TASK 1

Subtask 1:
Membuat database baru beserta tabel-tabelnya untuk data yang telah disiapkan

Terdapat 8 dataset berekstensi csv, sehingga kita akan membuat 8 tabel untuk menyimpan dataset serta sesuaikan tipe data untuk setiap kolom berdasarkan tipe data pada dataset di file csv.

```
CREATE TABLE customers(
	customer_id varchar,
	customer_unique_id varchar,
	customer_zip_code_prefix int,
	customer_city varchar,
	customer_state varchar
);

CREATE TABLE geolocation (
	geo_zip_code_prefix varchar,
	geo_lat varchar,
	geo_lng varchar,
	geo_city varchar,
	geo_state varchar
);

CREATE TABLE payments (
	order_id varchar,
	payment_sequential int,
	payment_type varchar,
	payment_installment int,
	payment_value float
);

CREATE TABLE order_item (
	order_id varchar,
	order_item_id int,
	product_id varchar,
	seller_id varchar,
	shipping_limit_date timestamp,
	price float,
	freight_value float
);

CREATE TABLE reviews (
	review_id varchar,
	order_id varchar,
	review_score int,
	review_comment_title varchar,
	review_comment_message text,
	review_creation_date timestamp,
	review_answer timestamp
);
```

```
CREATE TABLE products (
	nomor varchar,
	product_id varchar,
	product_category_name varchar,
	product_name_length numeric,
	product_description_length numeric,
	product_photos_qty numeric,
	product_weight_g numeric,
	product_length_cm numeric,
	product_height_cm numeric,
	product_width_cm numeric
);

CREATE TABLE orders (
	order_id varchar,
	customers_id varchar,
	order_status varchar,
	order_purchase_timestamp timestamp,
	order_approved_at timestamp,
	order_delivered_carrier_date timestamp,
	order_delivered_customer_date timestamp,
	order_estimated_delivered_date timestamp
);

CREATE TABLE sellers (
	seller_id varchar,
	seller_zip_code int,
	seller_city varchar,
	seller_state varchar
);
```

Subtask 2:
Importing data csv ke dalam database

Ketika melakukan pengimporan data csv ke database, tipe data dalam .csv harus sama dengan tipe data dari kolom database.

```
COPY orders (
	order_id,
	customers_id,
	order_status,
	order_purchase_timestamp,
	order_approved_at,
	order_delivered_carrier_date,
	order_delivered_customer_date,
	order_estimated_delivered_date
)
FROM 'I:\01. Project\Ecommerce\Dataset\orders_dataset.csv'
delimiter ','
csv header
```

```
;

COPY customers (
       customer_id,
       customer_unique_id,
       customer_zip_code_prefix,
       customer_city,
       customer_state
)
FROM 'I:\01. Project\Ecommerce\Dataset\customers_dataset.csv'
delimiter ','
csv header
;

COPY geolocation (
geo_zip_code_prefix,
       geo_lat,
       geo_lng,
       geo_city,
       geo_state)
FROM 'I:\01. Project\Ecommerce\Dataset\geolocation_dataset.csv'
delimiter ','
csv header
;

COPY order_item (
       order_id,
       order_item_id,
       product_id,
       seller_id,
       shipping_limit_date,
       price,
       freight_value)
FROM 'I:\01. Project\Ecommerce\Dataset\order_items_dataset.csv'
delimiter ','
csv header
;

COPY payments (
       order_id,
       payment_sequential,
       payment_type,
       payment_installment,
       payment_value)
FROM 'I:\01. Project\Ecommerce\Dataset\order_payments_dataset.csv'
delimiter ','
csv header
;

COPY products (
       nomor,
       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 'I:\01. Project\Ecommerce\Dataset\product_dataset.csv'
delimiter ','
csv header
;

COPY reviews (
       review_id,
       order_id,
       review_score,
       review_comment_title,
       review_comment_message,
       review_creation_date,
       review_answer)
FROM 'I:\01. Project\Ecommerce\Dataset\order_reviews_dataset.csv'
delimiter ','
csv header
;

COPY sellers (
       seller_id,
       seller_zip_code,
       seller_city,
       seller_state)
FROM 'I:\01. Project\Ecommerce\Dataset\sellers_dataset.csv'
delimiter ','
csv header
;
```

Subtask 3:
Membuat entity relationship antar tabel berdasarkan skema yang telah diberikan sebelumnya.

Berdasarkan skema yang telah diberikan, dapat dilihat di antara panah yang menghubungkan setiap tabel terdapat nama suatu kolom, yakni kolom yang menjadi kunci yang menghubungkan tabel dengan tabel lainnya. Entity relationship diagram dibuat dengan menggunakan ERD tool yang ada pada pgadmin. Selanjutnya memilih "One to Many" link feature yang terdapat pada ERD tool, menyesuaikan dengan skema yang telah diberikan. Script SQL berikut merupakan generated script dari entity relationship diagram yang telah dirancang berdasarkan skema yang telah diberikan.

```
ALTER TABLE IF EXISTS public.customers
    ADD FOREIGN KEY (customer_zip_code_prefix)
    REFERENCES public.geolocation (geo_zip_code_prefix) MATCH SIMPLE
    ON UPDATE NO ACTION
```

```
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.order_item
    ADD FOREIGN KEY (seller_id)
    REFERENCES public.sellers (seller_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.order_item
    ADD FOREIGN KEY (order_id)
    REFERENCES public.orders (order_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.order_item
    ADD FOREIGN KEY (product_id)
    REFERENCES public.products (product_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.orders
    ADD FOREIGN KEY (customers_id)
    REFERENCES public.customers (customer_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.payments
    ADD FOREIGN KEY (order_id)
    REFERENCES public.orders (order_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.reviews
    ADD FOREIGN KEY (order_id)
    REFERENCES public.orders (order_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.sellers
    ADD FOREIGN KEY (seller_zip_code)
    REFERENCES public.geolocation (geo_zip_code_prefix) MATCH SIMPLE
```

```
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;

END;
```

TASK 2

Subtask 1:
Menampilkan rata-rata jumlah customer aktif bulanan (*Monthly Active User*) untuk setiap tahun.

Monthly Active Users (MAU) Ratio measures the stickiness of our product - that is, how often people engage with our product. These help us understand how frequently active people are in using the products. Determining the number of active users over time helps with assessing the effectiveness of our marketing campaigns and the customer experience

```
select tahun,
       round(avg(mau),2) as average_mau
from (
       select
              date_part('year', o.order_purchase_timestamp) as tahun,
              date_part('month',o.order_purchase_timestamp) as bulan,
              count (distinct c.customer_unique_id) as mau
       from orders o
       join customers c on o.customers_id = c.customer_id
       group by 1,2
       ) tempQ
group by 1;
```

Monthly Active User (MAU) adalah metrik yang digunakan untuk mengukur jumlah pengguna unik suatu aplikasi selama periode 30 hari. Metrik ini menunjukkan kemampuan produk suatu perusahaan dalam menarik dan mempertahankan pengguna dari waktu ke waktu. Selanjutnya adalah mendefenisikan apa itu pengguna aktif. Pengguna aktif pada project ini didefenisikan sebagai pelanggan yang telah melakukan purchase.

Langkah pertama adalah membuat *nested query* yang bertujuan untuk menghitung MAU setiap bulannya. Nested query yang dilakukan berisi:

- Fungsi `date_part` digunakan untuk ekstraksi tanggal purchase. Ekstraksi yang dilakukan adalah mengambil tahun dan bulan dari kolom `order_purchase_timestamp`.
- Menghitung jumlah pelanggan aktif menggunakan `customer_unique_id`, bukan menggunakan `customer_id`. `Customer_id` adalah ID urutan pelanggan melakukan purchase pada riwayat orders. Lain halnya `customer_unique_id` adalah ID unik untuk setiap pelanggan.
- Mengambil data dari tabel customers untuk dimasukkan ke dalam tabel orders menggunakan primary key yakni `customer_id` pada tabel customers.

- Mengelompokkan jumlah pengguna aktif berdasarkan tahun order dan bulan order.

Setelah membangun tabel sementara menggunakan nested query, langkah berikutnya adalah menampilkan tabel yang berisi rata-rata jumlah MAU untuk setiap tahun. Perhitungan ini dilakukan dengan menggunakan SELECT statement pada kolom `tahun` dan `average_mau` dengan mengambil data dari tabel sementara yang telah dirancang sebelumnya menggunakan nested query (`tempQ`). Selanjutnya dilakukan pengelompokkan berdasarkan tahun order.

Subtask 2:
Menampilkan jumlah customer baru pada masing-masing tahun.

```
select
    date_part ('year', first_purchase_timestamp) as tahun,
    count (1) as new_customers
    from
    (
        select
        c.customer_unique_id,
        min (order_purchase_timestamp) as first_purchase_timestamp
        from orders o
        join customers c on o.customers_id = c.customer_id
        group by 1
    ) first_time
   group by 1;
```

Jumlah total customer_unique_id dalam tabel customers berjumlah 96.096 ID, yang artinya dalam database sudah ada 96.096 customers dari tahun 2016-2018. Semua customers tersebut tentunya pernah menjadi new customer disaat mereka melakukan purchase pertama kali. Sehingga dengan menemukan tahun pertama kali setiap pelanggan dalam melakukan purchase, akan menunjukkan jumlah new customer per tahunnya. Perhitungan dilakukan dengan menggunakan *nested query* untuk menemukan first_purchase_timestamp setiap customer dengan menggunakan MIN statement. Setelah mendapatkan tabel sementara yang berisikan kolom `customer_unique_id` dan `first_purchase_timestamp`, pada query utama dilakukan ekstrasi tahun order pada kolom `first_purchase_timestamp` menggunakan fungsi `date_part` dan pengelompokkan jumlah new customers berdasarkan tahun menggunakan `COUNT` dan `GROUP BY` statement.

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

```
select tahun,
count(distinct customer_unique_id) as repeat_customers
from
(
    select
```

```
    date_part ('year', o.order_purchase_timestamp) as tahun,
    c.customer_unique_id,
    count (1) as freq_purchase
    from orders o
    join customers c on c.customer_id = o.customers_id
    group by 1, 2
    having count (1) > 1
) freqpur
group by 1;
```

Tabel sementara hasil *nested query* (`freqpur`) digunakan untuk menghitung jumlah purchase yang dilakukan oleh masing-masing customer unique id pada tahun tertentu. Perhitungan ini dilakukan dengan menggunakan pengelompokkan COUNT statement berdasarkan tahun dan customer_unique_id. Selanjutnya untuk memfilter mana yang terkategorikan repeat order (yang mana telah melakukan order lebih dari 1 kali) digunakan HAVING statement. Pada main query, untuk mendapatkan jumlah repeat customers setiap tahunnya, dilakukan dengan agregasi COUNT yang dikelompokkan berdasarkan tahun order.

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

Average Order Frequency counts the average number of transactions per customer per period. This metric is used to better understand customer behavior and purchase patterns.

```
select tahun,
       round(avg (freq_purchase),2) as avg_yearly_order
from
(
       select
       date_part('year', o.order_purchase_timestamp) as tahun,
       c.customer_unique_id,
       count(1) as freq_purchase
       from orders o
       join customers c on c.customer_id = o.customers_id
       group by 1,2
) freqOrd
group by 1;
```

Tabel sementara hasil *nested query* (`freqOrd`) digunakan untuk menghitung jumlah purchase yang dilakukan oleh masing-masing customer unique id pada tahun tertentu. Perhitungan ini dilakukan dengan menggunakan pengelompokkan COUNT statement berdasarkan tahun dan customer_unique_id. Pada main query, untuk mendapatkan jumlah-rata order yang dilakukan oleh pelanggan pada tiap tahunnya dilakukan dengan menggunakan fungsi AVG dengan data inputannya adalah kolom freq_purchase yang sebelumnya telah dihitung menggunakan *nested query* (`freqOrd`). Kemudian untuk mengelompokkannya berdasarkan tahun order dilakukan dengan menggunakan GROUP BY statement.

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

```
WITH
	est_mau AS
				(
					select tahun,
					round(avg(mau),2) as average_mau
					from (
							select
							date_part('year',
o.order_purchase_timestamp) as tahun,
date_part('month',o.order_purchase_timestamp) as bulan,
							count (distinct c.customer_unique_id)
as mau
							from orders o
							join customers c on o.customers_id =
c.customer_id
							group by 1,2
						) tempQ
					group by 1
				),

	est_newcustomer AS
	(
		select
		date_part ('year', first_purchase_timestamp) as tahun,
		count (1) as new_customers
		from
		(
			select
			c.customer_unique_id,
			min (order_purchase_timestamp) as first_purchase_timestamp
			from orders o
			join customers c on o.customers_id = c.customer_id
			group by 1
		) first_time
	   group by 1
	),

	est_repeat_order AS
	(
		select tahun,
		count(distinct customer_unique_id) as repeat_customers
		from
		(
			select
			date_part ('year', o.order_purchase_timestamp) as tahun,
			c.customer_unique_id,
			count (1) as freq_purchase
			from orders o
			join customers c on c.customer_id = o.customers_id
			group by 1, 2
```

```
                    having count (1) > 1
            ) freqpur
            group by 1
      ),

      est_avg_order_yearly AS
      (
            select tahun,
            round(avg (freq_purchase),2) as avg_order_per_customer
            from
            (
                  select
                  date_part('year', o.order_purchase_timestamp) as tahun,
                  c.customer_unique_id,
                  count(1) as freq_purchase
                  from orders o
                  join customers c on c.customer_id = o.customers_id
                  group by 1,2
            ) freqOrd
            group by 1
      )

SELECT
      mau.tahun,
      mau.average_mau,
      newc.new_customers,
      rep.repeat_customers,
      freq.avg_order_per_customer
FROM est_mau mau
join est_newcustomer newc on mau.tahun = newc.tahun
join est_repeat_order rep on rep.tahun = mau.tahun
join est_avg_order_yearly freq on freq.tahun = mau.tahun;
```

Common Table Expression (CTE) digunakan untuk menggabungkan output dari 4 task sebelumnya. CTE syntax mirip seperti algoritma def function pada umumnya. Secara umum CTE dilakukan dengan mendefinisikan suatu serangkaian operasi ke dalam suatu function, kemudian pada akhir algoritma dilakukan pemanggilan menggunakan nama function yang telah didefinisikan sebelumnya. CTE syntax pada bahasa sql diawali dengan menggunakan WITH statement, kemudian mendefinisikan function dengan menggunakan AS statement. Pada query di atas digunakan CTE yang didalamnya terdapat 4 def function yakni: Task1, Task2, Task3, Task4. Kemudian pada akhir query dilakukan penggabungan seluruh output hasil masing masing nested query dengan menggunakan JOIN statement dan key tahun order. Hal ini dikarenakan tahun order merupakan kolom dimiliki masing-masing tabel sementara.

TASK 3

Subtask 1:
Membuat tabel berisi informasi total revenue tiap tahun.

```
create table Total_revenue as
select
       date_part('year', o.order_purchase_timestamp),
       sum (revenue_per_order) as revenue
from (
              select
                     order_id,
                     sum(price+freight_value) as revenue_per_order
                     from order_item
                     group by 1
) tempI
join orders o on tempI.order_id = o.order_id
where o.order_status = 'delivered'
group by 1;
```

Perhitungan pendapatan total (total revenue) dilakukan dengan menjumlahkan harga barang (`price`) dan ongkos kirim (`freight_value`) untuk barang-barang dengan order_status = 'delivered'. Kemudian dikelompokkan berdasarkan tahun order.

Subtask 2:
Membuat tabel berisi informasi jumlah cancel order untuk masing-masing tahun.

```
create table cancel_order as
select
       date_part('year', order_purchase_timestamp) as tahun,
       count (1) as canceled_order
from orders
where order_status ='canceled'
group by 1;
```

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

```
create table top_category_by_revenue as select
       tahun,
       product_category_name,
       revenue
from
(select
       date_part ('year', o.order_purchase_timestamp) as tahun,
       p.product_category_name,
       sum (oi.price + oi.freight_value) as revenue,
       rank () over(partition by
                           date_part('year', o.order_purchase_timestamp)
                           order by sum (oi.price + oi.freight_value) desc) as rcat
from order_item oi
```

```
join orders o on o.order_id = oi.order_id
join products p on p.product_id = oi.product_id
where o.order_status = 'delivered'
group by 1,2) tempM
where rcat = 1;
```

Pertama, membuat subquery untuk menghitung revenue yang dikelompokkan berdasarkan kategori dan tahun. Pengelompokkan berdasarkan product category dilakukan dengan join ke tabel products. Kemudian digunakan function RANK disimpan sebagai kolom rcat, bertujuan untuk mendapatkan rank dari perhitungan revenue by product category. Tak lupa dilakukan filter pada order dengan status yang sudah terkirim (‘`delivered`’) untuk memastikan perhitungan revenue valid. Pada main query dilakukan pembuatan table `top_category_by_revenue` berisi kolom tahun, product_category_name, dan revenue dengan filter rcat=1 untuk mendapatkan kategori produk yang mendapatkan peringkat pertama dalam memberikan banyak revenue untuk masing-masing tahun.

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

```
create table most_canceled_category as select
      tahun,
      product_category_name,
      canceled_order
from
(select
      date_part ('year', o.order_purchase_timestamp) as tahun,
      p.product_category_name,
      count (1) as canceled_order,
      rank () over(partition by
                        date_part('year', o.order_purchase_timestamp)
                        order by count (1) desc) as r_cat

from order_item oi
join orders o on o.order_id = oi.order_id
join products p on p.product_id = oi.product_id
where o.order_status = 'canceled'
group by 1,2) tempK
where r_cat=1;
```

Subtask 5:
Menampilkan seluruh hasil dalam satu tampilan tabel.

```
select
      a.tahun,
```

```
       b.revenue as year_total_revenue,
       a.product_category_name as top_product_category_by_revenue,
       a.revenue as category_revenue,
       c.product_category_name as most_canceled_product_category,
       c.canceled_order_num as category_number_canceled,
       d.canceled_order as year_total_number_canceled
from top_category_by_revenue a
join total_revenue b on a.tahun = b.tahun
join most_canceled_category c on a.tahun = c.tahun
join cancel_order d on d.tahun = a.tahun;
```

Task 4

Subtask 1:
Menampilkan jumlah pengguna masing-masing tipe pembayaran secara all the time diurutkan dari yang terfavorit.

```
select
              pay.payment_type,
       count (1) as jumlah_pengguna
from payments pay
join orders o on o.order_id = pay.order_id
group by 1
order by 2 DESC;
```

Subtask 2:
Menampilkan detail informasi jumlah pengguna masing-masing tipe pembayaran untuk setiap tahun.

```
with anpay as (
select
       date_part('year', o.order_purchase_timestamp) as tahun,
       pay.payment_type,
       count (1) as jumlah_pengguna
from payments pay
join orders o on o.order_id = pay.order_id
group by 1,2)

select payment_type,
       sum(case when tahun = '2016' then jumlah_pengguna else 0 end) as tahun_2016,
       sum(case when tahun ='2017' then jumlah_pengguna else 0 end) as tahun_2017,
       sum(case when tahun= '2018' then jumlah_pengguna else 0 end) as tahun_2018
from anpay
group by 1
order by 2 desc;
```