Analyzing eCommerce Business Performance with SQL

Created by:
Nur Imam Masri
Email : nurimammasri.01@gmail.com
LinkedIn : linkedin.com/in/nurimammasri
Github : github.com/nurimammasri
Portfolio : bit.ly/ImamProjectPortfolio

Overview :
In a company measuring business performance is very important to track, monitor, and assess the success or failure of various business processes. This can help us to see current market conditions, growth analysis, and product analysis, and to develop opportunities for new, more effective business methods. Therefore, this project will analyze the business performance of an eCommerce company, taking into account several business metrics, namely customer growth, product quality, and payment types.

In this project, an analysis will be carried out using PostgreSQL, and visualizing the results using Tableau.

Datasets :
The dataset used is provided by Rakamin Academy, Brazilian E-Commerce Public Dataset by Olist. Or it can be accessed on the kaggle dataset. This is a Brazilian e-commerce public dataset of orders made at the Olist Store. The dataset has information on 100k orders from 2016 to 2018 made at multiple marketplaces in Brazil. Its features allow viewing orders from various dimensions: from order status, price, payment, and freight performance to customer location, product attributes, and finally reviews written by customers. This also released a geolocation dataset that relates Brazilian zip codes to lat/long coordinates.

SQL Query Files :
- 01. Data Preparation.sql
- 02. Annual Customer Activity Growth Analysis.sql
- 03. Annual Product Category Quality Analysis.sql
- 04. Annual Payment Type Usage Analysis.sql

ERD Files :
- ERD Final.pgerd
- ERD Final.png

Tableau Data Visualization : Visualization

Tugas 1
Data Preparation

1. **Create Database**

   Create a database as a repository for datasets

```
CREATE DATABASE ecommerce;
```

2. **Create Tables**

   Create tables for 8 csv datasets and define column names, data types, and primary keys

- **Table product**

```
CREATE TABLE products (
	column1 int4 NULL,
	product_id varchar(50) NOT NULL,
	product_category_name varchar(50) NULL,
	product_name_lenght float8 NULL,
	product_description_lenght float8 NULL,
	product_photos_qty float8 NULL,
	product_weight_g float8 NULL,
	product_length_cm float8 NULL,
	product_height_cm float8 NULL,
	product_width_cm float8 NULL,
	CONSTRAINT products_pk PRIMARY KEY (product_id)
);
```

- **Table order_payment**

```
CREATE TABLE order_payments (
	order_id varchar(50) NULL,
	payment_sequential int4 NULL,
	payment_type varchar(50) NULL,
	payment_installments int4 NULL,
	payment_value float8 NULL
);
```

- **Table order_reviews**

```
CREATE TABLE order_reviews (
	review_id varchar(100) NULL,
	order_id varchar(100) NULL,
```

```
    review_score int4 NULL,
    review_comment_title varchar(100) NULL,
    review_comment_message varchar(400) NULL,
    review_creation_date timestamp NULL,
    review_answer_timestamp timestamp NULL
);
```

- **Table orders**

```
CREATE TABLE orders (
    order_id varchar(50) NOT NULL,
    customer_id varchar(50) NULL,
    order_status varchar(50) NULL,
    order_purchase_timestamp timestamp NULL,
    order_approved_at timestamp NULL,
    order_delivered_carrier_date timestamp NULL,
    order_delivered_customer_date timestamp NULL,
    order_estimated_delivery_date timestamp NULL,
    CONSTRAINT orders_pk PRIMARY KEY (order_id)
);
```

- **Table customers**

```
CREATE TABLE customers (
    customer_id varchar(50) NOT NULL,
    customer_unique_id varchar(50) NULL,
    customer_zip_code_prefix varchar(50) NULL,
    customer_city varchar(50) NULL,
    customer_state varchar(50) NULL,
    CONSTRAINT customers_pk PRIMARY KEY (customer_id)
);
```

- **Table geolocation (dirty)**

```
CREATE TABLE geolocation_dirty (
    geolocation_zip_code_prefix varchar(50) NULL,
    geolocation_lat float8 NULL,
    geolocation_lng float8 NULL,
    geolocation_city varchar(50) NULL,
    geolocation_state varchar(50) NULL
);
```

- **Table seller**

```
CREATE TABLE sellers (
    seller_id varchar(50) NOT NULL,
    seller_zip_code_prefix varchar(50) NULL,
    seller_city varchar(50) NULL,
    seller_state varchar(50) NULL,
    CONSTRAINT sellers_pk PRIMARY KEY (seller_id)
);
```

- **Table order_items**

```
CREATE TABLE order_items (
    order_id varchar(50) NULL,
    order_item_id int4 NULL,
    product_id varchar(50) NULL,
    seller_id varchar(50) NULL,
    shipping_limit_date timestamp NULL,
    price float8 NULL,
    freight_value float8 NULL
);
```

3. **Import Datasets**

After making sure column names and data types match, then we enter the .csv data into each table.

- **Import product dataset**

```
COPY products(
    column1,
    product_id,
    product_category_name,
    product_name_lenght,
    product_description_lenght,
    product_photos_qty,
    product_weight_g,
    product_length_cm,
    product_height_cm,
    product_width_cm
)
FROM
'\Dataset\product_dataset.csv'
DELIMITER ','
CSV HEADER;

ALTER TABLE products DROP COLUMN column1;
```

- **Import order_payments dataset**

```
COPY order_payments(
    order_id,
    payment_sequential,
    payment_type,
    payment_installments,
    payment_value
)
FROM
'\Dataset\order_payments_dataset.csv'
DELIMITER ','
CSV HEADER;
```

- **Import order_reviews dataset**

```
COPY order_reviews(
    review_id,
    order_id,
    review_score,
    review_comment_title,
    review_comment_message,
    review_creation_date,
    review_answer_timestamp
)
FROM
'\Dataset\order_reviews_dataset.csv'
DELIMITER ','
CSV HEADER;
```

- **Import orders dataset**

```
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
'\Dataset\orders_dataset.csv'
DELIMITER ','
CSV HEADER;
```

- **Import customers dataset**

```
COPY customers(
    customer_id,
    customer_unique_id,
    customer_zip_code_prefix,
    customer_city,
    customer_state
)
FROM
'\Dataset\customers_dataset.csv'
DELIMITER ','
CSV HEADER;
```

- **Import sellers dataset**

```
COPY sellers(
    seller_id,
    seller_zip_code_prefix,
    seller_city,
    seller_state
)
FROM
'\Dataset\sellers_dataset.csv'
DELIMITER ','
CSV HEADER;
```

- **Import order_items dataset**

```
COPY order_items(
    order_id,
    order_item_id,
    product_id,
    seller_id,
    shipping_limit_date,
    price,
    freight_value
)
FROM
'\Dataset\order_items_dataset.csv'
DELIMITER ','
CSV HEADER;
```

- **Import geolocation (dirty) dataset**

```
COPY geolocation_dirty(
        geolocation_zip_code_prefix,
        geolocation_lat,
        geolocation_lng,
        geolocation_city,
        geolocation_state
)
FROM
'\Dataset\geolocation_dataset.csv'
DELIMITER ','
CSV HEADER;
```

Specifically for geolocation data, cleaning will be carried out first including :

1. Drop Duplicate Rows

2. Change Special Character in City

3. Input new geolocations from customers and sellers

```
-- ======= geolocation (clean) =======
-- create geolocation clean

CREATE TABLE geolocation_dirty2 AS
SELECT geolocation_zip_code_prefix, geolocation_lat, geolocation_lng,
REPLACE(REPLACE(REPLACE(
TRANSLATE(TRANSLATE(TRANSLATE(TRANSLATE(
TRANSLATE(TRANSLATE(TRANSLATE(TRANSLATE(
geolocation_city, '£,³´.', ''), '`', ''''),
'é,ê', 'e,e'), 'á,â,ã', 'a,a,a'), 'ô,ó,õ', 'o,o,o'),
        'ç', 'c'), 'ú,ü', 'u,u'), 'í', 'i'),
        '4o', '4º'), '* ', ''), '%26apos%3b', ''''
) AS geolocation_city, geolocation_state
from geolocation_dirty gd;
CREATE TABLE geolocation AS
WITH geolocation 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 ROW_NUMBER
```

```
                FROM geolocation_dirty2
        ) TEMP
        WHERE ROW_NUMBER = 1
),
custgeo 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 ROW_NUMBER
                FROM (
                        SELECT customer_zip_code_prefix, geolocation_lat,
                        geolocation_lng, customer_city, customer_state
                        FROM customers cd
                        LEFT JOIN geolocation_dirty gdd
                        ON customer_city = geolocation_city
                        AND customer_state = geolocation_state
                        WHERE customer_zip_code_prefix NOT IN (
                                SELECT geolocation_zip_code_prefix
                                FROM geolocation gd
                        )
                ) geo
        ) TEMP
        WHERE ROW_NUMBER = 1
),
sellgeo 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 ROW_NUMBER
                FROM (
                        SELECT seller_zip_code_prefix, geolocation_lat,
                        geolocation_lng, seller_city, seller_state
                        FROM sellers cd
                        LEFT JOIN geolocation_dirty gdd
                        ON seller_city = geolocation_city
                        AND seller_state = geolocation_state
                        WHERE seller_zip_code_prefix NOT IN (
                                SELECT geolocation_zip_code_prefix
```

```
                                    FROM geolocation gd
                                    UNION
                                    SELECT customer_zip_code_prefix
                                    FROM custgeo cd
                                )
                        ) geo
                ) TEMP
                WHERE ROW_NUMBER = 1
        )
        SELECT *
        FROM geolocation
        UNION
        SELECT *
        FROM custgeo
        UNION
        SELECT *
        FROM sellgeo;

        ALTER TABLE geolocation ADD CONSTRAINT geolocation_pk PRIMARY KEY (geolocation_zip_code_prefix);
```

4. **Entity Relationship (Constraint & Foreign Key)**
   First of all, we will define the relationship between columns based on primary key to foreign key by storing them

   - **products → order_items**

     ```
     ALTER TABLE order_items
     ADD CONSTRAINT order_items_fk_product
     FOREIGN KEY (product_id) REFERENCES products(product_id)
     ON DELETE CASCADE ON UPDATE CASCADE;
     ```

   - **sellers → order_items**

     ```
     ALTER TABLE order_items
     ADD CONSTRAINT order_items_fk_seller
     FOREIGN KEY (seller_id) REFERENCES sellers(seller_id)
     ON DELETE CASCADE ON UPDATE CASCADE;
     ```

   - **orders → order_items**

     ```
     ALTER TABLE order_items
     ADD CONSTRAINT order_items_fk_order
     ```

```
FOREIGN KEY (order_id) REFERENCES orders(order_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

- **orders → order_payments**

```
ALTER TABLE order_payments
ADD CONSTRAINT order_payments_fk
FOREIGN KEY (order_id) REFERENCES orders(order_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

- **orders → order_reviews**

```
ALTER TABLE order_reviews
ADD CONSTRAINT order_reviews_fk
FOREIGN KEY (order_id) REFERENCES orders(order_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

- **customers → orders**

```
ALTER TABLE orders
ADD CONSTRAINT orders_fk
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

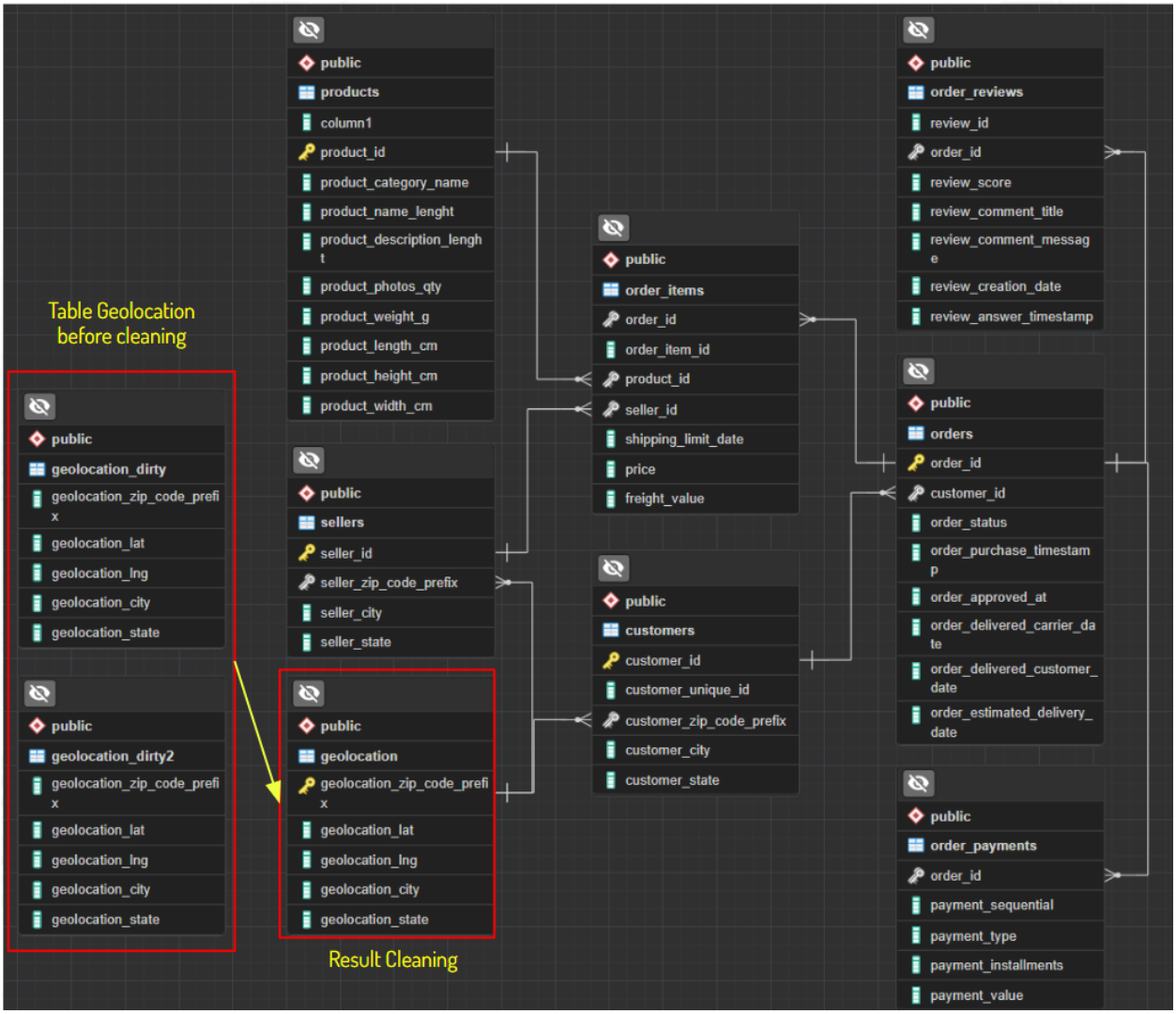
- **geolocation -> customers**

```
ALTER TABLE customers
ADD CONSTRAINT customers_fk
FOREIGN KEY (customer_zip_code_prefix) REFERENCES geolocation(geolocation_zip_code_prefix)
ON DELETE CASCADE ON UPDATE CASCADE;
```

- **geolocation → sellers**

```
ALTER TABLE sellers
ADD CONSTRAINT sellers_fk
FOREIGN KEY (seller_zip_code_prefix) REFERENCES geolocation(geolocation_zip_code_prefix)
ON DELETE CASCADE ON UPDATE CASCADE;
```

So that the Generate ERD Diagram results are as follows:



Tugas 2
Annual Customer Activity Growth Analysis

1. **Average Monthly Active User (MAU) per year**

   Displays the average number of monthly active users (monthly active users) for each year

   - Create a subquery that shows the number of customers in each month in each year
   - Perform the aggregation function to get the average monthly customer per year

```
SELECT
        year,
        floor(avg(n_customers)) AS avg_monthly_active_user
FROM (
        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 n_customers
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1,2
) monthly
GROUP BY 1
ORDER BY 1;
```

year double precision	avg_monthly_active_user numeric
2016	108
2017	3694
2018	5338

2. **Total new customers per year**

   Displays the number of new customers in each year

   - Create a subquery that shows first date orders from each customer
   - Perform the aggregation function to get the number of new customers per year

```
SELECT
        date_part('year', first_date_order) AS year,
        count(customer_unique_id) AS new_customers
FROM (
        SELECT
                c.customer_unique_id,
                min(o.order_purchase_timestamp) AS first_date_order
```

```
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1
) first_order
GROUP BY 1
ORDER BY 1;
```

year double precision	new_customers bigint
2016	326
2017	43708
2018	52062

3. **The number of customers who make repeat orders per year**

   Displays the number of customers who make purchases more than once (repeat orders) in each year

   - Create a subquery that shows the number of orders for each customer
   - Do a filter on customers who have the number of orders > 1
   - Perform the aggregation function to get the number of customers who make repeat orders

```
SELECT
        year,
        count(DISTINCT customer_unique_id) AS repeat_customers
FROM (
        SELECT
                date_part('year', o.order_purchase_timestamp) AS year,
                c.customer_unique_id,
                count(c.customer_unique_id) AS n_customer,
                count(o.order_id) AS n_order
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1,2
        HAVING count(o.order_id) > 1
) repeat_order
GROUP BY 1
ORDER BY 1;
```

year double precision	repeat_customers bigint
2016	3
2017	1256
2018	1167

4. **Average order frequency for each year**

   Displays the average number of orders made by customers for each year

   - Create a subquery that shows the number of orders for each customer
   - Perform aggregation function to get average orders per year

```
SELECT
        year,
        round(avg(n_order), 2) AS avg_num_orders
FROM (
        SELECT
                date_part('year', o.order_purchase_timestamp) AS year,
                c.customer_unique_id,
                count(c.customer_unique_id) AS n_customer,
                count(o.order_id) AS n_order
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1,2
) order_customer
GROUP BY 1
ORDER BY 1;
```

year double precision	avg_num_orders numeric
2016	1.01
2017	1.03
2018	1.02

5. **Master table which contains all the above information**

   Combines the four metrics that have been successfully displayed into one table view

   - Create a temporary table for the previous subtasks first

```
WITH tbl_mau AS (
        SELECT
```

```
        year,
        floor(avg(n_customers)) AS avg_monthly_active_user
    FROM (
        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 n_customers
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1,2
    ) monthly
    GROUP BY 1
),
tbl_newcust AS (
    SELECT
        date_part('year', first_date_order) AS year,
        count(customer_unique_id) AS new_customers
    FROM (
        SELECT
            c.customer_unique_id,
            min(o.order_purchase_timestamp) AS first_date_order
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1
    ) first_order
    GROUP BY 1
),
tbl_repcust AS (
    SELECT
        year,
        count(DISTINCT customer_unique_id) AS repeat_customers
    FROM (
        SELECT
            date_part('year', o.order_purchase_timestamp) AS year,
            c.customer_unique_id,
            count(c.customer_unique_id) AS n_customer,
            count(o.order_id) AS n_order
        FROM orders o
        JOIN customers c
        ON o.customer_id = c.customer_id
        GROUP BY 1,2
        HAVING count(o.order_id) > 1
```

```
        ) repeat_order
        GROUP BY 1
),
tbl_avgorder AS (
        SELECT
                year,
                round(avg(n_order), 2) AS avg_num_orders
        FROM (
                SELECT
                        date_part('year', o.order_purchase_timestamp) AS year,
                        c.customer_unique_id,
                        count(c.customer_unique_id) AS n_customer,
                        count(o.order_id) AS n_order
                FROM orders o
                JOIN customers c
                ON o.customer_id = c.customer_id
                GROUP BY 1,2
        ) order_customer
        GROUP BY 1
)
SELECT
        tm.year,
        tm.avg_monthly_active_user,
        tn.new_customers,
        tr.repeat_customers,
        ta.avg_num_orders
FROM tbl_mau tm
JOIN tbl_newcust tn
ON tm.year = tn.year
JOIN tbl_repcust tr
ON tm.year = tr.year
JOIN tbl_avgorder ta
ON tm.year = ta.year
ORDER BY 1;
```

year double precision	avg_monthly_active_user numeric	new_customers bigint	repeat_customers bigint	avg_num_orders numeric
2016	108	326	3	1.01
2017	3694	43708	1256	1.03
2018	5338	52062	1167	1.02

Tugas 3
Annual Product Category Quality Analysis

Reset Table

To ensure the results do not increase or differ from the initial input

```
DROP TABLE IF EXISTS total_revenue_year;
DROP TABLE IF EXISTS total_canceled_orders_year;
DROP TABLE IF EXISTS top_product_category_revenue_year;
DROP TABLE IF EXISTS top_product_category_canceled_year;
```

1. Table of revenue per year

Create a table that contains total company revenue information for each year

- Create CTE revenue_orders to store the amount of revenue for each order, this is to reduce computation on joins
- Revenue is obtained by adding up the price and freight value
- Using the aggregation function to add up the amount of revenue from orders each year
- Setting filter = 'delivered' because it is necessary to approve that the transaction is fully completed

```
CREATE TABLE total_revenue_year AS
WITH revenue_orders AS (
        SELECT
                order_id,
                sum(price + freight_value) AS revenue
        FROM order_items oi
        GROUP BY 1
)
SELECT
        date_part('year', o.order_purchase_timestamp) AS year,
        sum(po.revenue) AS revenue
FROM orders o
JOIN revenue_orders po
ON o.order_id = po.order_id
WHERE o.order_status = 'delivered'
GROUP BY 1
ORDER BY 1;
```

year double precision	revenue double precision
2016	46653.74000000004
2017	6921535.239999719
2018	8451584.769999959

2. **Table of the number of canceled orders per year**

Create a table that contains information on the total number of cancel orders for each year

- Using the aggregation function to add up the number of orders from orders each year
- Setting filter = 'canceled' because it is necessary to approve that the transaction is canceled

```
CREATE TABLE total_canceled_orders_year AS
SELECT
        date_part('year', order_purchase_timestamp) AS year,
        count(order_id) AS total_cancel
FROM orders o
WHERE order_status = 'canceled'
GROUP BY 1
ORDER BY 1;
```

year double precision	total_cancel bigint
2016	26
2017	265
2018	334

3. **Table of the top categories that generate the largest revenue per year**

Create a table containing the product category names that provide the highest total revenue for each year

- Create CTE for total revenue for each product category each year.
- Using the window function ROW_NUMBER() to rank revenue
- Setting filter rank = 1 to get the top product category each year

```
CREATE TABLE top_product_category_revenue_year AS
WITH revenue_category_orders AS (
        SELECT
                date_part('year', o.order_purchase_timestamp) AS year,
                p.product_category_name,
                sum(oi.price + oi.freight_value) AS revenue,
                ROW_NUMBER() OVER(
                        PARTITION BY date_part('year', o.order_purchase_timestamp)
                        ORDER BY sum(oi.price + oi.freight_value) desc
                ) AS rank
        FROM orders o
        JOIN order_items oi
        ON o.order_id = oi.order_id
        JOIN products p
        ON oi.product_id = p.product_id
```

```
        WHERE order_status = 'delivered'
        GROUP BY 1, 2
)
SELECT
        year,
        product_category_name,
        revenue
FROM revenue_category_orders
WHERE rank = 1;
```

year double precision	product_category_name character varying (50)	revenue double precision
2016	furniture_decor	6899.349999999999
2017	bed_bath_table	580949.2000000025
2018	health_beauty	866810.3399999987

4. **Table of categories that experienced the most canceled orders per year**

   Create a table containing the names of the product categories that have the highest number of canceled orders for each year

   - Create a CTE for the number of orders canceled for each product category each year.
   - Uses the window function ROW_NUMBER() to rank the number of canceled numbers
   - Setting filter rank = 1 to get the top cancel product category each year

```
CREATE TABLE top_product_category_canceled_year AS
WITH canceled_category_orders AS (
        SELECT
                date_part('year', o.order_purchase_timestamp) AS year,
                p.product_category_name,
                count(*) AS total_cancel,
                ROW_NUMBER() OVER(
                        PARTITION BY date_part('year', o.order_purchase_timestamp)
                        ORDER BY count(*) desc
                ) AS rank
        FROM orders o
        JOIN order_items oi
        ON o.order_id = oi.order_id
        JOIN products p
        ON oi.product_id = p.product_id
        WHERE order_status = 'canceled'
        GROUP BY 1, 2
)
```

```
SELECT
        year,
        product_category_name,
        total_cancel
FROM canceled_category_orders
WHERE rank = 1;
```

year double precision	product_category_name character varying (50)	total_cancel bigint
2016	toys	3
2017	sports_leisure	25
2018	health_beauty	27

5. **Master table containing the above information**

Combine the information that has been obtained into a single table view

```
SELECT
    tpr.year,
    tpr.product_category_name AS top_product_category_revenue,
    tpr.revenue AS top_category_revenue,
    try.revenue AS total_revenue_year,
    tpc.product_category_name AS top_product_category_canceled,
    tpc.total_cancel AS top_category_num_canceled,
    tco.total_cancel AS total_canceled_orders_year
FROM top_product_category_revenue_year tpr
JOIN total_revenue_year try
ON tpr.year = try.year
JOIN top_product_category_canceled_year tpc
ON tpr.year = tpc.year
JOIN total_canceled_orders_year tco
ON tpr.year = tco.YEAR;
```

year double precision	top_product_category_revenue character varying (50)	top_category_revenue double precision	total_revenue_year double precision
2016	furniture_decor	6899.35	46653.74000000004
2017	bed_bath_table	580949.200000002	6921535.239999719
2018	health_beauty	866810.3399999987	8451584.769999959

top_product_category_canceled character varying (50)	top_category_num_canceled bigint	total_canceled_orders_year bigint
toys	3	26
sports_leisure	25	265
health_beauty	27	334

Tugas 4
Annual Payment Type Usage Analysis

1. **Total usage of each type of payment all time**

   Displays the total usage of each type of payment at all time, sorted from the favorite

   - Create a table that shows the amount of payment usage for each type of payment for all time
   - Sort data number of usage from the largest to the smallest payment type

```
SELECT
        op.payment_type,
        count(*) AS num_of_usage
FROM orders o
JOIN order_payments op
ON o.order_id = op.order_id
GROUP BY 1
ORDER BY 2 DESC;
```

payment_type character varying (50)	num_of_usage bigint
credit_card	76795
boleto	19784
voucher	5775
debit_card	1529
not_defined	3

2. **Details of the amount used for each type of payment for each year**

   Displays detailed information on the amount of usage for each type of payment for each year

   - Create a table that shows the amount of payment usage for each type of payment for each year
   - Sort data number of usage from the largest to the smallest payment type

```
SELECT
        date_part('year', o.order_purchase_timestamp) as year,
        op.payment_type,
        count(*) AS num_of_usage
FROM orders o
JOIN order_payments op
ON o.order_id = op.order_id
GROUP BY 1,2
ORDER BY 1 ASC, 3 DESC;
```

year double precision	payment_type character varying (50)	num_of_usage bigint
2016	credit_card	258
2016	boleto	63
2016	voucher	23
2016	debit_card	2
2017	credit_card	34568
2017	boleto	9508
2017	voucher	3027
2017	debit_card	422
2018	credit_card	41969
2018	boleto	10213
2018	voucher	2725
2018	debit_card	1105
2018	not_defined	3

3. **One summary table of the number of payment types used for each year.**
   - Utilize the CASE WHEN function to pivot data (rows : payment_type, columns : year)
   - Combining CASE WHEN into an aggregate function with year value windowing for dividing values in the column
   - Sort data number of usage from the largest to the smallest payment type

   - *Method 1*

```
WITH type_payments AS (
        SELECT
                date_part('year', o.order_purchase_timestamp) as year,
                op.payment_type,
                count(*) AS num_of_usage
        FROM orders o
        JOIN order_payments op
        ON o.order_id = op.order_id
        GROUP BY 1,2
)
select
payment_type,
sum(case when year = '2016' then num_of_usage else 0 end) as year_2016,
sum(case when year = '2017' then num_of_usage else 0 end) as year_2017,
sum(case when year = '2018' then num_of_usage else 0 end) as year_2018
from type_payments
GROUP BY 1
ORDER BY 4 DESC;
```

- *Method 1*

```
SELECT
        payment_type,
        count(CASE WHEN date_part('year', order_purchase_timestamp) = '2016' THEN o.order_id END) AS year_2016,
        count(CASE WHEN date_part('year', order_purchase_timestamp) = '2017' THEN o.order_id END) AS year_2017,
        count(CASE WHEN date_part('year', order_purchase_timestamp) = '2018' THEN o.order_id END) AS year_2018
FROM orders o
JOIN order_payments op
ON o.order_id = op.order_id
GROUP BY 1
ORDER BY 4 DESC;
```

payment_type character varying (50)	year_2016 bigint	year_2017 bigint	year_2018 bigint
credit_card	258	34568	41969
boleto	63	9508	10213
voucher	23	3027	2725
debit_card	2	422	1105
not_defined	0	0	3

========== END ==========