Experiment No 7

Problem Statement:

Perform following SQL queries on the database created in assignment 1.

• Implementation of relational operators in SQL

• Boolean operators and pattern matching

• Arithmetic operations and built in functions

• Group functions

• Processing Date and Time functions

• Complex queries and set operators

Objective:

Understand and practice various SQL queries on the given database. Learn to use relational operators in SQL.

Instructions:

- Open your preferred SQL environment (e.g., MySQL Workbench, SQL Server Management Studio).
- Create a new database (if not already existing) and use it.

Theory:

In SQL (Structured Query Language), operators are symbols or keywords used to perform operations on one or more expressions or values. These operations can include mathematical calculations, comparisons, and logical operations. SQL operators are broadly categorized into several types, each serving a specific purpose. Here are the main types of operators in SQL:

1. **Arithmetic Operators:**
    - + (Addition)
    - - (Subtraction)
    - * (Multiplication)
    - / (Division)
    - % (Modulus)

Example:

1. SELECT salary + (salary * 0.1) AS increased_salary FROM employees;
2. SELECT CURRENT_YEAR - birth_year AS age FROM customers;
3. SELECT quantity * unit_price AS total_revenue FROM orders;
4. SELECT AVG(salary) AS average_salary FROM employees;

2. **Comparison Operators:**

- = (Equal to)
- <> or != (Not equal to)
- < (Less than)
- > (Greater than)
- <= (Less than or equal to)
- >= (Greater than or equal to)

Example:

SELECT * FROM products WHERE price > 100;

---

3. **Logical Operators**:

- AND (Logical AND)
- OR (Logical OR)
- NOT (Logical NOT)

Example:

SELECT * FROM employees WHERE department = 'IT' AND salary > 50000;

---

4. **Concatenation Operator:**

- || (Concatenates two strings)

Example:

SELECT first_name || ' ' || last_name AS full_name FROM employees;

---

5. **Pattern Matching Operators**:

- LIKE (Matches a pattern)

- IN (Matches any value in a set of values)
-

Example:

```
SELECT * FROM customers WHERE last_name LIKE 'Sm%';
```

6. **IS NULL Operator:**
    - IS NULL (Checks if a value is NULL)

Example:

```
SELECT * FROM orders WHERE ship_date IS NULL;
```

7. **BETWEEN Operator:**
    - BETWEEN (Checks if a value is within a range)

Example:

```
SELECT * FROM products WHERE price BETWEEN 50 AND 100;
```

These operators are crucial for constructing SQL queries, enabling users to filter, compare, and manipulate data within a database.

➢ Implementation of relational operators in SQL:

1. **EQUALS (=):**
    - Used to compare if two values are equal.
    - Example: **SELECT * FROM employees WHERE salary = 50000;**

2. **NOT EQUALS (<> or !=):**
    - Used to check if two values are not equal.
    - Examples:
        - **SELECT * FROM products WHERE category <> 'Electronics';**
        - **SELECT * FROM customers WHERE age != 25;**

3. **LESS THAN (<):**
    - Checks if the left operand is less than the right operand.
    - Example: **SELECT * FROM orders WHERE order_date < '2023-01-01';**

4. **GREATER THAN (>):**
    - Checks if the left operand is greater than the right operand.
    - Example: **SELECT * FROM products WHERE price > 100;**

5. **LESS THAN OR EQUAL TO (<=):**
    - Checks if the left operand is less than or equal to the right operand.
    - Example: **SELECT * FROM employees WHERE hire_date <= '2023-01-01';**

6. **GREATER THAN OR EQUAL TO (>=):**
    - Checks if the left operand is greater than or equal to the right operand.
    - Example: **SELECT * FROM orders WHERE total_amount >= 500;**

7. **BETWEEN:**
    - Used to check if a value falls within a specified range.
    - Example: **SELECT * FROM products WHERE price BETWEEN 50 AND 100;**

8. **IN:**
    - Checks if a value matches any value in a specified list.
    - Example: **SELECT * FROM customers WHERE country IN ('USA', 'Canada', 'Mexico');**

9. **LIKE:**
    - Used for pattern matching in strings.
    - Example: **SELECT * FROM employees WHERE last_name LIKE 'Sm%';**

10. **IS NULL:**
    - Checks if a column contains a NULL value.
    - Example: **SELECT * FROM orders WHERE ship_date IS NULL;**

11. **IS NOT NULL:**
    - Checks if a column does not contain a NULL value.
    - Example: **SELECT * FROM products WHERE expiry_date IS NOT NULL;**

➢ Boolean operators and pattern matching:

Boolean operators in SQL are used to combine or modify conditions in queries. The three primary Boolean operators are **AND**, **OR**, and **NOT**.

1. **AND Operator:**
    - The **AND** operator is used to retrieve rows where all specified conditions are true.
    - Example: Retrieve employees who are both in the 'IT' department and have a salary greater than 50,000.

```
SELECT * FROM employees WHERE department = 'IT' AND salary > 50000;
```

2. **OR Operator:**
    - The **OR** operator is used to retrieve rows where at least one of the specified conditions is true.
    - Example: Retrieve employees who are either in the 'HR' department or have a salary greater than 60,000.

```
SELECT * FROM employees WHERE department = 'HR' OR salary > 60000;
```

3. **NOT Operator:**
    - The **NOT** operator is used to retrieve rows where the specified condition is not true.
    - Example: Retrieve employees who are not in the 'Finance' department.

```
SELECT * FROM employees WHERE NOT department = 'Finance';
```

Pattern Matching:

Pattern matching in SQL involves using operators like **LIKE** to filter data based on specific patterns within strings.

1. **LIKE Operator:**
    - The **LIKE** operator is used for pattern matching with wildcard characters.
    - **%** represents zero or more characters, and **_** represents a single character.
    - Example: Retrieve employees whose last names start with "Sm".

```
SELECT * FROM employees WHERE last_name LIKE 'Sm%';
```

2. **IN Operator:**
    - The **IN** operator is not a traditional pattern matching operator, but it allows matching values in a specified list.
    - Example: Retrieve employees from specific departments.

```
SELECT * FROM employees WHERE department IN ('IT', 'HR', 'Marketing');
```

3. **Regular Expressions (REGEXP):**
    - Some database systems support regular expressions for more advanced pattern matching.
    - Example: Retrieve employees with names starting with "A" or "B".

```
SELECT * FROM employees WHERE first_name REGEXP '^[A-B]';
```

➢ Group functions:

1. COUNT():
The **COUNT()** function is used to count the number of rows in a group.

```
SELECT department, COUNT(*) AS employee_count FROM employees GROUP BY
department;
```

2. SUM():
The **SUM()** function calculates the sum of numeric values in a group.

```
SELECT department, SUM(salary) AS total_salary FROM employees GROUP BY
department;
```

3. AVG():
The **AVG()** function calculates the average of numeric values in a group.

```
SELECT department, AVG(salary) AS average_salary FROM employees GROUP BY
department;
```

4. MIN():
The **MIN()** function retrieves the minimum value from a group.

```
SELECT department, MIN(salary) AS lowest_salary FROM employees GROUP BY
department;
```

5. HAVING Clause:
The **HAVING** clause is used in combination with the **GROUP BY** clause to filter the results based on aggregate functions.

```
SELECT department, AVG(salary) AS avg_salary FROM employees GROUP BY
department HAVING AVG(salary) > 50000;
```

➢ Processing Date and Time functions:

Here are some common date and time functions in SQL:

1. CURRENT_DATE:

The **CURRENT_DATE** function retrieves the current date.

SELECT CURRENT_DATE AS current_date;

2. CURRENT_TIME:

The **CURRENT_TIME** function retrieves the current time.

SELECT CURRENT_TIME AS current_time;

3. CURRENT_TIMESTAMP:

The **CURRENT_TIMESTAMP** function retrieves the current date and time.

SELECT CURRENT_TIMESTAMP AS current_datetime;

4. DATEADD():

The **DATEADD()** function adds a specified time interval to a date.

SELECT DATEADD(MONTH, 3, order_date) AS three_months_later FROM orders;

5. DATEDIFF():

The **DATEDIFF()** function calculates the difference between two dates.

SELECT DATEDIFF(DAY, order_date, shipped_date) AS days_to_ship FROM orders;

6. EXTRACT():

The **EXTRACT()** function extracts a specific part of a date-time value.

SELECT EXTRACT(YEAR FROM hire_date) AS hire_year FROM employees;

7. DATE_FORMAT():

The **DATE_FORMAT()** function formats a date according to a specified format.

SELECT order_id, DATE_FORMAT(order_date, '%Y-%m-%d') AS formatted_date FROM orders;

8. NOW():

The **NOW()** function returns the current date and time.

```
SELECT NOW() AS current_datetime;
```

9. TIMESTAMPDIFF():

The **TIMESTAMPDIFF()** function calculates the difference between two date-time values.

```
SELECT TIMESTAMPDIFF(MINUTE, login_time, logout_time) AS session_duration FROM user_sessions;
```

10. YEAR(), MONTH(), DAY():

These functions extract the year, month, or day from a date.

```
SELECT order_id, YEAR(order_date) AS order_year FROM orders;
```

Conclusion:

In conclusion, a solid understanding of the implementation of different operators in SQL is fundamental for anyone working with databases. It empowers users to efficiently retrieve, manipulate, and analyze data, making SQL a powerful language for managing relational databases.