Analyzing eCommerce Business Performance

Theofilus Arifin

LinkedIn : linkedin.com/in/theofilusarifin/

Github : github.com/Theofilusarifin

Portfolio : datascienceportfol.io/theodev

Data Preparation

Query History:

1. Create Table

```
CREATE TABLE customers (
    customer_id VARCHAR(32),
    customer_unique_id VARCHAR(32),
    customer_zip_code_prefix INTEGER,
    customer_city VARCHAR(255),
    customer_state VARCHAR(2)
);

CREATE TABLE geolocations (
    zip_code_prefix INTEGER,
    at NUMERIC(10, 8),
    lng NUMERIC(11, 8),
    city VARCHAR(255),
    state VARCHAR(2)
);

CREATE TABLE order_items (
    order_id VARCHAR(32),
    order_item_id INTEGER,
    product_id VARCHAR(32),
    seller_id VARCHAR(32),
    shipping_limit_date TIMESTAMP,
    price NUMERIC(10, 2),
    freight_value NUMERIC(10, 2)
);

CREATE TABLE payments (
    order_id VARCHAR(32),
    sequential INTEGER,
    type VARCHAR(50),
    installments INTEGER,
    value NUMERIC(10, 2)
);
```

```
CREATE TABLE reviews (
   review_id VARCHAR(32),
   order_id VARCHAR(32),
   score INTEGER,
   comment_title VARCHAR(255),
   comment_message TEXT,
   creation_date TIMESTAMP,
   answer_timestamp TIMESTAMP
);

CREATE TABLE orders (
   order_id VARCHAR(32),
   customer_id VARCHAR(32),
   status VARCHAR(50),
   purchase_timestamp TIMESTAMP,
   approved_at TIMESTAMP,
   delivered_carrier_date TIMESTAMP,
   delivered_customer_date TIMESTAMP,
   estimated_delivery_date TIMESTAMP
);

CREATE TABLE products (
   product_id VARCHAR(32),
   category_name VARCHAR(255),
   name_length INTEGER,
   description_length INTEGER,
   photos_qty INTEGER,
   weight_g INTEGER,
   length_cm INTEGER,
   height_cm INTEGER,
   width_cm INTEGER
);

CREATE TABLE sellers (
   seller_id VARCHAR(32),
   zip_code_prefix INTEGER,
   city VARCHAR(255),
   state VARCHAR(2)
);
```

2. Add Primary Key

```
ALTER TABLE customers
ADD PRIMARY KEY (customer_id);

ALTER TABLE orders
ADD PRIMARY KEY (order_id);

ALTER TABLE products
ADD PRIMARY KEY (product_id);

ALTER TABLE sellers
ADD PRIMARY KEY (seller_id);
```

3. Clear duplicate data on geolocations

```
DELETE FROM
   geolocations
WHERE
   ctid IN (
      SELECT
         ctid
      FROM (
         SELECT
            ctid,
            ROW_NUMBER() OVER (PARTITION BY zip_code_prefix ORDER BY lat) AS
row_num
         FROM
            geolocations
      ) AS sub
      WHERE
         row_num > 1
   );
```

4. Fill missing geolocation data for both customers and sellers by imputing data with the closest available ID in the "geolocations" table

```
INSERT INTO geolocations (zip_code_prefix, lat, lng, city, state)
SELECT
   c.zip_code_prefix,
   g.lat,
   g.lng,
   g.city,
   g.state
FROM
   (
       SELECT DISTINCT zip_code_prefix
       FROM customers
       WHERE zip_code_prefix NOT IN (
           SELECT DISTINCT zip_code_prefix
           FROM geolocations
       )
   ) AS c
CROSS JOIN LATERAL (
   SELECT
       lat,
       lng,
       city,
       state
   FROM
       geolocations
   ORDER BY
       zip_code_prefix
   LIMIT 1
) AS g;


INSERT INTO geolocations (zip_code_prefix, lat, lng, city, state)
SELECT
   c.zip_code_prefix,
   g.lat,
   g.lng,
   g.city,
   g.state
FROM
   (
       SELECT DISTINCT zip_code_prefix
       FROM sellers
       WHERE zip_code_prefix NOT IN (
           SELECT DISTINCT zip_code_prefix
           FROM geolocations
       )
   ) AS c
CROSS JOIN LATERAL (
   SELECT
       lat,
       lng,
       city,
       state
   FROM
       geolocations
   ORDER BY
       zip_code_prefix
   LIMIT 1
) AS g;
```

5. Add relationship

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer_id
FOREIGN KEY (customer_id)
REFERENCES customers(customer_id);

ALTER TABLE reviews
ADD CONSTRAINT fk_order_id
FOREIGN KEY (order_id)
REFERENCES orders(order_id);

ALTER TABLE payments
ADD CONSTRAINT fk_order_id
FOREIGN KEY (order_id)
REFERENCES orders(order_id);

ALTER TABLE order_items
ADD CONSTRAINT fk_order_id
FOREIGN KEY (order_id)
REFERENCES orders(order_id);

ALTER TABLE order_items
ADD CONSTRAINT fk_product_id
FOREIGN KEY (product_id)
REFERENCES products(product_id);

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

```
ALTER TABLE sellers
ADD CONSTRAINT fk_zip_code_prefix
FOREIGN KEY (zip_code_prefix)
REFERENCES geolocations(zip_code_prefix);

ALTER TABLE customers
ADD CONSTRAINT fk_zip_code_prefix
FOREIGN KEY (zip_code_prefix)
REFERENCES geolocations(zip_code_prefix);
```

Annual Customer Activity Growth Analysis

Query History:

1. Common Table Expression

```
WITH temp AS (
   SELECT
      c.customer_unique_id AS id,
      DATE_PART('year', o.purchase_timestamp) AS year,
      DATE_PART('month', o.purchase_timestamp) AS month
   FROM
      orders o
   INNER JOIN
      customers c ON c.customer_id = o.customer_id
),

customer_2016 AS (
   SELECT DISTINCT id, year
   FROM temp
   WHERE year = 2016
),

customer_2017 AS (
   SELECT DISTINCT id, year
   FROM temp
   WHERE year = 2017
   AND id NOT IN (SELECT id FROM customer_2016)
),

customer_2018 AS (
   SELECT DISTINCT id, year
   FROM temp
   WHERE year = 2018
   AND id NOT IN (SELECT id FROM customer_2016)
   AND id NOT IN (SELECT id FROM customer_2017)
),

-- CTE for number 1 : monthly active users
```

```
monthly_active_users AS (
	SELECT
		ROUND(AVG(monthly_active_users)) AS average_monthly_active_users,
		year
	FROM (
		SELECT
			year,
			COUNT(DISTINCT id) AS monthly_active_users
		FROM
			temp
		GROUP BY
			year, month
	) AS monthly_active_users_per_month
	GROUP BY
		year
	ORDER BY
		year
),

-- CTE for number 2 : total new customers
new_customers AS (
	SELECT COUNT(id) as total_new_customer, year
	FROM customer_2016
	GROUP BY year
	UNION
	SELECT COUNT(id) as total_new_customer, year
	FROM customer_2017
	GROUP BY year
	UNION
	SELECT COUNT(id) as total_new_customer, year
	FROM customer_2018
	GROUP BY year
),

-- CTE for number 3 : total repeat orders
repeat_orders AS (
	SELECT
		SUM(total_repeat_orders) AS total_repeat_order_customer,
		year
	FROM (
		SELECT
			COUNT(*) - 1 AS total_repeat_orders,
			year
		FROM
```

```
                temp
            GROUP BY
                id,
                year
            HAVING COUNT(*) > 1
        ) AS repeat_orders_per_year
        GROUP BY
            year
        ORDER BY
            year
),

-- CTE for number 4 : average total order
total_orders AS (
        SELECT
            AVG(total_order) AS average_total_order,
            year
        FROM
            (
                SELECT
                    COUNT(*) AS total_order,
                    year
                FROM
                    temp
                GROUP BY
                    id, year
            ) as total_order_per_year
        GROUP BY
            year
        ORDER BY
            year
)
```

2. Create Master Table

```
CREATE TABLE master (
   year INTEGER PRIMARY KEY,
   average_monthly_active_users FLOAT,
   total_new_customer INTEGER,
   total_repeat_order_customer INTEGER,
   average_total_order FLOAT
);
```

3. Insert Master Data

```
INSERT INTO master (year, average_monthly_active_users, total_new_customer,
total_repeat_order_customer, average_total_order)
SELECT
   m.year,
   m.average_monthly_active_users,
   n.total_new_customer,
   r.total_repeat_order_customer,
   t.average_total_order
FROM
   (
      SELECT
         year,
         average_monthly_active_users
      FROM
         monthly_active_users
   ) AS m
JOIN
   (
      SELECT
         year,
         total_new_customer
      FROM
         new_customers
   ) AS n ON m.year = n.year
JOIN
   (
      SELECT
         year,
         total_repeat_order_customer
      FROM
         repeat_orders
   ) AS r ON m.year = r.year
JOIN
   (
      SELECT
         year,
         average_total_order
      FROM
         total_orders
   ) AS t ON m.year = t.year;
```

Annual Product Category Quality Analysis

Query History:

1. Common Table Expression

```
-- Nomor 1 : Total Revenue Yearly
WITH TotalRevenue AS (
   SELECT
      SUM(oi.price * oi.order_item_id) + SUM(freight_value) AS total_revenue,
      DATE_PART('year', o.purchase_timestamp) AS year
   FROM
      orders o
   INNER JOIN
      order_items oi
   ON o.order_id = oi.order_id
   WHERE
      o.status = 'delivered'
   GROUP BY
      DATE_PART('year', o.purchase_timestamp)
),

-- Nomor 2 : Total Cancel Order Yearly
TotalCancelOrder AS (
   SELECT
      COUNT(*) AS total_cancel_order,
      DATE_PART('year', purchase_timestamp) AS year
   FROM
      orders
   WHERE
      status = 'canceled'
   GROUP BY
      DATE_PART('year', purchase_timestamp)
),

-- Nomor 3 : highest_revenue_product_category
HighestRevenueProductCategory AS (
         SELECT
      category_name AS highest_revenue_product_category,
                    total_revenue,
        year
    FROM (
        SELECT
            p.category_name,
            DATE_PART('year', o.purchase_timestamp) AS year,
                            SUM(oi.price * oi.order_item_id) + SUM(freight_value) as total_revenue,
            RANK() OVER(PARTITION BY DATE_PART('year', o.purchase_timestamp) ORDER BY
SUM(oi.price * oi.order_item_id) + SUM(freight_value) DESC) AS rank_revenue
        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.status = 'delivered'
        GROUP BY
            p.category_name,
            DATE_PART('year', o.purchase_timestamp)
        ) AS subquery
    WHERE
        rank_revenue = 1
),

-- Nomor 4 : highest_total_cancel_order_product_category
HighestTotalCancelOrderProductCategory AS (
    SELECT
        category_name AS highest_total_cancel_order_product_category,
                    total_cancel_order,
        year
    FROM (
        SELECT
            p.category_name,
            DATE_PART('year', o.purchase_timestamp) AS year,
                            COUNT(*) as total_cancel_order,
            RANK() OVER(PARTITION BY DATE_PART('year', o.purchase_timestamp) ORDER BY
COUNT(*) DESC) AS rank_total_cancel_order
        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.status = 'canceled'
        GROUP BY
           p.category_name,
           DATE_PART('year', o.purchase_timestamp)
        ) AS subquery
    WHERE
        rank_total_cancel_order = 1
)
```

2. Create Master Table

```
CREATE TABLE master_2 (
           year INTEGER PRIMARY KEY,
           total_revenue FLOAT,
           total_cancel_order INTEGER,
           highest_revenue_product_category VARCHAR(255),
           highest_total_cancel_order_product_category VARCHAR(255)
);
```

3. Insert Master Data

```
INSERT INTO master_2 (year, total_revenue, total_cancel_order,
highest_revenue_product_category, highest_total_cancel_order_product_category)
SELECT
    tr.year,
    tr.total_revenue,
    tco.total_cancel_order,
    hrpc.highest_revenue_product_category,
    htcopc.highest_total_cancel_order_product_category
FROM
    TotalRevenue tr
INNER JOIN
    TotalCancelOrder tco ON tr.year = tco.year
INNER JOIN
    HighestRevenueProductCategory hrpc ON tr.year = hrpc.year
INNER JOIN
    HighestTotalCancelOrderProductCategory htcopc ON tr.year = htcopc.year;
```

Annual Payment Type Usage Analysis

Query History:

1. All-Time Payment Usage

```
SELECT
        p.type,
        count(*) as total_usage
FROM
        payments p
INNER JOIN
        orders o
ON o.order_id = p.order_id
GROUP BY
        p.type
ORDER BY
        2 DESC
```

2. Annual Payment Usage

```
-- Tipe Pembayaran Terfavorite All Time
SELECT
        p.type,
        count(*) as total_usage,
        DATE_PART('year', o.purchase_timestamp) AS year
FROM
        payments p
INNER JOIN
        orders o
ON o.order_id = p.order_id
GROUP BY
        p.type,
        DATE_PART('year', o.purchase_timestamp)
ORDER BY
        3,
        2;
```