```
# ------- Day 12 Topics ----------#
# -------   Eternaltek   -----------#
# ----- By Vamsidhar Reddy --------#


/*
-- OPERATORS
1)  Arithmetic Operators:
2)  Comparison Operators:
3)  Logical Operators:
4)  Concatenation Operator:
5)  LIKE Operator:
6)  IN Operator:
7)  IS NULL / IS NOT NULL:
8)  BETWEEN Operator:

-- SQL Date Expression
-- SQL Aggregate Functions
-- SQL String Functions
-- Joins
1)  QUERIES and SUB QUERIES
2)  CLAUSES
3)  JOINS
*/

#OPERATORS
/* operators are symbols or keywords used to perform operations on values
and expressions.
They are fundamental components in SQL queries, allowing you to filter,
compare, and manipulate data.
Here are some common types of operators in MySQL:
1)  Arithmetic Operators:
2)  Comparison Operators:
3)  Logical Operators:
4)  Concatenation Operator:
5)  LIKE Operator:
6)  IN Operator:
7)  IS NULL / IS NOT NULL:
8)  BETWEEN Operator:

*/

show databases;
drop database sum;
# Arithmetic Operators:
show databases;

create database sum;
use sum;
-- Creating a sample table
CREATE TABLE numbers (
    a INT,
    b INT
```

```
);

-- Inserting some sample data
INSERT INTO numbers (a, b) VALUES
    (10, 3),
    (15, 7),
    (25, 5);

-- Performing arithmetic operations
SELECT
    a,
    b,
    a + b AS sum_result,
    a - b AS difference_result,
    a * b AS product_result,
    a / b AS division_result,
    a % b AS modulus_result
FROM
    numbers;


# Comparison Operators
CREATE TABLE products (
    product_id INT,
    product_name varchar(30),
    price INT,
    stock_quantity int
);

insert into products (product_id,product_name,price,stock_quantity)
values
(1,'rice',30,40),
(2,'Dal',96,47),
(3,'fruit',70,30);

select * from products;

#Equal to (=)
SELECT * FROM products WHERE price = 30;

#Not equal to (<> or !=):
SELECT * FROM products WHERE product_id != 2;

#Less than (<):
SELECT * FROM products WHERE price < 90;

#Greater than (>):
SELECT * FROM products WHERE stock_quantity > 40;

#Less than or equal to (<=):
SELECT * FROM products WHERE price <= 25;

#Greater than or equal to (>=):
SELECT * FROM products WHERE stock_quantity >= 50;
```

```
# Logical Operators

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    salary INT,
    department VARCHAR(50)
);

INSERT INTO employees VALUES (1, 'John', 'Doe', 50000, 'Sales');
INSERT INTO employees VALUES (2, 'Jane', 'Smith', 60000, 'Marketing');
INSERT INTO employees VALUES (3, 'Bob', 'Johnson', 55000, 'Sales');
INSERT INTO employees VALUES (4, 'Alice', 'Williams', 70000, 'Finance');
INSERT INTO employees VALUES (5, 'Charlie', 'Brown', 48000, 'Marketing');

#AND Operator:
-- Retrieve employees from the Sales department with a salary greater than
50000
SELECT * FROM employees
WHERE department = 'Sales' AND salary > 50000;

#OR Operator:
-- Retrieve employees from the Sales department or with a salary greater
than 60000
SELECT * FROM employees
WHERE department = 'Sales' OR salary > 60000;

# NOT Operator:
-- Retrieve employees not from the Marketing department
SELECT * FROM employees
WHERE NOT department = 'Marketing';

#----- Concatenation Operator:-----------#
-- Create a database
CREATE DATABASE IF NOT EXISTS example_database;
USE example_database;

drop table employees;
-- Create a table
CREATE TABLE IF NOT EXISTS employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    salary INT
);

-- Insert some data
INSERT INTO employees VALUES (1, 'John', 'dev', 50000);
INSERT INTO employees VALUES (2, 'Sai', 'Smith', 60000);
INSERT INTO employees VALUES (3, 'Bob', 'John', 55000);

-- Concatenation using ||
```

```sql
SELECT employee_id, first_name || ' ' || last_name AS full_name
FROM employees;

-- Concatenation using CONCAT()
SELECT employee_id, CONCAT(first_name, ' ', last_name) AS full_name
FROM employees;

#-------------- LIKE Operator ------------#


-- Create a table
CREATE TABLE IF NOT EXISTS ele_products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100)
);

-- Insert some data
INSERT INTO ele_products VALUES (1, 'Mobile Phone');
INSERT INTO ele_products VALUES (2, 'Laptop');
INSERT INTO ele_products VALUES (3, 'Tablet');
INSERT INTO ele_products VALUES (4, 'Smartwatch');

-- Use LIKE to find products with names starting with 'Mobile'
SELECT * FROM ele_products
WHERE product_name LIKE 'Mobile%';

-- Find products with names containing 'Phone'
SELECT * FROM ele_products
WHERE product_name LIKE '%Phone%';

-- Find products with names ending with 'Tablet'
SELECT * FROM ele_products
WHERE product_name LIKE '%Tablet';

#------------- IN Operator: ---------------#
-- Create a departments table
-- Create a table
CREATE TABLE IF NOT EXISTS employees_dep (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department VARCHAR(50)
);

-- Insert some data
INSERT INTO employees_dep VALUES (1, 'John', 'sin', 'Sales');
INSERT INTO employees_dep VALUES (2, 'Sri', 'Smith', 'Marketing');
INSERT INTO employees_dep VALUES (3, 'Bob', 'John', 'Sales');
INSERT INTO employees_dep VALUES (4, 'Ali', 'Williams', 'Finance');
INSERT INTO employees_dep VALUES (5, 'Char', 'Brown', 'Marketing');

-- Use IN to find employees in 'Sales' or 'Marketing'
SELECT * FROM employees_dep
WHERE department IN ('Sales', 'Marketing');
```

```
#----------------- IS NULL / IS NOT NULL -------------#
-- Create a table
CREATE TABLE IF NOT EXISTS students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);

-- Insert some data with NULL values in the last_name column
INSERT INTO students VALUES (1, 'Vamsi', 'Reddy');
INSERT INTO students VALUES (2, 'Sai', NULL);
INSERT INTO students VALUES (3, 'Bob', 'John');
INSERT INTO students VALUES (4, 'Kumar', NULL);

SELECT * FROM students;
-- Find students with a null last_name
SELECT * FROM students WHERE first_name OR last_name IS NULL;


-- Find students with a non-null last_name
SELECT * FROM students
WHERE last_name IS NOT NULL;


#------------- BETWEEN Operator --------------#
-- Create a table
CREATE TABLE IF NOT EXISTS products_val (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    price DECIMAL(10, 2)
);

-- Insert some data
INSERT INTO products_val VALUES (1, 'Laptop', 800.00);
INSERT INTO products_val VALUES (2, 'Smartphone', 200.00);
INSERT INTO products_val VALUES (3, 'Tablet', 120.00);
INSERT INTO products_val VALUES (4, 'Headphones', 60.00);
INSERT INTO products_val VALUES (5, 'Camera', 150.00);

-- Find products with prices between $50 and $100
SELECT * FROM products_val WHERE price BETWEEN 50.00 AND 100.00;

-- Find products with prices outside the range $50 to $100
SELECT * FROM products_val WHERE price NOT BETWEEN 50.00 AND 100.00;

#------------------ SQL Date Expression ------------------#


-- Creating a table for orders
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    product_name VARCHAR(50),
    order_date DATE,
    delivery_date DATE
);

-- Inserting sample data into the orders table
INSERT INTO orders (order_id, product_name, order_date, delivery_date)
VALUES
(1, 'Laptop', '2022-01-15', '2022-01-20'),
(2, 'Smartphone', '2022-02-01', '2022-02-05'),
(3, 'Tablet', '2022-03-10', NULL),
(4, 'Headphones', '2022-04-05', '2022-04-10');

-- Current Date and Time
SELECT NOW() AS current_datetime;

-- Query 1: Select orders placed after a specific date
SELECT * FROM orders WHERE order_date > '2022-02-01';

-- Query 2: Calculate the difference in days between order_date and
delivery_date
SELECT
    order_id,
    DATEDIFF(delivery_date, order_date) AS days_to_delivery
FROM orders WHERE delivery_date IS NOT NULL;

-- Query 3: Format order_date using DATE_FORMAT
SELECT
    order_id,
    product_name,
    DATE_FORMAT(order_date, '%Y-%m-%d') AS formatted_order_date
FROM orders;


/*
SQL Aggregate Functions
SQL String Functions
*/
#database
create database functions;
use functions;

#------ SQL Aggregate Functions -------#
/*
SQL aggregate functions are used to perform calculations on sets of
values.
These functions operate on multiple rows of data and return a single
value, summarizing the information in the dataset.
*/
-- Creating a table for sales data
CREATE TABLE sales (
    sale_id INT PRIMARY KEY,
    product_name VARCHAR(50),
    sale_date DATE,
    quantity INT,
    unit_price DECIMAL(10, 2)
);

-- Inserting sample data into the sales table
INSERT INTO sales (sale_id, product_name, sale_date, quantity, unit_price)
VALUES
(1, 'Laptop', '2022-01-15', 3, 1200.00),
(2, 'Smartphone', '2022-01-20', 5, 599.99),
(3, 'Tablet', '2022-02-05', 2, 299.99),
(4, 'Headphones', '2022-02-15', 10, 49.99),
(5, 'Laptop', '2022-03-10', 2, 1300.00);

-- COUNT():Counts the number of rows in a result set.
SELECT COUNT(*) AS total_sales FROM sales;

-- SUM():Calculates the sum of a numeric column.
SELECT SUM(quantity) AS total_quantity, SUM(unit_price * quantity) AS
total_revenue FROM sales;

-- AVG():Calculates the average value of a numeric column.
SELECT AVG(unit_price) AS average_price FROM sales;

-- MIN() and MAX():Retrieve the minimum and maximum values from a column.
SELECT MIN(unit_price) AS min_price, MAX(unit_price) AS max_price FROM
sales;

-- GROUP BY:Groups the result set by one or more columns.
SELECT product_name, SUM(quantity) AS total_quantity FROM sales GROUP BY
product_name;

-- HAVING: Filters the results of a GROUP BY query based on a condition.
SELECT product_name, SUM(quantity) AS total_quantity FROM sales GROUP BY
product_name HAVING total_quantity > 5;



# --------- SQL String Functions ---------#
-- string functions that allow you to manipulate and perform operations on
character strings.


-- Creating a table for employee data
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    salary DECIMAL(10, 2)
);

-- Inserting sample data into the employees table
INSERT INTO employees (employee_id, first_name, last_name, email, salary)
VALUES
(1, 'John', 'Doe', 'john.doe@example.com', 50000.00),
```

```sql
(2, 'Jane', 'Smith', 'jane.smith@example.com', 60000.00),
(3, 'Bob', 'Johnson', 'bob.johnson@example.com', 55000.00);

-- CONCAT():Concatenates two or more strings.
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees;

-- LENGTH():Returns the length of a string.
SELECT first_name, LENGTH(first_name) AS name_length FROM employees;

-- UPPER() and LOWER():Converts a string to uppercase or lowercase.
SELECT first_name, UPPER(first_name) AS upper_name, LOWER(first_name) AS
lower_name FROM employees;

-- SUBSTRING():Extracts a substring from a string.
SELECT first_name, SUBSTRING(first_name, 1, 3) AS substring_name FROM
employees;

-- LEFT() and RIGHT():Retrieves a specified number of characters from the
left or right of a string.
SELECT first_name, LEFT(first_name, 1) AS left_name, RIGHT(first_name, 2)
AS right_name FROM employees;

-- TRIM():Removes leading and trailing spaces from a string.

SELECT email, TRIM(email) AS trimmed_email FROM employees;

-- REPLACE():Replaces occurrences of a specified substring with another
substring.
SELECT email, REPLACE(email, '@example.com', '@company.com') AS new_email
FROM employees;



/*
1)  QUERIES and SUB QUERIES
2)  CLAUSES
3)  JOINS
*/

-- drop database adv;
create database adv;
use adv;
# -------  QUERIES and SUB QUERIES --- #
/*
In MySQL, queries are used to retrieve data from one or more tables in a
database.
A query is essentially a request for information from the database.
Subqueries, on the other hand, are queries that are embedded within other
queries,
allowing you to perform more complex operations and retrieve specific data
based on the results of another query.
*/

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT,
    salary DECIMAL(10, 2)
);

INSERT INTO employees VALUES
(1, 'John', 'Doe', 101, 50000.00),
(2, 'Jane', 'Smith', 102, 60000.00),
(3, 'Bob', 'Johnson', 101, 55000.00),
(4, 'Alice', 'Williams', 103, 70000.00);

-- 1. Basic SELECT Query: This query retrieves all columns from the
employees table.
SELECT * FROM employees;

-- 2. Subquery in WHERE Clause:
SELECT first_name, last_name
FROM employees
WHERE department_id = (SELECT department_id FROM employees WHERE
first_name = 'John');

-- 3. Subquery in SELECT Clause:
SELECT first_name, last_name, (SELECT AVG(salary) FROM employees) AS
avg_salary FROM employees;


#--------------- CLAUSES ---------------#
--  Each clause serves a specific purpose and can be used in various
combinations to construct a comprehensive query or statement.

-- 1. SELECT Clause:The SELECT clause is used to specify the columns that
you want to retrieve in a query.
SELECT first_name, last_name FROM employees;

-- 2. FROM Clause:The FROM clause specifies the table from which to
retrieve the data.
SELECT * FROM employees;

-- WHERE Clause:The WHERE clause is used to filter the rows based on a
specified condition.
SELECT * FROM employees WHERE salary > 60000;

-- 4. ORDER BY Clause:The ORDER BY clause is used to sort the result set
based on one or more columns.
SELECT * FROM employees ORDER BY last_name ASC;
SELECT * FROM employees ORDER BY first_name ASC;

-- 5. GROUP BY Clause:The GROUP BY clause is used to group rows based on
one or more columns and apply aggregate functions.
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id;
```

```sql
-- 6. HAVING Clause:The HAVING clause is used in combination with GROUP BY
to filter the grouped data based on a condition.
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id
HAVING AVG(salary) < 65000;

#----------- JOIN ----------------- #

/*
In MySQL, JOINs are used to combine rows from two or more tables based on
a related column between them.
There are several types of JOINs,
1)  INNER JOIN,
2)  LEFT JOIN (or LEFT OUTER JOIN),
3)  RIGHT JOIN (or RIGHT OUTER JOIN),
4)  FULL JOIN (or FULL OUTER JOIN).

*/

-- drop database joins;

 create database joins;
 use joins;
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT,
    salary DECIMAL(10, 2)
);

CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50)
);

INSERT INTO employees VALUES
(1, 'vamsi', 'Reddy', 101, 50000.00),
(2, 'sai', 'Smith', 102, 60000.00),
(3, 'Bob', 'John', 101, 55000.00),
(4, 'Ali', 'Williams', 103, 70000.00);

INSERT INTO departments VALUES
(101, 'HR'),
(102, 'IT'),
(103, 'Finance');

select * from departments;

-- 1. INNER JOIN: An INNER JOIN returns 'rows when there is a match in
both tables based on the specified condition'.
SELECT employees.employee_id, employees.first_name, employees.last_name,
departments.department_name
FROM employees
```

```
INNER JOIN departments ON employees.department_id =
departments.department_id;

-- 2. LEFT JOIN (or LEFT OUTER JOIN):A LEFT JOIN returns all rows from the
left table and the matching rows from the right table.
-- If there is no match, NULL values are returned for columns from the
right table.

SELECT employees.employee_id, employees.first_name, employees.last_name,
departments.department_name
FROM employees
LEFT JOIN departments ON employees.department_id =
departments.department_id;

-- 3. RIGHT JOIN (or RIGHT OUTER JOIN):A RIGHT JOIN returns all rows from
the right table and the matching rows from the left table.
-- If there is no match, NULL values are returned for columns from the
left table.

SELECT employees.employee_id, employees.first_name, employees.last_name,
departments.department_name
FROM employees
RIGHT JOIN departments ON employees.department_id =
departments.department_id;

-- 4. FULL JOIN (or FULL OUTER JOIN):
-- A FULL JOIN returns all rows when there is a match in either the left
or right table.
-- If there is no match, NULL values are returned for columns from the
table without a match.

SELECT employees.employee_id, employees.first_name, employees.last_name,
departments.department_name
FROM employees
LEFT JOIN departments ON employees.department_id =
departments.department_id

UNION

SELECT employees.employee_id, employees.first_name, employees.last_name,
departments.department_name
FROM employees
RIGHT JOIN departments ON employees.department_id =
departments.department_id;
```