**Stage 1: Data Preparation**

**Step 1:**

**Create a new database and tables for the prepared data.**

create database ecommerce;

Create a table for nine csv data

There are 9 datasets with csv extension, so we create 9 tables to store the datasets, and customize the type of each column based on the datasets in the csv file.

create table customers (

customer\_id varchar(250),

customer\_unique\_id varchar(250),

customer\_zip\_code\_prefix int,

customer\_city varchar(250),

customer\_state varchar(250)

);

create table geolocation (

geo\_zip\_code\_prefix varchar(250),

geo\_lat varchar(250),

geo\_lng varchar(250),

geo\_city varchar(250),

geo\_state varchar(250)

);

create table order\_item (

order\_id varchar(250),

order\_item\_id int,

product\_id varchar(250),

seller\_id varchar(250),

shipping\_limit\_date timestamp,

price float,

freight\_value float

);

create table payments (

order\_id varchar(250),

payment\_sequential int,

payment\_type varchar(250),

payment\_installment int,

payment\_value float

);

create table reviews (

review\_id varchar(250),

order\_id varchar(250),

review\_score int,

review\_comment\_title varchar(250),

review\_comment\_message text,

review\_creation\_date timestamp,

review\_answer timestamp

);

create table orders (

order\_id varchar(250),

customers\_id varchar(250),

order\_status varchar(250),

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 products (

product\_id varchar(250),

product\_category\_name varchar(250),

product\_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

);

create table sellers (

seller\_id varchar(250),

seller\_zip\_code int,

seller\_city varchar(250),

seller\_state varchar(250)

);

**Step 2:**

**Importing csv data into the database**

When importing csv data into the database, the data type of the column in the database must match the dataset type in the csv file. If there is a discrepancy, the import operation will error. Furthermore, the dataset storage folder path must be complete till the name file.csv.

copy customers(

customer\_id,

customer\_unique\_id,

customer\_zip\_code\_prefix,

customer\_city,

customer\_state

)

from 'C:\Users\nurul\Documents\Rakamin\Mini Project\SQL\Dataset\customers\_dataset.csv'

delimiter ','

csv header;

copy geolocation(

geo\_zip\_code\_prefix,

geo\_lat,

geo\_lng,

geo\_city,

geo\_state

)

from 'C:\Users\nurul\Documents\Rakamin\Mini Project\SQL\Dataset\geolocation\_dataset.csv'

delimiter ','

csv header;

**Step 3:**

**Create entity relationships between tables, based on the schema below. Then export the Entity Relationship Diagram (ERD) in the form of a picture.**

The schematic graphic shows a column name in the centre of the arrows connecting each dataset. This demonstrates that the column is the key column that connects the dataset to other datasets.

For example, the order item dataset (orange color) is linked to the product dataset (yellow color), with the key column being the product id column.

However, in the product dataset, all of the product id column values are unique (single), yet in the order item dataset, the product id column values are non-unique (same value). As a result, the product id column is the product dataset's main key and a foreign key for the order item dataset. As a result, the right query is as follows:

alter table products add constraint pk\_products primary key (product\_id);

alter table order\_items add foreign key (product\_id) references products;

For relationships between other datasets, apply the same procedure as in the previous example to get the primary key and foreign key, such that the right query is obtained as follows.

Primary key for the other tables

alter table customers add constraint pk\_cust primary key (customer\_id);

alter table geolocation add constraint pk\_geo primary key (geo\_zip\_code\_prefix);

alter table orders add constraint pk\_orders primary key (order\_id);

alter table sellers add constraint pk\_seller primary key (seller\_id);

Foreign key for relationship between other tables

alter table customers add foreign key (customer\_zip\_code\_prefix) references geolocation;

alter table orders add foreign key (customer\_id) references customers;

alter table order\_items add foreign key (order\_id) references orders;

alter table order\_items add foreign key (seller\_id) references sellers;

alter table sellers add foreign key (seller\_zip\_code\_prefix) references geolocation;

alter table payments add foreign key (order\_id) references orders;

alter table order\_items add foreign key (product\_id) references products;

alter table reviews add foreign key (order\_id) references orders;

**Stage 2: Annual Customer Activity Growth Analysis**

**Step 1:**

**Displays the annual average number of monthly active users.**

select

year,

round(avg(mau), 2) as average\_mau

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 mau

from orders o

join customers c on o.customer\_id = c.customer\_id

group by 1,2

) subq

group by 1

The first step is to create a temporary table using a from clause subquery (colored yellow). The temporary table keeps track of the number of unique customers who actively place orders each month; this figure is known as the Monthly Active User (MAU). The date part function is used to extract the timestamp component, which in this case includes the year and month components as needed. After obtaining the monthly MAU figure, a larger aggregation can be performed to obtain the average MAU for each year, resulting in the avg (average) computation grouped by year.

**Step 2:**

**Displays the number of new customers (first time transactions) in each year.**

As in step 1, a subquery is utilized in the from clause to build a temporary table as a first step (colored yellow). However, this time the temporary table contains information about each customer's first purchase timestamp. This is simply done by using the MIN aggregation function on the order\_purchase\_timestamp column to determine the smallest date for each customer. After obtaining the first order date for each customer, the next step can be to extract the year using date\_part and then calculate how many customers there are for each year. This last figure reveals how many new consumers arrive each year.

select

date\_part('year', first\_purchase\_time) as year,

count(1) as new\_customers

from (

select

c.customer\_unique\_id,

min(o.order\_purchase\_timestamp) as first\_purchase\_time

from orders o

join customers c on c.customer\_id = o.customer\_id

group by 1

) subq

group by 1

**Step 3:**

**Displays the amount of customers who make multiple purchases (repeat orders) each year.**

select

year,

count(distinct customer\_unique\_id) as repeating\_customers

from (

select

date\_part('year', o.order\_purchase\_timestamp) as year,

c.customer\_unique\_id,

count(1) as purchase\_frequency

from orders o

join customers c on c.customer\_id = o.customer\_id

group by 1, 2

having count(1) > 1

) subq

group by 1

In step 3, the temporary table (yellow colored query) created indicates the quantity of orders placed by each customer in a specific year. This is accomplished by grouping the COUNT calculation by year and customer unique id. Because the purpose is to gain repeat customers, the HAVING filter is utilized to get customers who place more than one order. Following that, COUNT sorted by year can be aggregated to calculate how many customers place repeat orders (order more than one) for each year.

**Step 4:**

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

select

year,

round(avg(frequency\_purchase),3) as avg\_orders\_per\_customers

from (

select

date\_part('year', o.order\_purchase\_timestamp) as year,

c.customer\_unique\_id,

count(1) as frequency\_purchase

from orders o

join customers c on c.customer\_id = o.customer\_id

group by 1, 2

) a

group by 1

The temporary table produced this time (yellow-colored query) is similar to the one established in step 3, except that the HAVING filter is not used because the goal is to calculate the average order placed by each customer each year. After creating a table with order frequency information for each client each year, further aggregation can be accomplished by running AVG on the frequency column, which is then aggregated by year. This allows us to calculate the average number of orders per customer each year.

**Step 5:**

**Combine the three successfully displayed metrics into a single table view.**

Making each of these queries into a temporary table is one approach to present all of the results collected from Steps 1 - 4 in a single table. This can be accomplished by using Common Table Expressions (CTE). After that, we can join by year to aggregate all of the results because all of the queries from steps 1-4 have been grouped by year. The following is an example of the final query, which is a merge of steps 1 through 4, with the yellow colored query representing step 1, the blue colored query representing step 2, the red colored query representing step 3, and the green colored query representing step 4.

with

calc\_mau as (

select

year,

round(avg(mau), 2) as average\_mau

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 mau

from orders o

join customers c on o.customer\_id = c.customer\_id

group by 1,2

) subq

group by 1

),

calc\_newcust as (

select

date\_part('year', first\_purchase\_time) as year,

count(1) as new\_customers

from

(

select

c.customer\_unique\_id,

min(o.order\_purchase\_timestamp) as first\_purchase\_time

from orders o

join customers c on c.customer\_id = o.customer\_id

group by 1

) subq

group by 1

),

calc\_repeat as (

select

year,

count(distinct customer\_unique\_id) as repeating\_customers

from

(

select

date\_part('year', o.order\_purchase\_timestamp) as year,

c.customer\_unique\_id,

count(1) as purchase\_frequency

from orders o

join customers c on c.customer\_id = o.customer\_id

group by 1, 2

having count(1) > 1

) subq

group by 1

),

calc\_avg\_freq as (

select

year,

round(avg(frequency\_purchase),3) as avg\_orders\_per\_customers

from

(

select

date\_part('year', o.order\_purchase\_timestamp) as year,

c.customer\_unique\_id,

count(1) as frequency\_purchase

from orders o

join customers c on c.customer\_id = o.customer\_id

group by 1, 2

) a

group by 1

)

select

mau.year,

mau.average\_mau,

newc.new\_customers,

rep.repeating\_customers,

freq.avg\_orders\_per\_customers

from calc\_mau mau

join calc\_newcust newc on mau.year = newc.year

join calc\_repeat rep on rep.year = mau.year

join calc\_avg\_freq freq on freq.year = mau.year

**Stage 3: Annual Product Category Quality Analysis**

**Step 1:**

**Create a table with total company revenue statistics for each year.**

create table total\_revenue\_per\_year as

select

date\_part('year', o.order\_purchase\_timestamp) as year,

sum(revenue\_per\_order) as revenue

from (

select

order\_id,

sum(price+freight\_value) as revenue\_per\_order

from order\_items

group by 1

) subq

join orders o on subq.order\_id = o.order\_id

where o.order\_status = 'delivered'

group by 1

To create a new table based on a specified query, use the command create table <nama\_tabel> as <query>. To compute total revenue, a simple sum of the price of the goods (price) and the shipping cost (freight\_value) is performed, not forgetting to filter for orders that have actually been delivered (order\_status = ‘delivered’). After that, make certain that the grouping by year is completed at the end.

**Step 2:**

**Create a table that includes the total number of canceled orders for each year.**

create table total\_cancel\_per\_year as

select

date\_part('year', order\_purchase\_timestamp) as year,

count(1) as num\_canceled\_orders

from orders

where order\_status = 'canceled'

group by 1

**Step 3:**

**Created a table with the product category names that generated the most overall income each year.**

create table top\_product\_category\_by\_revenue\_per\_year as

select

year,

product\_category\_name,

revenue

from (

select

date\_part('year', o.order\_purchase\_timestamp) as year,

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 rk

from order\_items 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) sq

where rk = 1

The first step in this stage is to create a subquery with total revenue calculations organized by year and product category. Join to the products table to acquire the name of the product category. Remember to use the RANK window function to get the rk column, which is the ranking of product categories based on revenue for each year. Following that, a filter is applied to obtain product categories with a rank of 1 (indicating the highest income) for each year.

**Step 4:**

**Create a table with the names of the product categories with the most canceled orders for each year.**

create table most\_canceled\_product\_category\_per\_year as

select

year,

product\_category\_name,

num\_canceled

from (

select

date\_part('year', o.order\_purchase\_timestamp) as year,

p.product\_category\_name,

count(1) as num\_canceled,

rank() over(partition by

date\_part('year', o.order\_purchase\_timestamp)

order by count(1) desc) as rk

from order\_items 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) sq

where rk = 1

The query in this step are similar to the previous step’s. The RANK window function is the sole difference. It was ranked in the previous subtask based on the greatest total revenue, but it is now ranked based on the highest number of rows. Remember to filter for canceled orders (order\_status = ‘canceled’). This method successfully collected the number of cancellations for each product category. Following that, a filter to get rank 1 (the maximum number of cancels for that year) can be applied to obtain the necessary data results.

**Step 5:**

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

select

a.year,

a.product\_category\_name as top\_product\_category\_by\_revenue,

a.revenue as category\_revenue,

b.revenue as year\_total\_revenue,

c.product\_category\_name as most\_canceled\_product\_category,

c.num\_canceled as category\_num\_canceled,

d.num\_canceled\_orders as year\_total\_num\_canceled

from top\_product\_category\_by\_revenue\_per\_year a

join total\_revenue\_per\_year b on a.year = b.year

join most\_canceled\_product\_category\_per\_year c on a.year = c.year

join total\_cancel\_per\_year d on d.year = a.year

**Stage 4: Annual Payment Type Usage Analysis**

**Step 1:**

**Displays the number of usage of each payment type over time, sorted by favorite.**

select

op.payment\_type,

count(1) as num\_used

from order\_payments op

join orders o on o.order\_id = op.order\_id

group by 1

order by 2 desc

**Subtask 2:**

**Displays detailed information on the amount of each payment type used in each year.**

with

tmp as (

select

date\_part('year', o.order\_purchase\_timestamp) as year,

op.payment\_type,

count(1) as num\_used

from order\_payments op

join orders o on o.order\_id = op.order\_id

group by 1, 2

)

select \*,

case when year\_2017 = 0 then NULL

else round((year\_2018 - year\_2017) / year\_2017, 2)

end as pct\_change\_2017\_2018

from (

select

payment\_type,

sum(case when year = '2016' then num\_used else 0 end) as year\_2016,

sum(case when year = '2017' then num\_used else 0 end) as year\_2017,

sum(case when year = '2018' then num\_used else 0 end) as year\_2018

from tmp

group by 1) subq

order by 5 desc