

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 :

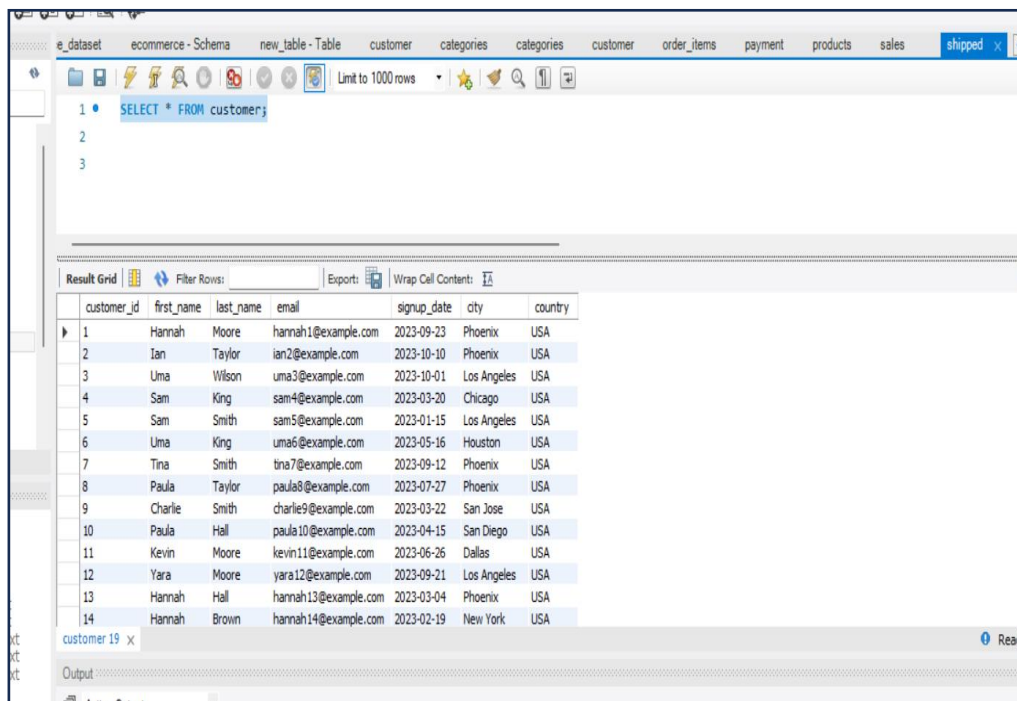
1. In MySQL Workbench:
 - Right-click the **target table** → **Table Data Import Wizard**.
 - Select the corresponding CSV.
 - Click **Next** → check columns → **Next** → **Import**.
2. Confirm that data is loaded correctly.
3. Repeat for all sheets
 - 3.1 One CSV → one table.
 - 3.2 If you have 50+ 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)

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

1. List all customers

SELECT * FROM customer;

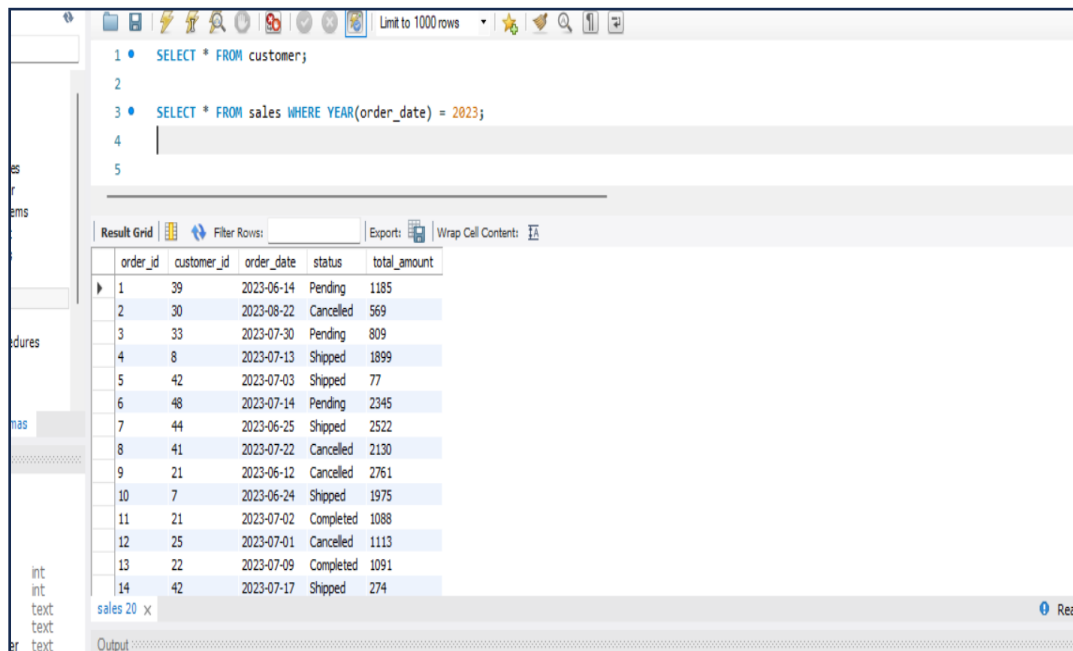


The screenshot shows the MySQL Workbench interface. The SQL editor at the top contains the query `SELECT * FROM customer;`. Below the editor, the 'Result Grid' tab is active, displaying the results of the query. The results are shown in a table with 7 columns: `customer_id`, `first_name`, `last_name`, `email`, `signup_date`, `city`, and `country`. There are 14 rows of data. The interface also shows a schema browser on the left with a tree view containing 'e_dataset', 'ecommerce - Schema', and 'new_table - Table'. The 'customer' table is selected under 'ecommerce - Schema'. The 'Output' and 'Action Output' tabs are visible at the bottom.

customer_id	first_name	last_name	email	signup_date	city	country
1	Hannah	Moore	hannah1@example.com	2023-09-23	Phoenix	USA
2	Ian	Taylor	ian2@example.com	2023-10-10	Phoenix	USA
3	Uma	Wilson	uma3@example.com	2023-10-01	Los Angeles	USA
4	Sam	King	sam4@example.com	2023-03-20	Chicago	USA
5	Sam	Smith	sam5@example.com	2023-01-15	Los Angeles	USA
6	Uma	King	uma6@example.com	2023-05-16	Houston	USA
7	Tina	Smith	tina7@example.com	2023-09-12	Phoenix	USA
8	Paula	Taylor	paula8@example.com	2023-07-27	Phoenix	USA
9	Charlie	Smith	charlie9@example.com	2023-03-22	San Jose	USA
10	Paula	Hall	paula10@example.com	2023-04-15	San Diego	USA
11	Kevin	Moore	kevin11@example.com	2023-06-26	Dallas	USA
12	Yara	Moore	yara12@example.com	2023-09-21	Los Angeles	USA
13	Hannah	Hall	hannah13@example.com	2023-03-04	Phoenix	USA
14	Hannah	Brown	hannah14@example.com	2023-02-19	New York	USA

2. Find order_sales placed in 2023

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

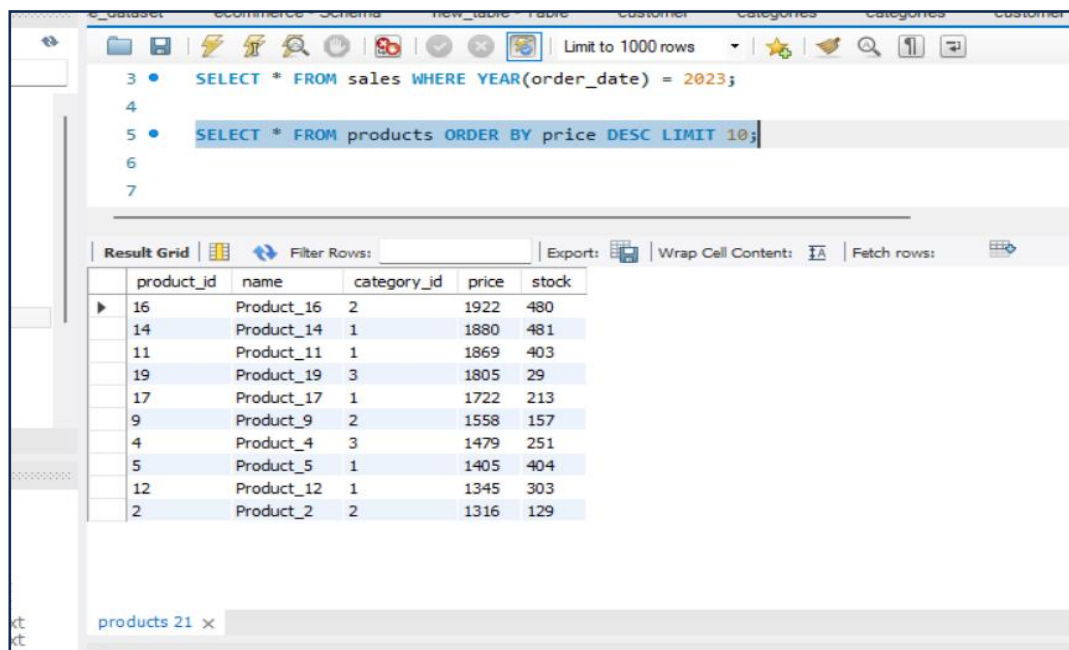


Result Grid

	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;
```

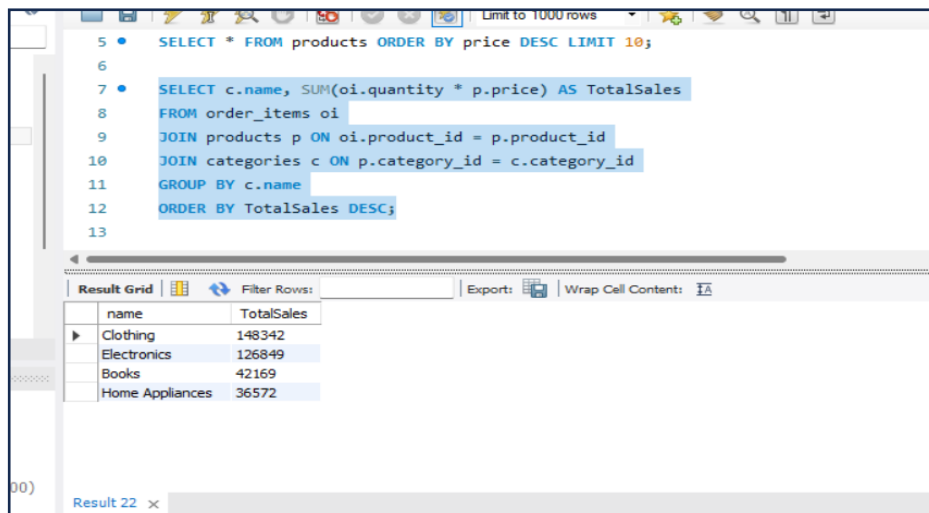


Result Grid

	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;
```



The screenshot shows a SQL query editor with the following query:

```
5 • SELECT * FROM products ORDER BY price DESC LIMIT 10;
6
7 • SELECT c.name, SUM(oi.quantity * p.price) AS TotalSales
8 FROM order_items oi
9 JOIN products p ON oi.product_id = p.product_id
10 JOIN categories c ON p.category_id = c.category_id
11 GROUP BY c.name
12 ORDER BY TotalSales DESC;
13
```

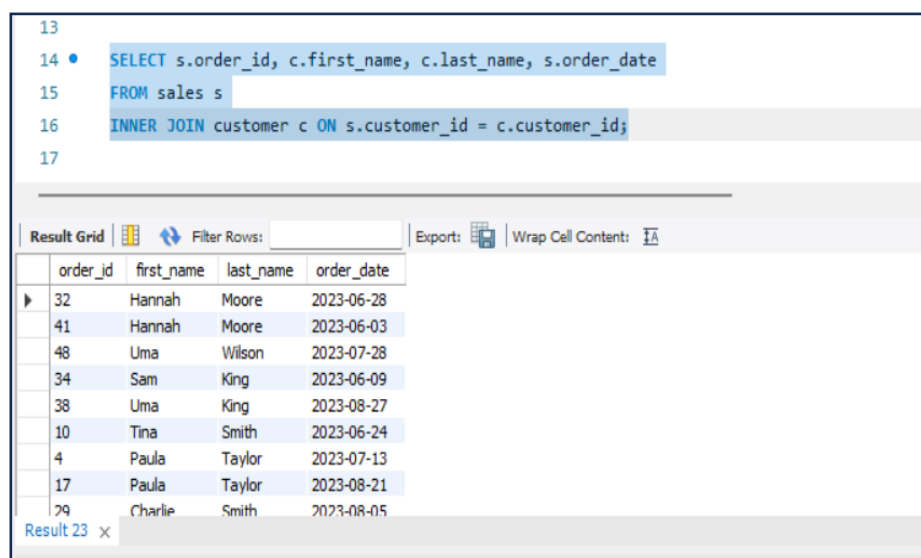
Below the query editor is the 'Result Grid' showing the results of the query:

name	TotalSales
Clothing	148342
Electronics	126849
Books	42169
Home Appliances	36572

-- b. JOINS (INNER, LEFT, RIGHT)

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;
```



The screenshot shows a SQL query editor with the following query:

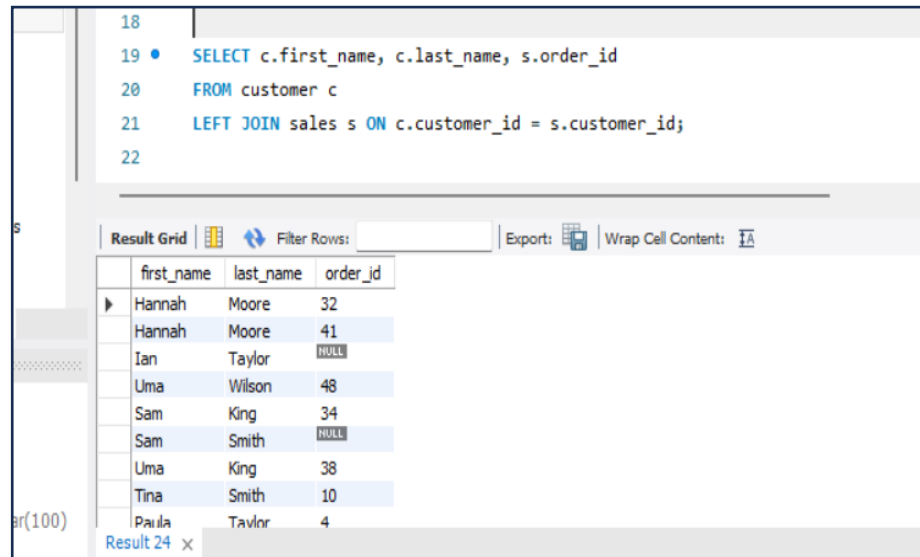
```
13
14 • SELECT s.order_id, c.first_name, c.last_name, s.order_date
15 FROM sales s
16 INNER JOIN customer c ON s.customer_id = c.customer_id;
17
```

Below the query editor is the 'Result Grid' showing the results of the query:

order_id	first_name	last_name	order_date
32	Hannah	Moore	2023-06-28
41	Hannah	Moore	2023-06-03
48	Uma	Wilson	2023-07-28
34	Sam	King	2023-06-09
38	Uma	King	2023-08-27
10	Tina	Smith	2023-06-24
4	Paula	Taylor	2023-07-13
17	Paula	Taylor	2023-08-21
29	Charlie	Smith	2023-08-05

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;
```



The screenshot shows a SQL query editor with the following query:

```
18  
19 • SELECT c.first_name, c.last_name, s.order_id  
20 FROM customer c  
21 LEFT JOIN sales s ON c.customer_id = s.customer_id;  
22
```

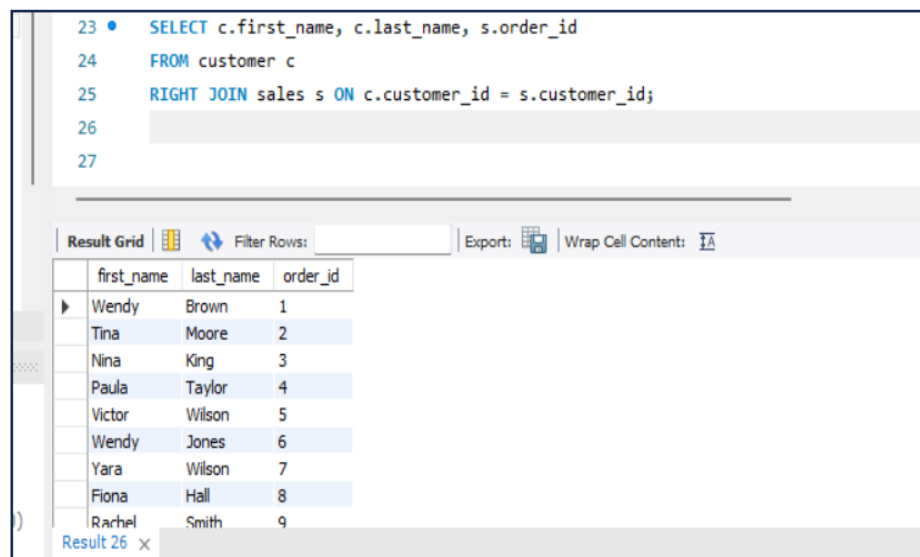
Below the query editor is a "Result Grid" showing the results of the query. The grid has columns for first_name, last_name, and order_id. The results are as follows:

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

The result grid also includes a "Filter Rows:" field, an "Export:" button, and a "Wrap Cell Content:" checkbox.

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;
```



The screenshot shows a SQL query editor with the following query:

```
23 • SELECT c.first_name, c.last_name, s.order_id  
24 FROM customer c  
25 RIGHT JOIN sales s ON c.customer_id = s.customer_id;  
26  
27
```

Below the query editor is a "Result Grid" showing the results of the query. The grid has columns for first_name, last_name, and order_id. The results are as follows:

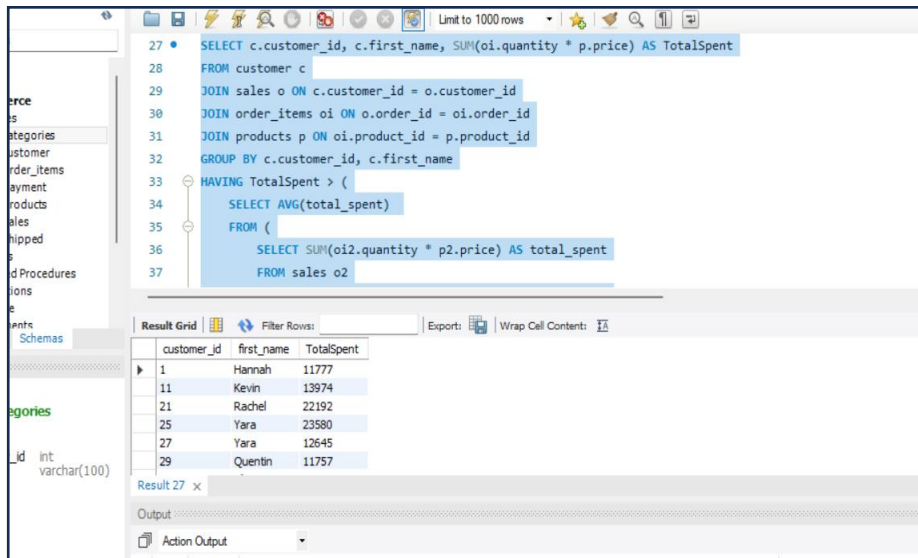
first_name	last_name	order_id
Wendy	Brown	1
Tina	Moore	2
Nina	King	3
Paula	Taylor	4
Victor	Wilson	5
Wendy	Jones	6
Yara	Wilson	7
Fiona	Hall	8
Rachel	Smith	9

The result grid also includes a "Filter Rows:" field, an "Export:" button, and a "Wrap Cell Content:" checkbox.

-- 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
);
```



The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
27 SELECT c.customer_id, c.first_name, SUM(oi.quantity * p.price) AS TotalSpent
28 FROM customer c
29 JOIN sales o ON c.customer_id = o.customer_id
30 JOIN order_items oi ON o.order_id = oi.order_id
31 JOIN products p ON oi.product_id = p.product_id
32 GROUP BY c.customer_id, c.first_name
33 HAVING TotalSpent > (
34     SELECT AVG(total_spent)
35     FROM (
36         SELECT SUM(oi2.quantity * p2.price) AS total_spent
37         FROM sales o2
38         JOIN order_items oi2 ON o2.order_id = oi2.order_id
39         JOIN products p2 ON oi2.product_id = p2.product_id
40         GROUP BY o2.customer_id
41     ) AS avg_sub
42 );
```

The result grid shows the following data:

customer_id	first_name	TotalSpent
1	Hannah	11777
11	Kevin	13974
21	Rachel	22192
25	Yara	23580
27	Yara	12645
29	Quentin	11757

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

9. Average product price

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

45	•	SELECT AVG(price) AS AvgProductPrice FROM products;
46		
Result Grid		
		AvgProductPrice
		1102.7500

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;
```

47	•	SELECT MAX(oi.quantity * p.price) AS MaxOrderValue
48		FROM order_items oi
49		JOIN products p ON oi.product_id = p.product_id;
50		
Result Grid		
		MaxOrderValue
		9610

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;
```

```
50
51 • SELECT SUM(oi.quantity * p.price) AS TotalRevenue
52 FROM order_items oi
53 JOIN products p ON oi.product_id = p.product_id;
54
55
```

Result Grid

TotalRevenue
353932

12. Number of unique customers

SELECT COUNT(DISTINCT customer_id) AS UniqueCustomers FROM sales;

```
56 • SELECT COUNT(DISTINCT customer_id) AS UniqueCustomers FROM sales;
57
```

Result Grid

UniqueCustomers
35

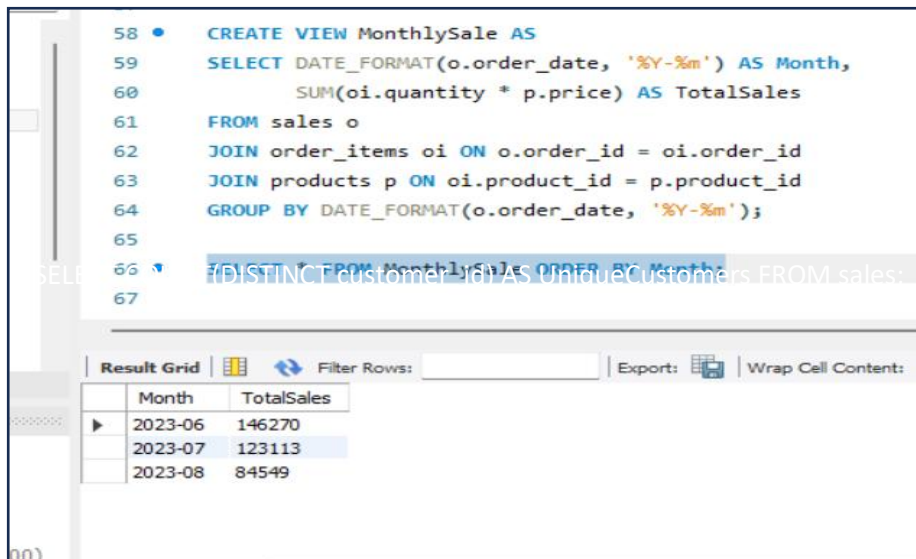
-- e. Views for analysis

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;



The screenshot shows a SQL IDE with a script editor and a result grid. The script editor contains the following SQL code:

```
58 • CREATE VIEW MonthlySale AS
59 SELECT DATE_FORMAT(o.order_date, '%Y-%m') AS Month,
60        SUM(oi.quantity * p.price) AS TotalSales
61 FROM sales o
62 JOIN order_items oi ON o.order_id = oi.order_id
63 JOIN products p ON oi.product_id = p.product_id
64 GROUP BY DATE_FORMAT(o.order_date, '%Y-%m');
65
66 SELECT * FROM MonthlySale ORDER BY Month;
67
```

The result grid displays the following data:

Month	TotalSales
2023-06	146270
2023-07	123113
2023-08	84549

-- f. Optimize queries with indexes

15. Index on customer_id for faster joins

CREATE INDEX idx_customer ON orders(customer_id);