Analyzing e-Commerce

Using SQL

Achmad Hilman Shadiqin
RAKAMIN MINI PROJECT

Create Table

```
CREATE TABLE IF NOT EXISTS customers (
    customer_id VARCHAR(255) PRIMARY KEY,
    customer_unique_id VARCHAR(255),
    customer_zip_code_prefix VARCHAR(255),
    customer_city VARCHAR(255),
    customer_state VARCHAR(255)
);
```

```
CREATE TABLE IF NOT EXISTS products (
    index_number VARCHAR(255),
    product_id VARCHAR(255) PRIMARY KEY,
    product_category_name VARCHAR(255),
    product_name_length INT,
    product_description_length INT,
    product_photos_qty INT,
    product_weight_g FLOAT,
    product_length_cm FLOAT,
    product_height_cm FLOAT,
    product_width_cm FLOAT
);
```

```
CREATE TABLE IF NOT EXISTS sellers (
    seller_id VARCHAR(255) PRIMARY KEY,
    seller_zip_code_prefix VARCHAR(255),
    seller_city VARCHAR(255),
    seller_state VARCHAR(255)
);
```

```
CREATE TABLE IF NOT EXISTS geolocation (
    geolocation_zip_code_prefix VARCHAR(255),
    geolocation_lat FLOAT,
    geolocation_lng FLOAT,
    geolocation_city VARCHAR(255),
    geolocation_state VARCHAR(255)
);
```

[Download Query disini]

```
CREATE TABLE IF NOT EXISTS orders (
    order_id VARCHAR(255) PRIMARY KEY,
    customer_id VARCHAR(255),
    order_status VARCHAR(255),
    order_purchase_timestamp TIMESTAMP,
    order_approved_at TIMESTAMP,
    order_delivered_carrier_date TIMESTAMP,
    order_delivered_customer_date TIMESTAMP,
    order_estimated_delivery_date TIMESTAMP
);
```

```
CREATE TABLE IF NOT EXISTS order_items (
    order_id VARCHAR(255),
    order_item_id VARCHAR(255)PRIMARY KEY,
    product_id VARCHAR(255),
    seller_id VARCHAR(255),
    shipping_limit_date TIMESTAMP,
    price NUMERIC(10,2),
    freight_value NUMERIC(10,2)
);
```

```
CREATE TABLE IF NOT EXISTS order_payments (
    order_id VARCHAR(255),
    payment_sequential INT,
    payment_type VARCHAR(255),
    payment_installments INT,
    payment_value NUMERIC(10,2)
);
```

```
CREATE TABLE IF NOT EXISTS order_reviews (
    review_id VARCHAR(255) PRIMARY KEY,
    order_id VARCHAR(255),
    review_score INT,
    review_comment_title VARCHAR(255),
    review_comment_message TEXT,
    review_creation_date TIMESTAMP,
    review_answer_timestamp TIMESTAMP
);
```

Handling Duplicates in Geolocation Table

```
-- check duplicated
SELECT geolocation_zip_code_prefix, COUNT(*)
FROM geolocation
GROUP BY geolocation_zip_code_prefix
HAVING COUNT(*) > 1;
```

```
-- remove duplicated
DELETE FROM geolocation
WHERE ctid NOT IN (
    SELECT MIN(ctid)
    FROM geolocation
    GROUP BY geolocation_zip_code_prefix
);
```

```
-- add pkey
ALTER TABLE public.geolocation
         ADD PRIMARY KEY (geolocation_zip_code_prefix);

-- add unique constraint
ALTER TABLE IF EXISTS public.geolocation
         ADD CONSTRAINT ukey_geolocation UNIQUE
(geolocation_zip_code_prefix);
```

Download Query disini

Add Foreign Key

```
ALTER TABLE IF EXISTS public.customers
    ADD FOREIGN KEY (customer_zip_code_prefix)
    REFERENCES public.geolocation (geolocation_zip_code_prefix)
MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
```

```
ALTER TABLE IF EXISTS public.order_items
    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_items
    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.order_items
    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_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.order_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.orders
    ADD FOREIGN KEY (customer_id)
    REFERENCES public.customers (customer_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_prefix)
    REFERENCES public.geolocation (geolocation_zip_code_prefix)
MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
```

Download Query disini

Annual Customer Activity Growth Analysis

```
-- monthly active user in year
select year, floor(avg(total_customer)) as monthly_active_user
from (
        select extract(
                month
                from order_purchase_timestamp
            ) as month, extract(
                year
                from order_purchase_timestamp
            ) as year, count(distinct customer_unique_id) as
total_customer
        from orders
            join customers on orders.customer_id =
customers.customer_id
        group by
            extract(
                month
                from order_purchase_timestamp
            ), extract(
                year
                from order_purchase_timestamp
            )
        order by 2 asc, 1 asc
    ) as sq1
group by
    year;
```

Download Query disini

```
-- total new customer
select year, sum(total_orders) as total_new_customer
from (
        select extract(
                year
                from orders.order_purchase_timestamp
            ) as year, customers.customer_unique_id,
count(orders.order_id) as total_orders
        from orders
            join customers on orders.customer_id =
customers.customer_id
        group by
            1, 2
    ) as sq2
where
    total_orders = 1
group by
    year;
```

```
-- total repeat customer
select year, sum(total_orders) as total_repeat_customer
from (
        select extract(
                year
                from orders.order_purchase_timestamp
            ) as year, customers.customer_unique_id,
count(orders.order_id) as total_orders
        from orders
            join customers on orders.customer_id =
customers.customer_id
        group by
            1, 2
    ) as sq3
where
    total_orders > 1
group by
    year;
```

```sql
-- avg frequency
select year, round(avg(total_orders),2) as average_orders
from (
        select extract(
                year
                from order_purchase_timestamp
            ) as year, customers.customer_unique_id,
count(orders.order_id) as total_orders
        from orders
            join customers on orders.customer_id =
customers.customer_id
        group by
            1, 2
    ) as sq4
group by
    year;
```

```sql
-- concat all metrics
with
    cte1 as (
        select year, floor(avg(total_customer)) as
monthly_active_user
        from (
                select extract(
                        month
                        from order_purchase_timestamp
                    ) as month, extract(
                        year
                        from order_purchase_timestamp
                    ) as year, count(distinct customer_unique_id)
as total_customer
                from orders
                    join customers on orders.customer_id =
customers.customer_id
                group by
                    extract(
                        month
                        from order_purchase_timestamp
                    ), extract(
                        year
                        from order_purchase_timestamp
                    )
                order by 2 asc, 1 asc
            ) as sq1
        group by
            year
    ),
    cte2 as (
        select year, sum(total_orders) as total_new_customer
        from (
                select extract(
                        year
                        from orders.order_purchase_timestamp
                    ) as year, customers.customer_unique_id,
count(orders.order_id) as total_orders
                from orders
                    join customers on orders.customer_id =
customers.customer_id
                group by
                    1, 2
            ) as sq2
        where
            total_orders = 1
        group by
            year
    ),

cte3 as (
    select year, sum(total_orders) as total_repeat_customer
    from (
            select extract(
                    year
                    from orders.order_purchase_timestamp
                ) as year, customers.customer_unique_id,
count(orders.order_id) as total_orders
            from orders
                join customers on orders.customer_id =
customers.customer_id
            group by
                1, 2
        ) as sq3
    where
        total_orders > 1
    group by
        year
```

```
),

cte4 as (
    select year, round(avg(total_orders),2) as average_orders
    from (
            select extract(
                    year
                    from order_purchase_timestamp
                ) as year, customers.customer_unique_id,
count(orders.order_id) as total_orders
            from orders
                join customers on orders.customer_id =
customers.customer_id
            group by
                1, 2
        ) as sq4
    group by
        year
)

select
    cte1.year,
    monthly_active_user,
    total_new_customer,
    total_repeat_customer,
    average_orders,
         round(monthly_active_user/total_new_customer,2)*100 as
ratio_monthly_active_user,
         round(total_repeat_customer/total_new_customer,2)*100 as
ratio_repeat_order
from
    cte1
    join cte2 on cte1.year = cte2.year
    join cte3 on cte2.year = cte3.year
    join cte4 on cte3.year = cte4.year;
```

Annual Product Quality Analysis

```
-- master table
create table master_table as
select
	o.customer_id,
	o.order_id,
	o.order_status,
	o.order_purchase_timestamp,
	pr.product_category_name,
	oi.price,
	oi.freight_value,
	sl.seller_city,
	sl.seller_state,
	op.payment_type,
	op.payment_sequential,
	op.payment_installments,
	op.payment_value,
	oi.price + oi.freight_value as total_price,
	op.payment_sequential * op.payment_value as
total_payment_value,
	(op.payment_sequential * op.payment_value) - (oi.price +
oi.freight_value) as total_revenue
from orders o join order_items oi on o.order_id = oi.order_id
join order_payments op on oi.order_id = op.order_id
join products pr on oi.product_id = pr.product_id
join sellers sl on oi.seller_id = sl.seller_id;
```

```
-- revenue yoy
select
	extract(year from order_purchase_timestamp) as year,
	sum(payment_value) as revenue
from
	master_table
where
	order_status = 'delivered'
group by
	1;
```

[Download Query disini]

```
-- cancel order
select
	extract(year from order_purchase_timestamp) as year,
	count(*) as total_cancel
from
	master_table
where
	order_status = 'canceled'
group by
	1;
```

```
-- top category by revenue yoy
-- create new table (product_revenue_yoy)
create table product_revenue_yoy as
select distinct
	product_category_name,
	extract(year from order_purchase_timestamp) as year,
	sum(payment_value) over(partition by
product_category_name, extract(year from
order_purchase_timestamp)) as revenue
from
	master_table
where
	order_status = 'delivered';
-- rank based on revenue
select
	product_category_name,
	year,
	revenue,
	rank_revenue
from
	(select
		product_category_name,
		year,
		revenue,
		rank() over(partition by year order by revenue
desc)as rank_revenue
	from product_revenue_yoy) as ranked
```

```
where rank_revenue = 1;
```

```
-- total cancel order by product category yoy
-- create new table (product_category_cancel_yoy)
create table product_category_cancel_yoy as
select distinct
        product_category_name,
        extract(year from order_purchase_timestamp) as year,
        count(*) over(partition by product_category_name,
extract(year from order_purchase_timestamp)) as total_cancel
from
        master_table
where
        order_status = 'canceled'
order by
        2 asc, 3 desc;
-- rank based on total cancel
select
        product_category_name,
        year,
        total_cancel,
        rank_cancel
from(
        select
                product_category_name,
                year,
                total_cancel,
                rank() over(partition by year order by
total_cancel desc) as rank_cancel
        from
                product_category_cancel_yoy) as ranked
where
        rank_cancel = 1;
```

```
-- concat all
with
cte_revenue as (
select
        extract(year from order_purchase_timestamp) as year,
        sum(payment_value) as revenue
from
        master_table
where
        order_status = 'delivered'
group by
        1),

cte_cancel_order as(
select
        extract(year from order_purchase_timestamp) as year,
        count(*) as total_cancel
from
        master_table
where
        order_status = 'canceled'
group by
        1),

cte_rank_top as(
select
        product_category_name,
        year,
        revenue,
        rank_revenue
from
        (select
                product_category_name,
                year,
                revenue,
                rank() over(partition by year order by revenue
desc)as rank_revenue
        from product_revenue_yoy) as ranked
where
        rank_revenue = 1),
```

```
cte_rank_cancel as(
select
        product_category_name,
        year,
        total_cancel,
        rank_cancel
from(
        select
                product_category_name,
                year,
                total_cancel,
                rank() over(partition by year order by
total_cancel desc) as rank_cancel
        from
                product_category_cancel_yoy) as ranked
where
        rank_cancel = 1)

select
        cte_revenue.year,
        cte_revenue.revenue,
        cte_cancel_order.total_cancel as total_order_canceled,
        cte_rank_top.product_category_name as top_ranked_product,
        cte_rank_top.revenue as total_revenue_top_rank_product,
        cte_rank_cancel.product_category_name as
most_canceled_product,
        cte_rank_cancel.total_cancel as
total_top_canceled_product
from cte_revenue
join cte_cancel_order on cte_revenue.year = cte_cancel_order.year
join cte_rank_top on cte_cancel_order.year = cte_rank_top.year
join cte_rank_cancel on cte_rank_top.year = cte_rank_cancel.year
order by year;
```

```
-- ratio revenue lost
-- optional
with
cte_real_revenue as (
select extract(year from order_purchase_timestamp) as year,
sum(payment_value) as real_revenue
from master_table where order_status = 'delivered' group by 1),

cte_expected_revenue as (
select extract(year from order_purchase_timestamp) as year,
sum(payment_value) as expect_revenue
from master_table where order_status in ('delivered','canceled')
group by 1),

cte_lost_revenue as (
select extract(year from order_purchase_timestamp) as year,
sum(payment_value) as lost_revenue
from master_table where order_status = 'canceled' group by 1 order
by 1 asc)

select
        crr.year,
        crr.real_revenue,
        cer.expect_revenue,
        clr.lost_revenue,
        round(clr.lost_revenue / cer.expect_revenue * 100,2) as
ratio_lost_revenue
from cte_real_revenue crr
join cte_expected_revenue cer on crr.year = cer.year
join cte_lost_revenue clr on cer.year = clr.year
order by 1;
```

Download Query disini

Analysis of Annual Payment Type Usage

```
with cte_payment as (
select
	extract(year from order_purchase_timestamp) as year,
	sum(case when payment_type = 'boleto' then 1 else 0 end)
as boleto,
	sum(case when payment_type = 'boleto' then payment_value
else 0 end) as boleto_values,
	sum(case when payment_type = 'debit_card' then 1 else 0
end) as debit_card,
	sum(case when payment_type = 'debit_card' then
payment_value else 0 end) as debit_card_values,
	sum(case when payment_type = 'voucher' then 1 else 0 end)
as voucher,
	sum(case when payment_type = 'voucher' then payment_value
else 0 end) as voucher_values,
	sum(case when payment_type = 'credit_card' then 1 else 0
end) as credit_card,
	sum(case when payment_type = 'credit_card' then
payment_value else 0 end) as credit_card_values
from
	master_table
where
	order_status = 'delivered'
group by
	1
order by
	1)

select
	year,
	boleto, boleto_values, round(boleto_values/boleto,2) as
boleto_per_transaction,
	debit_card, debit_card_values,
round(debit_card_values/debit_card,2) as
debit_card_per_transaction,
	voucher, voucher_values, round(voucher_values/voucher,2)
as voucher_per_transaction,
	credit_card, credit_card_values,
round(credit_card_values/credit_card,2) as
credit_card_per_transaction,
	boleto_values + voucher_values + debit_card_values +
credit_card_values as total_payment_values
from cte_payment;
```

Download Query disini