Ecommerce platform database modeling and analysis in PostrgreSQL

This project demonstrates the application of SQL skills in joins, aggregations and optimization to design and implement an efficient schema for an E-commerce platform.

The following is further documentation on each task for submission

Task 1: Database Design

The database contains for tables; `customers`, `products`, `orders` and `order_items`. We establish a relationship between the `orders` and `customers` table through a foreign key column `customer_id` in the `customers` table to the `customer_id` column in the `orders` table. The `order_items` table contains product and customer information for each made by customers. `order_id` and `product_id` columns in `order_item` link to the `order_id` and `product_id` columns in `orders` and `products` tables. The ER diagram below provides a visual representation of the schema

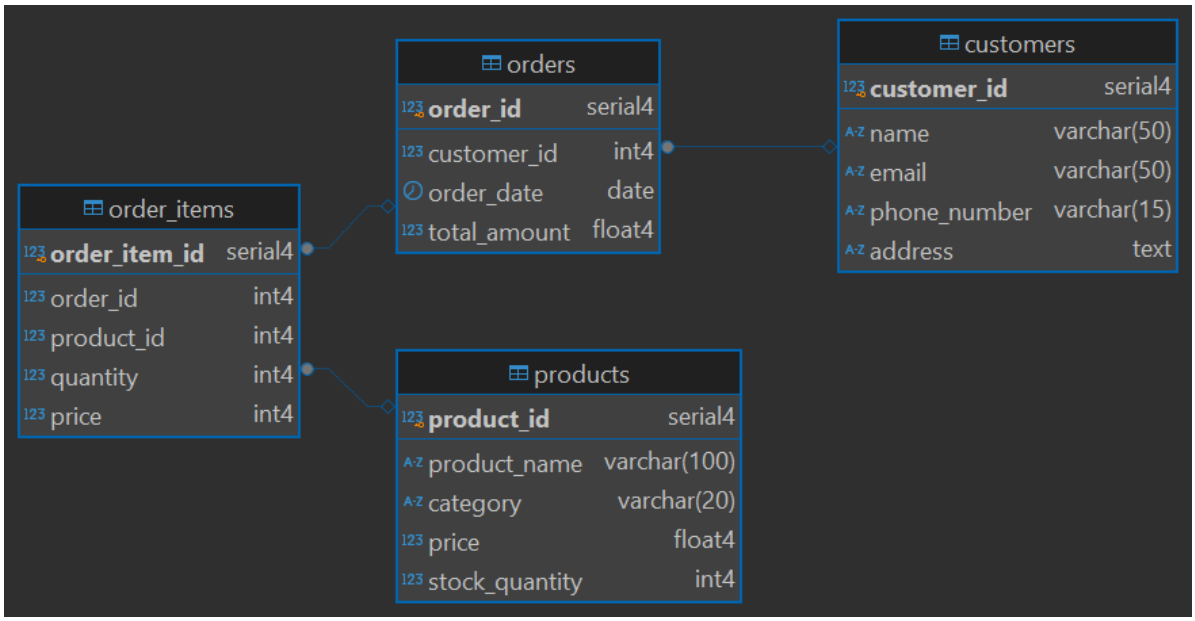


Figure 1: Schema

Task 2: Database Setup

To implement the schema in the database, we call `create table if not exists <table_name> (<col_1>,<col_2?>, ...)` for each table and the corresponding con-

straints and relationships in the columns.

```sql
create table customers (
    customer_id serial primary key,
    name varchar(50) not null,
    email varchar(50) unique not null,
    phone_number varchar(15) unique not null,
    address text not null
);

create table products (
    product_id serial primary key,
    product_name varchar(100) unique not null,
    category varchar(20) not null,
    price float(2) not null,
    stock_quantity int
);

create table orders (
    order_id serial primary key,
    customer_id int references customers(customer_id),
    order_date date,
    total_amount float(2)
);

create table order_items (
    order_item_id serial primary key,
    order_id int references orders(order_id),
    product_id int references products(product_id),
    quantity int not null,
    price int not null
)
```

Figure 2: create_tables

To persist data, we call `insert into <table_name> (<col_1>, <col_2>, ...) values (<val_1, <val_2>, ...)` for each table.

```sql
INSERT INTO customers (name, email, phone_number, address) VALUES
('Aisha Abubakar', 'aisha.abubakar@example.com', '0709117264', '123 Biashara Street, Eastleigh, Nairobi'),
('Benson Kiprotich', 'b.kiprotich@company.net', '0721752252', '456 Koinange Street, Nairobi CBD'),
('Catherine Wanjiku', 'c.wanjiku@mail.org', '0735233833', '789 Tom Mboya Street, Nairobi CBD'),
('David Omondi', 'david.omondi@web.io', '07100745296', '101 Moi Avenue, Nairobi CBD'),
('Esther Akinyi', 'e.akinyi@email.co.ke', '0772491745', '222 Ronald Ngala Street, Nairobi CBD'),
('Felix Musyoka', 'f.musyoka@online.com', '0766666666', '333 River Road, Nairobi CBD'),
('Grace Atieno', 'grace.atieno@mymail.com', '07753924777', '444 Accra Road, Nairobi CBD'),
('Hassan Mohammed', 'h.mohammed@domain.net', '0787833963', '555 Latema Road, Nairobi CBD'),
('Ivy Chebet', 'ivy.chebet@site.org', '0753926484', '666 Kirinyaga Road, Nairobi CBD'),
('John Kamau', 'john.kamau@mail.io', '0701234567', '777 Luthuli Avenue, Nairobi CBD'),
('Khadija Ali', 'k.ali@email.com', '0712345678', '888 Moi Drive, Nairobi West'),
```

Figure 3: Customers

```sql
INSERT INTO products (product_name, category, price, stock_quantity) VALUES
('Sony Bravia 55" TV', 'Electronics', 80000.00, 20),
('Savon Soap', 'Detergents', 80.00, 180),
('Garnier Micellar Water', 'Skincare', 600.00, 80),
('Kimbo Cooking Oil', 'Foods', 250.00, 80),
('Samsung Galaxy S23', 'Electronics', 120000.00, 30),
('Sunlight Dishwashing Liquid', 'Detergents', 180.00, 120),
('Pears Soap', 'Skincare', 100.00, 200),
('Raha Ugali', 'Foods', 120.00, 56),
('LG Refrigerator', 'Electronics', 60000.00, 15),
('Ariel Washing Powder', 'Detergents', 300.00, 75),
('Vaseline Petroleum Jelly', 'Skincare', 200.00, 150),
('Jogoo Maize Flour', 'Foods', 150.00, 100),
('Hisense Microwave', 'Electronics', 15000.00, 40),
('Jik Bleach', 'Detergents', 150.00, 90),
('Nivea Moisturizing Cream', 'Skincare', 450.00, 100),
('Brookside Milk', 'Foods', 60.00, 200),
('Skyworth 32" TV', 'Electronics', 25000.00, 35),
('Dettol Antiseptic Liquid', 'Detergents', 400.00, 60),
('Pemba Unga', 'Foods', 130.00, 90),
('Fresh Fri Cooking Oil', 'Foods', 280.00, 60);
```

Figure 4: Products

```sql
INSERT INTO orders (customer_id, order_date, total_amount) VALUES
(22, '2024-06-15', 125.50),
(4, '2024-07-07', 300.00),
(45, '2024-09-20', 75.00),
(22, '2024-08-03', 450.75),
(8, '2024-09-11', 220.20),
(36, '2024-08-18', 95.80),
(27, '2024-09-09', 180.50),
(4, '2024-08-07', 375.25),
(29, '2024-08-06', 60.00),
(11, '2024-07-06', 520.90),
(38, '2024-06-07', 110.75),
```

Figure 5: Orders

```sql
INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
(1, 8, 1, 120),
(2, 4, 1, 250),
(3, 2, 1, 80),
(4, 5, 1, 120000),
(5, 1, 1, 80000),
```

Figure 6: Order Items

Task 3 - Analytical Queries

Revenue Analysis

To calculate the total revenue by the platform we run the following query

```sql
-- Calculate all revenue for the
select sum(quantity*Price) as total_revenue
from order_items;
```

ustomers 1 | products 2 | orders 3 | order_items 4 | Results 5

lect sum(quantity*Price) as total_revenue from order_it | Enter a SQ

	total_revenue
1	2,525,660

Figure 7: Revenue

To find the total revenue per product

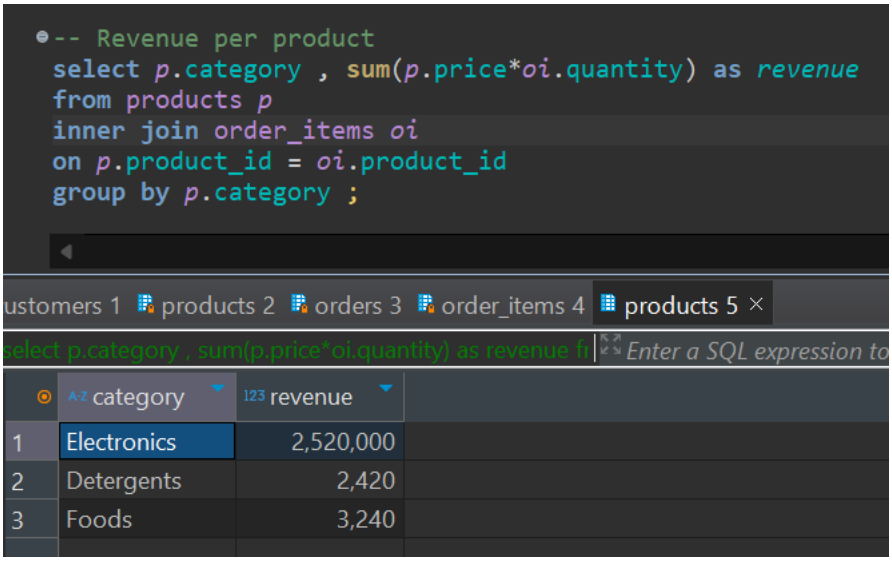


Figure 8: Revenue-Product

Customer Insights

To extract the top 5 customers by spending we run

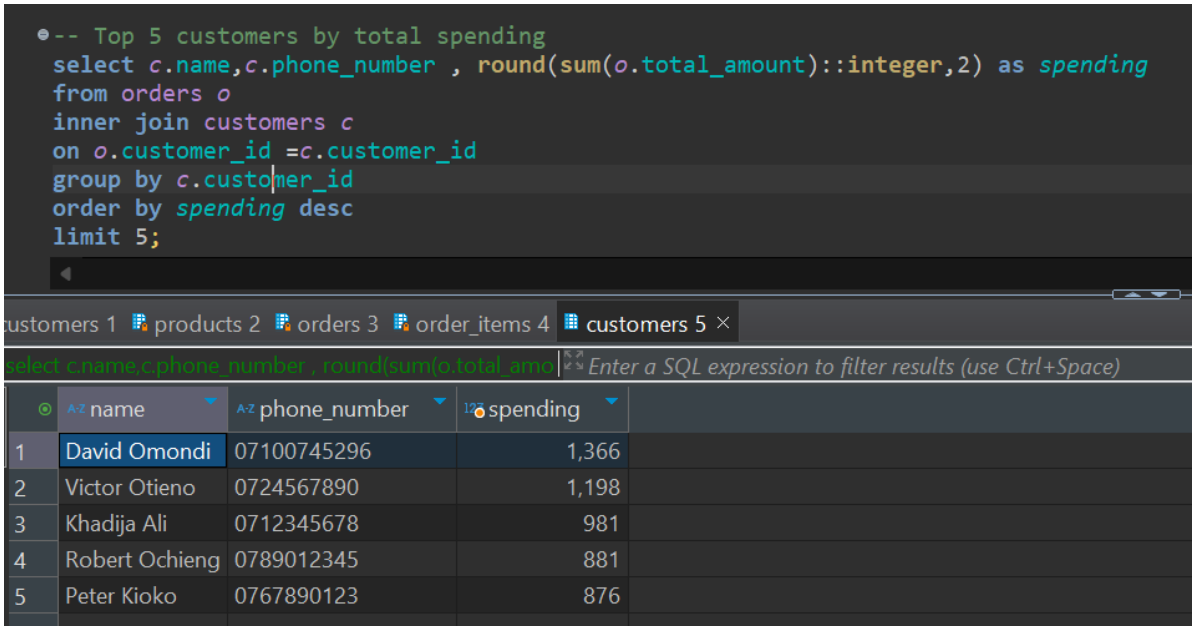


Figure 9: Top 5 Customers

To identify the customers who have not made a purchase

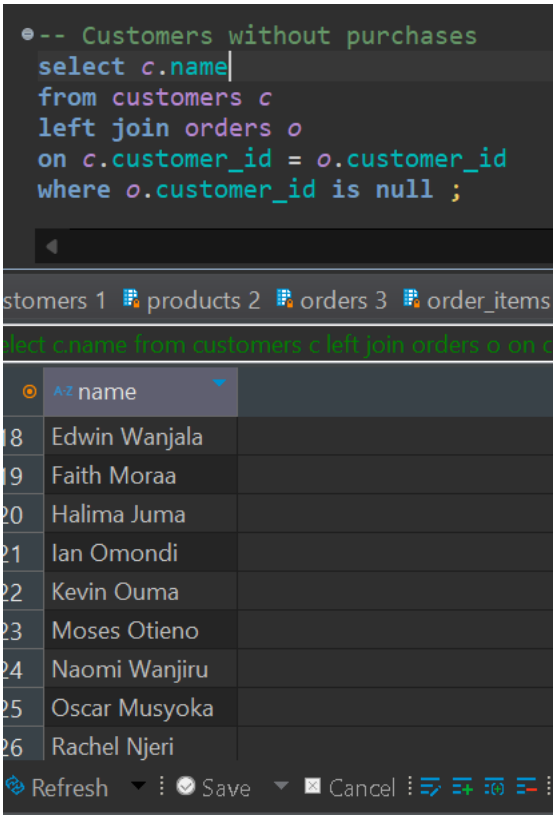


Figure 10: No Purchase

Product Trends

To find the 3 best selling product we run

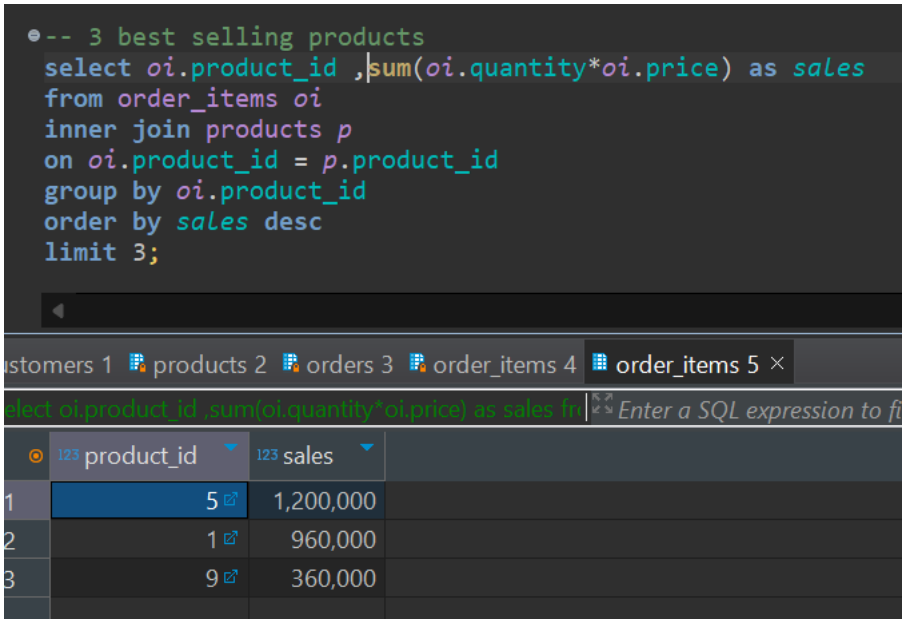


Figure 11: No Purchase

To identify the products that have ran out of stock we call the following query.

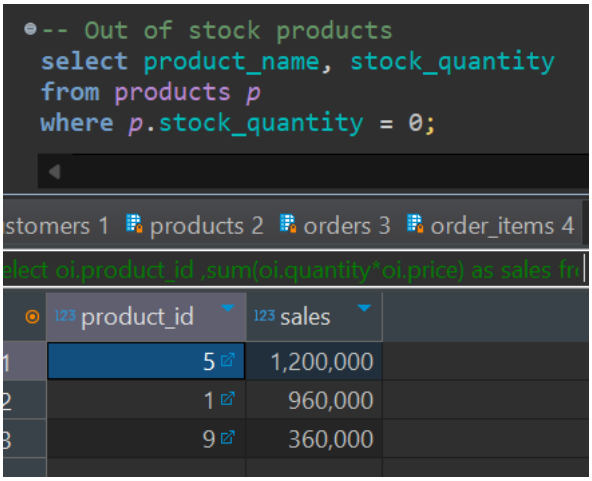


Figure 12: No Stock

Order Details

To retrieve all items in a specific order we call

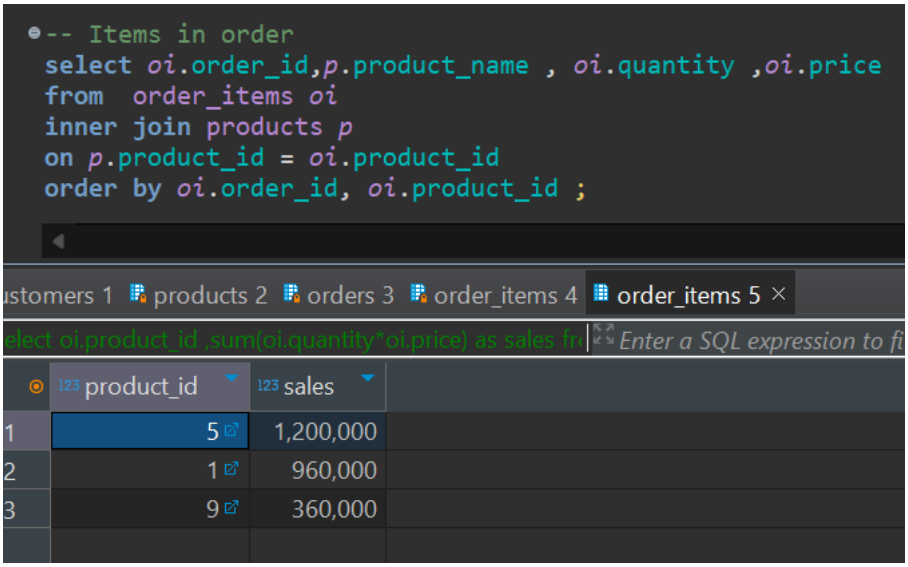


Figure 13: Items in order

To view the amount a given order call

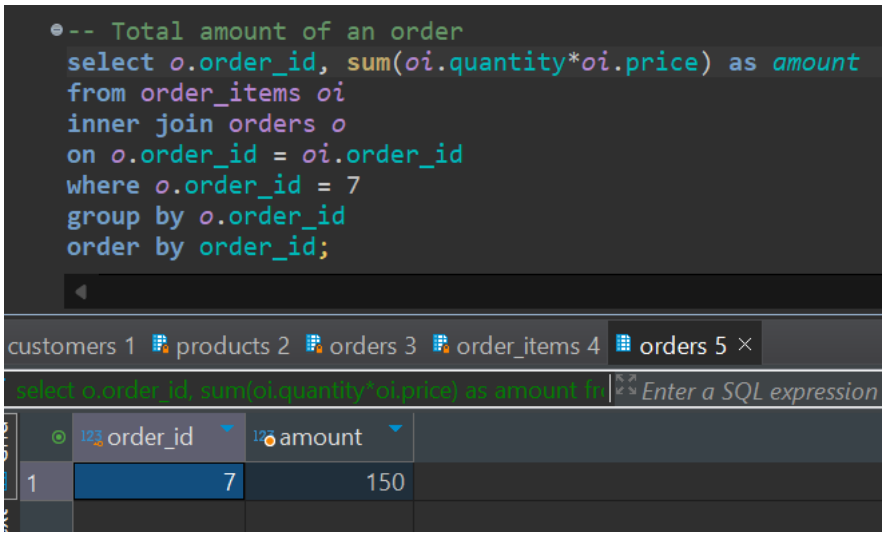


Figure 14: Order amount

Monthly Trends

To calculate orders and revenue per month

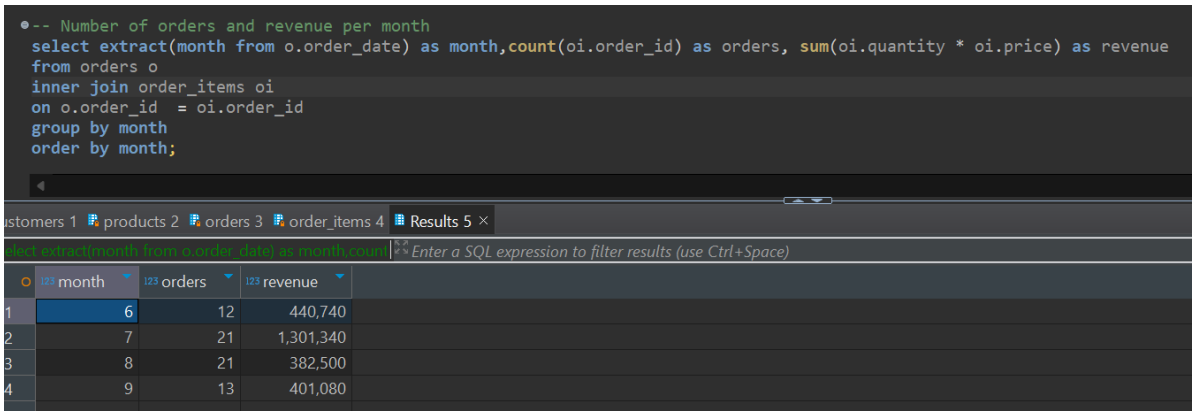
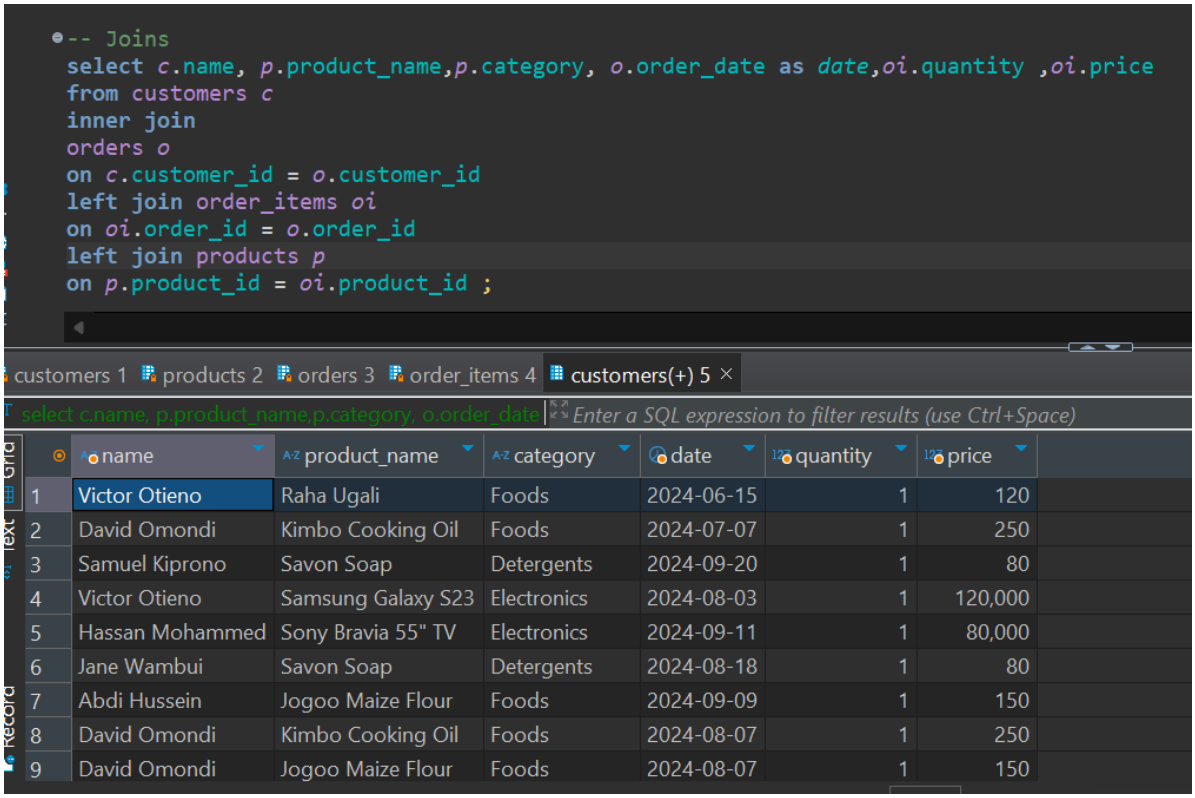


Figure 15: By month

Task 4 - Advanced SQL Concepts

Joins

To query data from multiple tables joins as follows



```sql
-- Joins
select c.name, p.product_name,p.category, o.order_date as date,oi.quantity ,oi.price
from customers c
inner join
orders o
on c.customer_id = o.customer_id
left join order_items oi
on oi.order_id = o.order_id
left join products p
on p.product_id = oi.product_id ;
```

customers 1 | products 2 | orders 3 | order_items 4 | customers(+) 5

select c.name, p.product_name,p.category, o.order_date | Enter a SQL expression to filter results (use Ctrl+Space)

	name	product_name	category	date	quantity	price
1	Victor Otieno	Raha Ugali	Foods	2024-06-15	1	120
2	David Omondi	Kimbo Cooking Oil	Foods	2024-07-07	1	250
3	Samuel Kiprono	Savon Soap	Detergents	2024-09-20	1	80
4	Victor Otieno	Samsung Galaxy S23	Electronics	2024-08-03	1	120,000
5	Hassan Mohammed	Sony Bravia 55" TV	Electronics	2024-09-11	1	80,000
6	Jane Wambui	Savon Soap	Detergents	2024-08-18	1	80
7	Abdi Hussein	Jogoo Maize Flour	Foods	2024-09-09	1	150
8	David Omondi	Kimbo Cooking Oil	Foods	2024-08-07	1	250
9	David Omondi	Jogoo Maize Flour	Foods	2024-08-07	1	150

Figure 16: Joins

Window functions

To rank customers based on total spending tun the following

```sql
-- Window functions RANK()
with spending as (
    select o.customer_id ,sum(oi.price * oi.quantity) as spending
    from orders o
    inner join order_items oi
    on oi.order_id = o.order_id
    group by o.customer_id
    )

select c.name, s.spending, rank() over (order by spending) as rank
from customers c
inner join spending s
on c.customer_id = s.customer_id;
```

customers 1 | products 2 | orders 3 | order_items 4 | customers 5 ×

with spending as (select o.customer_id ,sum(oi.price * oi Enter a SQL expression to filter results (u

	name	spending	rank
1	Caleb Kipkemboi	80	1
2	Samuel Kiprono	80	1
3	Peninah Atieno	80	1
4	Nicholas Ndegwa	80	1
5	Ivy Chebet	160	5
6	Geoffrey Mutua	230	6
7	Linet Chepkemoi	230	6
8	Winnie Achieng	400	8
9	Esther Akinyi	400	8

Figure 17: Rank

To assign a unique number to each order for a customer run

```sql
-- Window functions ROW_NUMBER()
select c.name,row_number() over (order by name) as order_number, o.order_date ,o.total_amount
from customers c
inner join orders o
on c.customer_id = o.customer_id ;
```

ustomers 1 products 2 orders 3 order_items 4 customers(+) 5 ×

select c.name,row_number() over (order by name) as ord Enter a SQL expression to filter results (use Ctrl+Space)

	name	order_number	order_date	total_amount
1	Abdi Hussein	1	2024-08-01	170.8
2	Abdi Hussein	2	2024-09-09	180.5
3	Abdi Hussein	3	2024-08-12	70
4	Caleb Kipkemboi	4	2024-08-06	60
5	Catherine Wanjiku	5	2024-07-02	80
6	David Omondi	6	2024-08-12	280
7	David Omondi	7	2024-09-08	100.5
8	David Omondi	8	2024-07-07	300
9	David Omondi	9	2024-08-07	375.25

Figure 18: Row number

CTEs and Sub-queries

To calculate total revenue per customer and find customers spending more than $500 proceed as follows

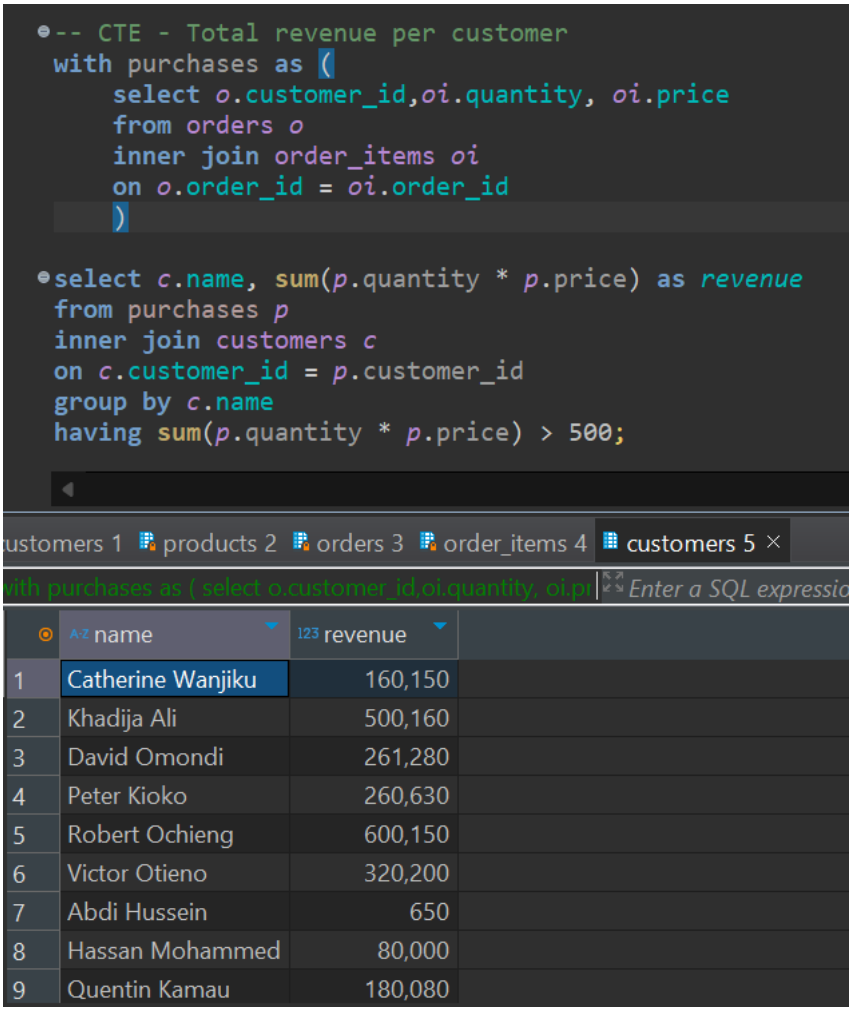


Figure 19: More then $500

To find the product with the highest price use a sub-query as follows

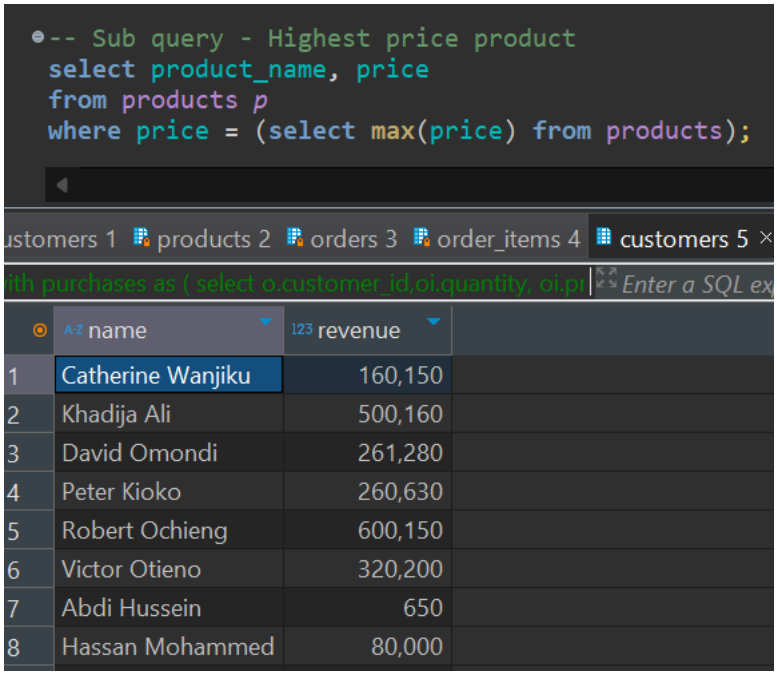


Figure 20: More then $500

Indexing

Use `explain analyze` to demonstrate efficiency of adding index columns

```
create unique index if not exists product_index on products (product_id);

-- Check effect of indexing on query speed by matching on ".com" in email column
explain analyze
select *from customers
where email ilike '%.com%';
```

ustomers 1 | products 2 | orders 3 | order_items 4 | Results 5 ×

xplain analyze select *from customers where email ilike | Enter a SQL expression to filter results (use Ctrl+Space)

	QUERY PLAN
1	Seq Scan on customers (cost=0.00..1.58 rows=1 width=320) (actual time=0.025..0.108 rows=12 loops=1)
2	Filter: ((email)::text ~~* '%.com%'::text)
3	Rows Removed by Filter: 34
4	Planning Time: 0.622 ms
5	Execution Time: 0.424 ms

Figure 21: Product index

•

```
-- Adding an index on customers table marginally increases query speed
create unique index customer_index on customers (customer_id);

explain analyze
select * from customers
where email ilike '%.com%';
```

ustomers 1 | products 2 | orders 3 | order_items 4 | Results 5 × | Statistics 5

reate unique index customer_index on customers (custo | Enter a SQL expression to filter results (use Ctr

	QUERY PLAN
1	Seq Scan on customers (cost=0.00..1.58 rows=1 width=320) (actual time=0.017..0.086 rows=12 lo
2	Filter: ((email)::text ~~* '%.com%'::text)
3	Rows Removed by Filter: 34
4	Planning Time: 0.822 ms
5	Execution Time: 0.096 ms

Figure 22: Customer index

Optimization

```sql
-- Optimization
-- Adjusting indexes, improving index coverage by using a composite index
create unique index if not exists product_name_index on products (product_id, product_name);

-- Join re-ordering
-- from
explain analyze
with purchases as (
    select o.customer_id,oi.quantity, oi.price
    from orders o
    inner join order_items oi
    on o.order_id = oi.order_id
    )

select c.name, sum(p.quantity * p.price) as revenue
from purchases p
inner join customers c
on c.customer_id = p.customer_id
group by c.name
having sum(p.quantity * p.price) > 500;
```

esults 1 ×

xplain analyze with purchases as (select o.customer_id,(Enter a SQL expression to filter results (use Ctrl+Space)

	QUERY PLAN
12	Buckets: 1024 Batches: 1 Memory Usage: 10kB
13	-> Seq Scan on orders o (cost=0.00..1.40 rows=40 width=8) (actual time=0.018..0.023 r
14	-> Hash (cost=1.46..1.46 rows=46 width=122) (actual time=0.045..0.045 rows=46 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 11kB
16	-> Seq Scan on customers c (cost=0.00..1.46 rows=46 width=122) (actual time=0.023..0.02
17	Planning Time: 8.906 ms
18	Execution Time: 0.418 ms

Figure 23: Index optimization

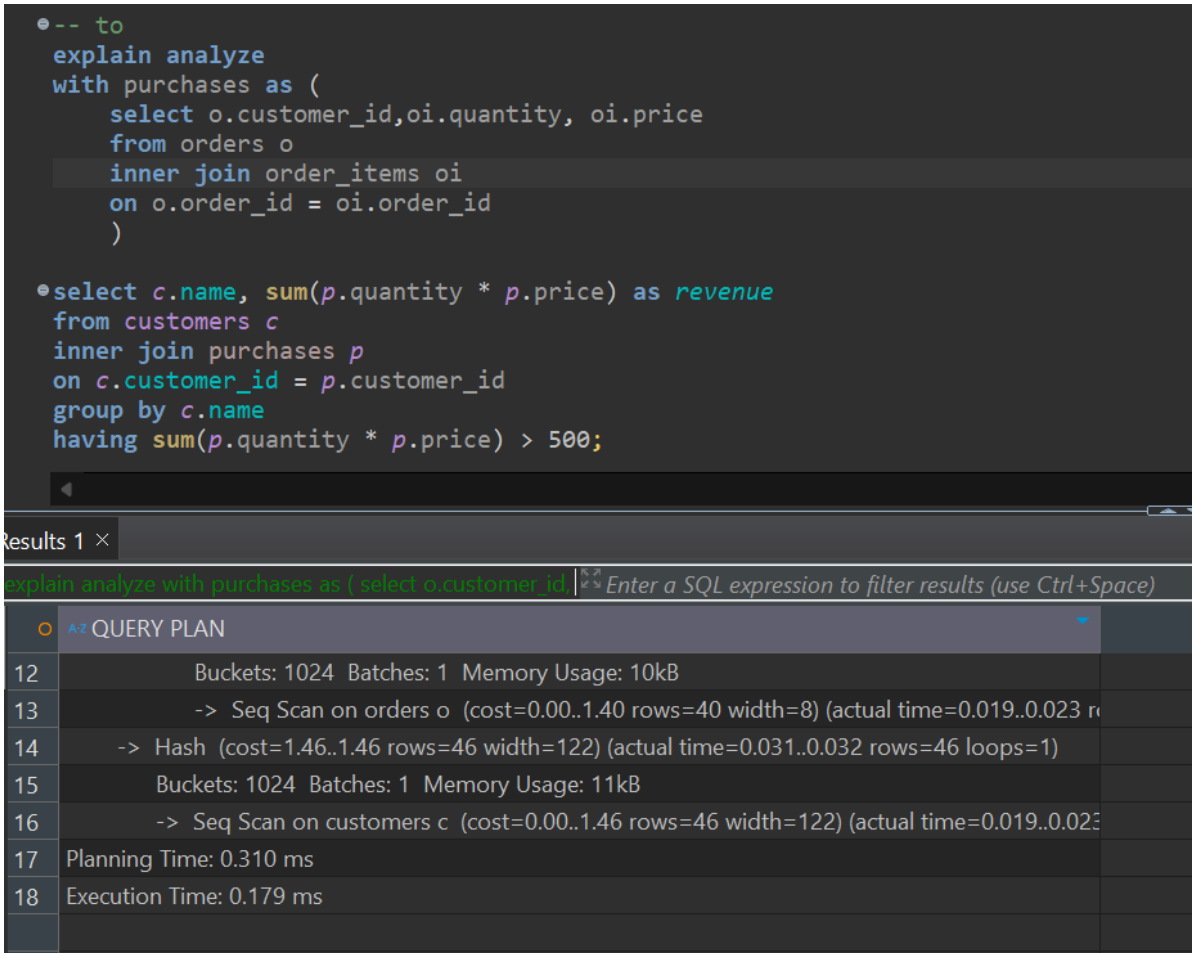


Figure 24: Reordering joins

Rewrite CTE to sub-query

```sql
-- Rewrite entire query
-- Subquery instead of CTE
select c.name, o.order_id, oi_details.product_id, oi_details.quantity, oi_details.price
from customers c
inner join orders o
on
c.customer_id = o.order_id
inner join (
    select product_id, order_id, quantity, price
from order_items
) as oi_details on
o.order_id = oi_details.order_id;
```

customers(+) 1 ×

select c.name, o.order_id, oi_details.product_id, oi_details | Enter a SQL expression to filter results (use Ctrl+Space)

	name	order_id	product_id	quantity	price
1	Aisha Abubakar	1	8	1	120
2	Benson Kiprotich	2	4	1	250
3	Catherine Wanjiku	3	2	1	80
4	David Omondi	4	5	1	120,000
5	Esther Akinyi	5	1	1	80,000
6	Felix Musyoka	6	2	1	80
7	Grace Atieno	7	12	1	150
8	Hassan Mohammed	8	4	1	250
9	Hassan Mohammed	8	12	1	150

Refresh | Save | Cancel | Export data | 200 | 67 | 67 row(s) fetc

Figure 25: CTE to subquery