ANALYZING E-COMMERCE BUSINESS PERFORMANCE WITH SQL

Rakamin Academy Mini Project 1 Report

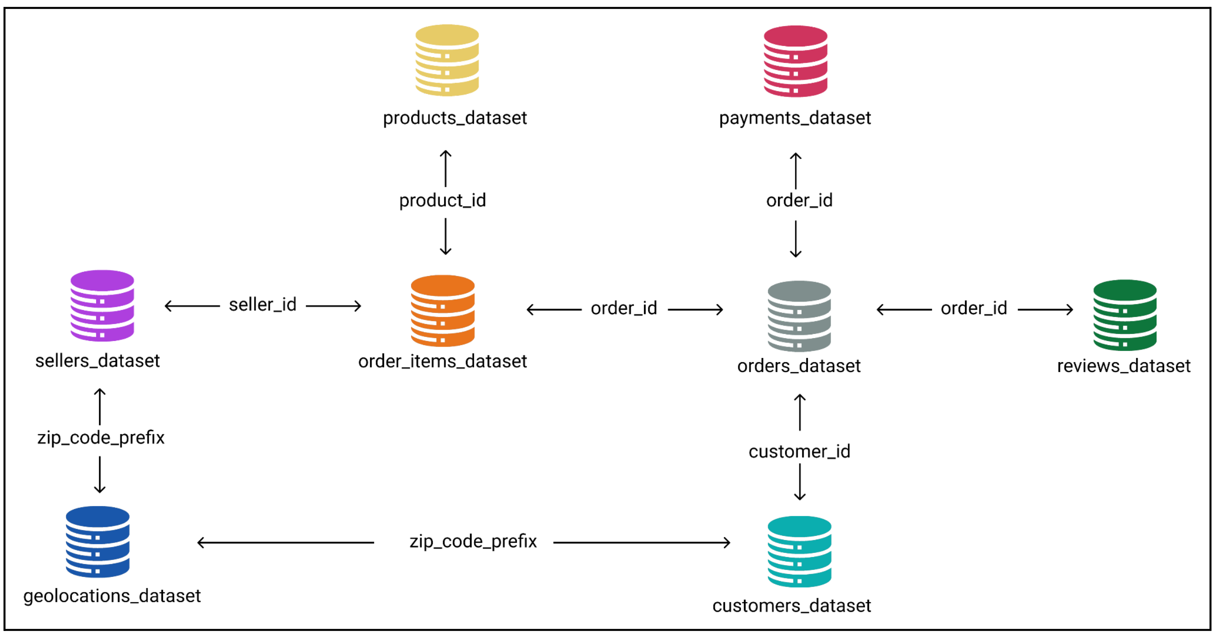
Written by:
Atthoriq Putra Pangestu
Data Science Bootcamp
JAP Batch 21

Stage 1: Preparation

In this stage, we prepared the raw data in the form of 8 CSV files consisting of:

1. Customers dataset
2. Geolocation dataset
3. Order + items dataset
4. Orders dataset
5. Payments dataset
6. Products dataset
7. Reviews dataset
8. Sellers dataset

The raw data was transformed and loaded into a PostgreSQL relational database based on the following Entity-Relationship Diagram (ERD):



The first step involved creating an empty database named ‘eCommerceDB’ within PostgreSQL using its built-in database creation feature. After that, eight empty tables were constructed for the CSV datasets, each with their respective field types, using the following query:

```
-- ========== TABLE CREATION ==========
-- geolocation_dataset
DROP TABLE IF EXISTS public.geolocation_dataset;
```

```sql
CREATE TABLE public.geolocation_dataset(
      geolocation_zip_code_prefix VARCHAR,  -- Can't set PK, duplicates
exist
      geolocation_lat FLOAT8,
      geolocation_lng FLOAT8,
      geolocation_city VARCHAR,
      geolocation_state CHAR(5)
);

-- customers_dataset
DROP TABLE IF EXISTS public.customers_dataset;

CREATE TABLE public.customers_dataset(
      customer_id CHAR(50) PRIMARY KEY,
      customer_unique_id CHAR(50),
      customer_zip_code_prefix VARCHAR,
      customer_city VARCHAR,
      customer_state CHAR(5)
);

-- sellers_dataset
DROP TABLE IF EXISTS public.sellers_dataset;

CREATE TABLE public.sellers_dataset(
      seller_id CHAR(50) PRIMARY KEY,
      seller_zip_code_prefix VARCHAR,
      seller_city VARCHAR,
      seller_state CHAR(5)
);

-- products_dataset
DROP TABLE IF EXISTS public.products_dataset;

CREATE TABLE public.products_dataset(
      idx INT,
      product_id CHAR(50) PRIMARY KEY,
      product_category_name VARCHAR,
      product_name_length FLOAT8,
      product_description_length FLOAT8,
      product_photos_qty FLOAT8,
      product_weight_g FLOAT8,
      product_length_cm FLOAT8,
      product_height_cm FLOAT8,
```

```sql
      product_width_cm FLOAT8
);

-- order_items_dataset
DROP TABLE IF EXISTS public.order_items_dataset;

CREATE TABLE public.order_items_dataset(
      order_id CHAR(50),
      order_item_id CHAR(50), -- Can't set PK, duplicates exist
      product_id CHAR(50),
      seller_id CHAR(50),
      shipping_limit_date TIMESTAMP,
      price FLOAT8,
      freight_value FLOAT8
);

-- orders_dataset
DROP TABLE IF EXISTS public.orders_dataset;

CREATE TABLE public.orders_dataset(
      order_id CHAR(50) PRIMARY KEY,
      customer_id CHAR(50),
      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
);

-- payments_dataset
DROP TABLE IF EXISTS public.payments_dataset;

CREATE TABLE public.payments_dataset(
      order_id CHAR(50),
      payment_sequential INT,
      payment_type VARCHAR,
      payment_installments INT,
      payment_value FLOAT8
);

-- reviews_dataset
DROP TABLE IF EXISTS public.reviews_dataset;
```

```sql
CREATE TABLE public.reviews_dataset(
	review_id CHAR(50), -- Can't set PK, duplicates exist
	order_id CHAR(50),
	review_score INT,
	review_comment_title VARCHAR,
	review_comment_message TEXT,
	review_creation_date TIMESTAMP,
	review_answer_timestamp TIMESTAMP
);
```

We noticed that there are several issues for fields that cannot be set into primary keys. This happened because the corresponding fields contain duplicate values. We will address it later in the process. For now, we're going to populate the empty tables with records from the raw tables (CSVs) we have using the following query:

```sql
-- ========== DATA IMPORT ==========
-- geolocation_dataset
COPY public.geolocation_dataset(
	geolocation_zip_code_prefix,
	geolocation_lat,
	geolocation_lng,
	geolocation_city,
	geolocation_state
)
FROM 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce Business Performance with SQL\DS\geolocation_dataset.csv'
DELIMITER ','
CSV HEADER;

-- customers_dataset
COPY public.customers_dataset(
	customer_id,
	customer_unique_id,
	customer_zip_code_prefix,
	customer_city,
	customer_state
)
FROM 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce Business Performance with SQL\DS\customers_dataset.csv'
DELIMITER ','
CSV HEADER;
```

```sql
-- sellers_dataset
COPY public.sellers_dataset(
	seller_id,
	seller_zip_code_prefix,
	seller_city,
	seller_state
)
FROM 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce 
Business Performance with SQL\DS\sellers_dataset.csv'
DELIMITER ','
CSV HEADER;

-- products_dataset
COPY public.products_dataset(
	idx,
	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 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce 
Business Performance with SQL\DS\product_dataset.csv'
DELIMITER ','
CSV HEADER;

ALTER TABLE public.products_dataset
DROP COLUMN idx;

-- order_items_dataset
COPY public.order_items_dataset(
	order_id,
	order_item_id,
	product_id,
	seller_id,
	shipping_limit_date,
	price,
	freight_value
```

```
)
FROM 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce
Business Performance with SQL\DS\order_items_dataset.csv'
DELIMITER ','
CSV HEADER;

-- orders_dataset
COPY public.orders_dataset(
      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 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce
Business Performance with SQL\DS\orders_dataset.csv'
DELIMITER ','
CSV HEADER;

-- payments_dataset
COPY public.payments_dataset(
      order_id,
      payment_sequential,
      payment_type,
      payment_installments,
      payment_value
)
FROM 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce
Business Performance with SQL\DS\order_payments_dataset.csv'
DELIMITER ','
CSV HEADER;

-- reviews_dataset
COPY public.reviews_dataset(
      review_id,
      order_id,
      review_score,
      review_comment_title,
      review_comment_message,
      review_creation_date,
```

```
      review_answer_timestamp
)
FROM 'D:\COURSES\Data Analytics\Rakamin\JAP\Projects\Analyzing E-Commerce Business Performance with SQL\DS\order_reviews_dataset.csv'
DELIMITER ','
CSV HEADER;
```

We encountered several issues with fields that could not be set as primary keys, particularly in the 'geolocation_dataset' table, which is essential for establishing relationships with the 'sellers_dataset' and 'customers_dataset' tables. The problem comes from duplicate values in the 'geolocation_zip_code_prefix' field, preventing it from being used as a primary key. Consequently, this would make the other fields in the 'sellers_dataset' and 'customers_dataset' tables, which reference 'geolocation_zip_code_prefix' as their foreign key, unable to be set.

Furthermore, we encountered an error when attempting to set 'customers_zip_code_prefix' and 'sellers_zip_code_prefix' as foreign keys referencing 'geolocation_zip_code_prefix' in the 'geolocation_dataset' table. This error arises due to records in the 'customers_dataset' and 'sellers_dataset' where the values in 'customers_zip_code_prefix' and 'sellers_zip_code_prefix' contain entries that do not exist from the referenced key, 'geolocation_zip_code_prefix'.

To address this issue, we began by dropping all duplicate values from the 'geolocation_zip_code_prefix' field in the 'geolocation_dataset' table. Upon further examination, we observed that both 'zip_code_prefix' fields in the 'customers_dataset' and 'sellers_dataset' tables contain city and state information. Hence, we populated the 'geolocation_dataset' table with new records corresponding to the previously unseen values of 'zip_code_prefix' in the 'customers_dataset' and 'sellers_dataset' tables. These new records were extracted from the 'customers_zip_code_prefix' and 'sellers_zip_code_prefix' fields that originally didn't have any match in the ‘geolocation_zip_code_prefix’ field. Here’s the query:

```
-- ========== NECESSARY CLEANING ==========
-- geolocation_dataset
CREATE TABLE geolocation_clean_dataset AS
WITH geolocation_clean AS (
      SELECT geolocation_zip_code_prefix,
             geolocation_lat,
             geolocation_lng,
             geolocation_city,
             geolocation_state
      FROM (
            SELECT ROW_NUMBER() OVER (PARTITION BY geolocation_zip_code_prefix) AS rn,
                   *
```

```
            FROM geolocation_dataset
      ) AS geo_tmp
      WHERE rn = 1
),
geolocation_customer_clean AS (
      SELECT customer_zip_code_prefix,
               geolocation_lat,
               geolocation_lng,
               customer_city,
               customer_state
      FROM(
            SELECT ROW_NUMBER() OVER (PARTITION BY
customer_zip_code_prefix) AS rn,
                     *
            FROM (
                  SELECT c.customer_zip_code_prefix,
                           g.geolocation_lat,
                           g.geolocation_lng,
                           c.customer_city,
                           c.customer_state
                  FROM customers_dataset AS c
                  LEFT JOIN geolocation_dataset AS g
                  ON c.customer_city = g.geolocation_city
                  AND c.customer_state = g.geolocation_state
                  WHERE c.customer_zip_code_prefix NOT IN (
                        SELECT geolocation_zip_code_prefix
                        FROM geolocation_dataset
                  )
            ) AS cust_geo
      ) AS cust_geo_tmp
      WHERE rn = 1
),
geolocation_seller_clean AS (
      SELECT seller_zip_code_prefix,
               geolocation_lat,
               geolocation_lng,
               seller_city,
               seller_state
      FROM(
            SELECT ROW_NUMBER() OVER (PARTITION BY seller_zip_code_prefix)
AS rn,
                     *
            FROM (
```

```
                  SELECT s.seller_zip_code_prefix,
                           g.geolocation_lat,
                           g.geolocation_lng,
                           s.seller_city,
                           s.seller_state
                  FROM sellers_dataset AS s
                  LEFT JOIN geolocation_dataset AS g
                  ON s.seller_city = g.geolocation_city
                  AND s.seller_state = g.geolocation_state
                  WHERE s.seller_zip_code_prefix NOT IN(
                        SELECT geolocation_zip_code_prefix
                        FROM geolocation_dataset
                        UNION
                        SELECT customer_zip_code_prefix
                        FROM geolocation_customer_clean
                  )
            ) AS seller_geo
      ) AS seller_geo_tmp
      WHERE rn = 1
)
SELECT *
FROM geolocation_clean
UNION
SELECT *
FROM geolocation_customer_clean
UNION
SELECT *
FROM geolocation_seller_clean;
```

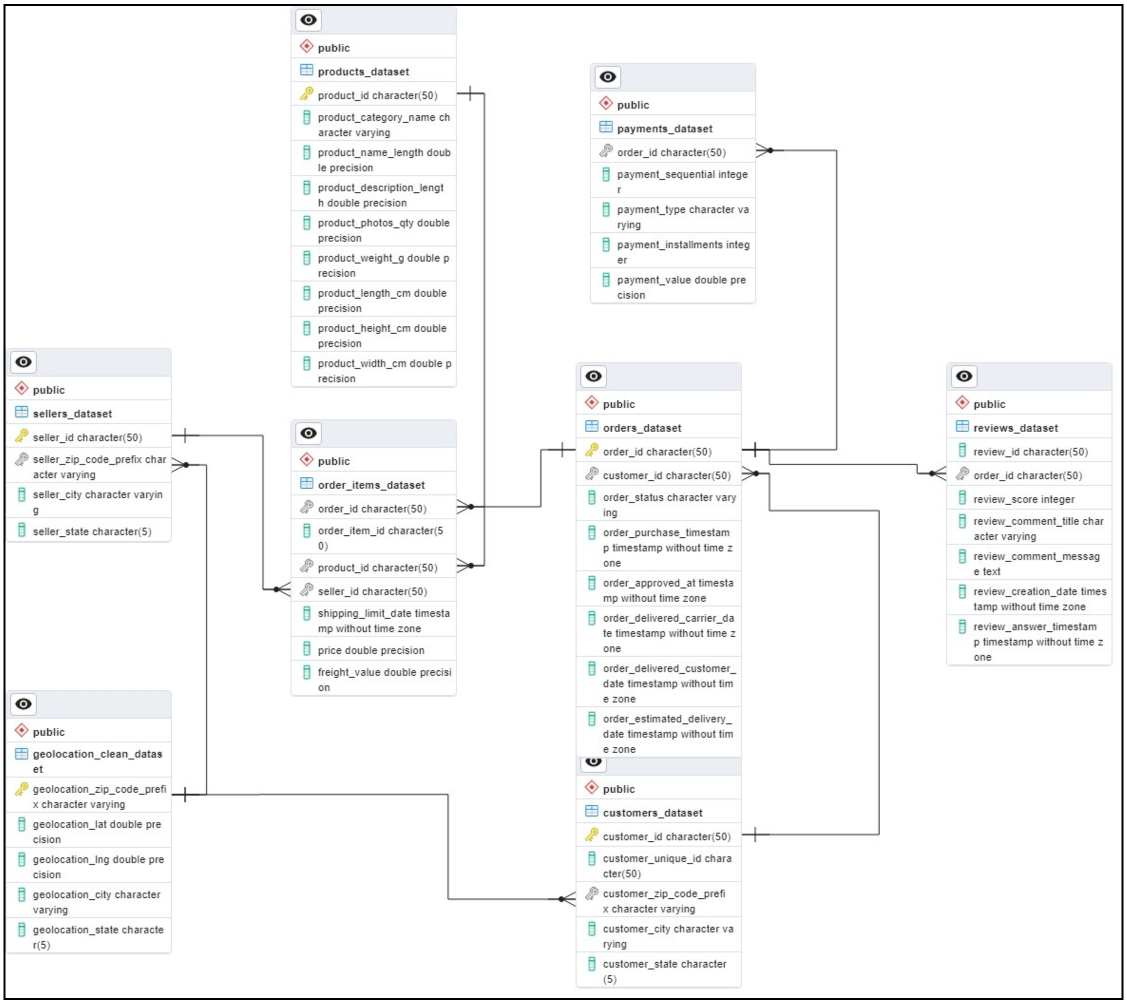
With the data consistency issues resolved, we can now establish the primary and foreign key constraints to define the relationships between tables as outlined in the Entity Relationship Diagram (ERD) discussed earlier. Here's the query:

```
-- ========== ADDING REQUIRED PRIMARY KEYS AND FOREIGN KEYS ==========
-- Primary key: geolocation_clean_dataset
ALTER TABLE geolocation_clean_dataset
ADD CONSTRAINT geolocation_pkey PRIMARY KEY (geolocation_zip_code_prefix);

-- Foreign key: customers_dataset & geolocation_clean_dataset
ALTER TABLE customers_dataset
ADD CONSTRAINT customers_fk_geolocation
FOREIGN KEY (customer_zip_code_prefix) REFERENCES geolocation_clean_dataset
```

```sql
(geolocation_zip_code_prefix)
ON DELETE CASCADE -- This ensures data integrity if there's a deletion in the geolocation_zip_code_prefix, the corresponding customer_zip_code_prefix will be deleted too
ON UPDATE CASCADE; -- This ensures data integrity if there's a value update in the geolocation_zip_code_prefix, the corresponding customer_zip_code_prefix value will be updated too

-- Foreign key: sellers_dataset & geolocation_clean_dataset
ALTER TABLE sellers_dataset
ADD CONSTRAINT sellers_fk_geolocation
FOREIGN KEY (seller_zip_code_prefix) REFERENCES geolocation_clean_dataset
(geolocation_zip_code_prefix)
ON DELETE CASCADE
ON UPDATE CASCADE;

-- Foreign key: order_items_dataset & sellers_dataset
ALTER TABLE order_items_dataset
ADD CONSTRAINT order_items_fk_sellers
FOREIGN KEY (seller_id) REFERENCES sellers_dataset (seller_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

-- Foreign key: order_items_dataset & products_dataset
ALTER TABLE order_items_dataset
ADD CONSTRAINT order_items_fk_products
FOREIGN KEY (product_id) REFERENCES products_dataset (product_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

-- Foreign key: order_items_dataset & orders_dataset
ALTER TABLE order_items_dataset
ADD CONSTRAINT order_items_fk_orders
FOREIGN KEY (order_id) REFERENCES orders_dataset (order_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

-- Foreign key: orders_dataset & customers_dataset
ALTER TABLE orders_dataset
ADD CONSTRAINT orders_fk_customers
FOREIGN KEY (customer_id) REFERENCES customers_dataset (customer_id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

```
-- Foreign key: payments_dataset & orders_dataset
ALTER TABLE payments_dataset
ADD CONSTRAINT payments_fk_orders
FOREIGN KEY (order_id) REFERENCES orders_dataset (order_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

-- Foreign key: reviews_dataset & orders_dataset
ALTER TABLE reviews_dataset
ADD CONSTRAINT reviews_fk_orders
FOREIGN KEY (order_id) REFERENCES orders_dataset (order_id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Once the database structure was established, we generated the following Entity Relationship Diagram (ERD) that follows the ERD plan outlined earlier.



Stage 2: Annual Customer Activity Growth Analysis

Our initial analysis process involved assessing the annual growth of customer activity from late 2016 to 2018. This analysis was conducted using the following metrics:

1. Monthly Active Users across years
2. New customer growth across years
3. Repeat customer growth across years
4. Average order per customer across years

We calculated and summarized these metrics from late 2016 to 2018 using the following query:

```
-- ================= Yearly Customer Activity Growth Analysis
=======================================
-- In this query, we measure the customer activity growth across the year
```

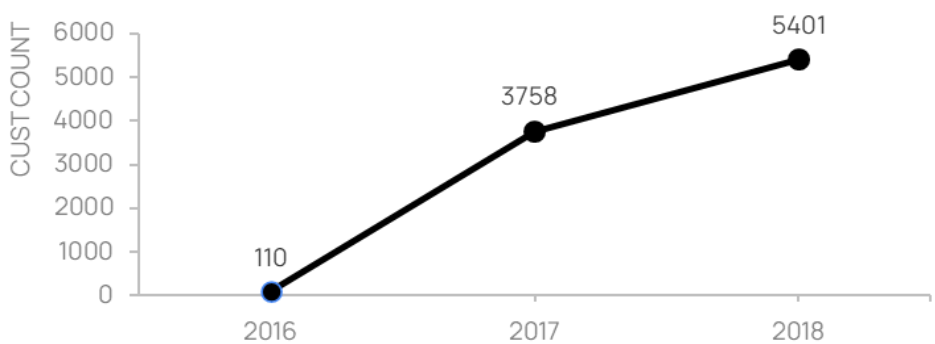
```
using the following metrics:
-- 1. Average active users
-- 2. New customer count
-- 3. Repeat customer count
-- 4. Average orders per user
--
===========================================================================
=======================
WITH yearly_avg_user_active AS (
      -- Average monthly active users across years
      SELECT year,
               ROUND(AVG(user_count)) AS avg_monthly_user
      FROM (
            SELECT EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
                     EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
                     COUNT(DISTINCT customer_id) AS user_count
            FROM orders_dataset
            GROUP BY month, year
            ORDER BY year
            ) AS monthly_count
      GROUP BY year
      ORDER BY year
),
new_cust_count AS (
      -- Count of new customers each year
      SELECT year, COUNT(DISTINCT customer_unique_id) AS new_customers
      FROM (
            SELECT EXTRACT(YEAR FROM ord.order_purchase_timestamp) AS year,
                                 cust.customer_unique_id,
                                 COUNT(DISTINCT ord.order_id) AS
order_count
            FROM orders_dataset AS ord
            INNER JOIN customers_dataset AS cust
            USING(customer_id)
            GROUP BY year, cust.customer_unique_id
            HAVING COUNT(ord.order_id) = 1
            ) AS new_cust
      GROUP BY year
      ORDER BY year
),
repeat_cust_count AS (
      -- Count of repeat customers each year
      SELECT year, COUNT(DISTINCT customer_unique_id) AS repeat_customers
```

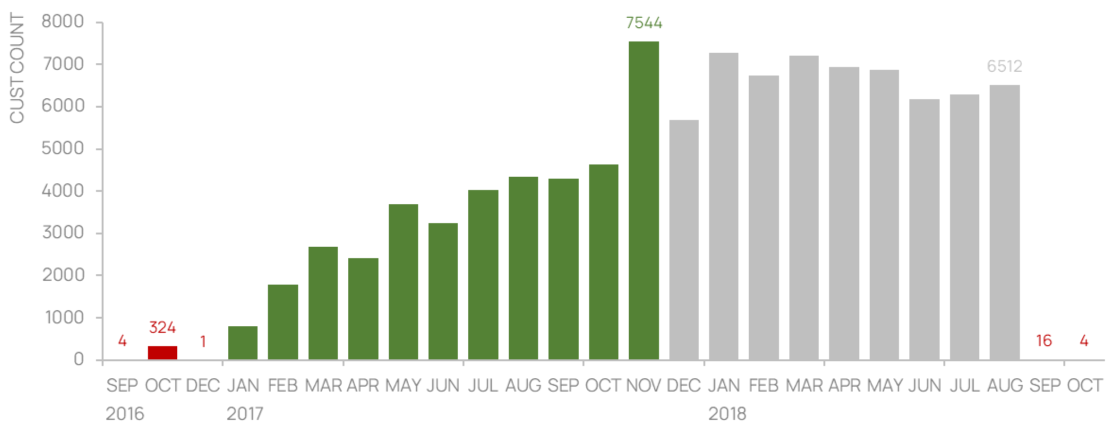
```sql
      FROM (
            SELECT EXTRACT(YEAR FROM ord.order_purchase_timestamp) AS year,
                                 cust.customer_unique_id,
                                 COUNT(DISTINCT ord.order_id) AS
order_count
            FROM orders_dataset AS ord
            INNER JOIN customers_dataset AS cust
            USING(customer_id)
            GROUP BY year, cust.customer_unique_id
            HAVING COUNT(ord.order_id) > 1
            ) AS repeat_cust
      GROUP BY year
      ORDER BY year
),
yearly_avg_order AS (
      -- Average order by customer across years
      SELECT year, ROUND(AVG(order_count), 2) AS avg_order_per_cust
      FROM (
            SELECT EXTRACT(YEAR FROM ord.order_purchase_timestamp) AS year,
                     cust.customer_unique_id,
                     COUNT(DISTINCT ord.order_id) AS order_count
            FROM orders_dataset AS ord
            INNER JOIN customers_dataset AS cust
            USING(customer_id)
            GROUP BY year, cust.customer_unique_id
            ORDER BY order_count DESC
            ) AS monthly_customer_order_count
      GROUP BY year
      ORDER BY year)
SELECT m1.year, m1.avg_monthly_user, m2.new_customers, m3.repeat_customers,
m4.avg_order_per_cust
FROM yearly_avg_user_active AS m1
INNER JOIN new_cust_count AS m2
USING(year)
INNER JOIN repeat_cust_count AS m3
USING(year)
INNER JOIN yearly_avg_order AS m4
USING(year);
```

We got the following table from the above query:

	year numeric	avg_monthly_user numeric	new_customers bigint	repeat_customers bigint	avg_order_per_cust numeric
1	2016	110	323	3	1.01
2	2017	3758	42457	1256	1.03
3	2018	5401	51582	1167	1.02

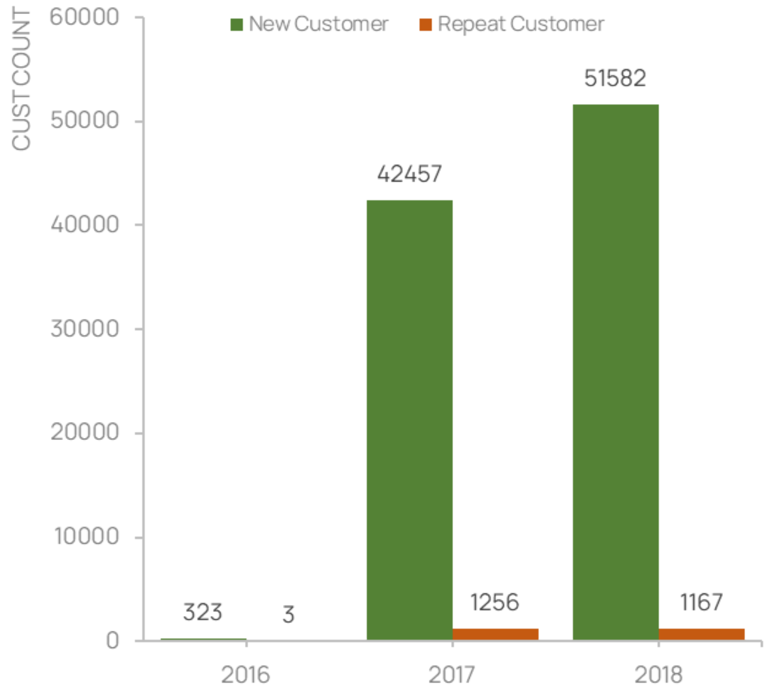
Overall, customer activity shows significant growth towards 2018. We went deeper into these metrics individually using visualizations created in Microsoft Excel. Let's begin by examining the Monthly Active Users across years.





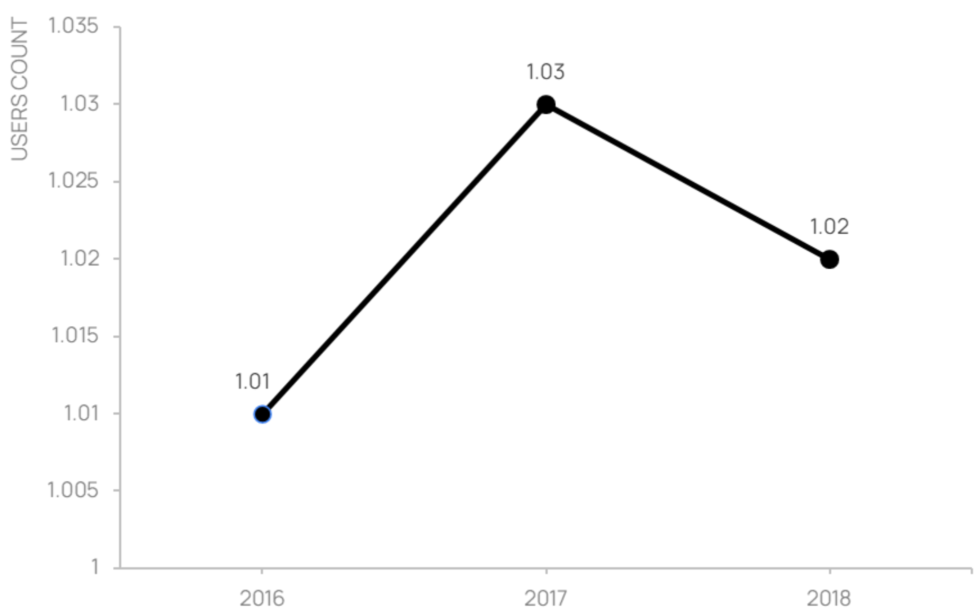
The number of active users went up steadily from 2016 to 2018. But starting in December 2017, the number of active users each month became stagnant and even dropped sharply in September-October 2018.

After that, we checked the frequency of new customers vs. repeat customers across late 2016 to 2018.



The growth in active users from 2016 to 2018 was primarily driven by new customers. However, the number of repeat customers remained relatively flat, despite a notable increase from 2016 to 2017.

Finally, we're going to take a look into the average order per customer across late 2016 to 2018



Even though the number of active users per month has grown significantly, the average order value per user has stayed fairly consistent. This is supported by the growth in e-commerce users mostly due to new customers who only bought something once from the store.

Stage 3: Annual Product Category Quality Analysis

Our second analysis focused on assessing the annual performance of fundamental financial metrics and product categories from late 2016 to 2018. The following metrics were measured in this analysis:

1. Total revenue across years
2. Total cancellations across years
3. Top-selling products across years
4. Top canceled products across years

We calculated and summarized these metrics from late 2016 to 2018 using the following query:

```
-- ================= Product Category Analysis
========================================
-- This query evaluates the performance of products across various
categories over the year, utilizing the following metrics:
-- 1. Revenue generated across years
-- 2. Total number of canceled orders across years
-- 3. Top-selling product for each year
-- 4. Product with the highest total cancellation for each year
--
===========================================================================
=======================
-- Revenue across years
WITH revenue AS (
      SELECT EXTRACT(YEAR FROM od.order_purchase_timestamp) AS year,
             ROUND(SUM(oid.price::numeric + oid.freight_value::numeric),
2) AS revenue
      FROM orders_dataset AS od
      INNER JOIN order_items_dataset AS oid
      USING(order_id)
      WHERE od.order_status = 'delivered'
      GROUP BY 1
),
-- Total canceled order across years
```

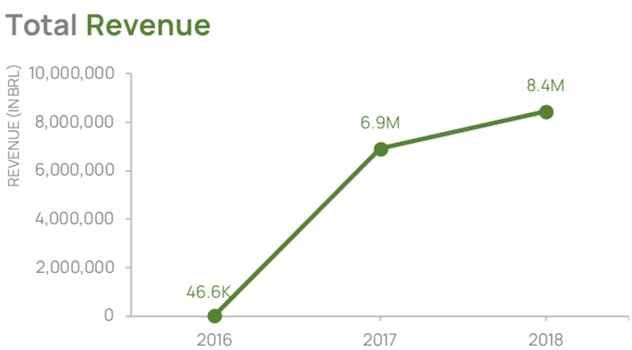
```sql
total_canceled_order AS (
      SELECT EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
               COUNT(DISTINCT order_id) AS canceled_order
      FROM orders_dataset
      WHERE order_status = 'canceled'
      GROUP BY 1
),
-- Top-selling product for each year
top_product AS (
      SELECT year,
               product_category_name,
               total_revenue
      FROM (
            SELECT DENSE_RANK() OVER (PARTITION BY year ORDER BY
total_revenue DESC) AS ranking,
                     year,
                     product_category_name,
                     total_revenue
            FROM (
                  SELECT year,
                           product_category_name,
                           ROUND(SUM(revenue)::numeric, 2) AS total_revenue
                  FROM (
                        SELECT EXTRACT(YEAR FROM order_purchase_timestamp)
AS year,
                                 pd.product_category_name,
                                 oid.price + oid.freight_value AS revenue
                        FROM orders_dataset AS od
                        INNER JOIN order_items_dataset AS oid
                        USING(order_id)
                        INNER JOIN products_dataset AS pd
                        USING(product_id)
                        WHERE od.order_status = 'delivered'
                        ) AS product_sales
                  GROUP BY 1, 2
                  ) AS product_total_revenue
            ) AS product_rank_revenue
      WHERE ranking = 1
),
-- Products with the highest total cancelation across years
top_canceled_product AS (
      SELECT year,
               product_category_name,
```

```
               cancel_count
      FROM(
            SELECT DENSE_RANK() OVER (PARTITION BY year ORDER BY
cancel_count DESC) AS ranking,
                     year,
                     product_category_name,
                     cancel_count
            FROM (
                  SELECT EXTRACT(YEAR FROM od.order_purchase_timestamp) AS
year,
                           pd.product_category_name,
                           COUNT(*) AS cancel_count
                  FROM orders_dataset AS od
                  INNER JOIN order_items_dataset AS oid
                  USING(order_id)
                  INNER JOIN products_dataset AS pd
                  USING(product_id)
                  WHERE od.order_status = 'canceled'
                  GROUP BY 1, 2
                  ) AS canceled_product_count
            ) AS canceled_product_rank
      WHERE ranking = 1
)
-- Concatenating the result
SELECT r.year,
         r.revenue AS total_revenue,
         tco.canceled_order AS total_canceled_order,
         tp.product_category_name AS top_selling_product,
         tp.total_revenue AS top_selling_product_revenue,
         tcp.product_category_name AS top_canceled_product,
         tcp.cancel_count AS top_canceled_product_cancel_count
FROM revenue AS r
INNER JOIN total_canceled_order AS tco
USING(year)
INNER JOIN top_product AS tp
USING(year)
INNER JOIN top_canceled_product AS tcp
USING(year);
```

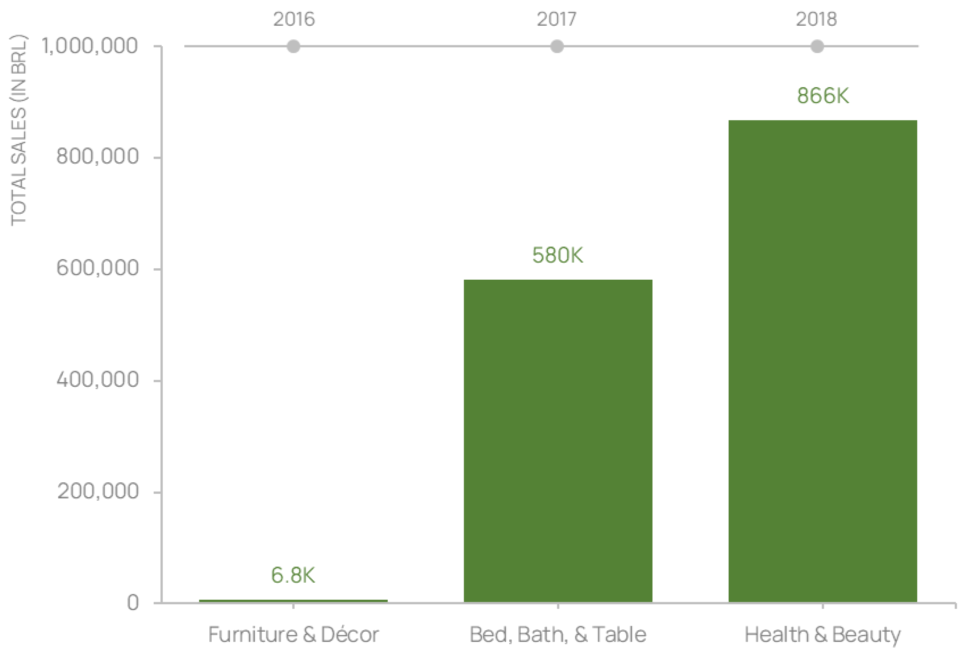
We got the following table from the above query:

	year numeric	total_revenue numeric	total_canceled_order bigint	top_selling_product character varying	top_selling_product_revenue numeric	top_canceled_product character varying	top_canceled_product_cancel_count bigint
1	2016	46653.74	26	furniture_decor	6899.35	toys	3
2	2017	6921535.24	265	bed_bath_table	580949.20	sports_leisure	25
3	2018	8451584.77	334	health_beauty	866810.34	health_beauty	27

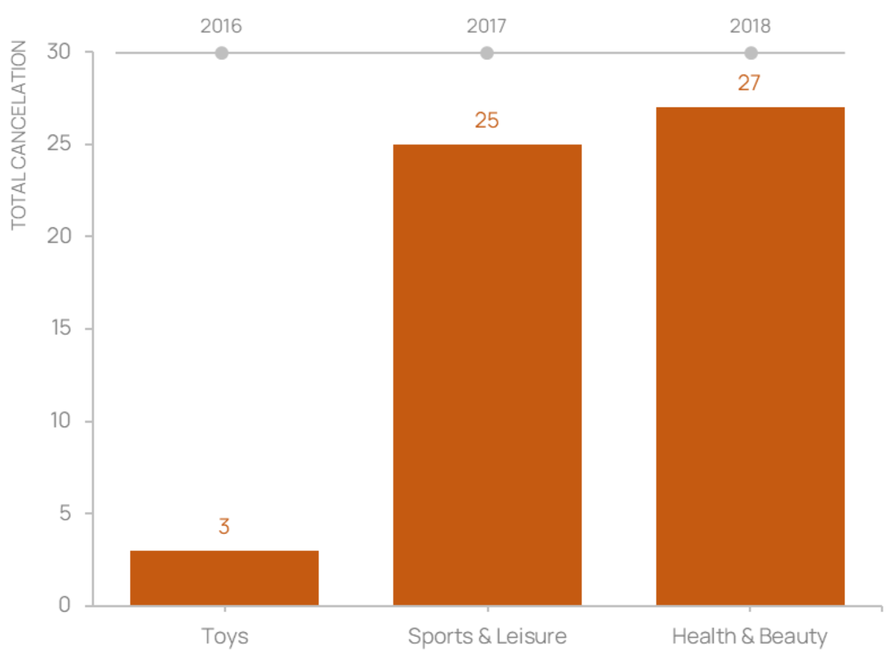
Overall, this e-commerce experienced an increased revenue surge from late 2016 to 2018, suggesting effective marketing strategies. However, this growth was accompanied by a rise in total canceled orders. This analysis also reveals evolving customer preferences, as supported by the annual change in top-selling and top-canceled products. To delve deeper into these trends, let's examine the total revenue and total canceled orders from late 2016 to 2018 using a visualization.



We can roughly see an association between total revenue and total canceled orders. This association, although not a direct cause-and-effect relationship, gives us a reflection of the underlying market dynamics. For example, increased competition and trending products in the e-commerce industry could lead to both higher revenue through price wars and increased canceled orders due to product comparisons and price sensitivity. Now, let’s progress into the next metric: top-selling products across years.



Customer preferences for top-selling product categories shifted notably over time, with Furniture and Home Décor dominating in late 2016, Bedding, Bath, and Tableware taking the lead in 2017, followed by Health and Beauty in 2018. The Health and Beauty category's sales surge in 2018 shows that more businesses are selling products that help people feel well and take care of themselves. This change is because more and more merchants and customers are interested in Health and Beauty. Finally, let's inspect the top canceled products across years.



In late 2016, Toys led the cancellation rate, followed by Sports and Leisure in 2017. By 2018, Health and Beauty had become the category with the highest cancellation rate. 2018 revealed a new trend where the top-selling product experienced the highest cancellation rate. Specifically, Health and Beauty products dominate top sales, but also saw the highest cancellation rates, suggesting increased e-commerce competition and customer price sensitivity in this category.

Stage 4: Annual Payment Type Usage Analysis

Our final analysis focused on assessing the annual and all time usage of payment methods usage from late 2016 to 2018. The following metrics were measured in this analysis:

1. Frequency usage of all payment methods across all time
2. Top used payment methods across years

We calculated and summarized these metrics from late 2016 to 2018 using the following query:

```
-- ================= Payment Type Usage Analysis
=======================================
-- This query evaluates the trend of payment method usage over the year,
utilizing the following metrics:
-- 1. Most used payment method all time
-- 2. Most used payment method across years
--
===========================================================================
======================
-- Top used payment method all time
SELECT payment_type,
         COUNT(*) AS total_used
FROM payments_dataset
WHERE payment_type != 'not_defined'
GROUP BY 1
ORDER BY 2 DESC;

-- Trend of most used payment method across years
SELECT year,
         payment_type,
         total_used
FROM (
      SELECT year,
               payment_type,
               total_used,
               DENSE_RANK() OVER (PARTITION BY year ORDER BY total_used
```

```
DESC) AS ranking
      FROM (
            SELECT EXTRACT(YEAR FROM od.order_purchase_timestamp) AS year,
                     pd.payment_type,
                     COUNT(*) AS total_used
            FROM orders_dataset AS od
            INNER JOIN payments_dataset AS pd
            USING(order_id)
            WHERE payment_type != 'not_defined'
            GROUP BY 1, 2
            ORDER BY 1
            ) AS total_method_used
      ORDER BY 1
      ) AS method_ranked
WHERE ranking = 1;
```

We got the following tables from the above query:

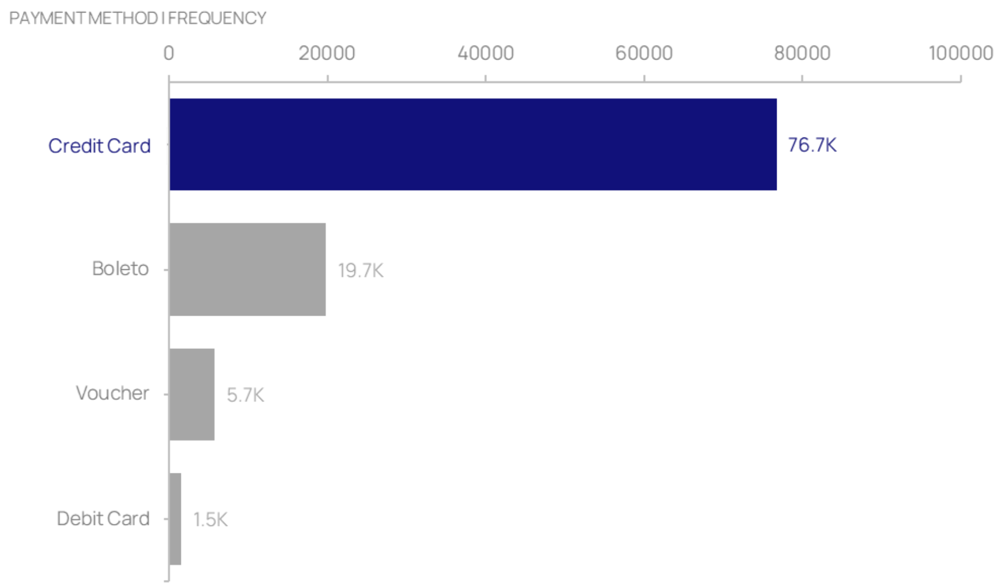
"All-time payment method usage"

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

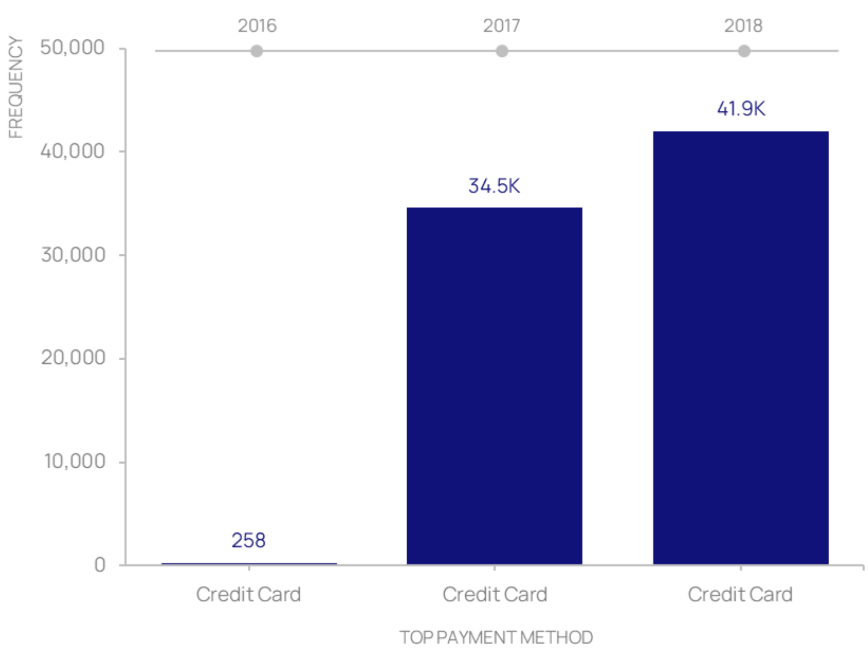
"Top used payment methods across years"

	year numeric	payment_type character varying	total_used bigint
1	2016	credit_card	258
2	2017	credit_card	34568
3	2018	credit_card	41969

Credit cards have consistently been the most popular payment method, both all-time and during the period from late 2016 to 2018. Let's inspect the payment types usage across all-time with a visualization.



Credit cards continue to be the most popular way to pay all-time, with Boleto also being a popular choice. This suggests that credit cards are so widely used because they are convenient and easy to use, with just a few clicks to make a purchase in this e-commerce. There is also a notable observation, where debit cards are significantly less used than other payment methods. We assume that debit cards are directly linked to users' banking accounts, potentially limiting spending flexibility compared to credit cards that offer a line of credit and installments. This could influence consumer choices when making e-commerce purchases. Finally, let's take a look at how payment methods have changed over time.



Credit cards remained the most popular payment method from late 2016 to 2018, confirming their offered convenience in e-commerce transactions. We can also observe that as the number of customers increases from late 2016 to 2018 based on the first analysis, so does the number of orders that use credit cards as the payment method. This also suggests that the customer preference for payment methods from late 2016 to 2018 is steady, where they mainly use credit cards as their way of completing orders.