Task 4: SQL for Data Analysis

Objective: Use SQL queries to extract and analyze data from a database.

Tools: MySQL or PostgreSQL or SQLite (I used MySQL)

Deliverables: SQL queries in a SQL file + screenshots of output

Hints/Mini Guide:

a. Use SELECT, WHERE, ORDER BY, GROUP BY

b. Use JOINS (INNER, LEFT, RIGHT)

c. Write subqueries

d. Use aggregate functions (SUM, AVG)

e. Create views for analysis

f. Optimize queries with indexes

Dataset: Ecommerce_Dataset

Work Done listed step by step

Step 1 :

The data set I have taken is named Ecommerce_dataset in excel format, which contains 7 sheets, named :

1. Customer
2. Categories
3. Products
4. Sales
5. Order_item
6. Payment
7. Shipped

These all are in excel format.

The very first step is to convert it into “csv” format.

All of them will not get changes at a time, so we will save it one by one in the same folder.

Step 2 : (Loading into database)

We will open our MySQL workbench, create a new schema named “ecommerce”, using query : **CREATE DATABASE ecommerce;**

and then use that database for creating tables and running queries, using query : **USE ecommerce;**

Now that the database is created we will load our csv file into our database in the form of table, all will not get loaded at a time, so we will do it one by one.

Steps involved for loading the files are as follow :

1. In MySQL Workbench:

- Right-click the **target table** → **Table Data Import Wizard**.
- Select the corresponding CSV.
- Click **Next** → check columns → **Next** → **Import**.

2. Confirm that data is loaded correctly.

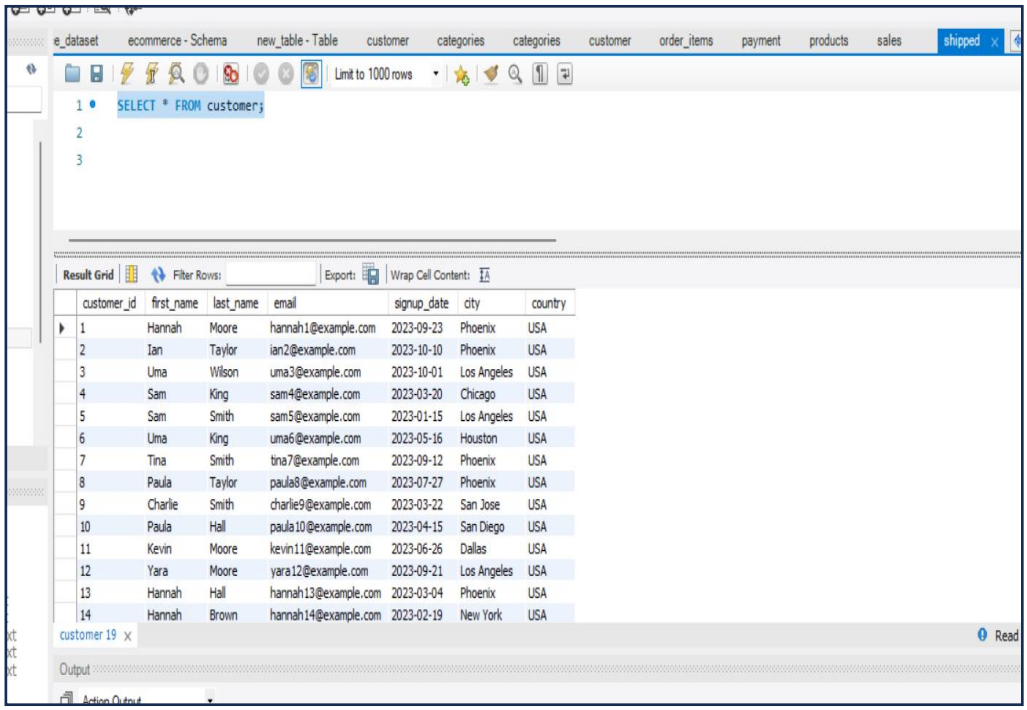
3. Repeat for all sheets
   3.1 One CSV → one table.
   3.2 If you have 50+ sheets, repeat the process for each.(I have 7 sheets)
   3.3 After import, all tables will appear under your schema.

Step 3 : (Perform SQL Queries)

<u>-- a. SELECT, WHERE, ORDER BY, GROUP BY</u>

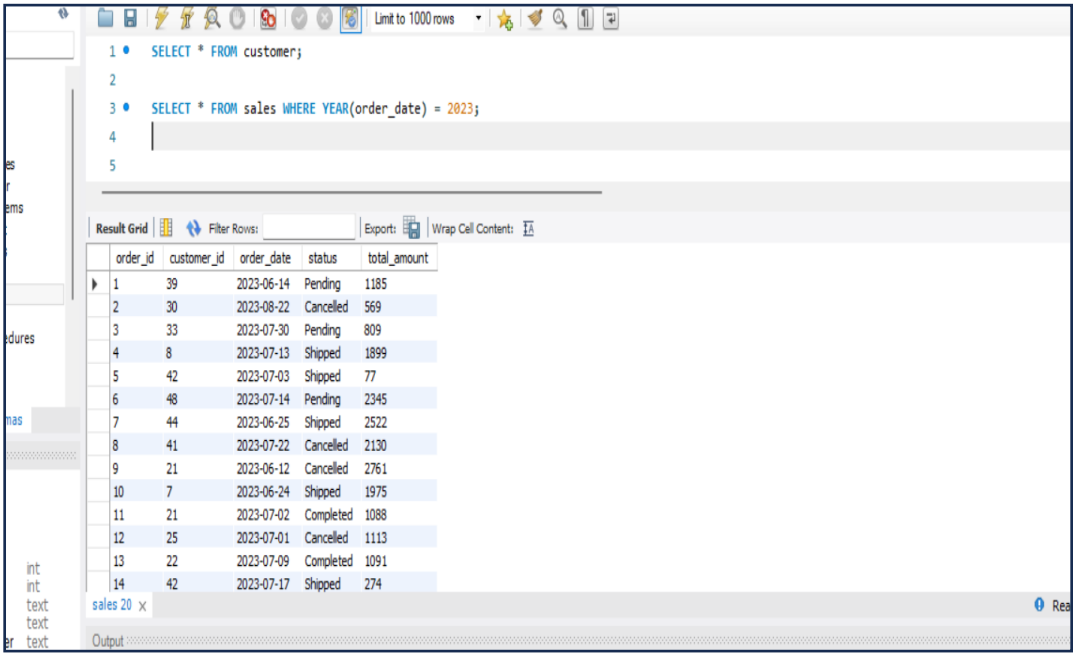
1. List all customers

SELECT * FROM customer;



2. Find order_sales placed in 2023

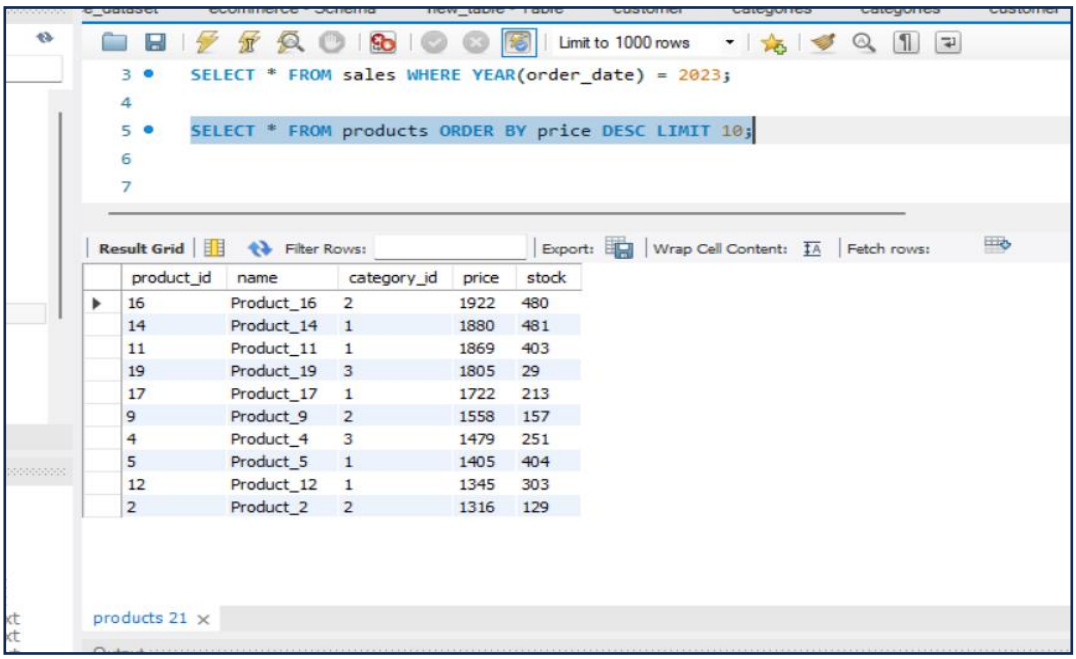
SELECT * FROM sales WHERE YEAR(order_date) = 2023;



order_id	customer_id	order_date	status	total_amount
1	39	2023-06-14	Pending	1185
2	30	2023-08-22	Cancelled	569
3	33	2023-07-30	Pending	809
4	8	2023-07-13	Shipped	1899
5	42	2023-07-03	Shipped	77
6	48	2023-07-14	Pending	2345
7	44	2023-06-25	Shipped	2522
8	41	2023-07-22	Cancelled	2130
9	21	2023-06-12	Cancelled	2761
10	7	2023-06-24	Shipped	1975
11	21	2023-07-02	Completed	1088
12	25	2023-07-01	Cancelled	1113
13	22	2023-07-09	Completed	1091
14	42	2023-07-17	Shipped	274

3. Top 10 most expensive products

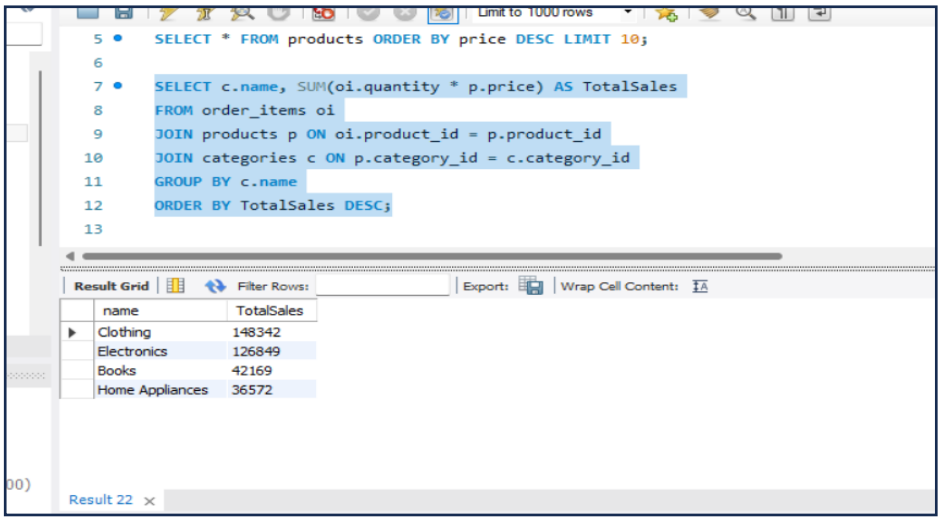
SELECT * FROM products ORDER BY price DESC LIMIT 10;



product_id	name	category_id	price	stock
16	Product_16	2	1922	480
14	Product_14	1	1880	481
11	Product_11	1	1869	403
19	Product_19	3	1805	29
17	Product_17	1	1722	213
9	Product_9	2	1558	157
4	Product_4	3	1479	251
5	Product_5	1	1405	404
12	Product_12	1	1345	303
2	Product_2	2	1316	129

4. Total sales per category

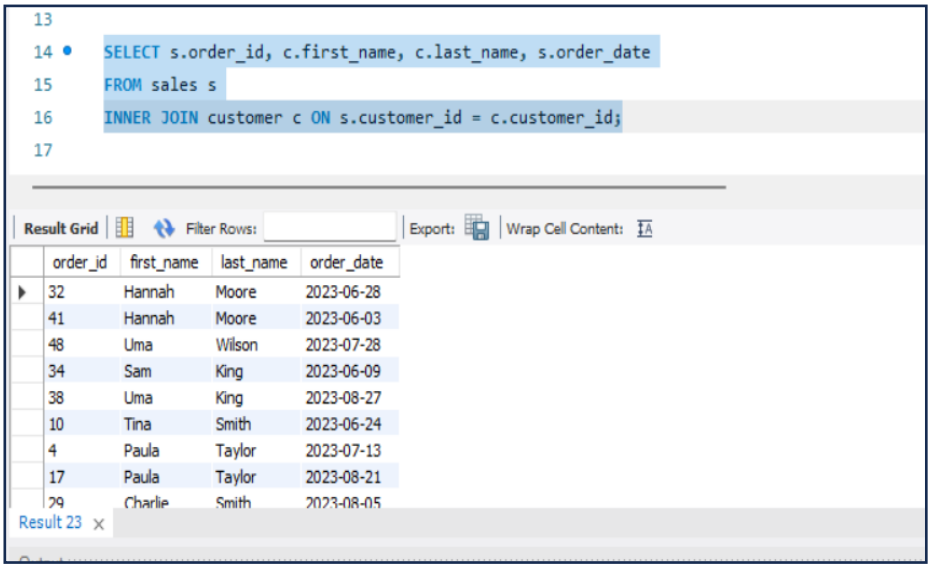
```
SELECT c.name, SUM(oi.quantity * p.price) AS TotalSales
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.name
ORDER BY TotalSales DESC;
```



<u>-- b. JOINS (INNER, LEFT, RIGHT)</u>

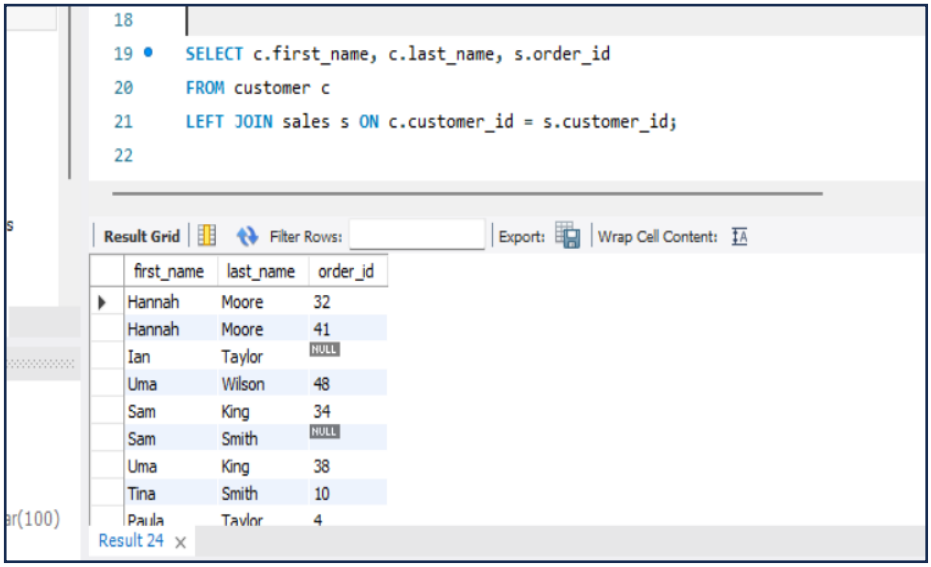
5. INNER JOIN: Orders with customer details

```
SELECT s.order_id, c.first_name, c.last_name, s.order_date
FROM sales s
INNER JOIN customer c ON s.customer_id = c.customer_id;
```



6. LEFT JOIN: All customers and their orders (even if no orders)

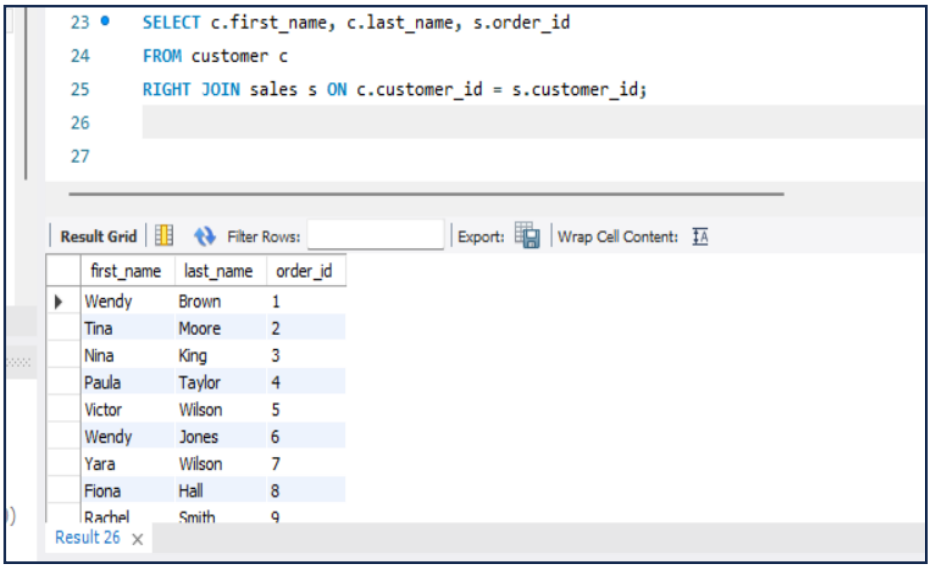
```
SELECT c.first_name, c.last_name, s.order_id
FROM customer c
LEFT JOIN sales s ON c.customer_id = s.customer_id;
```



first_name	last_name	order_id
Hannah	Moore	32
Hannah	Moore	41
Ian	Taylor	NULL
Uma	Wilson	48
Sam	King	34
Sam	Smith	NULL
Uma	King	38
Tina	Smith	10
Paula	Taylor	4

7. RIGHT JOIN: All orders, even if no customer record

```
SELECT c.first_name, c.last_name, s.order_id
FROM customer c
RIGHT JOIN sales s ON c.customer_id = s.customer_id;
```

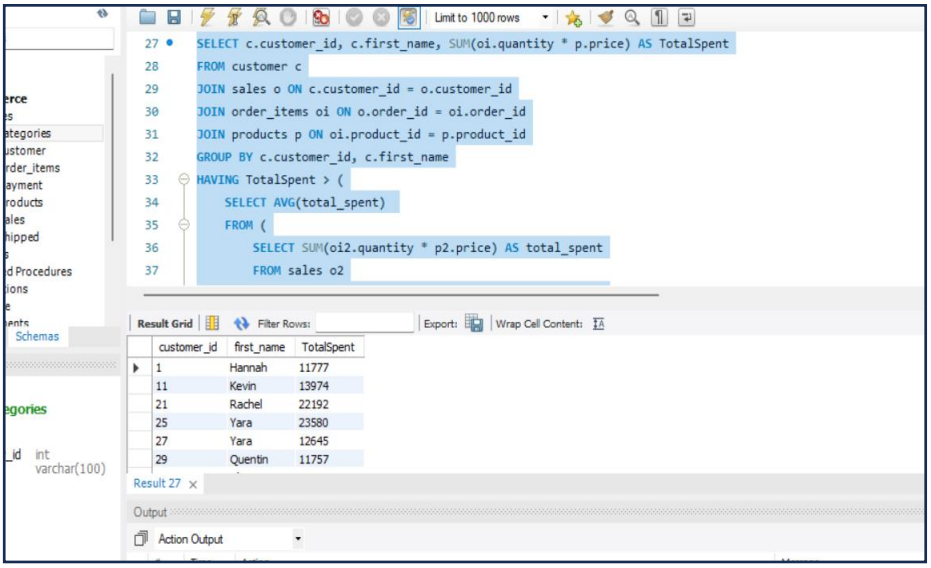


first_name	last_name	order_id
Wendy	Brown	1
Tina	Moore	2
Nina	King	3
Paula	Taylor	4
Victor	Wilson	5
Wendy	Jones	6
Yara	Wilson	7
Fiona	Hall	8
Rachel	Smith	9

-- c. Subqueries

8. Customers who spent above average

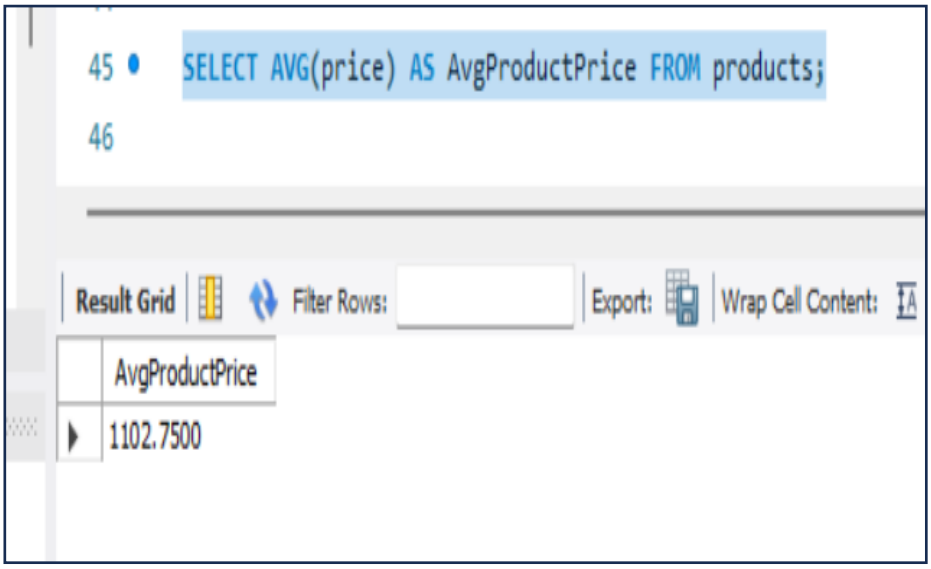
```
SELECT c.customer_id, c.first_name, SUM(oi.quantity * p.price) AS TotalSpent
FROM customer c
JOIN sales o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY c.customer_id, c.first_name
HAVING TotalSpent > (
   SELECT AVG(total_spent)
   FROM (
      SELECT SUM(oi2.quantity * p2.price) AS total_spent
      FROM sales o2
      JOIN order_items oi2 ON o2.order_id = oi2.order_id
      JOIN products p2 ON oi2.product_id = p2.product_id
      GROUP BY o2.customer_id
   ) AS avg_sub
);
```



-- d. Aggregate functions (SUM, AVG, COUNT, MAX, MIN)

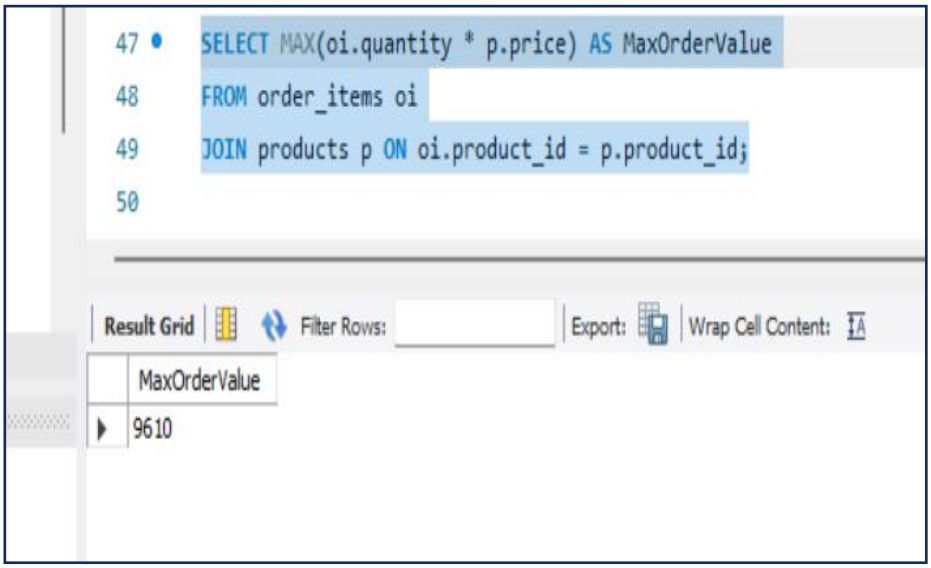
9. Average product price

```
SELECT AVG(price) AS AvgProductPrice FROM products;
```



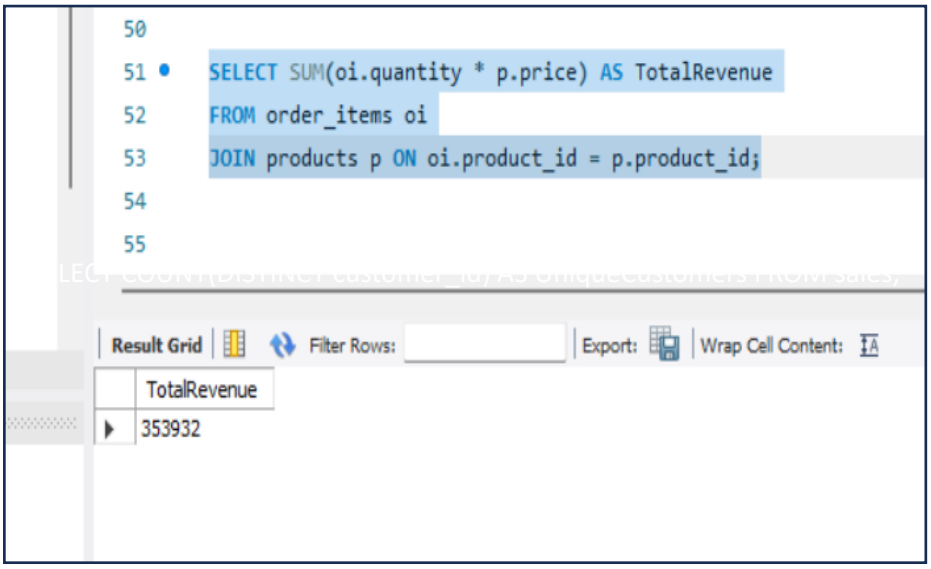
10. Maximum order amount

SELECT MAX(oi.quantity * p.price) AS MaxOrderValue
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id;



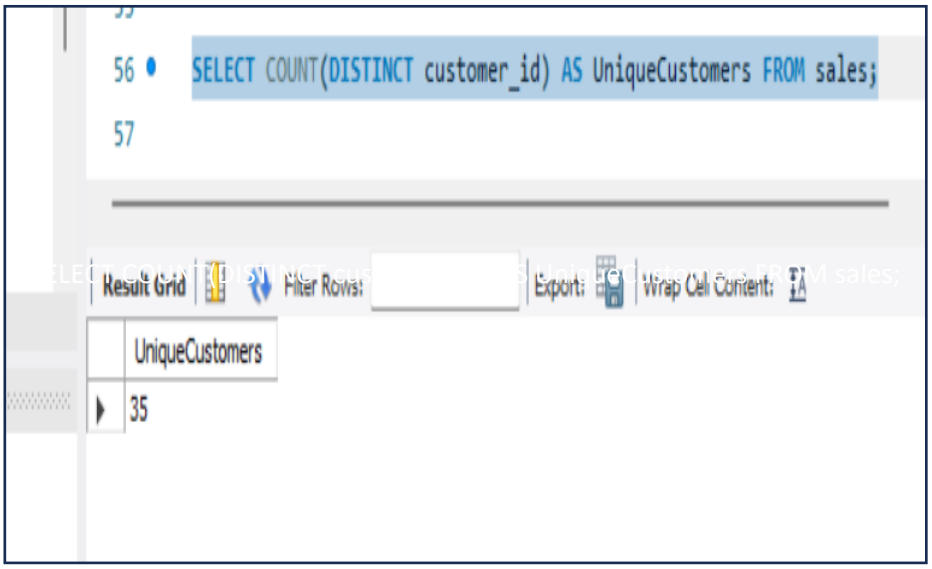
11. Total revenue

SELECT SUM(oi.quantity * p.price) AS TotalRevenue
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id;



12. Number of unique customers

SELECT COUNT(DISTINCT customer_id) AS UniqueCustomers FROM sales;



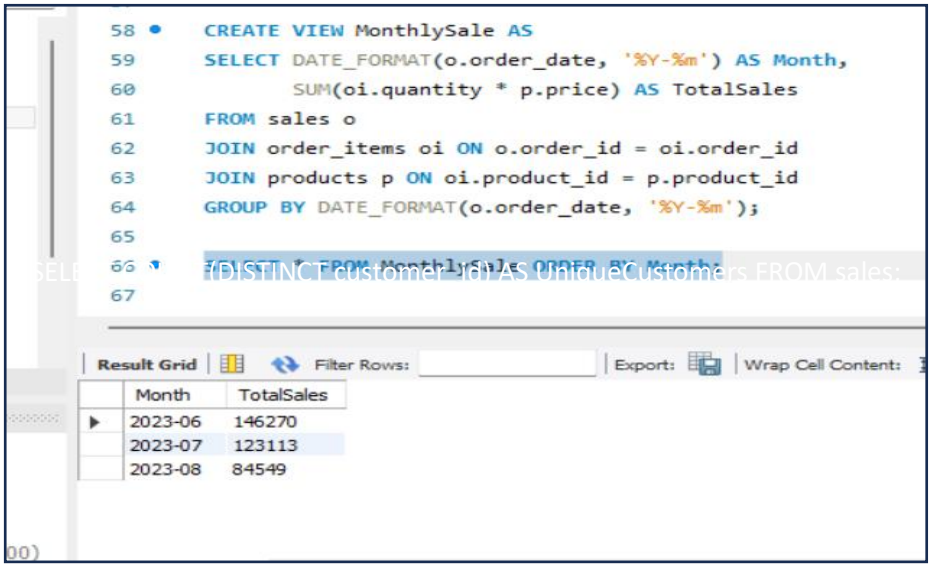
<u>-- e. Views for analysis</u>

13. Create a view for monthly sales

```
CREATE VIEW MonthlySale AS
SELECT DATE_FORMAT(o.order_date, '%Y-%m') AS Month,
    SUM(oi.quantity * p.price) AS TotalSales
FROM sales o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY DATE_FORMAT(o.order_date, '%Y-%m');
```

14. Use the view

SELECT * FROM MonthlySale ORDER BY Month;



<u>-- f. Optimize queries with indexes</u>

15. Index on customer_id for faster joins

CREATE INDEX idx_customer ON orders(customer_id);