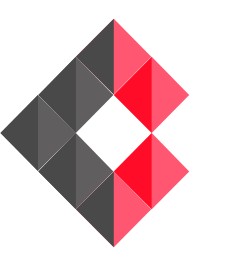
**Internship Task** - **RDBMS and SQL Task #3**



The case will be based on a hypothetical business scenario involving a retail store's database. The database contains tables for Customers, Orders, Products, OrderDetails, and Payments.

**Database Schema Customers Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| customer\_id | INT | Primary key |
| first\_name | VARCHAR(100) | Customer's first name |
| last\_name | VARCHAR(100) | Customer's last name |
| email | VARCHAR(100) | Customer's email address |
| phone | VARCHAR(20) | Customer's phone number |
| address | TEXT | Customer's address |
| join\_date | DATE | Date customer joined |

**Products Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| product\_id | INT | Primary key |
| product\_name | VARCHAR(100) | Product name |
| category | VARCHAR(50) | Product category |
| price | DECIMAL(10,2) | Product price |
| stock\_quantity | INT | Quantity available in stock |

**Orders Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| order\_id | INT | Primary key |
| customer\_id | INT | Foreign key (references  Customers) |
| order\_date | DATE | Date order was placed |
| total\_amount | DECIMAL(10,2) | Total order amount |
| order\_status | VARCHAR(20) | Order status (e.g., 'Pending',  'Shipped') |

**OrderDetails Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| order\_detail\_id | INT | Primary key |
| order\_id | INT | Foreign key (references Orders) |
| product\_id | INT | Foreign key (references Products) |
| quantity | INT | Quantity of product in the order |
| unit\_price | DECIMAL(10,2) | Price per product |

**Payments Table:**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| payment\_id | INT | Primary key |
| order\_id | INT | Foreign key (references  Orders) |
| payment\_date | DATE | Date the payment was made |
| payment\_amoun  t | DECIMAL(10,2) | Amount paid |
| payment\_metho d | VARCHAR(20) | Payment method (e.g., 'Credit  Card', 'PayPal') |

**Case Study: Retail Store Analysis Background:**

You are working as a database analyst for a retail store. The store has multiple customers who place orders for various products. The store wants to generate several reports to analyze customer behavior, sales trends, and payment methods.

**SQL Queries for the Case Study**

1. **Find the Total Number of Orders for Each Customer**

**Ans :**

SELECT customer.first\_name, customer.last\_name, COUNT(orderTable.order\_id) AS total\_orders

FROM Customers customer

LEFT JOIN Orders orderTable ON customer.customer\_id = orderTable.customer\_id

GROUP BY customer.customer\_id, customer.first\_name, customer.last\_name;

1. **Find the Total Sales Amount for Each Product (Revenue per Product)**

**Ans :**

SELECT product.product\_name, SUM(orderDetail.quantity \* orderDetail.unit\_price) AS total\_revenue

FROM Products product

JOIN OrderDetails orderDetail ON product.product\_id = orderDetail.product\_id

GROUP BY product.product\_id, product.product\_name;

1. **Find the Most Expensive Product Sold**

**Ans :**

SELECT product.product\_name

FROM Products product

JOIN OrderDetails orderDetail ON product.product\_id = orderDetail.product\_id

ORDER BY orderDetail.unit\_price DESC

LIMIT 1;

1. **Get the List of Customers Who Have Placed Orders in the Last 30 Days**

**Ans :**

SELECT customer.first\_name, customer.last\_name

FROM Customers customer

JOIN Orders orderTable ON customer.customer\_id = orderTable.customer\_id

WHERE orderTable.order\_date >= DATE('now', '-30 days'); -- SQLite syntax

-- For MySQL: WHERE orderTable.order\_date >= DATE\_SUB(CURDATE(), INTERVAL 30 DAY);

-- For SQL Server: WHERE orderTable.order\_date >= DATEADD(day, -30, GETDATE());

1. **Calculate the Total Amount Paid by Each Customer**

**Ans :**

SELECT customer.first\_name, customer.last\_name, SUM(orderTable.total\_amount) AS total\_spent

FROM Customers customer

JOIN Orders orderTable ON customer.customer\_id = orderTable.customer\_id

GROUP BY customer.customer\_id, customer.first\_name, customer.last\_name;

1. **Get the Number of Products Sold by Category**

**Ans :**

SELECT product.category, SUM(orderDetail.quantity) AS total\_products\_sold

FROM Products product

JOIN OrderDetails orderDetail ON product.product\_id = orderDetail.product\_id

GROUP BY product.category;

1. **List All Orders That Are Pending (i.e., Orders that haven't been shipped yet)**

**Ans :**

SELECT orderTable.order\_id

FROM Orders orderTable

WHERE orderTable.order\_status = 'Pending';

1. **Find the Average Order Value (Total Order Amount / Number of Orders)**

**Ans :**

SELECT AVG(orderTable.total\_amount) AS average\_order\_value

FROM Orders orderTable;

1. **List the Top 5 Customers Who Have Spent the Most Money**

**Ans :**

SELECT customer.first\_name, customer.last\_name, SUM(orderTable.total\_amount) AS total\_spent

FROM Customers customer

JOIN Orders orderTable ON customer.customer\_id = orderTable.customer\_id

GROUP BY customer.customer\_id, customer.first\_name, customer.last\_name

ORDER BY total\_spent DESC

LIMIT 5;

1. **Find the Products That Have Never Been Sold**

**Ans :**

SELECT product.product\_name

FROM Products product

WHERE product.product\_id NOT IN (SELECT DISTINCT orderDetail.product\_id FROM OrderDetails orderDetail);

**Task Summary:**

This SQL case study covers a variety of business analysis queries, including customer insights, product sales, revenue, order status, and payment methods. The queries leverage basic SQL concepts like JOIN, GROUP BY, aggregate functions (SUM(), AVG()), and filtering with WHERE. The goal is to help analyse the store's performance, customer purchasing behaviour, and sales trends.

Each query is designed to answer a specific business question and can be adapted to meet additional reporting needs.