E-Commerce Data Analysis by SQL

Dataset: E-Commerce Database

1. Dataset Overview

Two sample tables were created for analysis:

1.1 Customers Table

Column	**Description**
customer_id	Unique ID for each customer
customer_name	Name of the customer
email	Customer email address
registration_date	Date when the customer registered
city	City where the customer is located

1.2 Orders Table

Column	**Description**
order_id	Unique ID for each order
customer_id	References customer_id in Customers table
order_date	Date when the order was placed
total_amount	Total order amount ($)
status	Order status (e.g., "Completed", "Pending")

2. SQL Queries & Analysis

Query 1: Get all customers from New York

```
SELECT * FROM Customers

WHERE city = 'New York';
```

Purpose: Filters customers based on location.

Query 2: Get orders above $100 (sorted high to low)

```
SELECT order_id, customer_id, total_amount
FROM Orders
WHERE total_amount > 100
ORDER BY total_amount DESC;
```

Purpose: Identifies high-value orders.

2.2 GROUP BY & Aggregations

Query 3: Average order amount by city

```
SELECT c.city, AVG(o.total_amount) as avg_order_amount
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
GROUP BY c.city;
```

Purpose: Finds which cities have the highest spending customers.

Query 4: Count of orders by status

```
SELECT status, COUNT(*) as order_count
FROM Orders
GROUP BY status;
```

Purpose: Shows order completion rates.

2.3 JOIN Operations

Query 5: Customer details with their orders (INNER JOIN)

```
SELECT c.customer_name, o.order_id, o.order_date, o.total_amount
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id;
```

Purpose: Combines customer and order data.

Query 6: Customers with no orders (LEFT JOIN)

```
SELECT c.customer_name, c.email
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
```

```
WHERE o.order_id IS NULL;
```

Purpose: Identifies inactive customers.

2.4 Subqueries

Query 7: Customers who placed orders above average amount

```
SELECT customer_name, email
FROM Customers
WHERE customer_id IN (
   SELECT customer_id
   FROM Orders
   WHERE total_amount > (SELECT AVG(total_amount) FROM Orders)
);
```

Purpose: Targets high-spending customers.

Query 8: Orders from the most active city

```
SELECT order_id, total_amount, order_date
FROM Orders
WHERE customer_id IN (
   SELECT customer_id
   FROM Customers
   WHERE city = (
      SELECT city
      FROM Customers
      GROUP BY city
      ORDER BY COUNT(*) DESC
      LIMIT 1
   )
);
```

Purpose: Analyzes orders from the city with the most customers.

2.5 Views for Analysis

Query 9: Create a Customer Order Summary View

```
CREATE VIEW CustomerOrderSummary AS
SELECT
   c.customer_id,
   c.customer_name,
   c.city,
   COUNT(o.order_id) as total_orders,
   SUM(o.total_amount) as total_spent,
   AVG(o.total_amount) as avg_order_amount
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.customer_name, c.city;
```

Usage:

```
SELECT * FROM CustomerOrderSummary ORDER BY total_spent DESC;
```

Purpose: Simplifies repeated customer analysis.

2.6 Query Optimization with Indexes

Query 10: Create indexes for performance

```
CREATE INDEX idx_customer_city ON Customers(city);
CREATE INDEX idx_order_customer ON Orders(customer_id);
CREATE INDEX idx_order_amount ON Orders(total_amount);
```

Purpose: Speeds up filtering and JOIN operations.

Query 11: Check query performance

```
EXPLAIN ANALYZE
SELECT c.customer_name, o.order_date, o.total_amount
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE c.city = 'Chicago' AND o.total_amount > 50;
```

Purpose: Measures the impact of indexing.