

SQL for Data Analysis Report

Name: Ashish Giri

Task: Task 3 – SQL for Data Analysis

Tools Used: MySQL Workbench

Objective:

To extract insights from the e-commerce database using SQL queries and advanced concepts such as joins, subqueries, aggregate functions, views, and indexes.

1. Query: Active Users List

```
SELECT id, name, email
```

```
FROM users
```

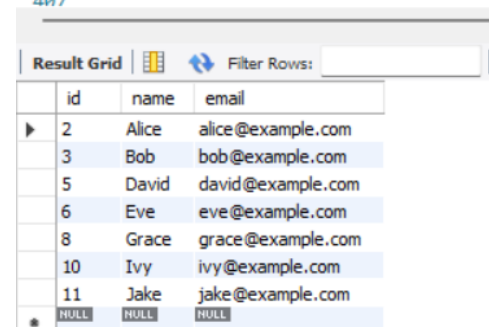
```
WHERE status = 'active'
```

```
ORDER BY created_at DESC;
```

Purpose: Fetch all active users ordered by their registration date (latest first).

Output:

```
403 • SELECT id, name, email
404 FROM users
405 WHERE status = 'active'
406 ORDER BY created_at DESC;
407
```



The screenshot shows the MySQL Workbench interface. The top part displays the SQL query: `SELECT id, name, email FROM users WHERE status = 'active' ORDER BY created_at DESC;`. Below the query editor, the 'Result Grid' tab is active, showing the results of the query. The results are displayed in a table with three columns: 'id', 'name', and 'email'. The table contains 11 rows of data, ordered by 'created_at' in descending order. The first row has id 2, name Alice, and email alice@example.com. The last row has id 11, name Jake, and email jake@example.com. The bottom row shows NULL values for all three columns.

	id	name	email
▶	2	Alice	alice@example.com
	3	Bob	bob@example.com
	5	David	david@example.com
	6	Eve	eve@example.com
	8	Grace	grace@example.com
	10	Ivy	ivy@example.com
	11	Jake	jake@example.com
*	NULL	NULL	NULL

2. Query: Orders by Payment Method

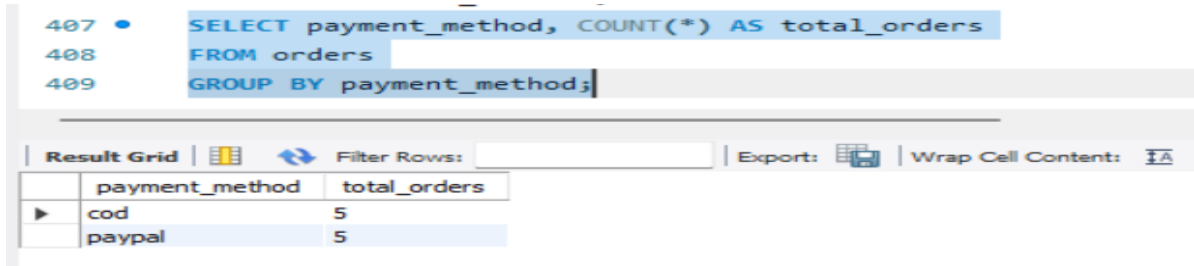
```
SELECT payment_method, COUNT(*) AS total_orders
```

```
FROM orders
```

```
GROUP BY payment_method;
```

Purpose: Show total number of orders for each payment method.

Output:



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

```
407 • SELECT payment_method, COUNT(*) AS total_orders
408 FROM orders
409 GROUP BY payment_method;
```

Below the editor is a 'Result Grid' tab. The grid has two columns: 'payment_method' and 'total_orders'. The first row is expanded, showing a sub-grid with two rows: 'cod' and 'paypal', both with a 'total_orders' value of 5.

	payment_method	total_orders
▶	cod	5
	paypal	5

3. Query: User with Most Orders

```
SELECT name, email
```

```
FROM users
```

```
WHERE id = (
```

```
    SELECT user_id
```

```
    FROM orders
```

```
    GROUP BY user_id
```

```
    ORDER BY COUNT(*) DESC
```

```
    LIMIT 1
```

```
);
```

Purpose: Identify the user who placed the most orders.

Output:

```
410 • SELECT c.title AS category, AVG(p.discount) AS avg_discount
411 FROM products p
412 JOIN categories c ON p.cat_id = c.id
413 GROUP BY c.title;
```

category	avg_discount
Mobiles	11.333333333333334
Laptops	15
Footwear	4
Cameras	8
Clothing	2
Watches	12

4. Query: User Spending Summary

```
SELECT user_id, SUM(total_amount) AS total_spent, AVG(total_amount) AS avg_order
FROM orders
GROUP BY user_id;
```

Purpose: Find total and average order value per user.

Output:

```
:21 • SELECT user_id, SUM(total_amount) AS total_spent, AVG(total_amount) AS avg_order
:22 FROM orders
:23 GROUP BY user_id;
:24
:25
:26
```

user_id	total_spent	avg_order
1	210	210
2	157	157
3	360	360
4	105	105
5	520	520
6	260	260
7	315	315
8	415	415
9	185	185
10	620	620

5. Query: Total Stock by Category

```
SELECT c.title AS category, SUM(p.stock) AS total_stock
FROM products p
```





JOIN categories c ON p.cat_id = c.id

GROUP BY c.id;

Purpose: Aggregate available product stock under each category.

Output:

```
425 • SELECT c.title AS category, SUM(p.stock) AS total_stock
426 FROM products p
427 JOIN categories c ON p.cat_id = c.id
428 GROUP BY c.id;
429
```

Result Grid   Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 		
	category	total_stock
✓	Mobiles	115
	Laptops	35
	Footwear	220
	Cameras	25
	Clothing	200
	Watches	40

6. View: Monthly Revenue View

CREATE VIEW monthly_revenue AS

SELECT DATE_FORMAT(created_at, '%Y-%m') AS month,

SUM(total_amount) AS total_revenue

FROM orders

GROUP BY month;

Purpose: Create a reusable view to show revenue generated monthly.


Follow-up Query:


SELECT * FROM monthly_revenue WHERE month >= '2024-01';


Output:

```
429 • CREATE VIEW monthly_revenue as
430     SELECT DATE_FORMAT(created_at, '%Y-%m') AS month,
431            SUM(total_amount) AS total_revenue
432     FROM orders
433     GROUP BY month;
434 • SELECT * FROM monthly_revenue WHERE month >= '2024-01';
435
```

Result Grid

 Filter Rows:

Export: 

Wrap Cell Content: 

	month	total_revenue
▶	2025-04	3147

7. Indexes for Optimization

CREATE INDEX idx_user_id ON orders(user_id);

CREATE INDEX idx_product_id ON carts(product_id);

CREATE INDEX idx_created_at ON orders(created_at);

Purpose: Improve query performance for filtering and joining on user ID, product ID, and created_at fields.

8. Query: Top 5 Users This Year by Spend

SELECT u.name, u.email, SUM(o.total_amount) AS total_spent

FROM users u

JOIN orders o ON u.id = o.user_id

WHERE YEAR(o.created_at) = YEAR(CURDATE())

GROUP BY u.id

ORDER BY total_spent DESC

LIMIT 5;

Purpose: Get top 5 highest spending users in the current year.

Output:

```
439 • SELECT u.name, u.email, SUM(o.total_amount) AS total_spent
440 FROM users u
441 JOIN orders o ON u.id = o.user_id
442 WHERE YEAR(o.created_at) = YEAR(CURDATE())
443 GROUP BY u.id
444 ORDER BY total_spent DESC
445 LIMIT 5;
446
```

Result Grid			
Filter Rows:		Exports:	Wrap Cell Content: Fetch rows:
	name	email	total_spent
▶	Ivy	ivy@example.com	620
	David	david@example.com	520
	Grace	grace@example.com	415
	Bob	bob@example.com	360
	Frank	frank@example.com	315

9. View: Detailed Order Info

CREATE OR REPLACE VIEW user_order_summary AS

SELECT

u.id AS user_id,

u.name,

o.id AS order_id,

o.total_amount,

o.created_at AS order_date,

c.product_id,

p.brand_id,

b.title AS brand_name

FROM users u

JOIN orders o ON u.id = o.user_id

JOIN carts c ON o.id = c.order_id

JOIN products p ON c.product_id = p.id

JOIN brands b ON p.brand_id = b.id;

Purpose: Create a unified view combining user, order, and brand details.

10. Final Complex Query: Multi-Table, Aggregates, Subqueries, and View Usage

SELECT

u.name AS user_name,

COUNT(DISTINCT o.order_id) AS total_orders,

SUM(o.total_amount) AS total_spent,

COUNT(c.product_id) AS total_products_bought,

MAX(o.order_date) AS last_order_date,

(

SELECT brand_name

FROM (

SELECT brand_name, COUNT(*) AS brand_count

FROM user_order_summary

WHERE user_id = u.id

GROUP BY brand_name

ORDER BY brand_count DESC

LIMIT 1

) AS fav_brand

) AS most_frequent_brand

```

FROM user_order_summary o

JOIN users u ON o.user_id = u.id

JOIN carts c ON o.order_id = c.order_id

GROUP BY u.id

ORDER BY total_spent DESC

LIMIT 5;

```

Purpose: Identify the top 5 customers with most spend, including product count, most purchased brand, and latest order.

Output:

```

35 • CREATE INDEX idx_user_id ON orders(user_id);
36 • CREATE INDEX idx_product_id ON carts(product_id);
37 • CREATE INDEX idx_created_at ON orders(created_at);
38
39 • SELECT u.name, u.email, SUM(o.total_amount) AS total_spent
40 FROM users u
41 JOIN orders o ON u.id = o.user_id
42 WHERE YEAR(o.created_at) = YEAR(CURDATE())
43 GROUP BY u.id
44 ORDER BY total_spent DESC
45 LIMIT 5;
46 -- Step 1: Create a VIEW for detailed order info
47 • CREATE OR REPLACE VIEW user_order_summary AS
48 SELECT
49     u.id AS user_id,
50     u.name,
51     o.id AS order_id,
52     o.total_amount,
53     o.created_at AS order_date,

```

result Grid Filter Rows: <input type="text"/> Export: Wrap Cell Content: Fetch rows:					
user_name	total_orders	total_spent	total_products_bought	last_order_date	most_frequent_brand
Ivy	1	620	1	2025-04-10 19:40:04	Puma
David	1	520	1	2025-04-10 19:40:04	Samsung
Grace	1	415	1	2025-04-10 19:40:04	Dell
Bob	1	360	1	2025-04-10 19:40:04	Asus
Frank	1	315	1	2025-04-10 19:40:04	Reebok

result 17

Reebok

Conclusion:

This report demonstrates the use of SQL to analyze user behavior, product data, and revenue trends using powerful techniques like views, subqueries, joins, and indexing.

Attachments:

- SQL File
- Screenshots of Output