Prepared by: A T M Minhazul Islam
ASTGD, Hi Tech Park, Rajshahi

Practical SQL Queries Using the MySQL Sample Database

Introduction

In this practical session, we will explore the MySQL sample database through various SQL queries. This hands-on approach will help students understand how to retrieve and manipulate data effectively.

---

1. Basic SELECT Statements

1.1 Retrieve All Columns from a Table

Query:

sql

```
SELECT * FROM customers;
```

Explanation:

- Retrieves all columns and rows from the `customers` table.

1.2 Retrieve Specific Columns

Query:

sql

```
SELECT customerNumber, customerName, city, country FROM customers;
```

Explanation:

- Selects specific columns: customer number, name, city, and country.

Prepared by: A T M Minhazul Islam
ASTGD, Hi Tech Park, Rajshahi

2. Filtering Data with WHERE Clause

2.1 Select Customers from a Specific Country

Query:

```sql
SELECT customerName, city, country FROM customers
WHERE country = 'USA';
```

Explanation:

- Filters customers located in the USA.

2.2 Using Comparison Operators

Query:

```sql
SELECT productCode, productName, quantityInStock FROM products
WHERE quantityInStock > 500;
```

Explanation:

- Retrieves products with more than 500 items in stock.

2.3 Using BETWEEN Operator

Query:

```sql
SELECT orderNumber, orderDate, status FROM orders
WHERE orderDate BETWEEN '2023-01-01' AND '2023-12-31';
```

Explanation:

- Selects orders placed within the year 2023.

---

3. Sorting Data with ORDER BY

3.1 Sort Customers Alphabetically

Query:

sql

```
SELECT customerName, country FROM customers
ORDER BY customerName ASC;
```

Explanation:

- Sorts customer names in ascending (A-Z) order.

3.2 Sort Orders by Status

Query:

sql

```
SELECT orderNumber, status FROM orders
ORDER BY status DESC;
```

Explanation:

- Sorts orders by status in descending order.

---

4. Aggregate Functions and GROUP BY

4.1 Count the Number of Customers per Country

Query:

sql

```
SELECT country, COUNT(*) AS totalCustomers FROM customers
GROUP BY country;
```

Explanation:

- Counts how many customers are in each country.

4.2 Calculate Average Quantity in Stock per Product Line

Query:

```sql
SELECT productLine, AVG(quantityInStock) AS avgQuantity FROM products
GROUP BY productLine;
```

Explanation:

- Calculates the average stock quantity for each product line.

---

5. Using HAVING Clause

5.1 Filter Groups After Aggregation

Query:

```sql
SELECT country, COUNT(*) AS totalCustomers FROM customers
GROUP BY country
HAVING COUNT(*) > 5;
```

Explanation:

- Displays countries with more than 5 customers.

---

6. JOINs

6.1 Inner Join Between Customers and Orders

Query:

```sql
SELECT c.customerName, o.orderNumber, o.orderDate FROM customers c
INNER JOIN orders o ON c.customerNumber = o.customerNumber;
```

Explanation:

- Retrieves customers and their corresponding orders.

6.2 Left Join Between Products and Order Details

Query:

sql

```sql
SELECT p.productCode, p.productName, od.quantityOrdered FROM products p
LEFT JOIN orderdetails od ON p.productCode = od.productCode;
```

Explanation:

- Shows all products and their order quantities if they have been ordered.

6.3 Self Join to Find Employees and Their Managers

Query:

sql

```sql
SELECT e.employeeNumber AS EmployeeID, e.firstName AS EmployeeName,
       m.employeeNumber AS ManagerID, m.firstName AS ManagerName
FROM employees e
LEFT JOIN employees m ON e.reportsTo = m.employeeNumber;
```

Explanation:

- Retrieves employees along with their managers' information.

---

7. Subqueries

7.1 Subquery in WHERE Clause

Query:

sql

```sql
SELECT customerName FROM customers
WHERE salesRepEmployeeNumber IN (SELECT employeeNumber FROM employees WHERE
officeCode = '1');
```

Explanation:

- Selects customers served by sales representatives from office number 1.

7.2 Correlated Subquery

Query:

```
sql

SELECT productName, buyPrice FROM products p
WHERE buyPrice > (SELECT AVG(buyPrice) FROM products WHERE productLine =
p.productLine);
```

Explanation:

- Retrieves products priced above the average price within their product line.

---

8. Transactions

8.1 Using Transactions for Atomic Operations

Scenario: Transfer stock quantities between two products.

Queries:

```
sql

START TRANSACTION;

UPDATE products SET quantityInStock = quantityInStock - 50
WHERE productCode = 'S10_1678';

UPDATE products SET quantityInStock = quantityInStock + 50
WHERE productCode = 'S10_1949';

COMMIT;
```

Explanation:

- Decreases stock of one product and increases another atomically.

8.2 Rolling Back a Transaction

Queries:

sql

```sql
START TRANSACTION;

DELETE FROM customers WHERE customerNumber = 999;

ROLLBACK;
```

Explanation:

- Attempts to delete a customer but rolls back the transaction to cancel the operation.

---

9. Advanced SQL Queries

9.1 Using CASE Statements

Query:

sql

```sql
SELECT customerName, country,
       CASE
           WHEN country = 'USA' THEN 'North America'
           WHEN country = 'France' THEN 'Europe'
           ELSE 'Other'
       END AS Region
FROM customers;
```

Explanation:

- Assigns a region based on the country.

9.2 Implementing Ranking with Window Functions (MySQL 8.0+)

Query:

sql

```sql
SELECT customerName, country, creditLimit,
       RANK() OVER (ORDER BY creditLimit DESC) AS CreditRank
FROM customers;
```

Explanation:

- Ranks customers based on their credit limit.

9.3 Using CONCAT and IFNULL Functions

Query:

```
sql

SELECT customerNumber,
       CONCAT(contactFirstName, ' ', contactLastName) AS ContactName,
       IFNULL(phone, 'No Phone') AS PhoneNumber
FROM customers;
```

Explanation:

- Concatenates first and last names and handles NULL phone numbers.

9.4 Calculating Cumulative Sales

Query:

```
sql

SELECT paymentDate, amount,
       SUM(amount) OVER (ORDER BY paymentDate) AS CumulativeSales
FROM payments;
```

Explanation:

- Calculates cumulative sales over time.

---

10. Views

10.1 Creating a View for Easy Access

Query:

```
sql

CREATE VIEW order_summary AS
SELECT o.orderNumber, o.orderDate, c.customerName, SUM(od.quantityOrdered *
od.priceEach) AS totalAmount
FROM orders o
INNER JOIN customers c ON o.customerNumber = c.customerNumber
INNER JOIN orderdetails od ON o.orderNumber = od.orderNumber
GROUP BY o.orderNumber, o.orderDate, c.customerName;
```

Explanation:

- Creates a view summarizing orders with total amounts.

Prepared by: A T M Minhazul Islam
ASTGD, Hi Tech Park, Rajshahi

10.2 Querying the View

Query:

sql

```
SELECT * FROM order_summary WHERE totalAmount > 50000;
```

Explanation:

- Retrieves orders with total amounts exceeding $50,000.