S1

Consider following Relation

Account (Acc_no, branch_name,balance)
Branch(branch_name,branch_city,assets)
Customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)
Borrower(cust_name,loan_no)

```
-- Account table
CREATE TABLE Account (
   Acc_no INT PRIMARY KEY,
   branch_name VARCHAR(50) NOT NULL,
   balance DECIMAL(10, 2) NOT NULL,
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

-- Branch table
CREATE TABLE Branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50) NOT NULL,
   assets DECIMAL(15, 2) NOT NULL
);

-- Customer table
CREATE TABLE Customer (
   cust_name VARCHAR(50) PRIMARY KEY,
   cust_street VARCHAR(50) NOT NULL,
   cust_city VARCHAR(50) NOT NULL
);

-- Depositor table (association between Customer and Account)
CREATE TABLE Depositor (
   cust_name VARCHAR(50),
   acc_no INT,
   PRIMARY KEY (cust_name, acc_no),
   FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
   FOREIGN KEY (acc_no) REFERENCES Account(Acc_no)
);

-- Loan table
CREATE TABLE Loan (
   loan_no INT PRIMARY KEY,
   branch_name VARCHAR(50) NOT NULL,
   amount DECIMAL(10, 2) NOT NULL,
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

-- Borrower table (association between Customer and Loan)
CREATE TABLE Borrower (
   cust_name VARCHAR(50),
   loan_no INT,
   PRIMARY KEY (cust_name, loan_no),
```

	FOREIGN KEY (cust_name) REFERENCES Customer(cust_name), FOREIGN KEY (loan_no) REFERENCES Loan(loan_no)); Create above tables with appropriate constraints like primary key, foreign key, not null etc. 1. Find the names of all branches in loan relation. SELECT DISTINCT branch_name FROM Loan; 2. Find all loan numbers for loans made at 'Wadia College' Branch with loan amount > 12000. SELECT loan_no FROM Loan WHERE branch_name = 'Wadia College' AND amount > 12000; 3. Find all customers who have a loan from bank. Find their names,loan_no and loan amount. SELECT Customer.cust_name, Loan.loan_no, Loan.amount FROM Customer JOIN Borrower ON Customer.cust_name = Borrower.cust_name JOIN Loan ON Borrower.loan_no = Loan.loan_no; 4. List all customers in alphabetical order who have loan from 'Wadia College' branch. SELECT Customer.cust_name FROM Customer JOIN Borrower ON Customer.cust_name = Borrower.cust_name JOIN Loan ON Borrower.loan_no = Loan.loan_no WHERE Loan.branch_name = 'Wadia College' ORDER BY Customer.cust_name; 5. Display distinct cities of branch. SELECT DISTINCT branch_city FROM Branch;
S2	Consider following Relation **Account (Acc_no, branch_name,balance)** **Branch(branch_name,branch_city,assets)** **Customer(cust_name,cust_street,cust_city)** **Depositor(cust_name,acc_no)** **Loan(loan_no,branch_name,amount)** **Borrower(cust_name,loan_no)** Create above tables with appropriate constraints like primary key, foreign key, not null etc. 1. Find all customers who have both account and loan at bank. SELECT DISTINCT Depositor.cust_name FROM Depositor JOIN Borrower ON Depositor.cust_name = Borrower.cust_name; 2. Find all customers who have an account or loan or both at bank. SELECT DISTINCT cust_name

```
    FROM (
        SELECT cust_name FROM Depositor
        UNION
        SELECT cust_name FROM Borrower
    ) AS AllCustomers;
```

3. Find all customers who have account but no loan at the bank.

```
SELECT DISTINCT Depositor.cust_name
FROM Depositor
LEFT JOIN Borrower ON Depositor.cust_name = Borrower.cust_name
WHERE Borrower.cust_name IS NULL;
```

4. Find average account balance at 'Wadia College' branch.

```
SELECT AVG(balance) AS average_balance
FROM Account
WHERE branch_name = 'Wadia College';
```

5. Find no. of depositors at each branch

```
SELECT Account.branch_name, COUNT(DISTINCT Depositor.cust_name) AS num_depositors
FROM Account
JOIN Depositor ON Account.Acc_no = Depositor.acc_no
GROUP BY Account.branch_name;
```

P10

Trigger: Write a after trigger for Insert, update and delete event considering following requirement:

Emp(Emp_no, Emp_name, Emp_salary)

```
-- Emp table
CREATE TABLE Emp (
  Emp_no INT PRIMARY KEY,
  Emp_name VARCHAR(50) NOT NULL,
  Emp_salary DECIMAL(10, 2) NOT NULL
);

-- Tracking table
CREATE TABLE Tracking (
  Emp_no INT,
  Emp_salary DECIMAL(10, 2),
  FOREIGN KEY (Emp_no) REFERENCES Emp(Emp_no)
);
```

a) Trigger should be initiated when salary tried to be inserted is less than Rs.50,000/-

```
-- Trigger for INSERT and UPDATE operations
CREATE TRIGGER trg_track_low_salary
AFTER INSERT OR UPDATE ON Emp
FOR EACH ROW
BEGIN
  -- Check if the inserted or updated salary is less than 50,000
```

```
        IF NEW.Emp_salary < 50000 THEN
          -- Insert the employee details into the Tracking table
          INSERT INTO Tracking (Emp_no, Emp_salary)
          VALUES (NEW.Emp_no, NEW.Emp_salary);
        END IF;
      END;
```

b) Trigger should be initiated when salary tried to be updated for value less than Rs. 50,000/-

```
-- Trigger for DELETE operation
CREATE TRIGGER trg_track_deleted_salary
AFTER DELETE ON Emp
FOR EACH ROW
BEGIN
  -- Log the deleted record (optional)
  INSERT INTO Tracking (Emp_no, Emp_salary)
  VALUES (OLD.Emp_no, OLD.Emp_salary);
END;
```

Also the new values expected to be inserted will be stored in new table Tracking(Emp_no,Emp_salary).

S3

Consider following Relation

Account (Acc_no, branch_name,balance)
Branch(branch_name,branch_city,assets)
Customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)
Borrower(cust_name,loan_no)

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Find the branches where average account balance > 15000.

```
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 15000;
```

2. Find number of tuples in customer relation.

```
SELECT COUNT(*) AS num_of_customers
FROM Customer;
```

3. Calculate total loan amount given by bank.

```
SELECT SUM(amount) AS total_loan_amount
FROM Loan;
```

4. Delete all loans with loan amount between 1300 and 1500.

```
DELETE FROM Loan
WHERE amount BETWEEN 1300 AND 1500;
```

5. Find the average account balance at each branch

```
SELECT branch_name, AVG(balance) AS average_balance
FROM Account
GROUP BY branch_name;
```

6. Find name of Customer and city where customer name starts with Letter P.

```
SELECT cust_name, cust_city
FROM Customer
WHERE cust_name LIKE 'P%';
```

S4

SQL Queries:

Create following tables with suitable constraints (primary key, foreign key, not null etc).

Insert record and solve the following queries:

Create table Cust_Master(Cust_no, Cust_name, Cust_addr)
Create table Order(Order_no, Cust_no, Order_date, Qty_Ordered)
Create Product (Product_no, Product_name, Order_no)

```
-- Cust_Master table
CREATE TABLE Cust_Master (
    Cust_no VARCHAR(10) PRIMARY KEY,
    Cust_name VARCHAR(50) NOT NULL,
    Cust_addr VARCHAR(100) NOT NULL
);

-- Order table
CREATE TABLE Orders (
    Order_no INT PRIMARY KEY,
    Cust_no VARCHAR(10),
    Order_date DATE NOT NULL,
    Qty_Ordered INT NOT NULL,
    FOREIGN KEY (Cust_no) REFERENCES Cust_Master(Cust_no)
);

-- Product table
CREATE TABLE Product (
    Product_no INT PRIMARY KEY,
    Product_name VARCHAR(50) NOT NULL,
```

```
    Order_no INT,
    FOREIGN KEY (Order_no) REFERENCES Orders(Order_no)
);

-- Sample Data Insertion
INSERT INTO Cust_Master (Cust_no, Cust_name, Cust_addr) VALUES
('C1001', 'Adam', 'Banglore'),
('C1002', 'Sara', 'Mumbai'),
('C1003', 'Ravi', 'Manglore'),
('C1004', 'Amanda', 'Delhi'),
('C1005', 'Raj', 'Banglore');

INSERT INTO Orders (Order_no, Cust_no, Order_date, Qty_Ordered)
VALUES
(1, 'C1001', '2023-11-01', 5),
(2, 'C1002', '2023-11-02', 10),
(3, 'C1003', '2023-11-03', 8),
(4, 'C1004', '2023-11-04', 3),
(5, 'C1005', '2023-11-05', 7);

INSERT INTO Product (Product_no, Product_name, Order_no) VALUES
(101, 'Laptop', 1),
(102, 'Tablet', 2),
(103, 'Monitor', 3),
(104, 'Mouse', 4),
(105, 'Keyboard', 5);
```

1. List names of customers having 'A' as second letter in their name.

```
SELECT Cust_name
FROM Cust_Master
WHERE Cust_name LIKE '_A%';
```

2. Display order from Customer no C1002, C1005, C1007 and C1008

```
SELECT *
FROM Orders
WHERE Cust_no IN ('C1002', 'C1005', 'C1007', 'C1008');
```

3. List Clients who stay in either 'Banglore or 'Manglore'

```
SELECT Cust_name
FROM Cust_Master
WHERE Cust_addr IN ('Banglore', 'Manglore');
```

4. Display name of customers& the product_name they have purchase

```
SELECT CM.Cust_name, P.Product_name
FROM Cust_Master AS CM
JOIN Orders AS O ON CM.Cust_no = O.Cust_no
JOIN Product AS P ON O.Order_no = P.Order_no;
```

5. Create view View1 consisting of Cust_name, Product_name.

```
CREATE VIEW View1 AS
SELECT CM.Cust_name, P.Product_name
FROM Cust_Master AS CM
JOIN Orders AS O ON CM.Cust_no = O.Cust_no
JOIN Product AS P ON O.Order_no = P.Order_no;
```

6. Disply product_name and quantity purchase by each customer

```
SELECT CM.Cust_name, P.Product_name, O.Qty_Ordered
FROM Cust_Master AS CM
JOIN Orders AS O ON CM.Cust_no = O.Cust_no
JOIN Product AS P ON O.Order_no = P.Order_no;
```

7. Perform different joint operation.

Inner Join: List customers with their orders.

```
SELECT CM.Cust_name, O.Order_no, O.Order_date, O.Qty_Ordered
FROM Cust_Master AS CM
JOIN Orders AS O ON CM.Cust_no = O.Cust_no;
```

Left Outer Join: List all customers with their orders, if any.

```
SELECT CM.Cust_name, O.Order_no, O.Order_date, O.Qty_Ordered
FROM Cust_Master AS CM
LEFT JOIN Orders AS O ON CM.Cust_no = O.Cust_no;
```

Right Outer Join: List all orders with customer information, if available.

```
SELECT CM.Cust_name, O.Order_no, O.Order_date, O.Qty_Ordered
FROM Cust_Master AS CM
RIGHT JOIN Orders AS O ON CM.Cust_no = O.Cust_no;
```

Full Outer Join (if supported by the DBMS): List all customers and all orders, matching where possible.

```
SELECT CM.Cust_name, O.Order_no, O.Order_date, O.Qty_Ordered
FROM Cust_Master AS CM
FULL OUTER JOIN Orders AS O ON CM.Cust_no = O.Cust_no;
```

P1 Write a **PL/SQL code** block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

```
CREATE TABLE areas (
  radius DECIMAL(5, 2),
  area DECIMAL(10, 2)
);

DECLARE
  radius DECIMAL(5, 2);
  area DECIMAL(10, 2);
BEGIN
  -- Loop through radius values from 5 to 9
  FOR radius IN 5..9 LOOP
    -- Calculate the area for the given radius (Area = π * r^2)
    area := 3.14159 * radius * radius;
```

```
    -- Insert the radius and calculated area into the areas table
    INSERT INTO areas (radius, area)
    VALUES (radius, area);
  END LOOP;

  -- Commit the changes
  COMMIT;

  -- Output a success message
  DBMS_OUTPUT.PUT_LINE('Areas calculated and stored successfully.');
END;
/
```

P3 Write a **PL/SQL block** of code using Cursor that will merge the data available in the newly created table N_Roll Call with the data available in the table O_RollCall. If the data in the first table already exist in the second table, then that data should be skipped.

```
DECLARE
    -- Declare cursor to select data from N_RollCall table
    CURSOR cur_rollcall IS
        SELECT roll_no, student_name
        FROM N_RollCall;

    -- Declare variables to hold cursor data
    v_roll_no INT;
    v_student_name VARCHAR(50);
BEGIN
    -- Open the cursor
    OPEN cur_rollcall;

    -- Loop through each record in the cursor
    LOOP
        FETCH cur_rollcall INTO v_roll_no, v_student_name;
        EXIT WHEN cur_rollcall%NOTFOUND;

        -- Check if the record already exists in O_RollCall table
        BEGIN
            -- If the record doesn't exist, insert it into O_RollCall
            IF NOT EXISTS (SELECT 1 FROM O_RollCall WHERE roll_no =
v_roll_no) THEN
                INSERT INTO O_RollCall (roll_no, student_name)
                VALUES (v_roll_no, v_student_name);
            END IF;
        EXCEPTION
            WHEN DUP_VAL_ON_INDEX THEN
```

```
                -- Skip insertion if duplicate value error occurs
(e.g., unique constraint violation)
                NULL;
        END;

    END LOOP;

    -- Close the cursor
    CLOSE cur_rollcall;

    -- Commit the changes
    COMMIT;

    -- Output a success message
    DBMS_OUTPUT.PUT_LINE('Data from N_RollCall merged into
O_RollCall successfully.');
END;
/
```

S5

Consider following Relation

Employee(emp_id,employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,city)
Manages(employee_name,manager_name)

```
-- Employee table
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   employee_name VARCHAR(50) NOT NULL UNIQUE,
   street VARCHAR(50),
   city VARCHAR(50)
);

-- Company table
CREATE TABLE Company (
   company_name VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50) NOT NULL
);

-- Works table
CREATE TABLE Works (
   employee_name VARCHAR(50) NOT NULL,
   company_name VARCHAR(50) NOT NULL,
   salary DECIMAL(10, 2) NOT NULL,
   FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
   FOREIGN KEY (company_name) REFERENCES Company(company_name)
);

-- Manages table
CREATE TABLE Manages (
   employee_name VARCHAR(50) NOT NULL,
   manager_name VARCHAR(50),
```

```
    FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
    FOREIGN KEY (manager_name) REFERENCES Employee(employee_name)
);
```

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Find the names of all employees who work for ‘TCS’.

```
SELECT employee_name
FROM Works
WHERE company_name = 'TCS';
```

2. Find the names and company names of all employees sorted in ascending order of company name and descending order of employee names of that company.

```
SELECT employee_name, company_name
FROM Works
ORDER BY company_name ASC, employee_name DESC;
```

3. Change the city of employee working with InfoSys to ‘Bangalore’

```
UPDATE Employee
SET city = 'Bangalore'
WHERE employee_name IN (
    SELECT employee_name
    FROM Works
    WHERE company_name = 'InfoSys'
);
```

4. Find the names, street address, and cities of residence for all employees who work for 'TechM' and earn more than $10,000.

```
SELECT E.employee_name, E.street, E.city
FROM Employee AS E
JOIN Works AS W ON E.employee_name = W.employee_name
WHERE W.company_name = 'TechM' AND W.salary > 10000;
```

5. Add Column Asset to Company table.

```
ALTER TABLE Company
ADD COLUMN Asset DECIMAL(15, 2);
```

S6

Consider following Relation

Employee(emp_id,employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,city)
Manages(employee_name,manager_name)

```
-- Employee table
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   employee_name VARCHAR(50) NOT NULL,
   street VARCHAR(50),
   city VARCHAR(50)
);

-- Works table
CREATE TABLE Works (
   employee_name VARCHAR(50) NOT NULL,
   company_name VARCHAR(50) NOT NULL,
   salary DECIMAL(10, 2) NOT NULL,
   FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
   FOREIGN KEY (company_name) REFERENCES Company(company_name)
);

-- Company table
CREATE TABLE Company (
   company_name VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50) NOT NULL
);

-- Manages table
CREATE TABLE Manages (
   employee_name VARCHAR(50) NOT NULL,
   manager_name VARCHAR(50),
   FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
   FOREIGN KEY (manager_name) REFERENCES Employee(employee_name)
);
```

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Change the city of employee working with InfoSys to 'Bangalore'

```
UPDATE Employee
SET city = 'Bangalore'
WHERE employee_name IN (
    SELECT employee_name
    FROM Works
    WHERE company_name = 'InfoSys'
);
```

2. Find the names of all employees who earn more than the average salary of all employees of their company. Assume that all people work for at most one company.

```
SELECT employee_name
FROM Works AS W1
```

```
    WHERE salary > (
      SELECT AVG(salary)
      FROM Works AS W2
      WHERE W1.company_name = W2.company_name
    );
```

3. Find the names, street address, and cities of residence for all employees who work for 'TechM' and earn more than $10,000.

```
SELECT E.employee_name, E.street, E.city
FROM Employee AS E
JOIN Works AS W ON E.employee_name = W.employee_name
WHERE W.company_name = 'TechM' AND W.salary > 10000;
```

4. Change name of table Manages to Management.

```
ALTER TABLE Manages RENAME TO Management;
```

5. Create Simple and Unique index on employee table.

```
CREATE INDEX idx_employee_city ON Employee (city);
CREATE UNIQUE INDEX idx_employee_name ON Employee (employee_name);
```

6. Display index Information

```
SHOW INDEX FROM Employee;
```

P8 Write a **Row Level Before and After Trigger** on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

```
CREATE TABLE Library (
  book_id INT PRIMARY KEY,
  title VARCHAR(100),
  author VARCHAR(100),
  published_year INT,
  quantity INT
);
CREATE TABLE Library_Audit (
  audit_id INT PRIMARY KEY AUTO_INCREMENT,
  book_id INT,
  title VARCHAR(100),
  author VARCHAR(100),
  published_year INT,
  quantity INT,
  action VARCHAR(10),  -- 'UPDATE' or 'DELETE'
  action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE TRIGGER before_update_library
BEFORE UPDATE ON Library
FOR EACH ROW
BEGIN
  -- Insert old values into Library_Audit table before updating
  INSERT INTO Library_Audit (book_id, title, author, published_year, quantity, action)
  VALUES (:OLD.book_id, :OLD.title, :OLD.author, :OLD.published_year, :OLD.quantity, 'UPDATE');
END;
```

```
/

CREATE OR REPLACE TRIGGER after_delete_library
AFTER DELETE ON Library
FOR EACH ROW
BEGIN
  -- Insert the old values of the deleted record into Library_Audit table
  INSERT INTO Library_Audit (book_id, title, author, published_year, quantity, action)
  VALUES (:OLD.book_id, :OLD.title, :OLD.author, :OLD.published_year, :OLD.quantity, 'DELETE');
END;
/
```

P9

Trigger: Create a row level trigger for the CUSTOMERS table that would fire INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.

```
CREATE TABLE CUSTOMERS (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(100),
  salary DECIMAL(10, 2)
);

CREATE OR REPLACE TRIGGER salary_difference_trigger
AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS
FOR EACH ROW
DECLARE
  salary_diff DECIMAL(10, 2);
BEGIN
  -- For INSERT operation, no old salary, just display message
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('New customer inserted: ' || :NEW.customer_name ||
                ' with salary ' || :NEW.salary);
  END IF;

  -- For UPDATE operation, calculate the salary difference
  IF UPDATING THEN
    salary_diff := :NEW.salary - :OLD.salary;
    DBMS_OUTPUT.PUT_LINE('Salary updated for customer: ' || :OLD.customer_name ||
                ' from ' || :OLD.salary || ' to ' || :NEW.salary ||
                '. Difference: ' || salary_diff);
  END IF;

  -- For DELETE operation, show the salary before the record is deleted
  IF DELETING THEN
    salary_diff := :OLD.salary;  -- Display the salary before deletion
    DBMS_OUTPUT.PUT_LINE('Customer deleted: ' || :OLD.customer_name ||
                ' with salary ' || :OLD.salary ||
                '. Difference: ' || salary_diff);
  END IF;
END;
/
```


S7	Consider following Relation **Account (Acc_no, branch_name,balance)** **Branch(branch_name,branch_city,assets)** **Customer(cust_name,cust_street,cust_city)** **Depositor(cust_name,acc_no)** **Loan(loan_no,branch_name,amount)** **Borrower(cust_name,loan_no)** Execute the following query: 1. Create a View1 to display List all customers in alphabetical order who have loan from Pune_Station branch. CREATE VIEW View1 AS SELECT Customer.cust_name FROM Customer JOIN Borrower ON Customer.cust_name = Borrower.cust_name JOIN Loan ON Borrower.loan_no = Loan.loan_no WHERE Loan.branch_name = 'Pune_Station' ORDER BY Customer.cust_name; 2. Create View2 on branch table by selecting any two columns and perform insert update delete operations. CREATE VIEW View2 AS SELECT branch_name, branch_city FROM Branch; INSERT INTO View2 (branch_name, branch_city) VALUES ('New_Branch', 'New_City'); UPDATE View2 SET branch_city = 'Updated_City' WHERE branch_name = 'New_Branch'; DELETE FROM View2 WHERE branch_name = 'New_Branch'; 3. Create View3 on borrower and depositor table by selecting any one column from each table perform insert update delete operations. CREATE VIEW View3 AS SELECT Borrower.cust_name AS borrower_name, Depositor.acc_no AS depositor_acc_no FROM Borrower JOIN Depositor ON Borrower.cust_name = Depositor.cust_name; INSERT INTO View3 (borrower_name, depositor_acc_no) VALUES ('John Doe', 12345); UPDATE View3 SET depositor_acc_no = 54321 WHERE borrower_name = 'John Doe';

DELETE FROM View3

WHERE borrower_name = 'John Doe';

4. Create Union of left and right joint for all customers who have an account or loan or both at bank

```
SELECT DISTINCT cust_name
FROM Depositor
LEFT JOIN Borrower ON Depositor.cust_name = Borrower.cust_name
UNION
SELECT DISTINCT cust_name
FROM Borrower
RIGHT JOIN Depositor ON Borrower.cust_name = Depositor.cust_name;
```

5. Create Simple and Unique index.

```
CREATE INDEX idx_customer_city ON Customer (cust_city);
CREATE UNIQUE INDEX idx_branch_name ON Branch (branch_name);
```

6. Display index Information.

```
SHOW INDEX FROM Customer;
SHOW INDEX FROM Branch;
```

S8

Consider following Relation:

Companies (comp_id, name, cost, year)

Orders (comp_id, domain, quantity) Execute

```
-- COMPANIES table
CREATE TABLE Companies (
    comp_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    cost DECIMAL(10, 2) NOT NULL,
    year INT
);

-- ORDERS table
CREATE TABLE Orders (
    comp_id INT,
    domain VARCHAR(50) NOT NULL,
    quantity INT NOT NULL,
    FOREIGN KEY (comp_id) REFERENCES
Companies(comp_id)
);

-- Sample Data Insertion
INSERT INTO Companies (comp_id, name, cost,
year) VALUES
```

(1, 'TechCorp', 20000, 2020),
(2, 'BizWare', 15000, 2021),
(3, 'DataSolutions', 30000, 2019);

INSERT INTO Orders (comp_id, domain, quantity) VALUES
(1, 'Software', 10),
(2, 'Hardware', 20),
(4, 'Consulting', 15);

the following query:

1. Find names, costs, domains and quantities for companies using inner join.

```
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
JOIN Orders ON Companies.comp_id = Orders.comp_id;
```

2. Find names, costs, domains and quantities for companies using left outer join.

```
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
LEFT JOIN Orders ON Companies.comp_id = Orders.comp_id;
```

3. Find names, costs, domains and quantities for companies using right outer join.

```
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
RIGHT JOIN Orders ON Companies.comp_id = Orders.comp_id;
```

4. Find names, costs, domains and quantities for companies using Union operator.

```
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
JOIN Orders ON Companies.comp_id = Orders.comp_id
UNION
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
LEFT JOIN Orders ON Companies.comp_id = Orders.comp_id
WHERE Orders.comp_id IS NULL;
```

5. Create View View1 by selecting both tables to show company name and quantities.

```
CREATE VIEW View1 AS
SELECT Companies.name, Orders.quantity
FROM Companies
JOIN Orders ON Companies.comp_id = Orders.comp_id;
```

6. Create View View2 by selecting any two columns and perform insert update delete operations.

	-- Creating View2 CREATE VIEW View2 AS SELECT name, cost FROM Companies; INSERT INTO View2 (name, cost) VALUES ('NewCo', 18000); UPDATE View2 SET cost = 22000 WHERE name = 'NewCo'; DELETE FROM View2 WHERE name = 'NewCo'; 7. Display content of View1, View2. SELECT * FROM View1; SELECT * FROM View2;
P5	Write a **PL/SQL Block** to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, whenever such salary updates take place, a record for same is maintained in the increment_salary table. emp(emp_no, salary) increment_salary(emp_no, salary) -- Create the emp table CREATE TABLE emp (emp_no INT PRIMARY KEY, salary DECIMAL(10, 2)); -- Create the increment_salary table to store salary increments CREATE TABLE increment_salary (emp_no INT, salary DECIMAL(10, 2), increment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP); DECLARE -- Declare a variable to store the average salary of all employees avg_salary DECIMAL(10, 2); -- Cursor to fetch employees with salary less than average salary CURSOR emp_cursor IS SELECT emp_no, salary FROM emp WHERE salary < avg_salary;

```
    -- Declare a variable to store the new salary after increment
    new_salary DECIMAL(10, 2);

BEGIN
    -- Calculate the average salary in the organization
    SELECT AVG(salary) INTO avg_salary FROM emp;

    -- Loop through each employee who earns less than the average
salary
    FOR emp_rec IN emp_cursor LOOP
        -- Calculate the new salary (10% increase)
        new_salary := emp_rec.salary * 1.10;

        -- Update the employee's salary in the emp table
        UPDATE emp
        SET salary = new_salary
        WHERE emp_no = emp_rec.emp_no;

        -- Insert the old salary and new salary into
increment_salary table
        INSERT INTO increment_salary (emp_no, salary)
        VALUES (emp_rec.emp_no, new_salary);

        -- Optionally, print a message for each update
        DBMS_OUTPUT.PUT_LINE('Employee ' || emp_rec.emp_no || '
salary updated to ' || new_salary);
    END LOOP;

    -- Commit the changes to the database
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        -- If an error occurs, rollback the transaction and print
error
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

S9

SQL Queries

Create following tables with suitable constraints. Insert data and solve the following queries:

CUSTOMERS(CNo, Cname, Ccity, CMobile)
ITEMS(INo, Iname, Itype, Iprice, Icount)
PURCHASE(PNo, Pdate, Pquantity, Cno, INo)

```
-- CUSTOMERS table
CREATE TABLE CUSTOMERS (
  CNo INT PRIMARY KEY,
  Cname VARCHAR(50) NOT NULL,
  Ccity VARCHAR(50),
  CMobile VARCHAR(15) UNIQUE
);
```

```
-- ITEMS table
CREATE TABLE ITEMS (
    INo INT PRIMARY KEY,
    Iname VARCHAR(50) NOT NULL,
    Itype VARCHAR(50) NOT NULL,
    Iprice DECIMAL(10, 2) NOT NULL,
    Icount INT NOT NULL
);

-- PURCHASE table
CREATE TABLE PURCHASE (
    PNo INT PRIMARY KEY,
    Pdate DATE NOT NULL,
    Pquantity INT NOT NULL,
    CNo INT,
    INo INT,
    FOREIGN KEY (CNo) REFERENCES CUSTOMERS(CNo),
    FOREIGN KEY (INo) REFERENCES ITEMS(INo)
);

-- Sample Data Insertion
INSERT INTO CUSTOMERS (CNo, Cname, Ccity, CMobile) VALUES
(1, 'Gopal', 'Mumbai', '1234567890'),
(2, 'Maya', 'Pune', '0987654321'),
(3, 'Amit', 'Delhi', '1122334455');

INSERT INTO ITEMS (INo, Iname, Itype, Iprice, Icount) VALUES
(1, 'Notebook', 'Stationary', 500, 100),
(2, 'Pen', 'Stationary', 50, 500),
(3, 'Calculator', 'Electronics', 1500, 50),
(4, 'Marker', 'Stationary', 300, 200);

INSERT INTO PURCHASE (PNo, Pdate, Pquantity, CNo, INo) VALUES
(1, '2023-10-01', 2, 1, 1),
(2, '2023-11-05', 1, 2, 3),
(3, '2023-09-15', 5, 2, 2);
```

1. List all stationary items with price between 400/- to 1000/-

```
SELECT *
FROM ITEMS
WHERE Itype = 'Stationary' AND Iprice BETWEEN 400 AND 1000;
```

2. Change the mobile number of customer “Gopal”

```
UPDATE CUSTOMERS
SET CMobile = '9876543210'
WHERE Cname = 'Gopal';
```

3. Display the item with maximum price

```
SELECT *
FROM ITEMS
WHERE Iprice = (SELECT MAX(Iprice) FROM ITEMS);
```

4. Display all purchases sorted from the most recent to the oldest

```
SELECT *
FROM PURCHASE
ORDER BY Pdate DESC;
```

5. Count the number of customers in every city

```
SELECT Ccity, COUNT(*) AS num_of_customers
FROM CUSTOMERS
GROUP BY Ccity;
```

6. Display all purchased quantity of Customer Maya

```
SELECT Pquantity
FROM PURCHASE
JOIN CUSTOMERS ON PURCHASE.CNo = CUSTOMERS.CNo
WHERE CUSTOMERS.Cname = 'Maya';
```

7. Create view which shows Iname, Price and Count of all stationary items in descending order of price.

```
CREATE VIEW StationaryItemsView AS
SELECT Iname, Iprice AS Price, Icount AS Count
FROM ITEMS
WHERE Itype = 'Stationary'
ORDER BY Iprice DESC;
```

M1

Design and Develop MongoDB Queries using CRUD operations:

Create Employee collection by considering following Fields:

i. Name: Embedded Doc (FName, LName)

ii. Company Name: String

iii. Salary: Number

iv. Designation: String

v. Age: Number

vi. Expertise: Array

vii. DOB: String or Date

viii. Email id: String ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following queries:

```
db.Employee.insertMany([
 {
  "Name": { "FName": "John", "LName": "Doe" },
  "Company Name": "TCS",
  "Salary": 50000,
  "Designation": "Programmer",
  "Age": 28,
  "Expertise": ["Java", "Spring", "MongoDB"],
  "DOB": "1995-06-15",
  "Email id": "john.doe@tcs.com",
  "Contact": "9876543210",
  "Address": [
   { "PAddr": "123 Main St", "LAddr": "Mumbai" }
  ]
 },
 {
  "Name": { "FName": "Jane", "LName": "Smith" },
  "Company Name": "Infosys",
  "Salary": 45000,
  "Designation": "Tester",
  "Age": 24,
  "Expertise": ["Testing", "Automation", "Selenium"],
  "DOB": "1999-04-12",
  "Email id": "jane.smith@infosys.com",
  "Contact": "9871234567",
  "Address": [
   { "PAddr": "456 Elm St", "LAddr": "Pune" }
```

```
  ]
 },
 {
  "Name": { "FName": "Alice", "LName": "Johnson" },
  "Company Name": "Infosys",
  "Salary": 55000,
  "Designation": "Programmer",
  "Age": 30,
  "Expertise": ["Java", "Python", "C++"],
  "DOB": "1993-05-20",
  "Email id": "alice.johnson@infosys.com",
  "Contact": "5559876543",
  "Address": [
   { "PAddr": "789 Oak St", "LAddr": "Chennai" }
  ]
 },
 {
  "Name": { "FName": "Bob", "LName": "Williams" },
  "Company Name": "TCS",
  "Salary": 60000,
  "Designation": "Manager",
  "Age": 35,
  "Expertise": ["Project Management", "Agile", "Scrum"],
  "DOB": "1988-11-30",
  "Email id": "bob.williams@tcs.com",
  "Contact": "7776543210",
  "Address": [
   { "PAddr": "123 Pine St", "LAddr": "Bangalore" }
  ]
 },
 {
  "Name": { "FName": "Eve", "LName": "Davis" },
  "Company Name": "TCS",
  "Salary": 70000,
  "Designation": "Programmer",
  "Age": 27,
  "Expertise": ["Java", "Spring Boot", "Microservices"],
  "DOB": "1996-03-15",
  "Email id": "eve.davis@tcs.com",
  "Contact": "6669876543",
```

```
  "Address": [
   { "PAddr": "234 Maple St", "LAddr": "Chennai" }
  ]
 }
]);
```

1. Select all documents where the Designation field has the value "Programmer" and the value of the salary field is greater than 30000.

```
db.Employee.find(
 {
  "Designation": "Programmer",
  "Salary": { $gt: 30000 }
 }
);
```

2. Creates a new document if no document in the employee collection contains {Designation: "Tester", Company_name: "TCS", Age: 25}

```
var existingDoc = db.Employee.findOne({ "Designation": "Tester", "Company Name": "TCS", "Age": 25 });
if (!existingDoc) {
 db.Employee.insertOne({
  "Name": { "FName": "Sam", "LName": "Taylor" },
  "Company Name": "TCS",
  "Salary": 40000,
  "Designation": "Tester",
  "Age": 25,
  "Expertise": ["Automation", "Selenium", "JMeter"],
  "DOB": "1998-01-10",
  "Email id": "sam.taylor@tcs.com",
  "Contact": "9876543212",
  "Address": [
   { "PAddr": "123 Park Ave", "LAddr": "Pune" }
  ]
 });
}
```

3. Increase salary of each Employee working with "Infosys" 10000.

```
db.Employee.updateMany(
  { "Company Name": "Infosys" },
  { $inc: { "Salary": 10000 } }
);
```

4. Finds all employees working with "TCS" and reduce their salary by 5000.

```
db.Employee.updateMany(
```

{ "Company Name": "TCS" },

{ $inc: { "Salary": -5000 } }

);

5. Return documents where Designation is not equal to "Tester".

```
db.Employee.find(
 { "Designation": { $ne: "Tester" } }
);
```

6. Find all employee with Exact Match on an Array having Expertise: ['Mongodb','Mysql','Cassandra']

```
db.Employee.find(
 { "Expertise": ["Mongodb", "Mysql", "Cassandra"] }
);
```

M2

Design and Develop MongoDB Queries using CRUD operations:

Create Employee collection by considering following Fields:

i. Name: Embedded Doc (FName, LName)
ii. Company Name: String
iii. Salary: Number
iv. Designation: String
v. Age: Number
vi. Expertise: Array
vii. DOB: String or Date
viii. Email id: String ix. Contact: String
x. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following queries:

```
db.Employee.insertMany([
 {
  "Name": { "FName": "John", "LName": "Doe" },
  "Company Name": "TCS",
  "Salary": 120000,
  "Designation": "Developer",
  "Age": 28,
  "Expertise": ["Java", "Spring", "Hibernate"],
  "DOB": "1995-08-10",
  "Email id": "john.doe@tcs.com",
  "Contact": "9876543210",
  "Address": [
   { "PAddr": "123 Main St", "LAddr": "Mumbai" }
  ]
 },
 {
  "Name": { "FName": "Jane", "LName": "Smith" },
  "Company Name": "Infosys",
  "Salary": 95000,
  "Designation": "Tester",
  "Age": 24,
  "Expertise": ["Testing", "Selenium", "Automation"],
  "DOB": "1999-06-15",
  "Email id": "jane.smith@infosys.com",
  "Contact": "9871234567",
  "Address": [
   { "PAddr": "456 Elm St", "LAddr": "Pune" }
  ]
```

```
},
{
 "Name": { "FName": "Alice", "LName": "Johnson" },
 "Company Name": "TCS",
 "Salary": 80000,
 "Designation": "Developer",
 "Age": 32,
 "Expertise": ["Python", "Machine Learning"],
 "DOB": "1991-05-25",
 "Email id": "alice.johnson@tcs.com",
 "Contact": "5559876543",
 "Address": [
  { "PAddr": "789 Oak St", "LAddr": "Delhi" }
 ]
},
{
 "Name": { "FName": "Bob", "LName": "Williams" },
 "Company Name": "Wipro",
 "Salary": 105000,
 "Designation": "Manager",
 "Age": 35,
 "Expertise": ["Project Management", "Agile"],
 "DOB": "1988-11-30",
 "Email id": "bob.williams@wipro.com",
 "Contact": "7776543210",
 "Address": [
  { "PAddr": "123 Pine St", "LAddr": "Bangalore" }
 ]
},
{
 "Name": { "FName": "Eve", "LName": "Davis" },
 "Company Name": "Infosys",
 "Salary": 115000,
 "Designation": "Developer",
 "Age": 26,
 "Expertise": ["Java", "Spring Boot", "Microservices"],
 "DOB": "1997-03-21",
 "Email id": "eve.davis@infosys.com",
 "Contact": "6669876543",
 "Address": [
  { "PAddr": "234 Maple St", "LAddr": "Chennai" }
 ]
```

```
 }
]);
```

1. Final name of Employee where age is less than 30 and salary more than 50000.

```
db.Employee.find(
 { "Age": { $lt: 30 }, "Salary": { $gt: 50000 } },
 { "Name": 1, "_id": 0 }
);
```

2. Creates a new document if no document in the employee collection contains
   {Designation: "Tester", Company_name: "TCS", Age: 25}

```
db.Employee.find(
 { "Age": { $lt: 30 }, "Salary": { $gt: 50000 } },
 { "Name": 1, "_id": 0 }
);
var existingDoc = db.Employee.findOne({ "Designation": "Tester", "Company Name": "TCS", "Age": 25 });
if (!existingDoc) {
  db.Employee.insertOne({
    "Name": { "FName": "Sam", "LName": "Taylor" },
    "Company Name": "TCS",
    "Salary": 90000,
    "Designation": "Tester",
    "Age": 25,
    "Expertise": ["Automation", "Selenium", "TestNG"],
    "DOB": "1998-02-15",
    "Email id": "sam.taylor@tcs.com",
    "Contact": "9876543211",
    "Address": [
     { "PAddr": "123 Park Ave", "LAddr": "Pune" }
    ]
  });
}
```

3. Selects all documents in the collection where the field age has a value less than 30 or the value of the salary field is greater than 40000.

```
db.Employee.find({
 $or: [
  { "Age": { $lt: 30 } },
  { "Salary": { $gt: 40000 } }
 ]
});
```

4. Find documents where Designation is not equal to "Developer".

```
db.Employee.find({ "Designation": { $ne: "Developer" } });
```

5. Find _id, Designation, Address and Name from all documents where Company_name is "Infosys".

```
db.Employee.find(
 { "Company Name": "Infosys" },
 { "_id": 1, "Designation": 1, "Address": 1, "Name": 1 }
);
```

6. Display only FName and LName of all Employees

```
db.Employee.find(
  {},
  { "Name.FName": 1, "Name.LName": 1, "_id": 0 }
);
```

P4

Write a **PL/SQL block** for following requirements and handle the exceptions. Roll no. of students will be entered by the user. Attendance of roll no. entered by user will be checked in the Stud table. If attendance is less than 75% then display the message "Term not granted" and set the status in stud table as "Detained". Otherwise display message "Term granted" and set the status in stud table as "Not Detained". **Student (Roll, Name, Attendance, Status)**

```
CREATE TABLE stud (
    roll INT PRIMARY KEY,
    name VARCHAR(100),
    attendance DECIMAL(5, 2),  -- Attendance percentage
    status VARCHAR(20)         -- Status: Detained / Not Detained
);
```

```
DECLARE
  -- Variable to store the roll number entered by the user
  v_roll INT;

  -- Variable to store the attendance of the student
  v_attendance DECIMAL(5, 2);
```

```
    -- Variable to store the current student's status
    v_status VARCHAR(20);

BEGIN
    -- Prompt the user to input the roll number
    v_roll := &roll_no;  -- This will prompt the user to enter the roll number

    -- Fetch the student's attendance from the stud table
    BEGIN
        SELECT attendance, status
        INTO v_attendance, v_status
        FROM stud
        WHERE roll = v_roll;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            -- Handle the case when no student with the given roll number is found
            DBMS_OUTPUT.PUT_LINE('No student found with roll number ' || v_roll);
            RAISE;  -- Raise exception to exit the block
        WHEN OTHERS THEN
            -- Handle any other exceptions
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
            RAISE;  -- Raise exception to exit the block
    END;

    -- Check the attendance and update the status accordingly
    IF v_attendance < 75 THEN
        -- If attendance is less than 75%, set status to "Detained"
        DBMS_OUTPUT.PUT_LINE('Term not granted');
        UPDATE stud
        SET status = 'Detained'
        WHERE roll = v_roll;
    ELSE
        -- If attendance is 75% or more, set status to "Not Detained"
        DBMS_OUTPUT.PUT_LINE('Term granted');
        UPDATE stud
        SET status = 'Not Detained'
        WHERE roll = v_roll;
    END IF;

    -- Commit the changes to the database
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        -- In case of any error, rollback the transaction
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

P7

Create a **stored function** titled **'Age_calc'**.
Accept the date of birth of a person as a parameter.
Calculate the age of the person in years, months and days e.g. 3 years, 2months, 10 days.
Return the age in years directly (with the help of Return statement).
The months and days are to be returned indirectly in the form of OUT parameters.

```
 CREATE OR REPLACE FUNCTION Age_calc (
    dob IN DATE,  -- Date of birth of the person
    months OUT NUMBER,  -- The calculated months, returned as OUT
parameter
    days OUT NUMBER  -- The calculated days, returned as OUT
parameter
)
RETURN NUMBER  -- The age in years
IS
    v_age_in_years NUMBER;  -- Variable to store the age in years
    v_age_in_months NUMBER;  -- Variable to store the age in months
    v_age_in_days NUMBER;  -- Variable to store the age in days
BEGIN
    -- Calculate the age in months from the date of birth till the
current date
    v_age_in_months := MONTHS_BETWEEN(SYSDATE, dob);

    -- Calculate the years part (truncate months to years)
    v_age_in_years := TRUNC(v_age_in_months / 12);

    -- Calculate remaining months after extracting the years
    months := TRUNC(v_age_in_months) - (v_age_in_years * 12);

    -- Calculate remaining days by subtracting the full years and
months from the current date
    v_age_in_days := SYSDATE - (ADD_MONTHS(dob, v_age_in_years * 12
+ months));
    days := TRUNC(v_age_in_days);

    -- Return the age in years directly
    RETURN v_age_in_years;

EXCEPTION
    WHEN OTHERS THEN
        -- Handle exceptions and return 0 for error case
        RETURN 0;
END Age_calc;
/
```

M3

Design and Develop MongoDB Queries using CRUD operations:

Create Employee collection by considering following Fields:

i. Emp_id : Number

ii. Name: Embedded Doc (FName, LName)

iii. Company Name: String

iv. Salary: Number

v. Designation: String

vi. Age: Number

vii. Expertise: Array

viii. DOB: String or Date

ix. Email id: String

x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following queries:

```
db.Employee.insertMany([
 {
  "Emp_id": 1001,
  "Name": { "FName": "John", "LName": "Doe" },
  "Company Name": "TCS",
  "Salary": 120000,
  "Designation": "Tester",
  "Age": 25,
  "Expertise": ["Automation", "Testing", "Java"],
  "DOB": "1998-01-15",
  "Email id": "john.doe@tcs.com",
  "Contact": "987-654-3210",
  "Address": [
   { "PAddr": "123 Main St", "LAddr": "Pune", "Pin_code": "411001" }
  ]
 },
 {
  "Emp_id": 1002,
  "Name": { "FName": "Jane", "LName": "Smith" },
  "Company Name": "TCS",
  "Salary": 100000,
  "Designation": "Developer",
  "Age": 30,
  "Expertise": ["NodeJS", "MongoDB", "JavaScript"],
  "DOB": "1993-07-20",
  "Email id": "jane.smith@tcs.com",
  "Contact": "987-123-4567",
```

```
  "Address": [
   { "PAddr": "456 Elm St", "LAddr": "Mumbai", "Pin_code": "400001" }
  ]
 },
 {
  "Emp_id": 1003,
  "Name": { "FName": "Bob", "LName": "Williams" },
  "Company Name": "Wipro",
  "Salary": 140000,
  "Designation": "Tester",
  "Age": 28,
  "Expertise": ["Manual Testing", "SQL", "Agile"],
  "DOB": "1995-05-30",
  "Email id": "bob.williams@wipro.com",
  "Contact": "666-432-9876",
  "Address": [
   { "PAddr": "789 Oak St", "LAddr": "Bangalore", "Pin_code": "560001" }
  ]
 },
 {
  "Emp_id": 1004,
  "Name": { "FName": "Alice", "LName": "Johnson" },
  "Company Name": "TCS",
  "Salary": 110000,
  "Designation": "Developer",
  "Age": 26,
  "Expertise": ["Java", "Spring", "Hibernate"],
  "DOB": "1997-08-10",
  "Email id": "alice.johnson@tcs.com",
  "Contact": "555-555-5555",
  "Address": [
   { "PAddr": "321 Pine St", "LAddr": "Pune", "Pin_code": "411001" }
  ]
 },
 {
  "Emp_id": 1005,
  "Name": { "FName": "Eve", "LName": "Davis" },
  "Company Name": "Accenture",
  "Salary": 130000,
  "Designation": "Manager",
  "Age": 35,
  "Expertise": ["Project Management", "Leadership", "Agile"],
```

```
  "DOB": "1988-03-20",
  "Email id": "eve.davis@accenture.com",
  "Contact": "111-222-3333",
  "Address": [
   { "PAddr": "500 Maple St", "LAddr": "Chennai", "Pin_code": "600001" }
  ]
 }
]);
```

1. Creates a new document if no document in the employee collection contains
   {Designation: "Tester", Company_name: "TCS", Age: 25}

```
var existingDoc = db.Employee.findOne({ "Designation": "Tester", "Company Name": "TCS", "Age": 25 });
if (!existingDoc) {
 db.Employee.insertOne({
  "Emp_id": 1006,
  "Name": { "FName": "Sam", "LName": "Taylor" },
  "Company Name": "TCS",
  "Salary": 90000,
  "Designation": "Tester",
  "Age": 25,
  "Expertise": ["Automation", "Selenium", "Testing"],
  "DOB": "1998-03-25",
  "Email id": "sam.taylor@tcs.com",
  "Contact": "123-321-4321",
  "Address": [
   { "PAddr": "777 Park Ave", "LAddr": "Pune", "Pin_code": "411001" }
  ]
 });
}
```

2. Finds all employees working with Company_name: "TCS" and increase their salary by 2000.

```
db.Employee.updateMany(
 { "Company Name": "TCS" },
 { $inc: { "Salary": 2000 } }
);
```

3. Matches all documents where the value of the field Address is an embedded document that contains only the field city with the value "Pune" and the field Pin_code with the value "411001".

```
db.Employee.find({
  "Address": {
    $elemMatch: {
      "LAddr": "Pune",
      "Pin_code": "411001"
    }
```

	}
	});
	4. Find employee details who are working as "Developer" or "Tester".
	db.Employee.find({
	"Designation": { $in: ["Developer", "Tester"] }
	});
	5. Drop Single documents where designation="Developer".
	db.Employee.deleteOne({ "Designation": "Developer" });
	6. Count number of documents in employee collection.
	db.Employee.countDocuments();

M6	**Design MongoDB database and perform following Map reduce operation:**
	Create Employee collection by considering following Fields:
	i. Name: Embedded Doc (FName, LName)
	ii. Company Name: String
	iii. Salary: Number
	iv. Designation: String
	v. Age: Number
	vi. Expertise: Array
	vii. DOB: String or Date
	viii. Email id: String ix. Contact: String
	x. Address: Array of Embedded Doc (PAddr, LAddr) Execute the following query:
	{
	"Name": {

```
    "FName": "John",
    "LName": "Doe"
   },
   "Company Name": "TCS",
   "Salary": 120000,
   "Designation": "DBA",
   "Age": 32,
   "Expertise": ["SQL", "MongoDB", "Database Management"],
   "DOB": "1991-03-15",
   "Email id": "john.doe@example.com",
   "Contact": "123-456-7890",
   "Address": [
    { "PAddr": "123 Main St", "LAddr": "Pune" }
   ]
  }
```

1. Display the total salary of per company

```
var map = function() {
  emit(this['Company Name'], this['Salary']);
};

var reduce = function(key, values) {
  return Array.sum(values);
};

db.Employee.mapReduce(map, reduce, { out: "total_salary_per_company" });
```

2. Display the total salary of company Name:"TCS"

```
var map = function() {
  if (this['Company Name'] == 'TCS') {
    emit(this['Company Name'], this['Salary']);
  }
};

var reduce = function(key, values) {
  return Array.sum(values);
};

db.Employee.mapReduce(map, reduce, { out: "total_salary_tcs" });
```

3. Return the average salary of company whose address is “Pune".

```
var map = function() {
  for (var i = 0; i < this.Address.length; i++) {
    if (this.Address[i].LAddr == 'Pune') {
      emit(this['Company Name'], { salary: this['Salary'], count:
1 });
    }
```

```
  }
};

var reduce = function(key, values) {
  var totalSalary = 0;
  var totalCount = 0;
  values.forEach(function(value) {
    totalSalary += value.salary;
    totalCount += value.count;
  });
  return { salary: totalSalary, count: totalCount };
};

var finalize = function(key, reducedValue) {
  reducedValue.averageSalary = reducedValue.salary /
reducedValue.count;
  return reducedValue;
};

db.Employee.mapReduce(map, reduce, { out: "average_salary_pune",
finalize: finalize });
```

4. Display total count for "City=Pune"

```
var map = function() {
 for (var i = 0; i < this.Address.length; i++) {
  if (this.Address[i].LAddr == 'Pune') {
   emit('Pune', 1);
  }
 }
};

var reduce = function(key, values) {
 return Array.sum(values);
};

db.Employee.mapReduce(map, reduce, { out: "count_pune" });
```

5. Return count for city pune and age greater than 40.

```
var map = function() {
  for (var i = 0; i < this.Address.length; i++) {
    if (this.Address[i].LAddr == 'Pune' && this['Age'] > 40) {
      emit('Pune_Above_40', 1);
    }
  }
};

var reduce = function(key, values) {
  return Array.sum(values);
};
```

```
db.Employee.mapReduce(map, reduce, { out: "count_pune_above_40"
});
```

M4

Design and Develop MongoDB Queries using Aggregation operations:

Create Employee collection by considering following Fields:

i. Emp_id : Number

ii. Name: Embedded Doc (FName, LName)

iii. Company Name: String

iv. Salary: Number

v. Designation: String

vi. Age: Number

vii. Expertise: Array

viii. DOB: String or Date

ix. Email id: String x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

```
db.Employee.insertMany([
 {
  "Emp_id": 1001,
  "Name": { "FName": "John", "LName": "Doe" },
  "Company Name": "TechCorp",
  "Salary": 120000,
  "Designation": "DBA",
  "Age": 32,
  "Expertise": ["SQL", "MongoDB", "Database Management"],
  "DOB": ISODate("1991-03-15"),
  "Email id": "john.doe@example.com",
  "Contact": "123-456-7890",
  "Address": [
   { "PAddr": "123 Main St", "LAddr": "Cityville" },
   { "PAddr": "456 Elm St", "LAddr": "Townsville" }
  ]
 },
 {
  "Emp_id": 1002,
  "Name": { "FName": "Jane", "LName": "Smith" },
  "Company Name": "WebSoft",
  "Salary": 200000,
  "Designation": "DBA",
  "Age": 30,
  "Expertise": ["NoSQL", "MongoDB", "DBA"],
  "DOB": ISODate("1993-07-20"),
  "Email id": "jane.smith@example.com",
  "Contact": "987-654-3210",
  "Address": [
   { "PAddr": "789 Oak St", "LAddr": "Lakeside" },
   { "PAddr": "321 Pine St", "LAddr": "Riverside" }
  ]
 },
 {
  "Emp_id": 1003,
  "Name": { "FName": "Swapnil", "LName": "Jadhav" },
  "Company Name": "TechCorp",
  "Salary": 180000,
```

```
    "Designation": "Engineer",
    "Age": 28,
    "Expertise": ["Python", "MongoDB"],
    "DOB": ISODate("1995-01-10"),
    "Email id": "swapnil.jadhav@example.com",
    "Contact": "555-789-1234",
    "Address": [
     { "PAddr": "500 Maple St", "LAddr": "Cityville" },
     { "PAddr": "700 Cedar St", "LAddr": "Uptown" }
    ]
   },
   {
    "Emp_id": 1004,
    "Name": { "FName": "Amit", "LName": "Patel" },
    "Company Name": "WebSoft",
    "Salary": 250000,
    "Designation": "DBA",
    "Age": 34,
    "Expertise": ["MySQL", "MongoDB", "DBA"],
    "DOB": ISODate("1989-05-05"),
    "Email id": "amit.patel@example.com",
    "Contact": "666-432-9876",
    "Address": [
     { "PAddr": "123 Elm St", "LAddr": "Woodland" },
     { "PAddr": "900 Birch St", "LAddr": "Hillside" }
    ]
   },
   {
    "Emp_id": 1005,
    "Name": { "FName": "Emily", "LName": "White" },
    "Company Name": "TechCorp",
    "Salary": 220000,
    "Designation": "Manager",
    "Age": 40,
    "Expertise": ["Leadership", "Project Management"],
    "DOB": ISODate("1983-11-15"),
    "Email id": "emily.white@example.com",
    "Contact": "888-555-1234",
    "Address": [
     { "PAddr": "100 Pine St", "LAddr": "Mountainview" },
     { "PAddr": "400 Oak St", "LAddr": "Seaside" }
    ]
   }
]);
```

Insert at least 5 documents in collection by considering above attribute and execute following:

1. Using aggregation Return Designation with Total Salary is Above 200000.

```
db.Employee.aggregate([
 { $group: { _id: "$Designation", totalSalary: { $sum: "$Salary" } } },
 { $match: { totalSalary: { $gt: 200000 } } }
]);
```

2. Using Aggregate method returns names and _id in upper case and in alphabetical order.

```
db.Employee.aggregate([
```

```
 { $project: {
    Name: { $concat: [{ $toUpper: "$Name.FName" }, " ", { $toUpper: "$Name.LName" }] },
    _id: 1
  }
 },
 { $sort: { Name: 1 } }
]);
```

3. Using aggregation method find Employee with Total Salary for Each City with Designation="DBA".

```
db.Employee.aggregate([
 { $match: { Designation: "DBA" } },
 { $unwind: "$Address" },
 { $group: { _id: "$Address.LAddr", totalSalary: { $sum: "$Salary" } } }
]);
```

4. Create Single Field Indexes on Designation field of employee collection

```
db.Employee.createIndex({ "Designation": 1 });
```

5. To Create Multikey Indexes on Expertise field of employee collection.

```
db.Employee.createIndex({ "Expertise": 1 });
```

6. Create an Index on Emp_id field, compare the time require to search Emp_id before and after creating an index. (Hint Add at least 10000 Documents)

```
let start = new Date();
db.Employee.find({ "Emp_id": 5000 });
let end = new Date();
print("Time before index: " + (end - start) + " ms");

db.Employee.createIndex({ "Emp_id": 1 });

start = new Date();
db.Employee.find({ "Emp_id": 5000 });
end = new Date();
print("Time after index: " + (end - start) + " ms");
```

7. Return a List of Indexes on created on employee Collection.

```
db.Employee.getIndexes();
```

P6

Write a **Stored Procedure** namely **proc_Grade** for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 categories is first class, if marks 899 and 825 category is Higher Second Class.
Write a PL/SQL block for using procedure created with above requirement.
Stud_Marks(name, total_marks),
Result (Roll,Name, Class)

```
CREATE TABLE Stud_Marks (
   name VARCHAR2(50) NOT NULL,      -- Student name
   total_marks NUMBER NOT NULL,     -- Total marks scored by the student
   CONSTRAINT pk_stud_marks PRIMARY KEY (name)  -- Primary key constraint
);

CREATE TABLE Result (
   Roll NUMBER PRIMARY KEY,         -- Unique roll number
   Name VARCHAR2(50) NOT NULL,      -- Student name
   Class VARCHAR2(50) NOT NULL      -- Categorized class (Distinction, First Class, etc.)
);


CREATE OR REPLACE PROCEDURE proc_Grade (
   p_name IN VARCHAR2,    -- Name of the student
   p_marks IN NUMBER      -- Total marks scored by the student
)
IS
   v_class VARCHAR2(20);  -- Variable to hold the category/class of the student
BEGIN
   -- Categorization based on marks
   IF p_marks >= 990 AND p_marks <= 1500 THEN
      v_class := 'Distinction';
   ELSIF p_marks >= 900 AND p_marks <= 989 THEN
      v_class := 'First Class';
   ELSIF p_marks >= 825 AND p_marks <= 899 THEN
      v_class := 'Higher Second Class';
   ELSE
      v_class := 'Not Categorized';  -- In case marks don't fit into any category
   END IF;

   -- Insert the result into Result table
   INSERT INTO Result (Name, Class)
   VALUES (p_name, v_class);

   COMMIT;  -- Commit the transaction
EXCEPTION
   WHEN OTHERS THEN
      -- Handle any exceptions (e.g., insertion failure)
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Error occurred while categorizing the student.');
END proc_Grade;
/
```

C1

Write a program to implement **MongoDB database connectivity** with PHP /python /Java Implement Database navigation CRUD operations (add, delete, edit etc.)

composer require mongodb/mongodb

```
<?php
require 'vendor/autoload.php';  // Autoload the MongoDB library

// Create MongoDB Client
$client = new MongoDB\Client("mongodb://localhost:27017");

// Select the database and collection
$collection = $client->mydatabase->employees;

// CREATE - Insert a new employee
$insertResult = $collection->insertOne([
   'name' => ['first' => 'John', 'last' => 'Doe'],
   'age' => 28,
   'company' => 'TCS',
   'salary' => 50000,
   'designation' => 'Programmer',
]);

echo "Inserted employee with ID: " . $insertResult->getInsertedId() . "\n";

// READ - Find all employees
$employees = $collection->find();
foreach ($employees as $employee) {
   echo "Name: " . $employee['name']['first'] . " " . $employee['name']['last'] . "\n";
}

// UPDATE - Update salary of employee where name is 'John'
$updateResult = $collection->updateOne(
   ['name.first' => 'John'],  // Filter
   ['$set' => ['salary' => 55000]]  // Update
);
echo "Matched " . $updateResult->getMatchedCount() . " document(s)\n";

// DELETE - Delete employee with name 'John'
$deleteResult = $collection->deleteOne(['name.first' => 'John']);
echo "Deleted " . $deleteResult->getDeletedCount() . " document(s)\n";
?>
```

C2

Implement **MYSQL/Oracle database connectivity** with PHP /python /Java Implement Database navigation operations (add, delete, edit,).

PHP (MySQL)

Step 1: MySQL Database Connection in PHP

php

Copy code

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
```

```
$dbname = "your_database_name";

// Create connection
$conn = new mysqli($servername, $username, $password,
$dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Step 2: Database Navigation (Add, Delete, Edit)

- **Add (Insert):**

```
php
Copy code
<?php
$sql = "INSERT INTO users (name, email) VALUES ('John Doe',
'john@example.com')";
if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
?>
```

- **Delete:**

```
php
Copy code
<?php
$sql = "DELETE FROM users WHERE id = 1";
if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
?>
```

- **Edit (Update):**

```
php
Copy code
<?php
$sql = "UPDATE users SET email='newemail@example.com' WHERE
id=1";
if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
?>
```

Step 3: Closing the Connection

```
php
Copy code
$conn->close();
?>
```

P2

Write an **Unnamed PL/SQL** of code for the following requirements: -
Schema:
Borrower (Rollin, Name, DateofIssue, NameofBook, Status)
Fine (Roll_no,Date,Amt)
Accept roll_no & name of book from user.
Check the number of days (from date of issue).

1. If days are between 15 to 30 then fine amounts will be Rs 5 per day.
2. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
3. After submitting the book, status will change from I to R.
4. If condition of fine is true, then details will be stored into fine table.

```
-- Create the Borrower table to store information about borrowed
books
CREATE TABLE Borrower (
    Rollin NUMBER PRIMARY KEY,                  -- Roll number of the
student (Primary Key)
    Name VARCHAR2(100) NOT NULL,                 -- Name of the
student
    DateofIssue DATE NOT NULL,                   -- Date when the
book was issued
    NameofBook VARCHAR2(255) NOT NULL,           -- Name of the
borrowed book
    Status CHAR(1) CHECK (Status IN ('I', 'R')) -- Status of the
book (I = Issued, R = Returned)
);

-- Create the Fine table to store the fine details
CREATE TABLE Fine (
    Roll_no NUMBER,                              -- Roll number of
the student
    Date DATE DEFAULT SYSDATE,                   -- Date when the
fine is registered
    Amt NUMBER(10, 2),                           -- Amount of the
fine
    CONSTRAINT fk_rollno FOREIGN KEY (Roll_no) REFERENCES
Borrower(Rollin) -- Foreign Key referencing Borrower table
);


DECLARE
  v_roll_no   NUMBER;      -- Variable to hold roll number
  v_name      VARCHAR2(100);  -- Variable to hold name of the book
  v_dateofissue DATE;      -- Variable to hold date of issue of the book
  v_days      NUMBER;      -- Variable to calculate number of days
  v_fine_amt  NUMBER := 0;  -- Variable to hold the fine amount
  v_status    CHAR(1);     -- Variable to hold the current status of the book
BEGIN
  -- Accept roll_no and name of the book as inputs from the user
```

```
    v_roll_no := &roll_no;
    v_name := '&book_name';

    -- Retrieve the DateofIssue and Status for the provided roll_no and NameofBook
    SELECT DateofIssue, Status INTO v_dateofissue, v_status
    FROM Borrower
    WHERE Rollin = v_roll_no AND NameofBook = v_name;

    -- Calculate the number of days from DateofIssue to today's date
    v_days := TRUNC(SYSDATE) - v_dateofissue;

    -- If days between 15 and 30, fine will be Rs. 5 per day
    IF v_days BETWEEN 15 AND 30 THEN
        v_fine_amt := v_days * 5;

    -- If days are greater than 30, fine is Rs. 50 per day for first 30 days and Rs. 5 per day after
    ELSIF v_days > 30 THEN
        v_fine_amt := (30 * 50) + ((v_days - 30) * 5);
    END IF;

    -- If there is a fine, insert into Fine table and update Status in Borrower table
    IF v_fine_amt > 0 THEN
        -- Insert the fine details into Fine table
        INSERT INTO Fine (Roll_no, Date, Amt)
        VALUES (v_roll_no, SYSDATE, v_fine_amt);

        -- Update the Status in Borrower table to 'R' (Returned)
        UPDATE Borrower
        SET Status = 'R'
        WHERE Rollin = v_roll_no AND NameofBook = v_name;

        -- Commit the changes to database
        COMMIT;
    ELSE
        -- If no fine, just update the status of the book to 'R'
        UPDATE Borrower
        SET Status = 'R'
        WHERE Rollin = v_roll_no AND NameofBook = v_name;

        -- Commit the changes to database
        COMMIT;
    END IF;

    -- Output the fine amount if any
    IF v_fine_amt > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Fine to be paid: Rs. ' || v_fine_amt);
    ELSE
        DBMS_OUTPUT.PUT_LINE('No fine. Book returned on time.');
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
```

```
    DBMS_OUTPUT.PUT_LINE('No record found for the provided Roll number and Book name.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

M5

Design and Develop MongoDB Queries using Aggregation operations:

Create Employee collection by considering following Fields:

i. Emp_id : Number
ii. Name: Embedded Doc (FName, LName)
iii. Company Name: String
iv. Salary: Number
v. Designation: String
vi. Age: Number
vii. Expertise: Array
viii. DOB: String or Date
ix. Email id: String
x. Contact: String
xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following:

1. Using aggregation Return separates value in the Expertise array and return sum of each element of array.

```
db.Employee.aggregate([
 { $unwind: "$Expertise" },  // Unwind the expertise array to separate each expertise into a document
 { $group: { _id: "$Expertise", totalCount: { $sum: 1 } } } // Group by expertise and count the occurrences
])
```

2. Using Aggregate method return Max and Min Salary for each company.

```
db.Employee.aggregate([
  { $group: {
      _id: "$Company Name",
      maxSalary: { $max: "$Salary" },
      minSalary: { $min: "$Salary" }
  }}
])
```

3. Using Aggregate method find Employee with Total Salary for Each City with Designation="DBA".

```
db.Employee.aggregate([
 { $match: { "Designation": "DBA" } },  // Filter by designation "DBA"
 { $group: {
    _id: { "city": { $arrayElemAt: ["$Address.LAddr", 0] } },  // Group by city
    totalSalary: { $sum: "$Salary" }
 }}
])
```

4. Using aggregation method Return separates value in the Expertise array for employee name where Swapnil Jadhav

```
db.Employee.aggregate([
 { $match: { "Name.FName": "Swapnil", "Name.LName": "Jadhav" } },  // Filter by employee name
 { $unwind: "$Expertise" },  // Unwind the expertise array
 { $project: { Expertise: 1 } }  // Return expertise field
])
```

5. To Create Compound Indexes on Name: 1, Age: -1

```
db.Employee.createIndex({ "Name.FName": 1, "Age": -1 })
```

6. Create an Index on Emp_id field, compare the time require to search Emp_id before and after creating an index. (Hint Add at least 10000 Documents)

```
// To test search time without index
let startTime = new Date();
db.Employee.find({ "Emp_id": 5000 }).toArray(); // Search for a specific Emp_id
let endTime = new Date();
print("Time without index:", endTime - startTime, "ms");
db.Employee.find({ "Emp_id": 5000 }).explain("executionStats")
db.Employee.createIndex({ "Emp_id": 1 })
db.Employee.find({ "Emp_id": 5000 }).explain("executionStats")
// To test search time after index creation
startTime = new Date();
db.Employee.find({ "Emp_id": 5000 }).toArray(); // Search for a specific Emp_id
endTime = new Date();
print("Time with index:", endTime - startTime, "ms");
```

7. Return a List of Indexes on created on employee Collection.

```
db.Employee.getIndexes()
```