford Cloth	5
18	19	Oxford Cloth	4
19	20	Oxford Cloth	3
20	21	Oxford Cloth	7
21	22	Oxford Cloth	4
22	23	Oxford Cloth	5
23	24	Oxford Cloth	5
24	25	Oxford Cloth	4
25	26	Oxford Cloth	8
26	27	Oxford Cloth	3
27	28	Oxford Cloth	7
28	30	Oxford Cloth	5
29	31	Oxford Cloth	5
30	32	Oxford Cloth	3

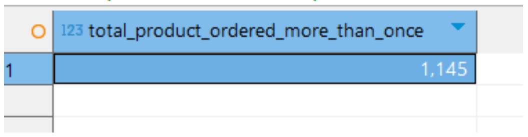
   c. **Description of Query Result:**
   Flannel, Peacoat and Bomber are the Top 3 most ordered products

2. From question number 1, How many products have been ordered more than once?
   a. **SQL Query Syntax:**



```sql
-- Case 2
-- From question number 1, How many products have been ordered more than once?
with
    total_product
as
(
    select
        product_id,
        product_name,
        count(order_id) as count_order
    from
        sales s
    join
        products p
    using
        (product_id)
    group by
        product_id,
        product_name
    having
        count(order_id) > 1
    order by
        product_id
)
select
    count(product_name) as total_product_ordered_more_than_once
from
    total_product;
-- Ada Total 1,145 Produk yang dipesan lebih dari 1 kali
```

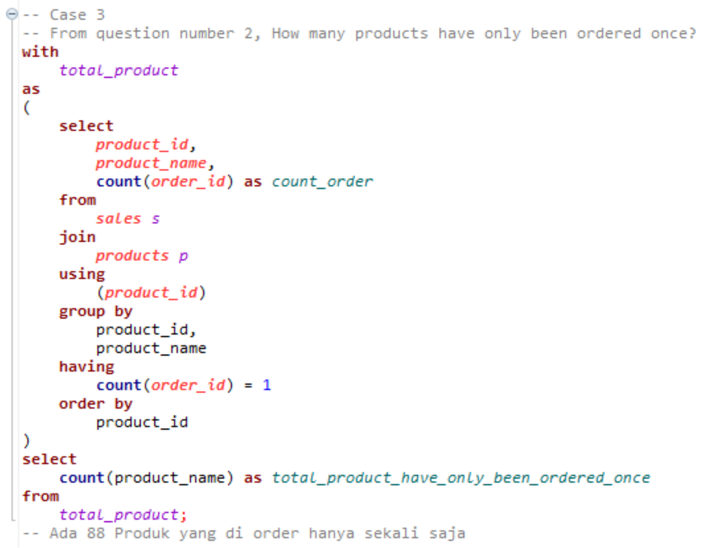
b. **Query Result:**



```
  123 total_product_ordered_more_than_once
1                                    1,145
```

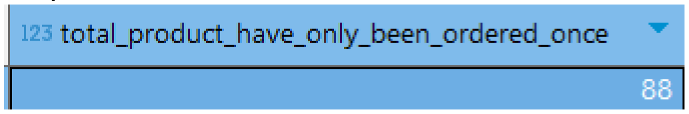
c. **Description of Query Result:**
Total 1,145 Products ordered more than once

3. From question number 2, How many products have only been ordered once?
   a. **SQL Query Syntax:**



```
-- Case 3
-- From question number 2, How many products have only been ordered once?
with
    total_product
as
(
    select
        product_id,
        product_name,
        count(order_id) as count_order
    from
        sales s
    join
        products p
    using
        (product_id)
    group by
        product_id,
        product_name
    having
        count(order_id) = 1
    order by
        product_id
)
select
    count(product_name) as total_product_have_only_been_ordered_once
from
    total_product;
-- Ada 88 Produk yang di order hanya sekali saja
```

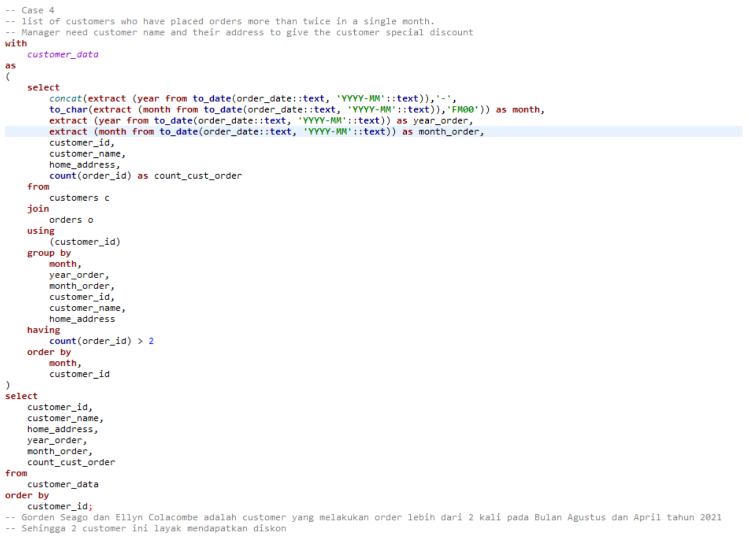
b. **Query Result:**



```
123 total_product_have_only_been_ordered_once
                                          88
```

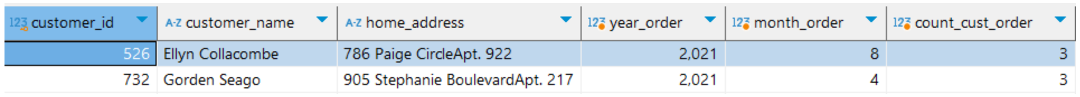
c. **Description of Query Result:**
There are 88 products that are ordered only once

4. List of customers who have placed orders more than twice in a single month. Manager need customer name and their address to give the customer special discount.
   a. **SQL Query Syntax:**



```
-- Case 4
-- list of customers who have placed orders more than twice in a single month.
-- Manager need customer name and their address to give the customer special discount
with
    customer_data
as
(
    select
        concat(extract (year from to_date(order_date::text, 'YYYY-MM'::text)),'-',
        to_char(extract (month from to_date(order_date::text, 'YYYY-MM'::text)),'FM00')) as month,
        extract (year from to_date(order_date::text, 'YYYY-MM'::text)) as year_order,
        extract (month from to_date(order_date::text, 'YYYY-MM'::text)) as month_order,
        customer_id,
        customer_name,
        home_address,
        count(order_id) as count_cust_order
    from
        customers c
    join
        orders o
    using
        (customer_id)
    group by
        month,
        year_order,
        month_order,
        customer_id,
        customer_name,
        home_address
    having
        count(order_id) > 2
    order by
        month,
        customer_id
)
select
    customer_id,
    customer_name,
    home_address,
    year_order,
    month_order,
    count_cust_order
from
    customer_data
order by
    customer_id;
-- Gorden Seago dan Ellyn Colacombe adalah customer yang melakukan order lebih dari 2 kali pada Bulan Agustus dan April tahun 2021
-- Sehingga 2 customer ini layak mendapatkan diskon
```

b. Query Result:



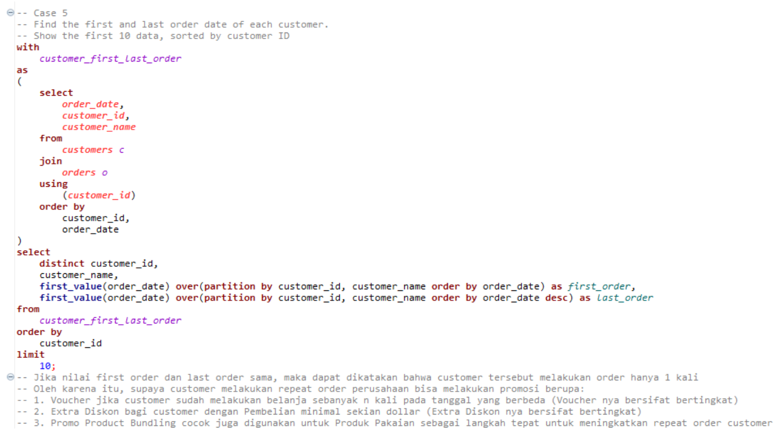
customer_id	customer_name	home_address	year_order	month_order	count_cust_order
526	Ellyn Collacombe	786 Paige CircleApt. 922	2,021	8	3
732	Gorden Seago	905 Stephanie BoulevardApt. 217	2,021	4	3

c. Description of Query Result:

Seago Curtains and Ellyn Colacombe are customers who placed orders more than 2 times in August and April 2021. Therefore, these 2 customers are eligible for a discount.

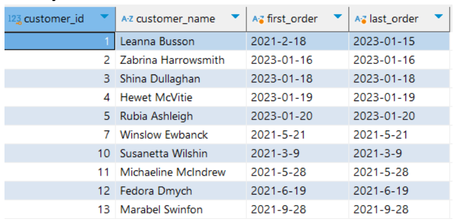
5. Find the first and last order date of each customer. Show the first 10 data, sorted by customer ID

a. SQL Query Syntax:



```sql
-- Case 5
-- Find the first and last order date of each customer.
-- Show the first 10 data, sorted by customer ID
with
    customer_first_last_order
as
(
    select
        order_date,
        customer_id,
        customer_name
    from
        customers c
    join
        orders o
    using
        (customer_id)
    order by
        customer_id,
        order_date
)
select
    distinct customer_id,
    customer_name,
    first_value(order_date) over(partition by customer_id, customer_name order by order_date) as first_order,
    first_value(order_date) over(partition by customer_id, customer_name order by order_date desc) as last_order
from
    customer_first_last_order
order by
    customer_id
limit
    10;
-- Jika nilai first order dan last order sama, maka dapat dikatakan bahwa customer tersebut melakukan order hanya 1 kali
-- Oleh karena itu, supaya customer melakukan repeat order perusahaan bisa melakukan promosi berupa:
-- 1. Voucher jika customer sudah melakukan belanja sebanyak n kali pada tanggal yang berbeda (Voucher nya bersifat bertingkat)
-- 2. Extra Diskon bagi customer dengan Pembelian minimal sekian dollar (Extra Diskon nya bersifat bertingkat)
-- 3. Promo Product Bundling cocok juga digunakan untuk Produk Pakaian sebagai langkah tepat untuk meningkatkan repeat order customer
```

b. Query Result:



customer_id	customer_name	first_order	last_order
1	Leanna Busson	2021-2-18	2023-01-15
2	Zabrina Harrowsmith	2023-01-16	2023-01-16
3	Shina Dullaghan	2023-01-18	2023-01-18
4	Hewet McVitie	2023-01-19	2023-01-19
5	Rubia Ashleigh	2023-01-20	2023-01-20
7	Winslow Ewbanck	2021-5-21	2021-5-21
10	Susanetta Wilshin	2021-3-9	2021-3-9
11	Michaeline McIndrew	2021-5-28	2021-5-28
12	Fedora Dmych	2021-6-19	2021-6-19
13	Marabel Swinfon	2021-9-28	2021-9-28

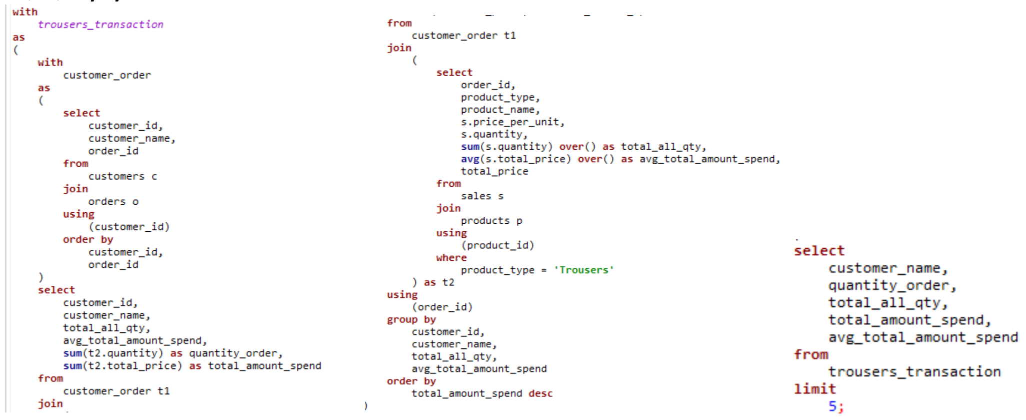
c. Description of Query Result:

If the first order and last order values are the same, it can be said that the customer placed an order only once. Therefore, so that customers make repeat orders, companies can make promotions in the form of:

- Vouchers if the customer has shopped n times on different dates (the vouchers are tiered)
- Extra Discount for customers with a minimum purchase of a certain dollar (Extra Discount is tiered)
- Product Bundling Promo is also suitable for Clothing Products as the right step to increase customer repeat orders.

6. Retrieve the top 5 customers who have spent the most amount of money on products within the "Trousers" category, including the customer's name, the quantity and total amount spent in this category. Additionally, find the total number of products sold in this category and calculate the average total price spent in this category, compare with the top 5 customers who have spent the most amount of money on products within the "Trousers" category . Finally, sort the results by the total amount spent in descending order.

a. SQL Query Syntax:

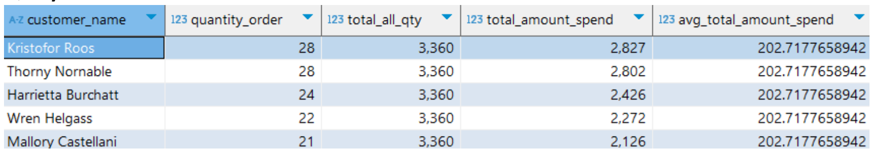


```sql
with
    trousers_transaction
as
(
    with
        customer_order
    as
    (
        select
            customer_id,
            customer_name,
            order_id
        from
            customers c
        join
            orders o
        using
            (customer_id)
        order by
            customer_id,
            order_id
    )
    select
        customer_id,
        customer_name,
        total_all_qty,
        avg_total_amount_spend,
        sum(t2.quantity) as quantity_order,
        sum(t2.total_price) as total_amount_spend
    from
        customer_order t1
    join
```

```sql
    from
        customer_order t1
    join
        (
            select
                order_id,
                product_type,
                product_name,
                s.price_per_unit,
                s.quantity,
                sum(s.quantity) over() as total_all_qty,
                avg(s.total_price) over() as avg_total_amount_spend,
                total_price
            from
                sales s
            join
                products p
            using
                (product_id)
            where
                product_type = 'Trousers'
        ) as t2
    using
        (order_id)
    group by
        customer_id,
        customer_name,
        total_all_qty,
        avg_total_amount_spend
    order by
        total_amount_spend desc
)
```

```sql
select
    customer_name,
    quantity_order,
    total_all_qty,
    total_amount_spend,
    avg_total_amount_spend
from
    trousers_transaction
limit
    5;
```

b. **Query Result:**



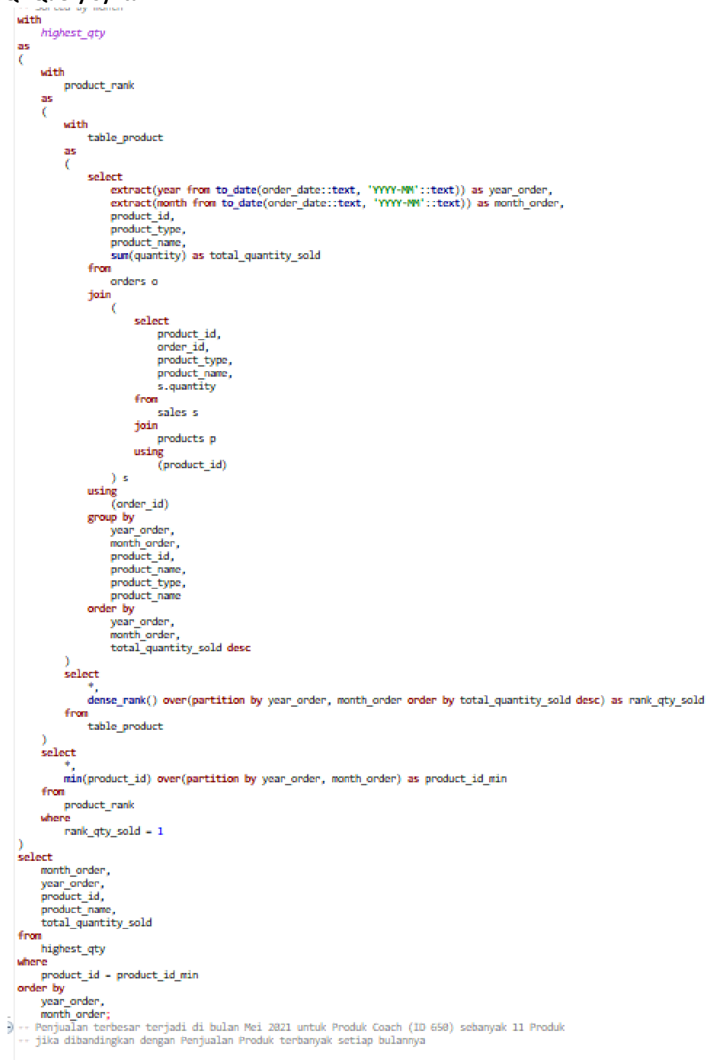
customer_name	quantity_order	total_all_qty	total_amount_spend	avg_total_amount_spend
Kristofor Roos	28	3,360	2,827	202.7177658942
Thorny Nornable	28	3,360	2,802	202.7177658942
Harrietta Burchatt	24	3,360	2,426	202.7177658942
Wren Helgass	22	3,360	2,272	202.7177658942
Mallory Castellani	21	3,360	2,126	202.7177658942

c. **Description of Query Result:**

The average Total Spending Customer for Trousers is 202.72 and the Top 5 Spender Customers for Trousers are above this average. In addition, the total Order Quantity of these Top 5 Customers contributed 3.66% of the overall sales (3,360).

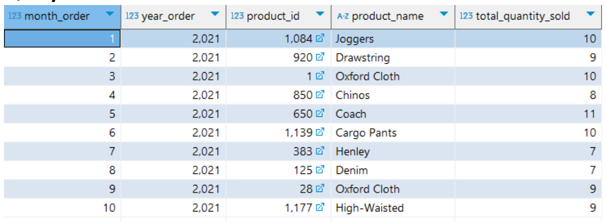
7. Find the top-selling (Top 1) product for each month. You want to know the product with the highest **total quantity sold** in each month. If there are products that have the same total quantity sold, choose the smallest product ID. Return the product name, the corresponding month, and the total quantity sold for each month's top-selling product. Sorted by month!

a. **SQL Query Syntax:**



```
with
    highest_qty
as
(
    with
        product_rank
    as
    (
        with
            table_product
        as
        (
            select
                extract(year from to_date(order_date::text, 'YYYY-MM'::text)) as year_order,
                extract(month from to_date(order_date::text, 'YYYY-MM'::text)) as month_order,
                product_id,
                product_type,
                product_name,
                sum(quantity) as total_quantity_sold
            from
                orders o
            join
                (
                    select
                        product_id,
                        order_id,
                        product_type,
                        product_name,
                        s.quantity
                    from
                        sales s
                    join
                        products p
                    using
                        (product_id)
                ) s
            using
                (order_id)
            group by
                year_order,
                month_order,
                product_id,
                product_name,
                product_type,
                product_name
            order by
                year_order,
                month_order,
                total_quantity_sold desc
        )
        select
            *,
            dense_rank() over(partition by year_order, month_order order by total_quantity_sold desc) as rank_qty_sold
        from
            table_product
    )
    select
        *,
        min(product_id) over(partition by year_order, month_order) as product_id_min
    from
        product_rank
    where
        rank_qty_sold = 1
)
select
    month_order,
    year_order,
    product_id,
    product_name,
    total_quantity_sold
from
    highest_qty
where
    product_id = product_id_min
order by
    year_order,
    month_order;
-- Penjualan terbesar terjadi di bulan Mei 2021 untuk Produk Coach (ID 650) sebanyak 11 Produk
-- jika dibandingkan dengan Penjualan Produk terbanyak setiap bulannya
```

b. **Query Result:**



month_order	year_order	product_id	product_name	total_quantity_sold
1	2,021	1,084	Joggers	10
2	2,021	920	Drawstring	9
3	2,021	1	Oxford Cloth	10
4	2,021	850	Chinos	8
5	2,021	650	Coach	11
6	2,021	1,139	Cargo Pants	10
7	2,021	383	Henley	7
8	2,021	125	Denim	7
9	2,021	28	Oxford Cloth	9
10	2,021	1,177	High-Waisted	9

c. Description of Query Result:

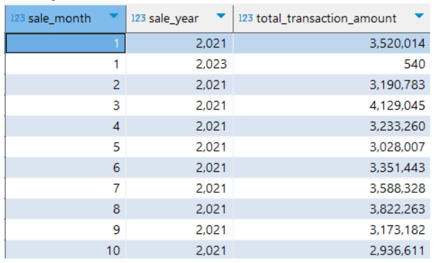
The largest sales occurred in May 2021 for Coach Products (ID 650) as many as 11 Products when compared to the largest Product Sales each month

8. Create a view to store a query for calculating monthly total payment

a. SQL Query Syntax:

```
-- Case 8
-- Create a view to store a query for calculating monthly total payment.
create view
    monthly_total_payment
 as
 select
     extract(month from to_date((o.order_date)::text, 'yyyy-mm'::text)) as sale_month,
     extract(year from to_date((o.order_date)::text, 'yyyy-mm'::text)) as sale_year,
     sum(o.payment) as total_transaction_amount
 from
    orders o
 group by
    (extract(month from to_date((o.order_date)::text, 'yyyy-mm'::text))),
    (extract(year from to_date((o.order_date)::text, 'yyyy-mm'::text)))
 order by
    (extract(month from to_date((o.order_date)::text, 'yyyy-mm'::text))),
    (extract(year from to_date((o.order_date)::text, 'yyyy-mm'::text)));
```

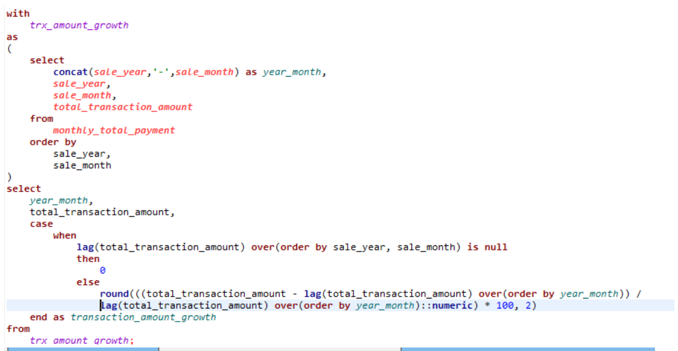
b. Query Result:



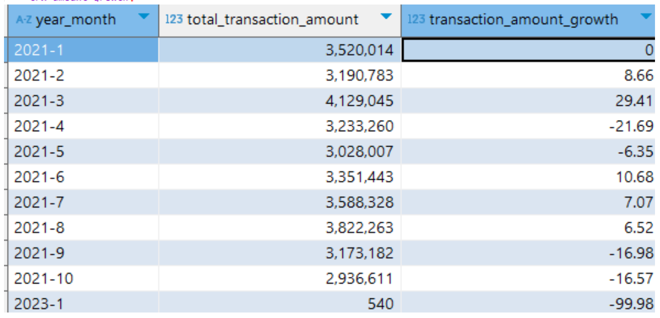
sale_month	sale_year	total_transaction_amount
1	2,021	3,520,014
1	2,023	540
2	2,021	3,190,783
3	2,021	4,129,045
4	2,021	3,233,260
5	2,021	3,028,007
6	2,021	3,351,443
7	2,021	3,588,328
8	2,021	3,822,263
9	2,021	3,173,182
10	2,021	2,936,611

c. Description of Query Result:

- The highest sales occurred in March 2021 with a nominal value of 4,129,045.
- The highest growth in the number of transactions occurred from February to March with a growth of 29.41%, but from February to March experienced a decrease in growth of -21.69%.



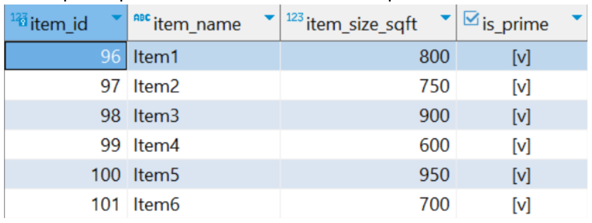
```
with
    trx_amount_growth
as
(
    select
        concat(sale_year,'-',sale_month) as year_month,
        sale_year,
        sale_month,
        total_transaction_amount
    from
        monthly_total_payment
    order by
        sale_year,
        sale_month
)
select
    year_month,
    total_transaction_amount,
    case
        when
            lag(total_transaction_amount) over(order by sale_year, sale_month) is null
            then
                0
            else
                round(((total_transaction_amount - lag(total_transaction_amount) over(order by year_month)) /
                lag(total_transaction_amount) over(order by year_month)::numeric) * 100, 2)
    end as transaction_amount_growth
from
    trx_amount_growth;
```



year_month	total_transaction_amount	transaction_amount_growth
2021-1	3,520,014	0
2021-2	3,190,783	8.66
2021-3	4,129,045	29.41
2021-4	3,233,260	-21.69
2021-5	3,028,007	-6.35
2021-6	3,351,443	10.68
2021-7	3,588,328	7.07
2021-8	3,822,263	6.52
2021-9	3,173,182	-16.98
2021-10	2,936,611	-16.57
2023-1	540	-99.98

- Positive growth in the Nominal Amount of Transactions occurred for 3 consecutive months from July to September, which also tended to be stable (no significant increase in nominal transaction growth).
- The behavior of Transaction Amounts during 2021 on a MoM basis is that in the first 2 months there is positive growth then the next 2 months experience a decline, then experience positive growth again 2-3 months later. There is a need for more comprehensive and in-depth research and analysis to find out the causes and consequences of this pattern of transaction amount behavior.

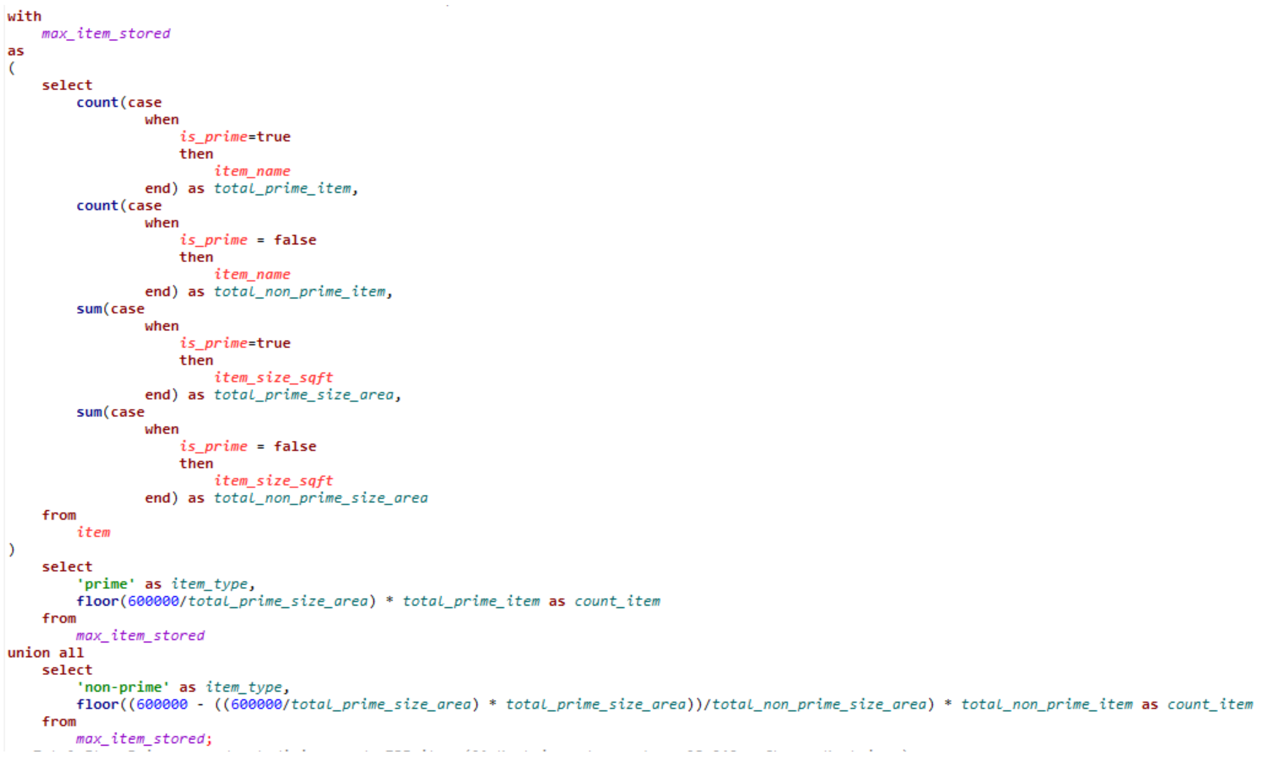
The Item table is a database table that stores information about items to be stored in the warehouse. It includes details such as item_id, the size of the item (item_size), and whether the item is a prime product or not. Here's an example of the data:



item_id	item_name	item_size_sqft	is_prime
96	Item1	800	[v]
97	Item2	750	[v]
98	Item3	900	[v]
99	Item4	600	[v]
100	Item5	950	[v]
101	Item6	700	[v]

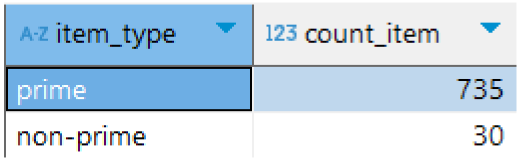
9. As a warehouse manager responsible for stock management in your company's warehouse, you oversee a warehouse with a total area of 600,000 sqft. There are two types of items: prime items and non-prime items. These items come in various sizes, with priority given to prime items. Your task is to determine the maximum number of prime and non-prime items that can be stored in the warehouse
   - Prime and non-prime items are stored in their respective containers. For example, In the database, there are 15 non-prime items and 35 prime items. Each prime container must contain 35 prime items, and each non-prime container must contain 15 non-prime items
   - Non-prime items must always be available in stock to meet customer demand, so the non-prime item count should never be zero.

   a. **SQL Query Syntax:**



```
with
    max_item_stored
as
(
    select
        count(case
                when
                    is_prime=true
                then
                    item_name
                end) as total_prime_item,
        count(case
                when
                    is_prime = false
                then
                    item_name
                end) as total_non_prime_item,
        sum(case
                when
                    is_prime=true
                then
                    item_size_sqft
                end) as total_prime_size_area,
        sum(case
                when
                    is_prime = false
                then
                    item_size_sqft
                end) as total_non_prime_size_area
    from
        item
)
    select
        'prime' as item_type,
        floor(600000/total_prime_size_area) * total_prime_item as count_item
    from
        max_item_stored
union all
    select
        'non-prime' as item_type,
        floor((600000 - ((600000/total_prime_size_area) * total_prime_size_area))/total_non_prime_size_area) * total_non_prime_item as count_item
    from
        max_item_stored;
```

   b. **Query Result:**

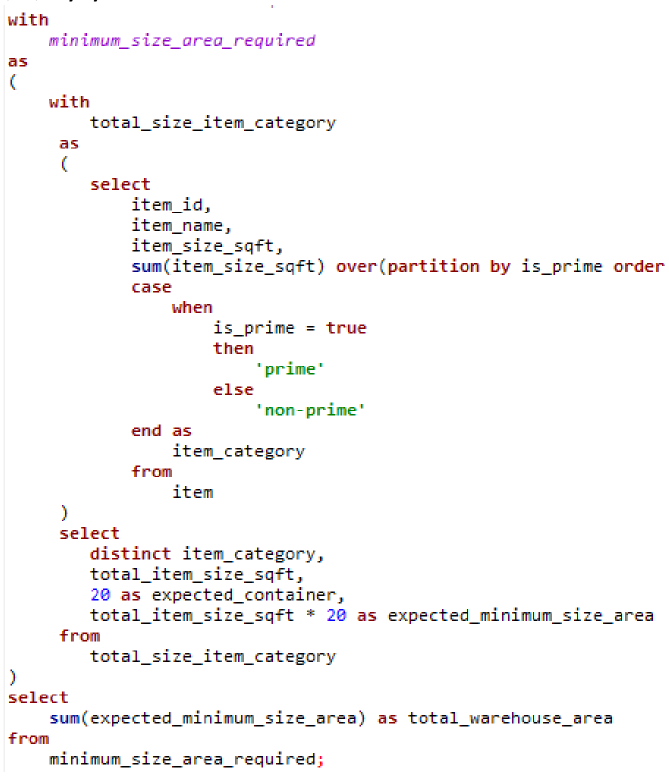


item_type	count_item
prime	735
non-prime	30

   c. **Description of Query Result:**
      - Total Prime Items that can be stored are 735 items (21 Containers at 28,040 sqft per Container)
      - Total Non-Prime Items that can be stored are 30 items (2 Containers with an Area of 5,500 sqft per Container)
      - Thus, to store the entire minimum Item Count of Prime and Non-Prime Items requires 23 Containers for a Warehouse Area of 600,000 sqft (588,840 sqft for Prime Items and 11,160 sqft for Non-Prime Items).

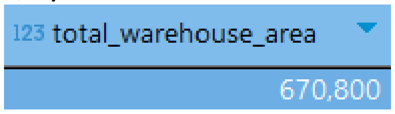
10. The warehouse manager is planning to find a new warehouse to store their products. The warehouse is expected to accommodate 20 containers for each prime and non-prime item. What is the minimum required size for the warehouse?

a. **SQL Query Syntax:**



```
with
    minimum_size_area_required
as
(
    with
        total_size_item_category
     as
     (
        select
            item_id,
            item_name,
            item_size_sqft,
            sum(item_size_sqft) over(partition by is_prime order by is_prime desc) as total_item_size_sqft,
            case
                when
                    is_prime = true
                    then
                        'prime'
                    else
                        'non-prime'
            end as
                item_category
            from
                item
     )
     select
        distinct item_category,
        total_item_size_sqft,
        20 as expected_container,
        total_item_size_sqft * 20 as expected_minimum_size_area
     from
        total_size_item_category
)
select
    sum(expected_minimum_size_area) as total_warehouse_area
from
    minimum_size_area_required;
```

b. **Query Result:**



123 total_warehouse_area
670,800

c. **Description of Query Result:**

Minimum Warehouse Area of approximately 670,800 sqft to store prime and non-prime items with 20 containers for Prime Items (with a Total Area of 28,040 sqft per container) with a Minimum Warehouse Area of 560,800 sqft and 20 containers for Non-Prime Items (with a Total Area of 5,550 sqft per container) with a Minimum Warehouse Area of 110,000 sqft.