Basic Queries

1. List all unique cities where customers are located.

2. Count the number of orders placed in 2017.

3. Find the total sales per category.

4. Calculate the percentage of orders that were paid in installments.

5. Count the number of customers from each state.

Intermediate Queries

1. Calculate the number of orders per month in 2018.

2. Find the average number of products per order, grouped by customer city.

3. Calculate the percentage of total revenue contributed by each product category.

4. Identify the correlation between product price and the number of times a product has been purchased.

5. Calculate the total revenue generated by each seller, and rank them by revenue.

Advanced Queries

1. Calculate the moving average of order values for each customer over their order history.

2. Calculate the cumulative sales per month for each year.

3. Calculate the year-over-year growth rate of total sales.

4. Calculate the retention rate of customers, defined as the percentage of customers who make another purchase within 6 months of their first purchase.

5. Identify the top 3 customers who spent the most money in each year.

Importing Data From CSV File

-- importing customers data from CSV file

```
create table customers (
customer_id varchar(50),
        customer_unique_id varchar(50),
        customer_zip_code_prefix int,
        customer_city varchar(50),
        customer_state varchar(20)
);

Select* from customers

copy customers(customer_id,
        customer_unique_id,
        customer_zip_code_prefix,
        customer_city,
        customer_state)

        from 'D:\SQL_Sales_Project\DataSet\customers.csv'
        delimiter ','
        CSV HEADER
```

```
-- importing geolocation data from CSV file
create table geolocation
(
geolocation_zip_code_prefix int,
	geolocation_lat decimal(15,10),
	geolocation_lng decimal(15,10),
	geolocation_city varchar(50),
	geolocation_state varchar(3)
)
copy geolocation
(
geolocation_zip_code_prefix,
	geolocation_lat,
	geolocation_lng,
	geolocation_city,
	geolocation_state
)

	from 'D:\SQL_Sales_Project\DataSet\geolocation.csv'
	delimiter ','
	CSV HEADER
```

```
-- importing order_items data from CSV file
create table order_items
(
order_id varchar(50),
        order_item_id int,
        product_id varchar(50),
        seller_id varchar(50),
        shipping_limit_date date,
        price decimal(8,3),
        freight_value decimal(8,3)
);

copy order_items(
        order_id,
    order_item_id,
        product_id,
        seller_id,
        shipping_limit_date,
        price,
        freight_value)
from 'D:\SQL_Sales_Project\DataSet\order_items.csv'
delimiter ','
CSV HEADER
```

-- importing orders data from CSV file

```
create table orders
(
order_id varchar(50),
	customer_id varchar(50),
	order_status varchar(20),
	order_purchase_timestamp timestamp,
	order_approved_at timestamp,
	order_delivered_carrier_date timestamp,
	order_delivered_customer_date timestamp,
	order_estimated_delivery_date timestamp
);

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 'D:\SQL_Sales_Project\DataSet\orders.csv'
delimiter ','
CSV Header;
```

--importing payments data from CSV file

```
create table payments
(
order_id varchar(50),
        payment_sequential int,
        payment_type varchar (15),
        payment_installment int,
        payment_value decimal (10,2)
);

copy payments (
order_id,
        payment_sequential,
        payment_type,
        payment_installment,
        payment_value
)
from 'D:\SQL_Sales_Project\DataSet\payments.csv'
delimiter ','
CSV Header;
```

```
-- importing products data from CSV file
create table products
	(product_id varchar(50),
	product_category varchar(50),
	payment_name_length int,
	product_description_length int,
	product_photos_qty int,
	product_weight_g int,
	product_length_cm int,
	product_height_cm int,
	product_width_cm int );
copy products (
product_id,
	product_category,
	payment_name_length,
	product_description_length,
	product_photos_qty,
	product_weight_g,
	product_length_cm,
	product_height_cm,
	product_width_cm
)
from 'D:\SQL_Sales_Project\DataSet\products.csv'
delimiter ','
CSV Header;
```

```
-- importing sellers data from CSV file
create table sellers
(
seller_id varchar(50),
        seller_zip_code_prefix int,
        seller_city varchar(50),
        seller_state varchar(5)

);
copy sellers (
seller_id ,
        seller_zip_code_prefix ,
        seller_city ,
        seller_state
)
from 'D:\SQL_Sales_Project\DataSet\sellers.csv'
delimiter ','
CSV Header;
```

Basic Queries

1. List all unique cities where customers are located.

```
Select distinct(customer_city) from customers
```

2. Count the number of orders placed in 2017.

```
select count(order_id) from orders where order_purchase_timestamp like
'2017%';
```

3. Find the total sales per category.

```
SELECT
   UPPER(products.product_category) AS category,
   ROUND(SUM(payments.payment_value), 2) AS sales
FROM
   products
JOIN
   order_items ON products.product_id = order_items.product_id
JOIN
   payments ON payments.order_id = order_items.order_id
GROUP BY
   UPPER(products.product_category);
```

4. Calculate the percentage of orders that were paid in installments.

```
SELECT ((SUM(CASE WHEN payment_installments >= 1 THEN 1 ELSE 0
END)::decimal / COUNT(*)) * 100) AS percentage FROM payments;
```

5. Count the number of customers from each state.

```
SELECT customer_state, COUNT(customer_id) AS customer_count FROM
customers GROUP BY customer_state;
```

Intermediate Queries

1. Calculate the number of orders per month in 2018.

```
SELECT to_char(order_purchase_timestamp, 'Month') AS months,
COUNT(order_id) AS order_count FROM orders WHERE EXTRACT(YEAR FROM
order_purchase_timestamp) = 2018 GROUP BY months;
```

2. Find the average number of products per order, grouped by customer city.

```
WITH count_per_order AS (
   SELECT
      orders.order_id,
      orders.customer_id,
      COUNT(order_items.order_id) AS oc
   FROM
      orders
   JOIN
      order_items ON orders.order_id = order_items.order_id
   GROUP BY
      orders.order_id, orders.customer_id
)
SELECT
   customers.customer_city,
   ROUND(AVG(count_per_order.oc), 2) AS average_orders
FROM customers
JOIN
   count_per_order ON customers.customer_id = count_per_order.customer_id
GROUP BY
   customers.customer_city
ORDER BY average_orders DESC;
```

3. **Calculate the percentage of total revenue contributed by each product category.**

```
SELECT
    UPPER(products.product_category) AS category,
    ROUND((SUM(payments.payment_value) / (SELECT
SUM(payment_value) FROM payments)) * 100, 2) AS sales_percentage
FROM
    products
JOIN
    order_items ON products.product_id = order_items.product_id
JOIN
    payments ON payments.order_id = order_items.order_id
GROUP BY
    category
ORDER BY
    sales_percentage DESC;
```

4. Calculate the total revenue generated by each seller, and rank them by revenue.

```
SELECT seller_id, revenue,
   DENSE_RANK() OVER (ORDER BY revenue DESC) AS rn
FROM (
   SELECT
      order_items.seller_id,
      SUM(payments.payment_value) AS revenue
   FROM order_items
   JOIN  payments ON order_items.order_id = payments.order_id
   GROUP BY
      order_items.seller_id
) AS a;
```

Advanced Queries

1. **Calculate the cumulative sales per month for each year.**

```
SELECT years, months, payment,
     SUM(payment) OVER (ORDER BY years, months) AS cumulative_sales
FROM (
   SELECT EXTRACT(YEAR FROM orders.order_purchase_timestamp) AS years,
        EXTRACT(MONTH FROM orders.order_purchase_timestamp) AS months,
        ROUND(SUM(payments.payment_value), 2) AS payment
   FROM orders
   JOIN payments ON orders.order_id = payments.order_id
   GROUP BY years, months
   ORDER BY years, months
) AS a;
```

2. **Calculate the year-over-year growth rate of total sales.**

```
WITH a AS (SELECT
      EXTRACT(YEAR FROM orders.order_purchase_timestamp) AS years,
      ROUND(SUM(payments.payment_value), 2) AS payment
   FROM orders
   JOIN payments ON orders.order_id = payments.order_id
   GROUP BY   years
   ORDER BY   years )
SELECT years,
   ((payment - LAG(payment, 1) OVER (ORDER BY years)) /
    LAG(payment, 1) OVER (ORDER BY years)) * 100 AS "yoy % growth"
FROM a;
```