ECOMMERCE_QUERIES

Owner:	Preeti Lata Biswal
Last Updated:	29 Oct 2025

▢ Steps:-

- **Create a new Database.**

```
CREATE DATABASE ecommerce_analysis;

USE ecommerce_analysis;
```

- **Create a Table.**

```
CREATE TABLE regions (

region_id INT PRIMARY KEY,

region_name VARCHAR(100)

);
```

- CREATE TABLE customers (

```
customer_id INT PRIMARY KEY,

customer_name VARCHAR(100),
```

gender VARCHAR(10),

city VARCHAR(100),

region_id INT,

FOREIGN KEY (region_id) REFERENCES regions(region_id)

);

- CREATE TABLE products (

product_id INT PRIMARY KEY,

product_name VARCHAR(100),

category VARCHAR(50),

unit_price DECIMAL(10,2)

);

- CREATE TABLE orders (

order_id INT PRIMARY KEY,

order_date DATE,

customer_id INT,

payment_mode VARCHAR(50),

total_amount DECIMAL(12,2),

FOREIGN KEY (customer_id) REFERENCES customers(customer_id)

);

- CREATE TABLE order_details (

order_detail_id INT PRIMARY KEY,

```
order_id INT,
product_id INT,
quantity INT,
total_price DECIMAL(12,2),
FOREIGN KEY (order_id) REFERENCES orders(order_id),
FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

▢ Import the Data

You'll load each Excel sheet into its respective SQL table.
Here's how:

✅ **In MySQL Workbench:**

1. **Open Table → Import Wizard**
2. **Select the Excel file → choose the correct sheet**
3. **Map columns → click "Import"**
4. **Repeat for all 5 sheets**

⚠️ **Import order matters because of foreign keys:**

regions
customers
products
orders
order_details

🔍 **Verify Data Imported Correctly**

Run:

```
SELECT COUNT(*) FROM customers;
SELECT COUNT(*) FROM products;
SELECT COUNT(*) FROM orders;
```

```
SELECT COUNT(*) FROM order_details;
```

```
SELECT COUNT(*) FROM regions;
```

If you see ~1000 customers, ~4500 orders, etc., you're good to go ✅

🧠 **Start Analysis (Run SQL Queries)**

You can now start running the analysis queries:-

▢ SQL queries-Question with their answers

Q1. Display all unique cities from the customer table.

```
SELECT DISTINCT city FROM customers;
```

Q2. Count the total number of customers.

```
SELECT COUNT(*) AS total_customers FROM customers;
```

Q3. Find total number of orders placed.

```
SELECT COUNT(order_id) AS total_orders FROM orders;
```

Q4. Show total sales (sum of total_amount).

```
SELECT SUM(total_amount) AS total_sales FROM orders;
```

Q5. List top 5 products with highest unit price.

```
SELECT product_name, unit_price
FROM products
```

```
ORDER BY unit_price DESC
LIMIT 5;
```

Q6. Find total revenue generated by each product category.

```
SELECT p.category, SUM(od.total_price) AS total_revenue
FROM order_details od
JOIN products p ON od.product_id = p.product_id
GROUP BY p.category
ORDER BY total_revenue DESC;
```

Q7. Find the average order value (AOV).

```
SELECT ROUND(AVG(total_amount), 2) AS avg_order_value
FROM orders;
```

Q8. Show monthly total sales for 2024.

```
SELECT
   DATE_FORMAT(order_date, '%Y-%m') AS month,
   SUM(total_amount) AS total_sales
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m')
ORDER BY month;
```

Q9. Find top 10 customers by total purchase value.

```
SELECT
    c.customer_name,
    SUM(o.total_amount) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name
ORDER BY total_spent DESC
LIMIT 10;
```

Q10. Show total orders per payment mode.

```
SELECT payment_mode, COUNT(*) AS total_orders
FROM orders
GROUP BY payment_mode
ORDER BY total_orders DESC;
```

Q11. Rank customers by total purchase using window functions.

```
SELECT
    c.customer_name,
    SUM(o.total_amount) AS total_spent,
    RANK() OVER (ORDER BY SUM(o.total_amount) DESC) AS customer_rank
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name;
```

Q12. Find repeat customers (who ordered more than 3 times).

```
SELECT
    c.customer_name,
    COUNT(o.order_id) AS total_orders
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name
HAVING COUNT(o.order_id) > 3;
```

Q13. Identify best-selling product by region.

```
SELECT
    r.region_name,
    p.product_name,
    SUM(od.total_price) AS total_revenue
FROM order_details od
JOIN orders o ON od.order_id = o.order_id
JOIN products p ON od.product_id = p.product_id
JOIN customers c ON o.customer_id = c.customer_id
JOIN regions r ON c.region_id = r.region_id
GROUP BY r.region_name, p.product_name
ORDER BY r.region_name, total_revenue DESC;
```

Q14. Compare current month vs previous month sales using LAG().

```
SELECT
    DATE_FORMAT(order_date, '%Y-%m') AS month,
    SUM(total_amount) AS total_sales,
    LAG(SUM(total_amount)) OVER (ORDER BY DATE_FORMAT(order_date, '%Y-%m')) AS prev_month_sales,
    (SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY DATE_FORMAT(order_date, '%Y-%m'))) AS sales_diff
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m')
ORDER BY month;
```

Q15. Find the top product category contributing to 80% of total sales (Pareto analysis).

```
WITH category_sales AS (
    SELECT
        p.category,
        SUM(od.total_price) AS revenue
    FROM order_details od
    JOIN products p ON od.product_id = p.product_id
    GROUP BY p.category
),
running_total AS (
    SELECT
        category,
```

```
        revenue,
        SUM(revenue) OVER (ORDER BY revenue DESC) AS cumulative_revenue,
        SUM(revenue) OVER () AS total_revenue
    FROM category_sales
)
SELECT
    category,
    revenue,
    cumulative_revenue,
    ROUND((cumulative_revenue / total_revenue) * 100, 2) AS cumulative_percent
FROM running_total
WHERE (cumulative_revenue / total_revenue) <= 0.8;
```