Name:Vaibhav kumar gupta

Date:-11-02-2025

1. Bitwise Operations

Creating the Permissions Table

```
CREATE TABLE permissions (
    user_id INT PRIMARY KEY,
    username VARCHAR(50),
    permission_flags INT  -- Stores permission bits
);
```

Inserting Sample Data

```
INSERT INTO permissions (user_id, username, permission_flags) VALUES
(1, 'admin', 7),     -- Binary: 111 (Read: 1, Write: 1, Execute: 1)
(2, 'developer', 6), -- Binary: 110 (Read: 1, Write: 1, Execute: 0)
(3, 'viewer', 4),    -- Binary: 100 (Read: 1, Write: 0, Execute: 0)
(4, 'guest', 1);     -- Binary: 001 (Read: 0, Write: 0, Execute: 1)
```

Bitwise Operations

Granting Write Permission (010) if Missing

```
UPDATE permissions
SET permission_flags = permission_flags | 2
WHERE (permission_flags & 2) = 0;
```

Uses bitwise OR (|) to add write permission (2) if not already set.

Toggling Execute Permission (001)

```
UPDATE permissions
SET permission_flags = permission_flags ^ 1;
```

Uses bitwise XOR (^) to flip the execute permission (1).

Checking Read Permission (100)

```
SELECT
    user_id,
    username,
    CASE
        WHEN (permission_flags & 4) > 0 THEN 'Has Read Permission'
        ELSE 'No Read Permission'
    END AS ReadPermissionStatus
FROM permissions;
```

Uses bitwise AND (&) to check if Read (4) is enabled.

---

2. Bit Shifting Operations

Creating the Bit Shift Demo Table

```
CREATE TABLE bit_shift_demo (
    id INT PRIMARY KEY,
    value INT
);
```

Inserting Sample Data

```
INSERT INTO bit_shift_demo (id, value) VALUES
(1, 8),   -- Binary: 1000
(2, 12),  -- Binary: 1100
(3, 16);  -- Binary: 10000
```

Bit Shifting Queries

Left Shift (Multiply by 2)

```
SELECT id, value, (value << 1) AS left_shift_1, (value << 2) AS left_shift_2 FROM bit_shift_demo;
```

Left shift (<<) doubles the value for each shift.

Right Shift (Divide by 2)

```
SELECT id, value, (value >> 1) AS right_shift_1, (value >> 2) AS right_shift_2 FROM
bit_shift_demo;
```

Right shift (>>) halves the value for each shift.

---

3. SQL Clauses (NOT, BETWEEN, EXISTS)

Creating Customers and Orders Tables

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Country VARCHAR(50),
    IsActive BIT,
    CreditLimit DECIMAL(10,2)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10,2),
    Status VARCHAR(20)
);
```

Query Examples

Finding Products in Stock (NOT Operator)

```
SELECT * FROM Products WHERE InStock != 0;
```

Retrieves all products that are currently in stock.

Finding Orders within a Specific Range (BETWEEN)

```
SELECT * FROM Orders WHERE TotalAmount BETWEEN 1000 AND 2000;
```

Filters records where TotalAmount falls within the specified range.

Checking if Customers Have Orders (EXISTS)

```
SELECT Name
FROM Customers C
WHERE EXISTS (SELECT 1 FROM Orders O WHERE O.CustomerID = C.CustomerID);
```

Uses EXISTS to return customers who have placed at least one order.

---

4. SQL Joins (INNER, LEFT, RIGHT JOINs)

Creating Employees and Departments Tables

```
CREATE TABLE Employees (EmpID INT PRIMARY KEY, Name VARCHAR(50), DeptID INT);
CREATE TABLE Departments (DeptID INT PRIMARY KEY, DeptName VARCHAR(50));
```

Query Examples

Inner Join (Matching Records Only)

```
SELECT E.Name, D.DeptName
FROM Employees E
INNER JOIN Departments D ON E.DeptID = D.DeptID;
```

Retrieves only the matching records from both tables.

Left Join (All Employees, Even Without a Department)

```
SELECT E.Name, D.DeptName
FROM Employees E
LEFT JOIN Departments D ON E.DeptID = D.DeptID;
```

Includes all employees, even those without a department.

Right Join (All Departments, Even Without Employees)

```
SELECT E.Name, D.DeptName
FROM Employees E
RIGHT JOIN Departments D ON E.DeptID = D.DeptID;
```

Ensures all departments appear, even if they have no assigned employees.

---

5. Ranking Functions (RANK(), DENSE_RANK())

Creating the Employees Table

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100),
   Department VARCHAR(50),
   Salary DECIMAL(10,2)
);
```

Inserting Sample Data

```
INSERT INTO Employees VALUES
(1, 'John Doe', 'HR', 5000),
(2, 'Jane Smith', 'IT', 7000),
(3, 'Alice Brown', 'IT', 7000),
(4, 'Bob Johnson', 'Finance', 6000),
(5, 'Charlie Wilson', 'Finance', 4000);
```

Query Examples

Comparing RANK() and DENSE_RANK()

```
SELECT
   EmployeeID,
   Name,
   Salary,
   RANK() OVER (ORDER BY Salary DESC) AS RankValue,
   DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRankValue
FROM Employees;
```

RANK() skips rankings for duplicate salaries, while DENSE_RANK() assigns continuous ranks.

Using PARTITION BY for Department-wise Ranking

```
SELECT
    EmployeeID,
    Name,
    Department,
    Salary,
    RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS RankValue
FROM Employees;
```

Generates rankings within each department separately.

---

6. LAG() Function

Retrieve Previous Salary Using LAG()

```
SELECT
    EmployeeID,
    Name,
    Salary,
    LAG(Salary, 1, 0) OVER (ORDER BY Salary DESC) AS PreviousSalary
FROM Employees;
```

Fetches the previous salary of each employee in descending order.

Classifying Employees Based on Salary (CASE Statement)

```
SELECT
    Name,
    Salary,
    CASE
        WHEN Salary > 6000 THEN 'High Salary'
        WHEN Salary BETWEEN 4000 AND 6000 THEN 'Medium Salary'
        ELSE 'Low Salary'
    END AS SalaryCategory
FROM Employees;
```