I. DATA DAN ENTITY RELATIONSHIP DIAGRAM (ERD)

1. Membuat tabel.

   Pada data CSV yang tersedia terdapat 8 dataset yang nantinya saling berinteraksi. Untuk memasukan dataset tersebut kedalam database yang tersedia, perlu untuk membuat tabel dari masing-masing dataset dan tidak lupa untuk menyesuaikan tipe data dari masing-masing kolom. Dalam membuat tabel dapat dilakukan dengan memilih menu create tabel pada database yang sebelumnya telah dibuat.

   Ataupun dapat menggunakan query dengan fungsi CREATE untuk membuat tabel seperti pada berikut ini.

```
-- This script was generated by a beta version of the ERD tool in pgAdmin 4.
-- Please log an issue at https://redmine.postgresql.org/projects/pgadmin4/issues/new if
    you find any bugs, including reproduction steps.
BEGIN;


CREATE TABLE IF NOT EXISTS public.customers_dataset
(
    customer_id character varying COLLATE pg_catalog."default",
    """customer_unique_id""" character varying COLLATE pg_catalog."default",
    """customer_zip_code_prefix""" numeric,
    """customer_city""" character varying COLLATE pg_catalog."default",
    """customer_state""" character varying COLLATE pg_catalog."default"
);

CREATE TABLE IF NOT EXISTS public.geolocation_dataset
(
    geolocation_zip_code_prefix numeric,
    """geolocation_lat""" double precision,
    """geolocation_lng""" double precision,
    """geolocation_city""" character varying COLLATE pg_catalog."default",
    """geolocation_state""" character varying COLLATE pg_catalog."default"
);

CREATE TABLE IF NOT EXISTS public.order_items_dataset
(
    order_id character varying COLLATE pg_catalog."default",
    """order_item_id""" numeric,
    """product_id""" character varying COLLATE pg_catalog."default",
    """seller_id""" character varying COLLATE pg_catalog."default",
    """shipping_limit_date""" timestamp with time zone,
    """price""" double precision,
    """freight_value""" double precision
);

CREATE TABLE IF NOT EXISTS public.order_payments_dataset
(
    order_idorder_id character varying COLLATE pg_catalog."default",
    """payment_sequential""" numeric,
    """payment_type""" character varying COLLATE pg_catalog."default",
    """payment_installments""" numeric,
    """payment_value""" double precision
```

```
);

CREATE TABLE IF NOT EXISTS public.order_reviews_dataset
(
    review_id character varying COLLATE pg_catalog."default",
    """order_id""" character varying COLLATE pg_catalog."default",
    """review_score""" numeric,
    """review_comment_title""" character varying COLLATE pg_catalog."default",
    """review_comment_message""" character varying COLLATE pg_catalog."default",
    """review_creation_date""" timestamp without time zone,
    """review_answer_timestamp""" timestamp without time zone
);

CREATE TABLE IF NOT EXISTS public.orders_dataset
(
    order_id character varying COLLATE pg_catalog."default",
    """customer_id""" character varying COLLATE pg_catalog."default",
    """order_status""" character varying COLLATE pg_catalog."default",
    """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
);

CREATE TABLE IF NOT EXISTS public.product_dataset
(
    "number" numeric,
    product_id character varying COLLATE pg_catalog."default",
    product_category_name character varying COLLATE pg_catalog."default",
    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
);

CREATE TABLE IF NOT EXISTS public.sellers_dataset
(
    seller_id character varying COLLATE pg_catalog."default",
    """seller_zip_code_prefix""" numeric,
    """seller_city""" character varying COLLATE pg_catalog."default",
    """seller_state""" character varying COLLATE pg_catalog."default"
);
END;
```

2. Mengecek arah hubungan antara tabel.
   Dalam melakukan analisis lebih lanjut terhadap database yang terdiri dari 8 tabel perlu dilakukan pengecekan terlebih bagaimana arah relasi antara tabel. Contohnya relasi One-to-One kolom customer_id pada tabel orders_dataset dan pada tabel customers_dataset yang

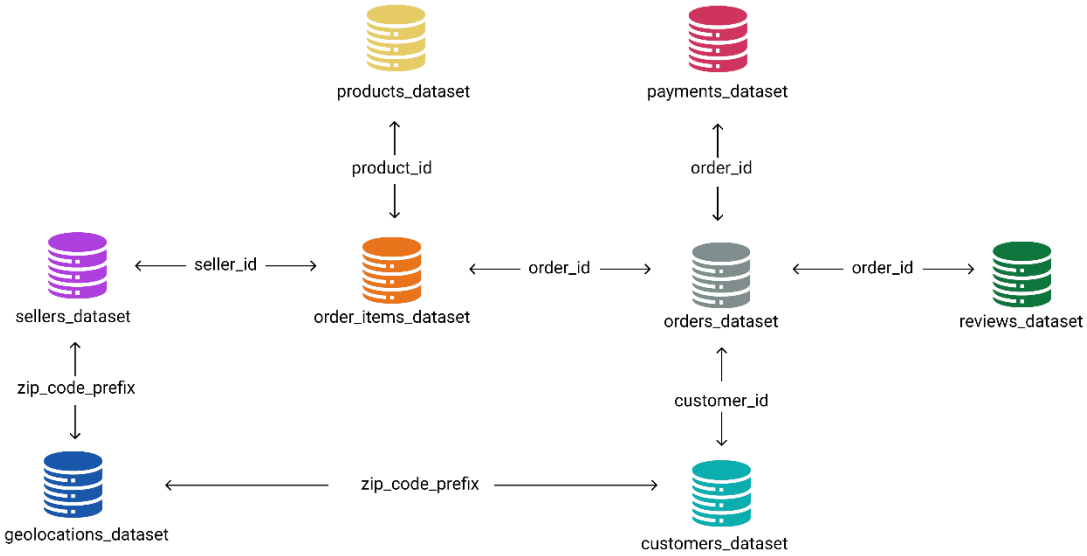
mana setiap satu baris pada tabel orders_dataset hanya berhubungan satu baris pada tabel customers_dataset dikarenakan tidak adanya duplikat value pada kedua tabel. Selain itu contoh relasi One-to-Many kolom product_id pada tabel product_dataset dan tabel order_items_dataset dikarenakan product_id dari order_items_dataset terdapat duplikat value. Untuk pengecekan kolom lainnya dapat dilihat melalui query berikut.

```
--MENGECEK KOLOM PRODUCT_ID--
select * from product_dataset where product_id in (select product_id from (
select product_id, count(*)
from product_dataset
group by product_id
HAVING count(*) > 1) as foo);

select * from order_items_dataset where product_id in (select product_id from (
select product_id, count(*)
from order_items_dataset
group by product_id
HAVING count(*) > 1) as foo);

--Mengecek ID antar tabel
SELECT pd.product_id
FROM product_dataset as PD
INNER JOIN order_items_dataset as OID
ON (PD.product_id = OID.product_id);

/*Kesimpulan Product_id dari product_dataset one to many, dikarenakan
Product_id dari order_items_dataset terdapat duplikat */

--MENGECEK KOLOM SELLER_ID--
select * from order_items_dataset where seller_id in (select seller_id from (
select seller_id, count(*)
from order_items_dataset
group by seller_id
HAVING count(*) > 1) as foo);

select * from sellers_dataset where seller_id in (select seller_id from (
select seller_id, count(*)
from sellers_dataset
group by seller_id
HAVING count(*) > 1) as foo);

--Mengecek ID antar tabel
SELECT SD.seller_id
FROM sellers_dataset as SD
INNER JOIN order_items_dataset as OID
ON (SD.seller_id = OID.seller_id);

/*Kesimpulan seller_id dari sellers_dataset one to many
dikarenakan seller_id dari order_items_dataset terdapat duplikat */

--MENGECEK KOLOM ORDER_ID--
select * from orders_dataset where order_id in (select order_id from (
select order_id, count(*)
```

```sql
from orders_dataset
group by order_id
HAVING count(*) > 1) as foo);
--tidak ada duplikat

select * from order_items_dataset where order_id in (select order_id from (
select order_id, count(*)
from order_items_dataset
group by order_id
HAVING count(*) > 1) as foo);
--terdapat duplikat

select * from order_payments_dataset where order_id in (select order_id from (
select order_id, count(*)
from order_payments_dataset
group by order_id
HAVING count(*) > 1) as foo);
--terdapat duplikat

select * from order_reviews_dataset where order_id in (select order_id from (
select order_id, count(*)
from order_reviews_dataset
group by order_id
HAVING count(*) > 1) as foo);
--terdapat duplikat

--Mengecek ID antar tabel
SELECT OD.order_id
FROM orders_dataset as OD
INNER JOIN order_items_dataset as OID
ON (OD.order_id = OID.order_id)
INNER JOIN order_payments_dataset as OPD
ON (OD.order_id = OPD.order_id)
INNER JOIN order_reviews_dataset as ORD
ON (OD.order_id = ORD.order_id);
/*Kesimpulan order_id dari orders_dataset one to many dikarenakan pada tabel
order_items_dataset, order_payments_dataset dan order_reviews_dataset terdapat
    duplikat*/

--MENGECEK KOLOM CUSTOMER_ID--
select * from orders_dataset where customer_id in (select customer_id from (
select customer_id, count(*)
from orders_dataset
group by customer_id
HAVING count(*) > 1) as foo);
-- tidak terdapat duplikat

select * from customers_dataset where customer_id in (select customer_id from (
select customer_id, count(*)
from customers_dataset
group by customer_id
HAVING count(*) > 1) as foo);
-- tidak terdapat duplikat
```

```sql
--Mengecek ID antar tabel
SELECT OD.customer_id
FROM orders_dataset as OD
INNER JOIN customers_dataset as CD
ON (OD.customer_id = CD.customer_id);
-- Kesimpulan customer_id one to one karena dari kedua tabel tidak terdapat duplikat

--MENGECEK KOLOM ZIP_CODE_PREFIX--
select * from customers_dataset where customer_zip_code_prefix in (select
    customer_zip_code_prefix from (
select customer_zip_code_prefix, count(*)
from customers_dataset
group by customer_zip_code_prefix
HAVING count(*) > 1) as foo);
--terdapat duplikat

select * from geolocation_dataset where geolocation_zip_code_prefix in (select
    geolocation_zip_code_prefix from (
select geolocation_zip_code_prefix, count(*)
from geolocation_dataset
group by geolocation_zip_code_prefix
HAVING count(*) > 1) as foo);
--terdapat duplikat

select * from sellers_dataset where seller_zip_code_prefix in (select
    seller_zip_code_prefix from (
select seller_zip_code_prefix, count(*)
from sellers_dataset
group by seller_zip_code_prefix
HAVING count(*) > 1) as foo);
--terdapat duplikat

SELECT CD.customer_zip_code_prefix, GD.geolocation_zip_code_prefix,
    SD.seller_zip_code_prefix
FROM customers_dataset as CD
INNER JOIN geolocation_dataset as GD
ON (CD.customer_zip_code_prefix = GD.geolocation_zip_code_prefix)
INNER JOIN sellers_dataset as SD
ON (CD.customer_zip_code_prefix = SD.seller_zip_code_prefix);
--Kesimpulan many to many
```

3. Menyesuaikan kolom dengan menambahkan kolom primary key dan foreign key.



Gambar. 1 Data dan Relationship eCommerce.

Setelah mengetahui arah relasi antara tabel pada dataset dan juga berdasarkan skema tabel pada Gambar 1, maka kita dapat menyesuaikan dengan menambahkan *primary key* dan *foreign key* menggunakan fungsi ALTER TABLE pada masing-masing tabel untuk kemudian dikeluarkan menjadi Entity Relationship Diagram (ERD) dengan query seperti berikut.

```
--customize primary key
alter table product_dataset add constraint pk_products primary key (product_id);
alter table customers_dataset add constraint pk_cust primary key (customer_id);
alter table geolocation_dataset add constraint pk_geo primary key (geo_zip_code_prefix);
alter table orders_dataset add constraint pk_orders primary key (order_id);
alter table sellers_dataset add constraint pk_seller primary key (seller_id);

--customize foreign key
alter table customers_dataset add foreign key (customer_zip_code_prefix) references
    geolocation;
alter table orders_dataset add foreign key (customer_id) references customers;
alter table order_items_dataset add foreign key (order_id) references orders;
alter table order_items_dataset add foreign key (seller_id) references sellers;
alter table order_items_dataset add foreign key (product_id) references products;
alter table sellers_dataset add foreign key (seller_zip_code_prefix) references
    geolocation;
alter table payments_dataset add foreign key (order_id) references orders;
alter table reviews_dataset add foreign key (order_id) references orders;
```

II. ANNUAL CUSTOMER ACTIVITY GROWTH

4. Menampilkan rata-rata jumlah customer aktif bulanan (monthly active user/MAU) untuk setiap tahun.

   Karena tidak terdapat kolom tahun untuk membantu menampilkan MAU setiap tahunnya, maka kita bisa mengambil tahunnya saja pada kolom order_purchase_timestamp yang

memiliki format timestamp without time zone dengan menggunakan fungsi DATE_PART dan mengambil bagian waktu year dan month. Setelah itu menghitung jumlah unik customer_unique_id menggunakan fungsi COUNT DISTINCT seperti pada query berikut ini. Sedangkan untuk mengambil nilai rata-rata jumlah customer menggunakan fungsi AVG dan fungsi ROUND untuk menentukan berapa digit angka di belakang koma.

```
select year, round(avg(total_customer),2) as avg_active_user
from
(select date_part('year', od.order_purchase_timestamp) as year,
        date_part('month', od.order_purchase_timestamp) as month,
        count(distinct cd.customer_unique_id) as total_customer
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
group by 1,2) a
group by 1;
```

5. Menampilkan jumlah customer baru pada masing-masing tahun.
   Seperti halnya pada nomor 4 menggunakan fungsi DATE_PART dan hanya mengambil bagian tahunnya saja, kemudian menghitung id yang unik menggunakan fungsi COUNT DISTICNT. Jumlah pelanggan baru merupakan pelanggan yang melakukan order pertama kali dimana kita bisa mencari melalui tanggal order yang paling awal yaitu menggunakan fungsi NEWEST yang diterapkan pada kolom order_purchase_timestamp.

```
select date_part('year', newest) as year, new_customer
from
(select date_part('year', od.order_purchase_timestamp) as year,
        min(order_purchase_timestamp) as newest,
        count(distinct cd.customer_unique_id) as new_customer
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
where order_status = 'delivered'
group by 1) a
group by 1,2;
```

6. Menampilkan jumlah customer yang melakukan pembelian lebih dari satu kali (repeat order) pada masing-masing tahun. Pada berikut ini yang dihitung merupakan jumlah pelanggan yang melakukan repeat order yaitu pelanggan yang melakukan order lebih dari 1 kali pada tiap tahunnya untuk menampilkan hal tersebut dapat menggunakan fungsi HAVING COUNT kolom order_id yang lebih dari 1.

```
select year, count(total_customer) as repear_order
from
(select date_part('year', od.order_purchase_timestamp) as year,
        cd.customer_unique_id,
        count(cd.customer_unique_id) as total_customer,
        count(od.order_id) as total_order
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
group by 1,2
having count(order_id) >1
```

```
) a
group by 1
order by 1;
```

7. Menampilkan rata-rata jumlah order yang dilakukan customer untuk masing-masing tahun. Untuk mendapatkan nilai rata-rata dari jumlah order dapat menggunakan fungsi AVG dari hasil COUNT DISTINCT kolom order_id sedangkan fungsi ROUND dan diakhir angka 2 untuk mengatur agar nilai yang dihasilkan hanya memiliki 2 angka di belakang koma.

```
select year, round(avg(total_order),2) as avg_frequency_order
from
(select date_part('year', od.order_purchase_timestamp) as year,
        cd.customer_unique_id,
        count(distinct order_id) as total_order
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
group by 1,2
) a
group by 1
order by 1;
```

8. Menggabungkan keempat metrik yang telah berhasil ditampilkan menjadi satu tampilan tabel. Untuk menggabungkan hasil matrix antara lain jumlah MAU, jumlah pelanggan baru, jumlah pelanggan repeat order dan rata-rata jumlah order yang sebelumnya telah dibuat dapat menggunakan Common Table Expression (CTE) yaitu dengan fungsi WITH … AS () atau dapat juga disebut sebagai temporary table seperti pada query berikut ini.

```
with count_mau as (
select year, round(avg(total_customer),2) as avg_active_user
from
(select date_part('year', od.order_purchase_timestamp) as year,
        date_part('month', od.order_purchase_timestamp) as month,
        count(distinct cd.customer_unique_id) as total_customer
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
group by 1,2) a
group by 1
),

count_newcust as(
select date_part('year', newest) as year, new_customer
from
(select date_part('year', od.order_purchase_timestamp) as year,
        min(order_purchase_timestamp) as newest,
        count(distinct cd.customer_unique_id) as new_customer
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
where order_status = 'delivered'
group by 1) a
```

```
group by 1,2
),

count_repeat_order as(
select year, count(total_customer) as repeat_order
from
(select date_part('year', od.order_purchase_timestamp) as year,
        cd.customer_unique_id,
        count(cd.customer_unique_id) as total_customer,
        count(od.order_id) as total_order
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
group by 1,2
having count(order_id) >1
) a
group by 1
order by 1
),

avg_order as (
select year, round(avg(total_order),2) as avg_frequency_order
from
(select date_part('year', od.order_purchase_timestamp) as year,
        cd.customer_unique_id,
        count(distinct order_id) as total_order
from orders_dataset as od
join customers_dataset as cd on od.customer_id = cd.customer_id
group by 1,2
) a
group by 1
order by 1)

select
cm.year,
cm.avg_active_user,
cn.new_customer,
cro.repeat_order,
ao.avg_frequency_order
from count_mau cm
join count_newcust cn on cm.year=cn.year
join count_repeat_order cro on cm.year=cro.year
join avg_order ao on cm.year=ao.year;
```

III. ANNUAL PRODUCT CATEGORY QUALITY ANALYSIS

9. Membuat tabel yang berisi informasi pendapatan/revenue perusahaan total untuk masing-masing tahun. Revenue merupakan harga barang dan juga biaya kirim, untuk query berikut

ini tinggal menambahkan kolom price dan freight_value menggunakan tanda “+”. Selain itu juga memastikan bahwa order_status adalah delivered dengan cara filtering menggunakan fungsi WHERE. Untuk membuat tabel menggunakan fungsi CREATE TABLE.

```
select * from order_payments_dataset;
select * from order_items_dataset;

create table if not exists revenue_year as (
select year, sum(revenue) as total_revenue
from
(select date_part('year', od.order_purchase_timestamp) as year,
        oid.price + oid.freight_value as revenue
        from orders_dataset od
        join order_items_dataset oid
        on oid.order_id=od.order_id
        where od.order_status = 'delivered') a
group by 1);
```

10. Membuat tabel yang berisi informasi jumlah cancel order total untuk masing-masing tahun. Menggunakan fungsi COUNT pada kolom order_id dan filtering WHERE dimana order_status adalah ‘canceled’.

```
select order_status, count(order_id) from orders_dataset
group by 1;

create table if not exists cancel_order_year as (
select date_part('year', order_purchase_timestamp) as year,
        count(distinct order_id) as total_cancel
        from orders_dataset
        where order_status = 'canceled'
group by 1);
```

11. Membuat tabel yang berisi nama kategori produk yang memberikan pendapatan total tertinggi untuk masing-masing tahun. Untuk mempermudah mendapatkan total pendapatan tertinggi perlu membuat subquery yaitu membuat query didalam query, dengan tujuan mendapatkan value dari tabel lain. Pada subquery tersebut diterapkan window function untuk mendapatkan total pendapatan tertinggi dengan menggunakan fungsi RANK() OVER (PARTITION BY <tahun>) dan ORDER BY <total pendapatan> karena yang diminta adalah setiap tahunnya dan diurutakan berdasarkan total pendapatan. Penggunaan window function seperti halnya agregasi menggunakan fungsi GROUP BY namun perbedaanya adalah hasil agregat tidak dikelompokkan menjadi satu baris sehingga ketika menggunakan window function hanya menampilkan per baris dan rank/urutan dari valuenya. Seperti pada query berikut ini.

```
select product_category_name, count(distinct product_id) from product_dataset
group by 1;
```

```
create table if not exists rank_product_revenue as(
select year, product_category_name, revenue
from (
select
    date_part('year', od.order_purchase_timestamp) as year,
    pd.product_category_name,
    rank() over(partition by date_part('year', od.order_purchase_timestamp)
    order by sum(oid.price + oid.freight_value) desc) as product_rank,
    sum(oid.price + oid.freight_value) as revenue
from orders_dataset od
        join order_items_dataset oid
        on oid.order_id=od.order_id
        join product_dataset as pd
        on oid.product_id = pd.product_id
        where od.order_status = 'delivered'
        group by 1,2) a
where product_rank = 1);
```

12. Membuat tabel yang berisi nama kategori produk yang memiliki jumlah cancel order terbanyak untuk masing-masing tahun. Seperti halnya nomor 11, dalam soal ini menerapkan penggunaan window function namun filtering yang digunakan adalah dimana order_status adalah ‘canceled’ setelah itu menghitung jumlah cancel ordernya dan mencari product_category_name apa yang memiliki rank/peringkat tertinggi.

```
create table if not exists rank_cancel_order as (
select year, product_category_name, total_cancel
from (
    select
        date_part('year', order_purchase_timestamp) as year,
        pd.product_category_name,
        rank() over(partition by date_part('year', od.order_purchase_timestamp)
        order by count(distinct od.order_id) desc) as cancel_rank,
        count(distinct od.order_id) as total_cancel
    from orders_dataset od
    join order_items_dataset oid
    on oid.order_id=od.order_id
    join product_dataset as pd
    on oid.product_id = pd.product_id
    where order_status = 'canceled'
    group by 1,2) a
where cancel_rank = 1);
```

13. Menggabungkan informasi-informasi yang telah didapatkan ke dalam satu tampilan tabel. Seperti halnya menggabungkan matrix pada nomer 8, namun pada kali ini tidak perlu menggunakan fungsi WITH AS, dikarenakan tabel sudah tersedia dengan cara CREATE TABLE sebelumnya sehingga hanya perlu menggunakan teknik JOIN pada kolom-kolom

yang dipilih dan menggunakan kolom year sebagai kunci untuk menggabungkan kolom-kolom lainnya.

```
select
    rpr.year,
    rpr.product_category_name,
    rpr.revenue,
    ry.total_revenue as total_revenue_order_per_year,
    rco.product_category_name,
    rco.total_cancel,
    coy.total_cancel as total_cancel_order_per_year
from revenue_year ry
join cancel_order_year coy on ry.year = coy.year
join rank_product_revenue rpr on ry.year = rpr.year
join rank_cancel_order rco on coy.year = rco.year;
```

IV. ANNUAL PAYMENT TYPE USAGE ANALYSIS

14. Menampilkan jumlah penggunaan masing-masing tipe pembayaran secara all time diurutkan dari yang terfavorit. Kita dapat mengurutkan jumlah order berdasarkan payment_type dengan menggunakan fungsi ORDER BY yang diikuti setelah penggunaan fungsi ORDER BY.

```
select * from order_payments_dataset;
select payment_type, count (order_id) from order_payments_dataset
group by 1
order by 2;
```

15. Menampilkan detail informasi jumlah penggunaan masing-masing tipe pembayaran untuk setiap tahun. Karena yang diminta adalah setiap tahunnya maka kita akan membuat menjadi kolom baru setiap tahun (2016-2018) berapa total order untuk masing-masing tahun, untuk itu kita dapat menggunakan fungsi CASE WHEN seperti pada query berikut ini.

```
select * from order_payments_dataset;
with
year_payment as (
select
    date_part('year', od.order_purchase_timestamp) as year,
    opd.payment_type,
    count(opd.order_id) as total_order
from order_payments_dataset opd
join orders_dataset od on od.order_id = opd.order_id
group by 1, 2
    order by 1
)
select
  payment_type,
  sum(case when year = '2016' then total_order else 0 end) as yr_2016,
  sum(case when year = '2017' then total_order else 0 end) as yr_2017,
  sum(case when year = '2018' then total_order else 0 end) as yr_2018
from year_payment
```

```
group by 1
order by 2,3,4 desc;
```