Q16. Write a function to insert and update the particular value in the table (Assume any Table)

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

position VARCHAR(50),

salary DECIMAL(10, 2)

);

DELIMITER $$

CREATE PROCEDURE insert\_or\_update\_employee(

IN emp\_id INT,

IN f\_name VARCHAR(50),

IN l\_name VARCHAR(50),

IN position VARCHAR(50),

IN salary DECIMAL(10, 2)

)

BEGIN

-- Check if the employee with the given emp\_id exists

IF EXISTS (SELECT 1 FROM Employees WHERE emp\_id = emp\_id) THEN

-- If the employee exists, update the record

UPDATE Employees

SET first\_name = f\_name,

last\_name = l\_name,

position = position,

salary = salary

WHERE emp\_id = emp\_id;

ELSE

-- If the employee does not exist, insert a new record

INSERT INTO Employees (emp\_id, first\_name, last\_name, position, salary)

VALUES (emp\_id, f\_name, l\_name, position, salary);

END IF;

END$$

DELIMITER ;

-- Insert a new employee

CALL insert\_or\_update\_employee(101, 'John', 'Doe', 'Software Engineer', 60000);

-- Update an existing employee

CALL insert\_or\_update\_employee(101, 'John', 'Doe', 'Senior Software Engineer', 75000);

Q17. .write a Stored Procedure to insert and delete a value in the table (Assume any Table)

CREATE TABLE students (

student\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

course VARCHAR(100)

);

DELIMITER $$

CREATE PROCEDURE InsertDeleteStudent(

IN action VARCHAR(10), -- action: 'insert' or 'delete'

IN student\_id INT, -- student\_id for delete action

IN student\_name VARCHAR(100), -- student\_name for insert action

IN student\_course VARCHAR(100) -- student\_course for insert action

)

BEGIN

-- Insert a new student if the action is 'insert'

IF action = 'insert' THEN

INSERT INTO students (name, course)

VALUES (student\_name, student\_course);

-- Delete a student by student\_id if the action is 'delete'

ELSEIF action = 'delete' THEN

DELETE FROM students

WHERE student\_id = student\_id;

END IF;

END$$

CALL InsertDeleteStudent('insert', NULL, 'Alice Brown', 'Mathematics');

DELIMITER ;

CALL InsertDeleteStudent('delete', 1, NULL, NULL);

CALL InsertDeleteStudent('insert', NULL, 'Bob Johnson', 'Physics');

CALL InsertDeleteStudent('delete', 2, NULL, NULL);

**Q18 .Write a trigger to insert, and update records from the Library system table**

CREATE TABLE books (

book\_id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255),

author VARCHAR(255),

publish\_date DATE,

status VARCHAR(20) -- e.g., 'available', 'checked out'

);

CREATE TABLE books\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

action\_type VARCHAR(10), -- 'insert' or 'update'

book\_id INT,

title VARCHAR(255),

author VARCHAR(255),

status VARCHAR(20),

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER log\_book\_insert

AFTER INSERT ON books

FOR EACH ROW

BEGIN

INSERT INTO books\_log (action\_type, book\_id, title, author, status)

VALUES ('insert', NEW.book\_id, NEW.title, NEW.author, NEW.status);

END$$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER log\_book\_update

AFTER UPDATE ON books

FOR EACH ROW

BEGIN

INSERT INTO books\_log (action\_type, book\_id, title, author, status)

VALUES ('update', NEW.book\_id, NEW.title, NEW.author, NEW.status);

END$$

DELIMITER ;

INSERT INTO books (title, author, publish\_date, status)

VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', '1925-04-10', 'available');

SELECT\*FROM books;

SELECT\*FROM books\_log;

UPDATE books

SET status = 'checked out'

WHERE book\_id = 1;

SELECT\*FROM books;

SELECT\*FROM books\_log;

**Q19. Write a cursor to insert, and delete records from the Employee table**

CREATE TABLE Employee (

employee\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(100),

status VARCHAR(20) -- 'active' or 'inactive'

);

CREATE TABLE temporary\_employee\_data (

name VARCHAR(100),

department VARCHAR(100),

status VARCHAR(20)

);

-- Insert some dummy data into the temporary table (for testing)

INSERT INTO temporary\_employee\_data (name, department, status)

VALUES

('John Doe', 'HR', 'active'),

('Jane Smith', 'Finance', 'inactive'),

('Mike Johnson', 'IT', 'active');

DELIMITER $$

CREATE PROCEDURE InsertEmployees()

BEGIN

-- Declare variables for fetching data from the cursor

DECLARE done INT DEFAULT 0;

DECLARE v\_name VARCHAR(100);

DECLARE v\_department VARCHAR(100);

DECLARE v\_status VARCHAR(20);

-- Declare the cursor to fetch records from the temporary\_employee\_data table

DECLARE cur CURSOR FOR

SELECT name, department, status

FROM temporary\_employee\_data;

-- Declare a handler for when the cursor has no more rows to fetch

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

-- Open the cursor

OPEN cur;

-- Loop through the rows returned by the cursor

read\_loop: LOOP

-- Fetch a row from the cursor into variables

FETCH cur INTO v\_name, v\_department, v\_status;

-- If no more rows, exit the loop

IF done THEN

LEAVE read\_loop;

END IF;

-- Insert the record into the Employee table

INSERT INTO Employee (name, department, status)

VALUES (v\_name, v\_department, v\_status);

END LOOP;

-- Close the cursor after use

CLOSE cur;

END$$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE DeleteInactiveEmployees()

BEGIN

-- Declare variables for fetching data from the cursor

DECLARE done INT DEFAULT 0;

DECLARE v\_employee\_id INT;

-- Declare the cursor to fetch employee IDs where status = 'inactive'

DECLARE cur CURSOR FOR

SELECT employee\_id

FROM Employee

WHERE status = 'inactive';

-- Declare a handler for when the cursor has no more rows to fetch

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

-- Open the cursor

OPEN cur;

-- Loop through the rows returned by the cursor

delete\_loop: LOOP

-- Fetch a row from the cursor into variables

FETCH cur INTO v\_employee\_id;

-- If no more rows, exit the loop

IF done THEN

LEAVE delete\_loop;

END IF;

-- Delete the employee from the Employee table

DELETE FROM Employee

WHERE employee\_id = v\_employee\_id;

END LOOP;

-- Close the cursor after use

CLOSE cur;

END$$

DELIMITER ;

CALL InsertEmployees();

CALL DeleteInactiveEmployees();

SELECT \* FROM Employee;

SELECT \* FROM Employee WHERE status = 'inactive';

**Q20. Implement MapReduce example in MongoDB with the suitable dataset. A. Create a sample collection order with 10 documents. B. Perform the map-reduce operation on the orders collection to group by the cust\_id, and calculate the sum of the price for each cust\_id.**

CREATE THE DATABASE IN THE MONGODB.

use Elect

switched to db Elect

db.orders.insertMany([

{ order\_id: 1, cust\_id: "A123", item: "Laptop", price: 1200, order\_date: new Date("2024-11-01") },

{ order\_id: 2, cust\_id: "B456", item: "Phone", price: 700, order\_date: new Date("2024-11-02") },

{ order\_id: 3, cust\_id: "A123", item: "Tablet", price: 400, order\_date: new Date("2024-11-03") },

{ order\_id: 4, cust\_id: "C789", item: "Monitor", price: 300, order\_date: new Date("2024-11-04") },

{ order\_id: 5, cust\_id: "B456", item: "Headphones", price: 150, order\_date: new Date("2024-11-05") },

{ order\_id: 6, cust\_id: "D012", item: "Keyboard", price: 80, order\_date: new Date("2024-11-06") },

{ order\_id: 7, cust\_id: "A123", item: "Mouse", price: 50, order\_date: new Date("2024-11-07") },

{ order\_id: 8, cust\_id: "C789", item: "Desk Chair", price: 120, order\_date: new Date("2024-11-08") },

{ order\_id: 9, cust\_id: "D012", item: "Monitor", price: 250, order\_date: new Date("2024-11-09") },

{ order\_id: 10, cust\_id: "B456", item: "Speakers", price: 90, order\_date: new Date("2024-11-10") }

]);

var mapFunction = function() {

emit(this.cust\_id, this.price); // Emit cust\_id as the key and price as the value

};

var reduceFunction = function(key, values) {

return Array.sum(values); // Sum up all the price values for the given cust\_id

};

db.orders.mapReduce(

mapFunction, // Map function

reduceFunction, // Reduce function

{

out: { inline: 1 } // Output the result inline (instead of saving to a collection)

}

);

**Q21.Create a function that takes two numbers as arguments and returns their sum, difference, product.**

DELIMITER $$

CREATE FUNCTION CalculateOperations(a INT, b INT)

RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE sum\_result INT;

DECLARE diff\_result INT;

DECLARE prod\_result INT;

DECLARE result VARCHAR(255);

-- Calculate sum, difference, and product

SET sum\_result = a + b;

SET diff\_result = a - b;

SET prod\_result = a \* b;

-- Prepare the result as a string (you can adjust the format)

SET result = CONCAT('Sum: ', sum\_result, ', Difference: ', diff\_result, ', Product: ', prod\_result);

-- Return the result

RETURN result;

END$$

DELIMITER ;

SELECT CalculateOperations(10, 5);

**Q22. Create a trigger so that every time a record is inserted into the users table, a corresponding log is inserted into an "Audit" table.**

-- Create the 'users' table

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Create the 'Audit' table to store logs

CREATE TABLE Audit (

audit\_id INT AUTO\_INCREMENT PRIMARY KEY,

action VARCHAR(50),

table\_name VARCHAR(50),

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

details VARCHAR(255)

);

DELIMITER $$

CREATE TRIGGER after\_user\_insert

AFTER INSERT ON users

FOR EACH ROW

BEGIN

-- Insert log into the 'Audit' table

INSERT INTO Audit (action, table\_name, details)

VALUES ('INSERT', 'users', CONCAT('Inserted user with username: ', NEW.username, ' and email: ', NEW.email));

END$$

DELIMITER ;

-- Insert a new user into the 'users' table

INSERT INTO users (username, email) VALUES ('john\_doe', 'john.doe@example.com');

-- Query the 'Audit' table to view the log

SELECT \* FROM Audit;

**23.Before any deletion on the "Users" table, create a trigger that will move the soon-to-be-deleted record to a "DeletedUsers" table.**

**-- Create the 'users' table (assuming it already exists)**

**CREATE TABLE users (**

**user\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**username VARCHAR(100) NOT NULL,**

**email VARCHAR(100) NOT NULL,**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

**-- Create the 'DeletedUsers' table to store deleted records**

**CREATE TABLE DeletedUsers (**

**user\_id INT,**

**username VARCHAR(100),**

**email VARCHAR(100),**

**deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**PRIMARY KEY (user\_id)**

**);**

DELIMITER $$

CREATE TRIGGER before\_user\_delete

BEFORE DELETE ON users

FOR EACH ROW

BEGIN

-- Insert the deleted user into the 'DeletedUsers' table

INSERT INTO DeletedUsers (user\_id, username, email)

VALUES (OLD.user\_id, OLD.username, OLD.email);

END$$

DELIMITER ;

-- Insert a user into the 'users' table

INSERT INTO users (username, email) VALUES ('john\_doe', 'john.doe@example.com');

-- Check the 'users' table to confirm the insertion

SELECT \* FROM users;

-- Delete the user from the 'users' table

DELETE FROM users WHERE username = 'john\_doe';

-- Check the 'users' table to confirm the deletion

SELECT \* FROM users;

-- Check the 'DeletedUsers' table to verify the record was moved

SELECT \* FROM DeletedUsers;

**Q 24.Use a cursor to fetch and display the average salary of each department.**

CREATE DATABASE IF NOT EXISTS company;

USE company;

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(50),

salary DECIMAL(10, 2)

);

-- Insert sample data into the employees table

INSERT INTO employees (employee\_id, name, department, salary) VALUES

(1, 'John Doe', 'Engineering', 80000),

(2, 'Jane Smith', 'Marketing', 60000),

(3, 'Sam Brown', 'Engineering', 95000),

(4, 'Emma White', 'Marketing', 72000),

(5, 'Mike Green', 'Sales', 65000),

(6, 'Olivia Black', 'Sales', 70000),

(7, 'Noah Blue', 'Engineering', 105000),

(8, 'Sophia Pink', 'Marketing', 78000),

(9, 'Liam Purple', 'Sales', 74000),

(10, 'Mia Brown', 'Engineering', 92000);

DELIMITER $$

-- Declare the procedure that uses the cursor

CREATE PROCEDURE GetAvgSalaryPerDepartment()

BEGIN

-- Declare variables

DECLARE done INT DEFAULT 0;

DECLARE dept VARCHAR(50);

DECLARE avg\_salary DECIMAL(10, 2);

-- Declare the cursor to fetch the department and average salary

DECLARE cur CURSOR FOR

SELECT department, AVG(salary)

FROM employees

GROUP BY department;

-- Declare a handler to set the 'done' variable when the cursor finishes

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

-- Open the cursor

OPEN cur;

-- Loop through all the results from the cursor

read\_loop: LOOP

FETCH cur INTO dept, avg\_salary;

-- Exit the loop when no more rows are fetched

IF done THEN

LEAVE read\_loop;

END IF;

-- Print the department and average salary

SELECT dept AS department, avg\_salary AS average\_salary;

END LOOP;

-- Close the cursor

CLOSE cur;

END$$

DELIMITER ;

SELECT department, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department;

CALL GetAvgSalaryPerDepartment();

**25.Write a Database Query for Nested queries Hotel table**

CREATE TABLE Hotel (

hotel\_id INT AUTO\_INCREMENT PRIMARY KEY,

hotel\_name VARCHAR(100),

city VARCHAR(100),

price\_per\_night DECIMAL(10, 2),

rating DECIMAL(3, 2)

);

INSERT INTO Hotel (hotel\_name, city, price\_per\_night, rating)

VALUES

('Hotel A', 'New York', 150.00, 4.5),

('Hotel B', 'Los Angeles', 200.00, 4.8),

('Hotel C', 'New York', 120.00, 4.2),

('Hotel D', 'Chicago', 180.00, 4.0),

('Hotel E', 'New York', 220.00, 5.0);

SELECT hotel\_name, price\_per\_night, city

FROM Hotel

WHERE city = 'New York'

AND price\_per\_night < (

SELECT AVG(price\_per\_night)

FROM Hotel

WHERE city = 'New York'

);

**26.Write a Database Query for Joins Banking table**

CREATE TABLE Customers (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

email VARCHAR(100)

);

CREATE TABLE Accounts (

account\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

account\_type VARCHAR(50),

balance DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

CREATE TABLE Transactions (

transaction\_id INT AUTO\_INCREMENT PRIMARY KEY,

account\_id INT,

transaction\_type VARCHAR(50), -- E.g., 'Deposit', 'Withdrawal'

amount DECIMAL(10, 2),

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (account\_id) REFERENCES Accounts(account\_id)

);

-- Insert customers

INSERT INTO Customers (first\_name, last\_name, email)

VALUES ('John', 'Doe', 'john.doe@example.com'),

('Jane', 'Smith', 'jane.smith@example.com');

-- Insert accounts

INSERT INTO Accounts (customer\_id, account\_type, balance)

VALUES (1, 'Checking', 5000.00),

(1, 'Savings', 10000.00),

(2, 'Checking', 1500.00);

-- Insert transactions

INSERT INTO Transactions (account\_id, transaction\_type, amount)

VALUES (1, 'Deposit', 1000.00),

(1, 'Withdrawal', 200.00),

(2, 'Deposit', 500.00);

SELECT c.first\_name, c.last\_name, c.email, a.account\_id, a.account\_type, a.balance FROM Customers c

INNER JOIN

Accounts a ON c.customer\_id = a.customer\_id;

SELECT c.first\_name, c.last\_name, c.email, a.account\_id, a.account\_type, a.balance

FROM Customers c

LEFT JOIN

Accounts a ON c.customer\_id = a.customer\_id;

SELECT c.first\_name, c.last\_name, SUM(t.amount) AS total\_transactions

FROM Customers c

INNER JOIN

Accounts a ON c.customer\_id = a.customer\_id

INNER JOIN

Transactions t ON a.account\_id = t.account\_id

GROUP BY

c.customer\_id;

SELECT t.transaction\_id, t.transaction\_type, t.amount, c.first\_name, c.last\_name, c.email

FROM Transactions t

RIGHT JOIN

Accounts a ON t.account\_id = a.account\_id

RIGHT JOIN

Customers c ON a.customer\_id = c.customer\_id;

SELECT c1.first\_name AS customer1\_first\_name, c1.last\_name AS customer1\_last\_name,

c2.first\_name AS customer2\_first\_name, c2.last\_name AS customer2\_last\_name

FROM Customers c1

INNER JOIN

Customers c2 ON c1.customer\_id != c2.customer\_id

WHERE

c1.first\_name = c2.first\_name;

**Q 27.Write a Database Query for Sub-queries of Manufacturing industry table**

CREATE TABLE manufacturing (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

category VARCHAR(50),

production\_cost DECIMAL(10, 2),

units\_produced INT,

production\_date DATE

);

-- Insert sample data

INSERT INTO manufacturing (product\_id, product\_name, category, production\_cost, units\_produced, production\_date) VALUES

(1, 'Widget A', 'Gadgets', 500.00, 100, '2024-11-01'),

(2, 'Widget B', 'Gadgets', 600.00, 150, '2024-11-02'),

(3, 'Gear A', 'Machinery', 1200.00, 80, '2024-11-03'),

(4, 'Bolt X', 'Tools', 200.00, 300, '2024-11-04'),

(5, 'Nut Y', 'Tools', 180.00, 400, '2024-11-05'),

(6, 'Widget C', 'Gadgets', 700.00, 120, '2024-11-06'),

(7, 'Gear B', 'Machinery', 1500.00, 50, '2024-11-07'),

(8, 'Bolt Y', 'Tools', 250.00, 250, '2024-11-08'),

(9, 'Nut X', 'Tools', 220.00, 200, '2024-11-09'),

(10, 'Widget D', 'Gadgets', 800.00, 90, '2024-11-10');

**. Find Products with Above-Average Production Costs**

To find all products whose production costs are above the average production cost:

sql

SELECT product\_name, production\_cost FROM manufacturing

WHERE production\_cost > ( SELECT AVG(production\_cost) FROM manufacturing );

SELECT product\_name, category FROM manufacturing

WHERE category = ( SELECT category FROM manufacturing

WHERE production\_cost = ( SELECT MIN(production\_cost)

FROM manufacturing) );

**Q. 28. Consider the following relational schema and briefly answer the questions that follow: Emp(eid: integer, ename: string, age: integer, salary: real) Works(eid: integer, did: integer, pct time: integer) Dept(did: integer, budget: real, managerid: integer)**

**1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000.**

**2. Define a table constraint on Dept that will ensure that all managers have age > 30.**

**3. Define an assertion on Dept that will ensure that all managers have age > 30.**

-- Step 1: Create the Emp table

CREATE TABLE Emp (

eid INT PRIMARY KEY,

ename VARCHAR(100),

age INT,

salary REAL,

CONSTRAINT chk\_salary CHECK (salary >= 10000) -- Salary >= $10,000

);

-- Step 2: Create the Dept table

CREATE TABLE Dept (

did INT PRIMARY KEY,

budget REAL,

managerid INT,

FOREIGN KEY (managerid) REFERENCES Emp(eid)

);

-- Step 3: Create a trigger to enforce the age > 30 rule for managers

DELIMITER $$

CREATE TRIGGER trg\_check\_manager\_age

BEFORE INSERT ON Dept

FOR EACH ROW

BEGIN

DECLARE manager\_age INT;

-- Check the age of the manager

SELECT age INTO manager\_age FROM Emp WHERE eid = NEW.managerid;

-- If the manager's age is <= 30, throw an error

IF manager\_age <= 30 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Manager must be older than 30.';

END IF;

END;

$$

DELIMITER ;

-- Insert valid employees

INSERT INTO Emp (eid, ename, age, salary)

VALUES

(1, 'John Doe', 45, 15000),

(2, 'Jane Smith', 28, 12000),

(3, 'Alice Johnson', 35, 18000);

-- Insert valid department data

INSERT INTO Dept (did, budget, managerid)

VALUES

(101, 500000, 1), -- Manager: John Doe (age 45, valid)

(102, 300000, 3); -- Manager: Alice Johnson (age 35, valid)

-- This will fail because Jane Smith (eid = 2) has age 28.

INSERT INTO Dept (did, budget, managerid)

VALUES (103, 200000, 2);