ANALYZING ECOMMERCE BUSINESS PERFORMANCE USING SQL

WORKING PROCEDURE AND SQL QUERY

Personal project of Putrini Nur Amalina H.

A. DATA PREPROCESSING

1. Create database

```
CREATE  DATABASE ecommerce_performance;
```

2. Create table

Create table for 9 csv datasets and set each column with its data type.

```
CREATE TABLE customers (
   customer_id varchar,
   customer_unique_id varchar,
   customer_zip_code_prefix int,
   customer_city varchar,
   customer_state varchar);

CREATE TABLE geolocation (
   geolocation_zip_code_prefix varchar,
   geolocation_lat double precision,
   geolocation_lng double precision,
   geolocation_city varchar,
   geolocation_state varchar);

CREATE TABLE order_items (
   order_id varchar,
   order_item_id smallint,
   product_id varchar,
   seller_id varchar,
   shipping_limit_date timestamp,
   price real,
   freight_value real);

CREATE TABLE order_payments (
   order_id varchar,
   payment_sequential smallint,
   payment_type varchar,
   payment_installments varchar,
   payment_value real);

CREATE TABLE order_reviews (
   review_id varchar,
   order_id varchar,
   review_score smallint,
   review_comment_title varchar,
   review_comment_message varchar,
   review_creation_date timestamp,
   review_answer_time timestamp);
```

```
CREATE TABLE orders (
   order_id varchar,
   customer_id varchar,
   order_status varchar,
   order_purchase_timestamp timestamp,
   order_approved_at timestamp,
   order_delivered_carrier_date timestamp,
   order_delivered_customer_date timestamp,
   order_estimated_delivery_date timestamp);

CREATE TABLE product (
   indeks int,
   product_id varchar,
   product_category_name varchar,
   product_name_length real,
   product_description_length real,
   product_photos_qty real,
   product_weight_g real,
   product_length_cm real,
   product_height_cm real,
   product_width_cm real);

CREATE TABLE sellers (
   seller_id varchar,
   seller_zip_code_prefix varchar,
   seller_city varchar,
   seller_state varchar);
```

3. Import dataset

Import csv dataset to each table using COPY statement.

```
COPY customers (
   customer_id,
   customer_unique_id,
   customer_zip_code_prefix,
   customer_city,
   customer_state)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\customers_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY geolocation (
   geolocation_zip_code_prefix,
   geolocation_lat,
   geolocation_lng,
   geolocation_city,
   geolocation_state)
```

```
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\geolocation_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY order_items (
   order_id,
   order_item_id,
   product_id,
   seller_id,
   shipping_limit_date,
   price,
   freight_value)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\order_items_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY order_payments (
   order_id,
   payment_sequential,
   payment_type,
   payment_installments,
   payment_value)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\order_payments_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY order_reviews (
   review_id,
   order_id,
   review_score,
   review_comment_title,
   review_comment_message,
   review_creation_date,
   review_answer_time)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\order_reviews_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY orders (
   order_id,
   customer_id,
   order_status,
   order_purchase_timestamp,
```

```
   order_approved_at,
   order_delivered_carrier_date,
   order_delivered_customer_date,
   order_estimated_delivery_date)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\orders_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY product (
   indeks,
   product_id,
   product_category_name,
   product_name_length,
   product_description_length,
   product_photos_qty,
   product_weight_g,
   product_length_cm,
   product_height_cm,
   product_width_cm)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\product_dataset.csv'
DELIMITER ','
CSV HEADER;

COPY sellers (
   seller_id,
   seller_zip_code_prefix,
   seller_city,
   seller_state)
FROM 'C:\Users\HP\Documents\DATA SCIENCE\RAKAMIN - Data Science
Bootcamp\Portofolio making\#1 Analyzing Ecommerce Business
Performance with SQL\Dataset\sellers_dataset.csv'
DELIMITER ','
CSV HEADER;
```

4. **Generate ERD diagram**

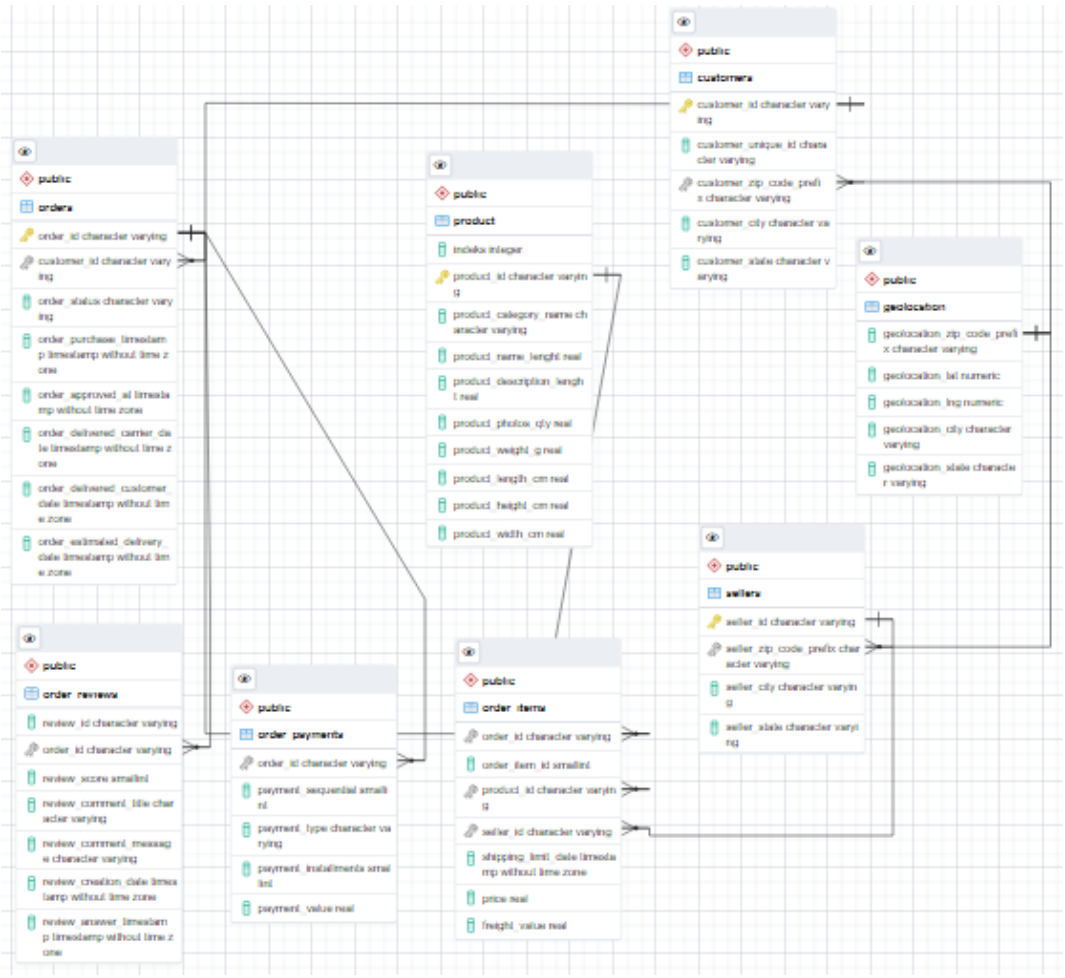
   First, we have to determine the primary key and foreign key for each table to define the relationship of each table using ALTER TABLE statement.

```
ALTER TABLE product ADD PRIMARY KEY (product_id);
ALTER TABLE sellers ADD PRIMARY KEY (seller_id);
ALTER TABLE orders ADD PRIMARY KEY (order_id);
ALTER TABLE customers ADD PRIMARY KEY (customer_id);

ALTER TABLE order_items ADD FOREIGN KEY (product_id) references
product(product_id);
ALTER TABLE order_items ADD FOREIGN KEY (order_id) references
orders(order_id);
```

```
ALTER TABLE order_items ADD FOREIGN KEY (seller_id) references
sellers(seller_id);
ALTER TABLE orders ADD FOREIGN KEY (customer_id) references
customers(customer_id);
ALTER TABLE order_reviews ADD FOREIGN KEY (order_id) references
orders(order_id);
ALTER TABLE order_payments ADD FOREIGN KEY (order_id) references
orders(order_id);
```

Then, generate the ERD diagram as shown below :



B. ANNUAL CUSTOMER ACTIVITY GROWTH ANALYSIS

1. Average monthly active customer per year

Create table to show the number of customers for each month per year.

```
WITH master as (
SELECT
   date_part ('year', o.order_purchase_timestamp) as year,
   date_part ('month', o.order_purchase_timestamp) as month,
   count (distinct c.customer_unique_id) as jumlah
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
```

```
GROUP BY 1,2
ORDER BY 1,2)
```

Then do agregation function to find the average of monthly customers per year.

```
SELECT
   year,
   round (avg (jumlah),2) as avg_mau
FROM master
GROUP BY 1
ORDER BY 1;
```

	year double precision	avg_mau numeric
1	2016	108.67
2	2017	3694.83
3	2018	5338.20

2. **New customers per year**
   Create table to show the customers id and the first purchase date for each customer.

```
WITH master as (
SELECT
   c.customer_unique_id as customer_id,
   min(o.order_purchase_timestamp) as first_purchase
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1)
```

Do agregation to find the average of new customers per year.

```
SELECT
   date_part('year',first_purchase) as year,
   count(customer_id)
FROM master
GROUP BY 1
ORDER BY 1;
```

	year double precision	count bigint
1	2016	326
2	2017	43708
3	2018	52062

3. **Returning customer per year**
   Create table to show the number of orders for each customer per year. Use WHERE statement to filter the number of order for more than 1 to define returning customer.

```
WITH master as (
SELECT
```

```
   date_part('year',o.order_purchase_timestamp) as year,
   c.customer_unique_id as customer,
   count(order_id)
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1,2
HAVING count(order_id) > 1
ORDER BY 1)
```

Use aggregate function to show the number of customers per year

```
SELECT
   year,
   count(customer) as returning_customer
FROM master
GROUP BY 1
ORDER BY 1;
```

	year double precision	returning_customer bigint
1	2016	3
2	2017	1256
3	2018	1167

4. **Average order per customer per year**

Create table to show the number of orders for each customer per year.

```
WITH master as (
SELECT
   date_part('year',o.order_purchase_timestamp) as year,
   c.customer_unique_id as customer_id,
   count(order_id) as jumlah_order
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1,2
ORDER BY 1)
```

Use aggregate function to show the average order per customer per year.

```
SELECT
   year,
   round(avg(jumlah_order),3) as avg_order_percustomer
FROM master
GROUP BY 1
ORDER BY 1;
```

	year double precision	avg_order_percustomer numeric
1	2016	1.009
2	2017	1.032
3	2018	1.024

5. **Compile the tables using CTE function**

```
WITH satu as (
WITH master as (
SELECT
   date_part ('year', o.order_purchase_timestamp) as year,
   date_part ('month', o.order_purchase_timestamp) as month,
   count (distinct c.customer_unique_id) as jumlah
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1,2
ORDER BY 1,2)
SELECT
   year,
   round (avg (jumlah),2) as avg_mau
FROM master
GROUP BY 1
ORDER BY 1),

dua as (
WITH master as (
SELECT
   c.customer_unique_id as customer_id,
   min(o.order_purchase_timestamp) as first_purchase
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1)
SELECT
   date_part('year',first_purchase) as year,
   count(customer_id) as new_customer
FROM master
GROUP BY 1
ORDER BY 1),

tiga as (
WITH master as (
SELECT
   date_part('year',o.order_purchase_timestamp) as year,
   c.customer_unique_id as customer,
   count(order_id)
FROM orders as o
```

```
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1,2
HAVING count(order_id) > 1
ORDER BY 1)
SELECT
   year,
   count(customer) as returning_customer
FROM master
GROUP BY 1
ORDER BY 1),

empat as (
WITH master as (
SELECT
   date_part('year',o.order_purchase_timestamp) as year,
   c.customer_unique_id as customer_id,
   count(order_id) as jumlah_order
FROM orders as o
JOIN customers as c
   on o.customer_id = c.customer_id
GROUP BY 1,2
ORDER BY 1)
SELECT
   year,
   round(avg(jumlah_order),3) as avg_order
FROM master
GROUP BY 1
ORDER BY 1)

SELECT
   satu.year,
   satu.avg_mau,
   dua.new_customer,
   tiga.returning_customer,
   empat.avg_order
FROM satu
JOIN dua
   on satu.year=dua.year
JOIN tiga
   on dua.year=tiga.year
JOIN empat
   on tiga.year=empat.year
GROUP BY 1,2,3,4,5
ORDER BY 1;
```

	year double precision	avg_mau numeric	new_customer bigint	returning_customer bigint	avg_order numeric
1	2016	108.67	326	3	1.009
2	2017	3694.83	43708	1256	1.032
3	2018	5338.20	52062	1167	1.024

C. ANNUAL PRODUCT CATEGORY QUALITY ANALYSIS

1. Total revenue per year

Create table to show the order_id and revenue for each order. Revenue is defined by adding the price and freight value.

```
CREATE TABLE total_revenue_per_year as
WITH master as (
SELECT
   order_id,
   sum(price + freight_value) as revenue
FROM order_items
GROUP BY 1)
```

Use aggregate function to show the number of revenue for each year. Make sure to filter the order status to ‘delivered’.

```
SELECT
   date_part ('year',orders.order_purchase_timestamp) as year,
   sum(revenue) as revenue_per_year
FROM master
JOIN orders
   on master.order_id = orders.order_id
WHERE orders.order_status = 'delivered'
GROUP BY 1
ORDER BY 1;
```

	year double precision	revenue_per_year double precision
1	2016	46682
2	2017	6924371
3	2018	8452044

2. Total cancel order per year

Create table with aggregate function to show the number of orders for each year. Filter the table where the order_status is ‘cancelled’.

```
CREATE TABLE total_cancel_per_year as
SELECT
   date_part('year',order_purchase_timestamp) as year,
   count(order_id)as total_cancel_order
FROM orders
```

```
WHERE order_status = 'canceled'
GROUP BY 1
ORDER BY 1;
```

	year double precision	total_cancel_order bigint
1	2016	26
2	2017	265
3	2018	334

3. **Highest revenue product per year**
   Create table to show the amount of revenue for each product by year. Create rank_revenue column to show the rank of each amount and sort the value from the largest amount using RANK and ORDER BY statement.

```
CREATE TABLE highest_revenue_product as (
WITH master as(
SELECT
   date_part('year',order_purchase_timestamp) as year,
   p.product_category_name,
   sum(oi.price+oi.freight_value) as revenue,
   rank () over (partition by date_part ('year',
   order_purchase_timestamp) order by sum (oi.price +
   oi.freight_value) desc) as rank_revenue
FROM product as p
JOIN order_items as oi
   on p.product_id = oi.product_id
JOIN orders as o
   on oi.order_id = o.order_id
GROUP BY 1,2)
```

Create table to show the product category for each year where the rank is 1.

```
SELECT
   year,
   product_category_name as highest_revenue_product,
   revenue
FROM master
WHERE rank_revenue =1
```

	year double precision	highest_revenue_product character varying	revenue double precision
1	2016	furniture_decor	7190
2	2017	bed_bath_table	590673
3	2018	health_beauty	885183

4. **The most cancelled product per year**
   Create table to show the number of order for each product category per year. Create rank_canceled_order column to show the rank of each order amount and sort the value from the largest amount using RANK and ORDER BY statement.

```
CREATE TABLE most_canceled_product as
WITH master as(
SELECT
   date_part('year',order_purchase_timestamp) as year,
   p.product_category_name,
   count (o.order_id),
   rank () over (partition by date_part ('year',
   order_purchase_timestamp)
   order by count(o.order_id) desc) as rank_canceled_order
FROM product as p
JOIN order_items as oi
   on p.product_id = oi.product_id
JOIN orders as o
   on oi.order_id = o.order_id
WHERE order_status = 'canceled'
GROUP BY 1,2)
```

Create table to show the product where generate the highest amount of cancel order using filter rank_canceled_order = 1.

```
SELECT
   year,
   product_category_name as highest_canceled_order
FROM master
WHERE rank_canceled_order = 1
```

	year double precision	highest_canceled_order character varying
1	2016	toys
2	2017	sports_leisure
3	2018	health_beauty

5. **Compile the tables**

```
SELECT
   tr.year,
   tr.revenue_per_year,
   hr.highest_revenue_product,
   tc.count,
   mc.highest_canceled_order
FROM total_revenue_per_year as tr
JOIN highest_revenue_product as hr
   on tr.year=hr.year
JOIN total_cancel_per_year as tc
   on hr.year=tc.year
JOIN most_canceled_product as mc
   on tc.year=mc.year
GROUP BY 1,2,3,4,5
ORDER BY 1
```

	year double precision	revenue_per_year double precision	highest_revenue_product character varying	total_cancel_order bigint	highest_canceled_order character varying
1	2016	46682	furniture_decor	26	toys
2	2017	6924371	bed_bath_table	265	sports_leisure
3	2018	8452044	health_beauty	334	health_beauty

D. ANNUAL PAYMENT TYPE USAGE ANALYSIS

1. Top payment method for all time

Create table to show the number of user for each payment type of all time. Order the data from large amount to show the most popular payment type.

```
SELECT
   op.payment_type,
   count(o.order_id) as jumlah_penggunaan
FROM orders as o
JOIN order_payments as op
   on o.order_id=op.order_id
GROUP BY 1
ORDER BY 2 desc;
```

	payment_type character varying	jumlah_penggunaan bigint
1	credit_card	76795
2	boleto	19784
3	voucher	5775
4	debit_card	1529
5	not_defined	3

2. Top payment method per year

Create table to show the number of user for each payment type per year.

```
WITH master as (
SELECT
   date_part('year',o.order_purchase_timestamp) as year,
   op.payment_type as payment_type,
   count(o.order_id) as num_used
FROM orders as o
JOIN order_payments as op
   on o.order_id=op.order_id
GROUP BY 1,2
ORDER BY 1,3 desc)
```

Create pct_change_2017_2018 column to define the rate of difference for each payment type user per year.

```
SELECT *,
   case when year_2017 = 0 then NULL
   else round((year_2018 - year_2017) / year_2017, 2)
   end as pct_change_2017_2018
FROM (
SELECT
```

```
   payment_type,
   sum(case when year = '2016' then num_used else 0 end) as
   year_2016,
   sum(case when year = '2017' then num_used else 0 end) as
   year_2017,
   sum(case when year = '2018' then num_used else 0 end) as
   year_2018
FROM master
GROUP BY 1) subq
ORDER BY 5 desc;
```

	payment_type character varying	year_2016 numeric	year_2017 numeric	year_2018 numeric	pct_change_2017_2018 numeric
1	not_defined	0	0	3	[null]
2	debit_card	2	422	1105	1.62
3	credit_card	258	34568	41969	0.21
4	boleto	63	9508	10213	0.07
5	voucher	23	3027	2725	-0.10