STAGE 1: Data Preparation

1. Membuat database melalui klik kanan Databases > Create > Database. dengan nama eCommerceDB.

2. Membuat tabel menggunakan statement CREATE TABLE dan memastikan tipe datanya sesuai.

```
CREATE TABLE customers (
   customer_id VARCHAR(255) PRIMARY KEY,
   customer_unique_id VARCHAR(255),
   customer_zip_code_prefix VARCHAR(10),
   customer_city VARCHAR(255),
   customer_state VARCHAR(255)
);


CREATE TABLE geolocation (
   geolocation_zip_code_prefix VARCHAR(10),
   geolocation_lat NUMERIC,
   geolocation_lng NUMERIC,
   geolocation_city VARCHAR(100),
   geolocation_state VARCHAR(5)
);

CREATE TABLE order_items (
   order_id VARCHAR(255),
   order_item_id INTEGER,
   product_id VARCHAR(255),
   seller_id VARCHAR(255),
   shipping_limit_date TIMESTAMP,
   price NUMERIC,
   freight_value NUMERIC
);

CREATE TABLE order_payments(
        order_id VARCHAR(255),
        payment_sequential INTEGER,
        payment_type VARCHAR(50),
        payment_installments INTEGER,
        payment_value NUMERIC
);
```

```
CREATE TABLE order_reviews(
        review_id VARCHAR(255),
        order_id VARCHAR(255),
        review_score INTEGER,
        review_comment_title TEXT,
        review_comment_message TEXT,
        review_creation_date TIMESTAMP,
        review_answer_timestamp TIMESTAMP,
);
CREATE TABLE orders(
   order_id VARCHAR(255),
   customer_id VARCHAR(255),
   order_status VARCHAR(50),
   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(
   product_id VARCHAR(255),
   product_category_name VARCHAR(255),
   product_name_length INTEGER,
   product_description_length INTEGER,
   product_photos_qty INTEGER,
   product_weight_g FLOAT,
   product_length_cm FLOAT,
   product_height_cm FLOAT,
   product_width_cm FLOAT
);

CREATE TABLE sellers(
   seller_id VARCHAR(255),
   seller_zip_code_prefix VARCHAR(10),
   seller_city VARCHAR(255),
   seller_state VARCHAR(5)
);
```

3. Mengimpor file csv ke dalam masing-masing tabel yang telah dibuat dengan klik kanan pada nama tabel > Import/Export Data.

4. Menentukan Primary Key dan Foreign Key agar terbentuk relasi antar tabelnya.

```
-- Menambahkan Primary Key ke tabel customers
-- Ket: Sudah dibuat saat CREATE TABLE customers

-- Menambahkan Primary Key ke tabel product
ALTER TABLE product
ADD CONSTRAINT product_pkey PRIMARY KEY (product_id);

-- Menambahkan Primary Key ke tabel sellers
ALTER TABLE sellers
ADD CONSTRAINT sellers_pkey PRIMARY KEY (seller_id);

-- Menambahkan Primary Key ke tabel orders
ALTER TABLE orders
ADD CONSTRAINT orders_pkey PRIMARY KEY (order_id);

-- Menambahkan Foreign Key ke tabel order_items
ALTER TABLE order_items
ADD CONSTRAINT fk_order_items_orders
FOREIGN KEY (order_id) REFERENCES orders(order_id);

ALTER TABLE order_items
ADD CONSTRAINT fk_order_items_product
FOREIGN KEY (product_id) REFERENCES product(product_id);

ALTER TABLE order_items
ADD CONSTRAINT fk_order_items_sellers
FOREIGN KEY (seller_id) REFERENCES sellers(seller_id);

-- Menambahkan Foreign Key ke tabel order_payments
ALTER TABLE order_payments
ADD CONSTRAINT fk_order_payments_orders
FOREIGN KEY (order_id) REFERENCES orders(order_id);

-- Menambahkan Foreign Key ke tabel order_reviews
ALTER TABLE order_reviews
ADD CONSTRAINT fk_order_reviews_orders
FOREIGN KEY (order_id) REFERENCES orders(order_id);
```

```
-- Menambahkan Foreign Key ke tabel orders
ALTER TABLE orders
ADD CONSTRAINT fk_orders_customers
FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
```

Catatan: Sebab tabel geolocation tidak memiliki nilai yang unik di semua kolom-nya, maka dibuat tabel dummy geolocation_temp yang berisi semua zip_code_prefix unik dari geolocation, customers, dan sellers agar tabel geolocation, customers, dan sellers dapat saling terhubung.

```
CREATE TABLE geolocation_temp AS
SELECT DISTINCT geolocation_zip_code_prefix
FROM geolocation;

INSERT INTO geolocation_temp (geolocation_zip_code_prefix)
SELECT DISTINCT customer_zip_code_prefix
FROM customers
WHERE customer_zip_code_prefix NOT IN (SELECT geolocation_zip_code_prefix
FROM geolocation_temp);

INSERT INTO geolocation_temp (geolocation_zip_code_prefix)
SELECT DISTINCT seller_zip_code_prefix
FROM sellers
WHERE seller_zip_code_prefix NOT IN (SELECT geolocation_zip_code_prefix
FROM geolocation_temp);

ALTER TABLE geolocation_temp
ADD CONSTRAINT uq_geolocation_temp_zip_code_prefix
UNIQUE (geolocation_zip_code_prefix);

ALTER TABLE customers
ADD CONSTRAINT fk_customers_geolocation_temp
FOREIGN KEY (customer_zip_code_prefix)
REFERENCES geolocation_temp(geolocation_zip_code_prefix);

ALTER TABLE geolocation
ADD CONSTRAINT fk_geolocation_geolocation_temp
FOREIGN KEY (geolocation_zip_code_prefix)
REFERENCES geolocation_temp(geolocation_zip_code_prefix);

ALTER TABLE sellers
ADD CONSTRAINT fk_sellers_geolocation_temp
FOREIGN KEY (seller_zip_code_prefix)
REFERENCES geolocation_temp(geolocation_zip_code_prefix);
```

5. Membuat ERD dengan cara klik kanan pada database eCommerceDB > Gererate ERD.

STAGE 2: Annual Customer Activity Growth Analysis

--1. Menampilkan rata-rata jumlah customer aktif bulanan (monthly active user) untuk setiap tahun.

```
WITH monthly_active_users AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
        c.customer_unique_id
    FROM orders o
    JOIN customers c ON o.customer_id = c.customer_id
),
active_users_per_month AS (
    SELECT
        year,
        month,
        COUNT(DISTINCT customer_unique_id) AS customer_total
    FROM monthly_active_users
    GROUP BY year, month
),
average_active_users_per_year AS (
    SELECT
        year,
        FLOOR(AVG(customer_total)) AS avg_mau
    FROM active_users_per_month
    GROUP BY year
)
SELECT
    year,
    avg_mau
FROM
    average_active_users_per_year
ORDER BY
    year;
```

```
-- 2. Menampilkan jumlah customer baru pada masing-masing tahun.
SELECT
	first_order_year AS year,
	COUNT(DISTINCT customer_unique_id) AS total_new_customers
FROM(
	SELECT
		customer_unique_id,
		MIN(EXTRACT(YEAR FROM order_purchase_timestamp)) AS first_order_year
	FROM customers c
	JOIN orders o ON c.customer_id = o.customer_id
	GROUP BY customer_unique_id
) AS df
GROUP BY year
ORDER BY year ASC;
```

```
-- 3. Menampilkan jumlah customer yang melakukan pembelian lebih dari satu kali (repeat order) pada masing-masing tahun.
WITH repeat_customers AS (
  SELECT
    c.customer_unique_id,
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
    COUNT(customer_unique_id) AS total_orders
  FROM orders o
  JOIN customers c ON o.customer_id = c.customer_id
  GROUP BY c.customer_unique_id, year
  HAVING COUNT(customer_unique_id) > 1
)
SELECT
  year,
  COUNT(DISTINCT customer_unique_id) AS repeat_customers
FROM repeat_customers
GROUP BY year
ORDER BY year;
```

```
-- 4. Menampilkan rata-rata jumlah order yang dilakukan customer untuk masing-masing tahun.
WITH order_frequency AS (
    SELECT
        c.customer_unique_id,
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        COUNT(order_id) AS total_orders
    FROM orders o
    JOIN customers c ON o.customer_id = c.customer_id
    GROUP BY c.customer_unique_id, year
),
average_orders_per_year AS (
    SELECT
        year,
        AVG(total_orders) AS avg_orders
    FROM order_frequency
    GROUP BY year
)
SELECT
    year,
    ROUND(AVG(avg_orders), 4) AS avg_orders_per_year
FROM average_orders_per_year
GROUP BY year
ORDER BY year;


-- 5. Menggabungkan semua hasil yang telah berhasil ditampilkan menjadi satu tampilan tabel master.
WITH monthly_active_users AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
        c.customer_unique_id AS customer_id
    FROM orders o
    JOIN customers c ON o.customer_id = c.customer_id
),
active_users_per_month AS (
    SELECT
        year,
        month,
        COUNT(DISTINCT customer_id) AS customer_total
    FROM monthly_active_users
    GROUP BY year, month
),
```

```
average_active_users_per_year AS (
    SELECT
        year,
        FLOOR(AVG(customer_total)) AS avg_mau
    FROM active_users_per_month
    GROUP BY year
),
new_customers AS (
    SELECT
        first_order_year AS year,
        COUNT(DISTINCT customer_unique_id) AS total_new_customers
    FROM (
        SELECT
            customer_unique_id,
                        MIN(EXTRACT(YEAR  FROM  order_purchase_timestamp))  AS
first_order_year
        FROM customers c
        JOIN orders o ON c.customer_id = o.customer_id
        GROUP BY customer_unique_id
    ) AS df
    GROUP BY year
),
repeat_customers AS (
    SELECT
        year,
        COUNT(DISTINCT customer_unique_id) AS repeat_customers
    FROM (
        SELECT
            c.customer_unique_id,
            EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
            COUNT(customer_unique_id) AS total_orders
        FROM orders o
        JOIN customers c ON o.customer_id = c.customer_id
        GROUP BY c.customer_unique_id, year
        HAVING COUNT(customer_unique_id) > 1
    ) AS repeat_customers
    GROUP BY year
),
average_orders_per_year AS (
    SELECT
        year,
        ROUND(AVG(avg_orders), 4) AS avg_orders_per_year
    FROM (
```

```
        SELECT
            year,
            AVG(total_orders) AS avg_orders
        FROM (
            SELECT
                c.customer_unique_id,
                EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
                COUNT(order_id) AS total_orders
            FROM orders o
            JOIN customers c ON o.customer_id = c.customer_id
            GROUP BY c.customer_unique_id, year
        ) AS order_frequency
        GROUP BY year
    ) AS average_orders_per_year
    GROUP BY year
)
SELECT
    mau.year,
    mau.avg_mau AS average_active_users,
    nc.total_new_customers AS new_customers,
    rc.repeat_customers AS repeat_customers,
    aoy.avg_orders_per_year AS avg_orders_per_year
FROM
    average_active_users_per_year mau
JOIN
    new_customers nc ON mau.year = nc.year
JOIN
    repeat_customers rc ON mau.year = rc.year
JOIN
    average_orders_per_year aoy ON mau.year = aoy.year
ORDER BY
    mau.year;
```

STAGE 3: Annual Product Category Quality Analysis

```
--1. Membuat tabel yang berisi informasi pendapatan/revenue perusahaan total
untuk masing-masing tahun.
SELECT
        SUM(price + freight_value) AS total_revenue,
        EXTRACT(YEAR FROM order_purchase_timestamp) AS year
FROM order_items oi
JOIN orders o ON oi.order_id = o.order_id
WHERE order_status IN ('delivered')
```

```
GROUP BY EXTRACT(YEAR FROM order_purchase_timestamp)
ORDER BY year ASC;

--2. Membuat tabel yang berisi informasi jumlah cancel order total untuk
masing-masing tahun.
SELECT
   EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
   COUNT(order_status) AS total_cancel_orders
FROM orders
WHERE order_status = 'canceled'
GROUP BY EXTRACT(YEAR FROM order_purchase_timestamp)
ORDER BY year;

--3. Membuat tabel yang berisi nama kategori produk yang memberikan pendapatan
total tertinggi untuk masing-masing tahun.
SELECT
        product_category_name AS product_name,
        year
FROM
        (
        SELECT
                product_category_name,
                SUM(price + freight_value) AS revenue,
                ROW_NUMBER() OVER(PARTITION BY  EXTRACT(YEAR FROM
order_purchase_timestamp) ORDER BY SUM(oi.price + oi.freight_value) DESC) AS
rank_product,
                EXTRACT(YEAR FROM order_purchase_timestamp) AS year
        FROM orders o
        JOIN order_items oi ON o.order_id = oi.order_id
        JOIN product p ON p.product_id = oi.product_id
        WHERE order_status = 'delivered'
        GROUP BY EXTRACT(YEAR FROM order_purchase_timestamp),
product_category_name
) AS df
WHERE rank_product = 1;

--4. Membuat tabel yang berisi nama kategori produk yang memiliki jumlah cancel
order terbanyak untuk masing-masing tahun.
SELECT
        product_category_name AS product_name,
        year
FROM
        (
        SELECT
```

```
            product_category_name,
            COUNT(*) AS total_cancel,
            ROW_NUMBER() OVER(PARTITION BY  EXTRACT(YEAR FROM
order_purchase_timestamp) ORDER BY COUNT(*) DESC) AS rank_product,
            EXTRACT(YEAR FROM order_purchase_timestamp) AS year
        FROM orders o
        JOIN order_items oi ON o.order_id = oi.order_id
        JOIN product p ON p.product_id = oi.product_id
        WHERE order_status = 'canceled'
        GROUP BY EXTRACT(YEAR FROM order_purchase_timestamp),
product_category_name
) AS df
WHERE rank_product = 1;


--5. Menggabungkan informasi-informasi yang telah didapatkan ke dalam satu
tampilan tabel.
WITH revenue_per_year AS (
  SELECT
    SUM(price + freight_value) AS total_revenue,
    EXTRACT(YEAR FROM order_purchase_timestamp) AS year
  FROM
    order_items oi
  JOIN
    orders o ON oi.order_id = o.order_id
  WHERE
    order_status = 'delivered'
  GROUP BY
    EXTRACT(YEAR FROM order_purchase_timestamp)
),
cancel_orders_per_year AS (
  SELECT
    EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
    COUNT(order_status) AS total_cancel_orders
  FROM
    orders
  WHERE
    order_status = 'canceled'
  GROUP BY
    EXTRACT(YEAR FROM order_purchase_timestamp)
),
top_revenue_category_per_year AS (
  SELECT
    product_category_name AS product_name,
```

```
        year
    FROM
        (
        SELECT
            product_category_name,
            SUM(price + freight_value) AS revenue,
            ROW_NUMBER() OVER(PARTITION BY EXTRACT(YEAR FROM
order_purchase_timestamp) ORDER BY SUM(oi.price + oi.freight_value) DESC) AS
rank_product,
            EXTRACT(YEAR FROM order_purchase_timestamp) AS year
        FROM
            orders o
        JOIN
            order_items oi ON o.order_id = oi.order_id
        JOIN
            product p ON p.product_id = oi.product_id
        WHERE
            order_status = 'delivered'
        GROUP BY
            EXTRACT(YEAR FROM order_purchase_timestamp),
product_category_name
        ) AS df
    WHERE
        rank_product = 1
),
top_cancel_category_per_year AS (
    SELECT
        product_category_name AS product_name,
        year
    FROM
        (
        SELECT
            product_category_name,
            COUNT(*) AS total_cancel,
            ROW_NUMBER() OVER(PARTITION BY EXTRACT(YEAR FROM
order_purchase_timestamp) ORDER BY COUNT(*) DESC) AS rank_product,
            EXTRACT(YEAR FROM order_purchase_timestamp) AS year
        FROM
            orders o
        JOIN
            order_items oi ON o.order_id = oi.order_id
        JOIN
            product p ON p.product_id = oi.product_id
        WHERE
```

```
            order_status = 'canceled'
        GROUP BY
            EXTRACT(YEAR FROM order_purchase_timestamp),
product_category_name
        ) AS df
    WHERE
        rank_product = 1
)
SELECT
    r.year,
    r.total_revenue,
    c.total_cancel_orders,
    tr.product_name AS top_revenue_category,
    tc.product_name AS top_cancel_category
FROM
    revenue_per_year r
JOIN
    cancel_orders_per_year c ON r.year = c.year
JOIN
    top_revenue_category_per_year tr ON r.year = tr.year
JOIN
    top_cancel_category_per_year tc ON r.year = tc.year
ORDER BY
    r.year;
```

STAGE 4: Annual Payment Type Usage Analysis

```
--1. Menampilkan jumlah penggunaan masing-masing tipe pembayaran secara all
time diurutkan dari yang terfavorit.
SELECT
        payment_type,
        COUNT(*) AS payment_count
FROM order_payments
GROUP BY payment_type
ORDER BY payment_count DESC;

--2. Menampilkan detail informasi jumlah penggunaan masing-masing tipe
pembayaran untuk setiap tahun.
SELECT
    payment_type,
    COUNT(CASE WHEN EXTRACT(YEAR FROM order_purchase_timestamp) =
2016 THEN 1 END) AS "2016",
```

```
   COUNT(CASE WHEN EXTRACT(YEAR FROM order_purchase_timestamp) =
2017 THEN 1 END) AS "2017",
   COUNT(CASE WHEN EXTRACT(YEAR FROM order_purchase_timestamp) =
2018 THEN 1 END) AS "2018"
FROM
   order_payments op
JOIN
   orders o ON op.order_id = o.order_id
GROUP BY
   payment_type
ORDER BY
   "2016" DESC, "2017" DESC, "2018" DESC;

--Menampilkan seluruh informasi jumlah penggunaan masing-masing tipe
pembayaran untuk setiap tahun.
WITH payment_counts AS (
   -- 1. Menampilkan jumlah penggunaan masing-masing tipe pembayaran secara all
time diurutkan dari yang terfavorit.
   SELECT
      payment_type,
      COUNT(*) AS payment_count
   FROM order_payments
   GROUP BY payment_type
),
yearly_payment_counts AS (
   -- 2. Menampilkan detail informasi jumlah penggunaan masing-masing tipe
pembayaran untuk setiap tahun.
   SELECT
      payment_type,
      COUNT(CASE WHEN EXTRACT(YEAR FROM order_purchase_timestamp) =
2016 THEN 1 END) AS "2016",
      COUNT(CASE WHEN EXTRACT(YEAR FROM order_purchase_timestamp) =
2017 THEN 1 END) AS "2017",
      COUNT(CASE WHEN EXTRACT(YEAR FROM order_purchase_timestamp) =
2018 THEN 1 END) AS "2018"
   FROM
      order_payments op
   JOIN
      orders o ON op.order_id = o.order_id
   GROUP BY
      payment_type
)
SELECT
   pc.payment_type,
```

```
    pc.payment_count,
    ypc."2016",
    ypc."2017",
    ypc."2018"
FROM
    payment_counts pc
JOIN
    yearly_payment_counts ypc ON pc.payment_type = ypc.payment_type
ORDER BY
    pc.payment_count DESC;
```