Mini Project 1

Analyzing eCommerce Business Performance with SQL

1. Data Preparation

Langkah-langkah:

1. **Download file "Dataset.rar", kemudian ekstrak file tersebut.**
2. **Membuat database baru beserta tabel-tabelnya untuk data yang sudah disiapkan**

   query :

```
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)
);

-- ======= 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
);

-- ======= 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
);

-- ======= 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)
);

-- ======= 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)
);

-- ======= 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
```

```
);

-- ======= 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)
);

-- ======= 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. **Importing Data CSV ke dalam database dengan SQL Shell**

   query:

```
\COPY order_items FROM 'order_items_dataset.csv' WITH (FORMAT CSV, HEADER);
\COPY products FROM 'product_dataset.csv' WITH (FORMAT CSV, HEADER);
 ALTER TABLE products DROP COLUMN column1;
\COPY order_items FROM 'customers_dataset.csv' WITH (FORMAT CSV, HEADER);
\COPY order_items FROM 'orders_dataset.csv' WITH (FORMAT CSV, HEADER);
\COPY order_items FROM 'order_payment_dataset.csv' WITH (FORMAT CSV, HEADER);
\COPY order_items FROM 'orders_reviews.csv' WITH (FORMAT CSV, HEADER);
\COPY order_items FROM 'sellers_dataset.csv' WITH (FORMAT CSV, HEADER);
\COPY order_items FROM 'geolocation_dataset.csv' WITH (FORMAT CSV, HEADER);
```

4. **Membuat Tabel Geolocation yang sudah di cleaning**

```
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, ' , , ,.', ''), '`', "''"),
  ' , ', 'e,e'), ' , , ', 'a,a,a'), ' , , ', 'o,o,o'),
      ' ', 'c'), ' , ', 'u,u'), ' ', 'i'),
      '4o', '4 '), '* ', ''), '%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);
```

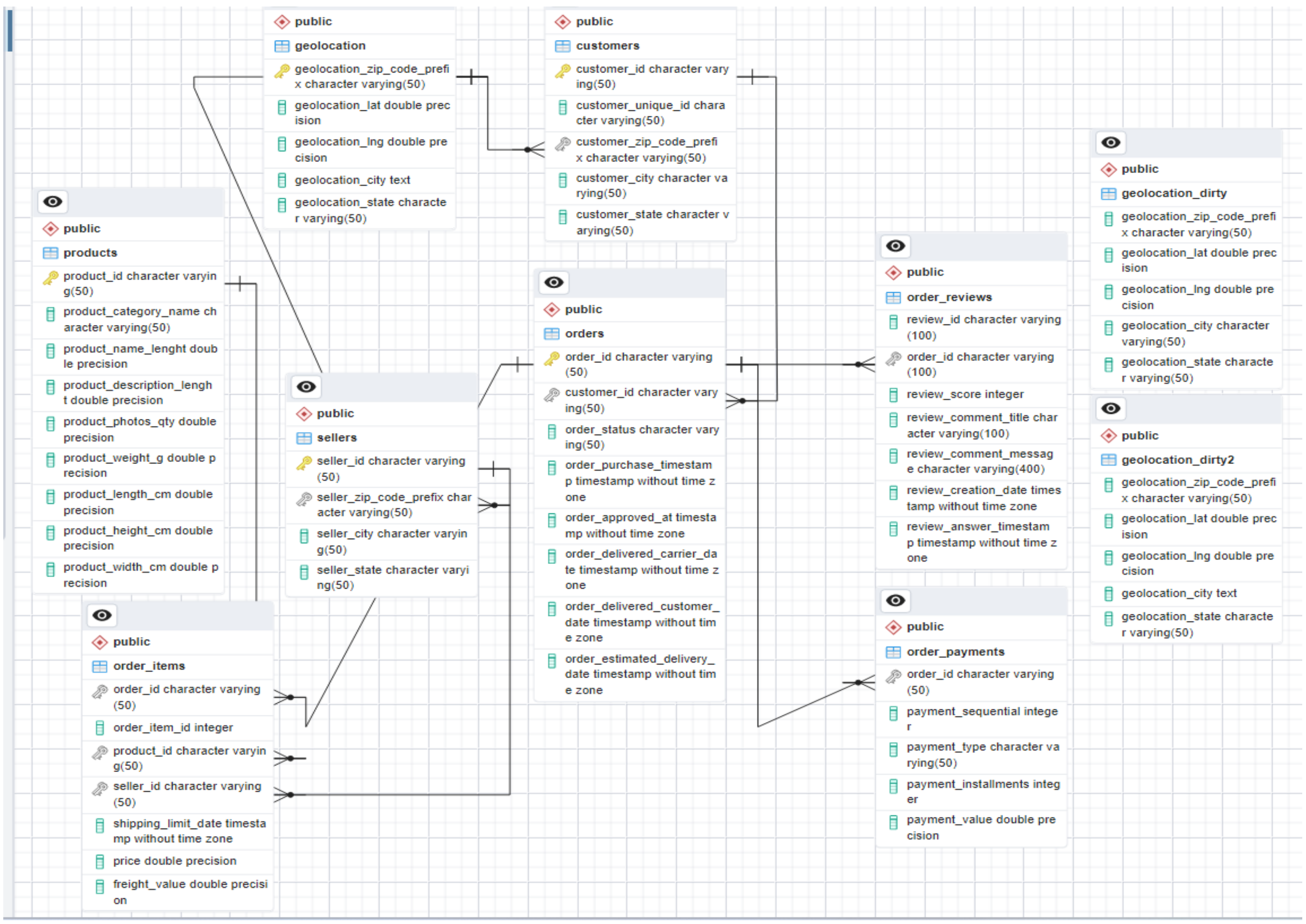
5. **Membuat Entity Relationship Diagram antar Tabel**

```
-- 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;
```

Gambar ERD:



public
geolocation
geolocation_zip_code_prefix character varying(50)
geolocation_lat double precision
geolocation_lng double precision
geolocation_city text
geolocation_state character varying(50)
public
customers
customer_id character varying(50)
customer_unique_id character varying(50)
customer_zip_code_prefix character varying(50)
customer_city character varying(50)
customer_state character varying(50)
public
geolocation_dirty
geolocation_zip_code_prefix character varying(50)
geolocation_lat double precision
geolocation_lng double precision
geolocation_city character varying(50)
geolocation_state character varying(50)
public
products
product_id character varying(50)
product_category_name character varying(50)
product_name_lenght double precision
product_description_lenght double precision
product_photos_qty double precision
product_weight_g double precision
product_length_cm double precision
product_height_cm double precision
product_width_cm double precision
public
orders
order_id character varying (50)
customer_id character varying(50)
order_status character varying(50)
order_purchase_timestamp timestamp without time zone
order_approved_at timestamp without time zone
order_delivered_carrier_date timestamp without time zone
order_delivered_customer_date timestamp without time zone
order_estimated_delivery_date timestamp without time zone
public
order_reviews
review_id character varying (100)
order_id character varying (100)
review_score integer
review_comment_title character varying(100)
review_comment_message character varying(400)
review_creation_date timestamp without time zone
review_answer_timestamp timestamp without time zone
public
sellers
seller_id character varying (50)
seller_zip_code_prefix character varying(50)
seller_city character varying(50)
seller_state character varying(50)
public
geolocation_dirty2
geolocation_zip_code_prefix character varying(50)
geolocation_lat double precision
geolocation_lng double precision
geolocation_city text
geolocation_state character varying(50)
public
order_items
order_id character varying (50)
order_item_id integer
product_id character varying(50)
seller_id character varying (50)
shipping_limit_date timestamp without time zone
price double precision
freight_value double precision
public
order_payments
order_id character varying (50)
payment_sequential integer
payment_type character varying(50)
payment_installments integer
payment_value double precision

2. Annual Customer Activity Growth Analysis

STEP 1

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

query:

```
SELECT
        year,
        floor(avg(n_customers)) AS avg_monthly_active_user
FROM (
        SELECT
        date_part('year',order_purchase_timestamp) AS year,
        date_part('month',order_purchase_timestamp) AS month,
        count(DISTINCT 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;
```

STEP 2

2. **Menampilkan jumlah customer baru pada masing-masing tahun**

query:

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

STEP 3

3. **Menampilkan jumlah customer yang melakukan pembelian lebih dari satu kali (repeat order) pada masing-masing tahun**

query:

```
SELECT
	year,
	count(DISTINCT customer_unique_id) AS customers_repeat
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
) order_repeat
GROUP BY 1
ORDER BY 1;
```

STEP 4

4. **Menampilkan rata-rata jumlah order yang dilakukan customer untuk masing-masing tahun**

query:

```
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;
```

STEP 5

5. **Menggabungkan ketiga metrik yang telah berhasil ditampilkan menjadi satu tampilan tabel**

query:

```
WITH table_mau AS (
        SELECT
                year,
                floor(avg(n_customers)) AS avg_monthly_active_user
        FROM (
                SELECT
                        date_part('year',order_purchase_timestamp) AS year,
                        date_part('month',order_purchase_timestamp) AS month,
                        count(DISTINCT 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
),
table_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(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
),
table_cust_repeat AS (
        SELECT
                year,
                count(DISTINCT customer_unique_id) AS customers_repeat
        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
        ) order_repeat
        GROUP BY 1
        ORDER BY 1
),
table_avg_order 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
        ORDER BY 1
)
SELECT
        tm.year,
        avg_monthly_active_user,
        new_customers,
        customers_repeat,
        avg_num_orders
FROM table_mau tm
JOIN table_newcust tn
ON tm.year = tn.year
JOIN table_cust_repeat tr
ON tm.year = tr.year
JOIN table_avg_order ta
ON tm.year = ta.year
ORDER BY 1;
```

3. Annual Product Category Quality Analysis

STEP 1

1. **Membuat tabel yang berisi informasi pendapatan/revenue perusahaan total untuk masing-masing tahun**

   query:

```
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(ro.revenue)AS revenue
        FROM orders o
        JOIN revenue_orders ro
        ON o.order_id = ro.order_id
        WHERE o.order_status = 'delivered'
        GROUP BY 1
        ORDER BY 1;
```

STEP 2

2. **Membuat tabel yang berisi informasi jumlah cancel order total untuk masing-masing tahun**

   query :

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

STEP 3

3. **Membuat tabel yang berisi nama kategori produk yang memberikan pendapatan total tertinggi untuk masing-masing tahun**

   query :

```
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(price + freight_value) AS revenue,
                ROW_NUMBER() OVER(
                    PARTITION BY date_part('year', o.order_purchase_timestamp)
                   ORDER BY sum(price + 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;
```

STEP 4

4. **Membuat tabel yang berisi nama kategori produk yang memiliki jumlah cancel order terbanyak untuk masing-masing tahun**

   query :

```
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_canceled,
                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_canceled
FROM canceled_category_orders
WHERE rank = 1;
```

STEP 5

5. **Menggabungkan informasi-informasi yang telah didapatkan ke dalam satu tampilan tabel**
   query :

```
        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_canceled AS top_category_canceled,
                tco.total_canceled 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
```

4. Analysis of Annual Payment Type Usage

STEP 1

1. **Menampilkan tabel yang berisi informasi jenis tipe pembayaran yang digunakan dalam pesanan beserta jumlah penggunaan (jumlah pesanan) untuk masing-masing jenis pembayaran**

   query:

```
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
```

STEP 2

2. **Menampilkan tabel yang berisi informasi daftar tahun, jenis tipe pembayaran, dan jumlah penggunaan tiap jenis tipe pembayaran dalam setiap tahun, yang diurutkan berdasarkan tahun secara naik dan jumlah penggunaan secara menurun.**

   query:

```
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;
```

STEP 3

3. **Menampilkan tabel yang berisi informasi jenis tipe pembayaran beserta jumlah pesanan pada tahun 2016, 2017, dan 2018, yang diurutkan berdasarkan jumlah pesanan pada tahun 2018 secara menurun (diurutkan dari yang terfavorit).**

   query:

```
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;
```