Task 3 – SQL for Data Analysis (Full Submission)

Objective

Use SQL queries to extract and analyze data from an Ecommerce database using SELECT, JOINs, GROUP BY, subqueries, aggregate functions, views, and query optimization techniques.

Tools Used

MySQL / PostgreSQL / SQLite (queries written to be portable across all).

Dataset Description

Tables used:

- users(user_id, email, signup_date, country)

- products(product_id, name, category, price)

- orders(order_id, user_id, order_date, status, total_amount)

- order_items(order_item_id, order_id, product_id, qty, unit_price)

- sessions(session_id, user_id, session_start, session_end, cart_value)

SQL Tasks Completed

1. SELECT, WHERE, ORDER BY queries

2. GROUP BY with SUM, AVG, COUNT aggregates

3. INNER JOIN, LEFT JOIN, RIGHT JOIN equivalents

4. Subqueries (in WHERE and FROM)

5. Window function example (if supported)

6. View creation for analysis

7. Query optimization using indexes

8. NULL handling examples

Key SQL Queries (Summarized)

1. Completed orders in March 2025:

```
SELECT order_id, user_id, order_date, total_amount
FROM orders
WHERE status='completed'
AND order_date BETWEEN '2025-03-01' AND '2025-03-31';
```

2. Monthly revenue:

```
SELECT strftime('%Y-%m', order_date) AS month,
SUM(total_amount) AS revenue
FROM orders
WHERE status='completed'
GROUP BY month;
```

3. INNER JOIN orders + order_items:

```
SELECT o.order_id, oi.product_id, oi.qty
FROM orders o
INNER JOIN order_items oi ON o.order_id=oi.order_id;
```

4. LEFT JOIN users + latest order:

```
SELECT u.user_id, u.email, o.order_id
FROM users u
```

LEFT JOIN orders o ON u.user_id=o.user_id;

5. Subquery – users with lifetime revenue > 100:

SELECT user_id, email

FROM users

WHERE (SELECT SUM(total_amount) FROM orders WHERE user_id=users.user_id) > 100;

6. View: vw_user_revenue

CREATE VIEW vw_user_revenue AS

SELECT u.user_id, u.email,

SUM(o.total_amount) AS lifetime_revenue

FROM users u LEFT JOIN orders o ON u.user_id=o.user_id

GROUP BY u.user_id;

7. ARPU calculation:

SELECT SUM(total_amount) / COUNT(DISTINCT user_id)

FROM orders WHERE status='completed';

Query Optimization Techniques Used

- Indexes created on user_id, order_id, product_id

- Avoided SELECT * in heavy queries

- Used EXPLAIN / QUERY PLAN to inspect performance

Handling NULL Values

- COALESCE(column, 0) used for replacing nulls

- IS NULL used for filtering

Interview Questions & Answers

1. WHERE vs HAVING:

- WHERE filters before GROUP BY; HAVING filters after aggregation.

2. Types of Joins:

- INNER, LEFT, RIGHT, FULL OUTER, CROSS.

3. Average Revenue Per User:

SUM(total_amount)/COUNT(DISTINCT user_id)

4. Subqueries:

Queries nested inside another query in WHERE/FROM/SELECT.

5. Optimizing SQL Queries:

Indexes, EXPLAIN plan, avoiding SELECT *, rewriting subqueries.

6. SQL View:

A saved SELECT query acting as a virtual table.

7. Handling NULL:

Use COALESCE/ISNULL; aggregates ignore NULL.

Submission Checklist

- SQL File uploaded to GitHub

- Dataset schema included

- All queries executed and screenshots taken

- PDF summary (this file)