

Problem Statement 1 (CRUD)

Create collection Student with fields as Roll_No, Name, Class, Marks, Address, Enrolled_Courses.

(Hint: One student can enrol in multiple courses. Use Array to store the names of courses enrolled)

Insert 10 documents in the collection Student. Write the queries for following.

1. List the names of students who have enrolled in the course "DBMS", "TOC".
2. List the Roll numbers and class of students who have marks more than 50 or class as TE.
3. Update the entire record of roll_no A10.
4. Display the names of students having 3rd and 4th highest marks.
5. Delete the records of students having marks less than 20.
6. Delete only first record from the collection.

ANS

use college;

db.Student.insertMany([

{ Roll_No: "A01", Name: "Riya", Class: "SE", Marks: 65, Address: "Pune", Enrolled_Courses: ["DBMS", "OS"] },

{ Roll_No: "A02", Name: "Amit", Class: "TE", Marks: 80, Address: "Mumbai", Enrolled_Courses: ["TOC", "CN"] },

{ Roll_No: "A03", Name: "Sneha", Class: "BE", Marks: 45, Address: "Pune", Enrolled_Courses: ["DBMS", "AI"] },

{ Roll_No: "A04", Name: "Raj", Class: "TE", Marks: 55, Address: "Nashik", Enrolled_Courses: ["TOC", "SE"] },

{ Roll_No: "A05", Name: "Karan", Class: "SE", Marks: 30, Address: "Pune", Enrolled_Courses: ["CN"] },

{ Roll_No: "A06", Name: "Priya", Class: "TE", Marks: 95, Address: "Nagpur", Enrolled_Courses: ["DBMS", "TOC"] },

{ Roll_No: "A07", Name: "Nisha", Class: "BE", Marks: 18, Address: "Kolhapur", Enrolled_Courses: ["AI", "SE"] },

{ Roll_No: "A08", Name: "Ankit", Class: "SE", Marks: 72, Address: "Pune", Enrolled_Courses: ["DBMS", "CN"] },

{ Roll_No: "A09", Name: "Tina", Class: "TE", Marks: 90, Address: "Mumbai", Enrolled_Courses: ["TOC", "AI"] },

{ Roll_No: "A10", Name: "Rohan", Class: "BE", Marks: 35, Address: "Nashik", Enrolled_Courses: ["OS", "DBMS"] }

])

Query 1: Names of students enrolled in "DBMS" or "TOC"

db.Student.find(

{ Enrolled_Courses: { \$in: ["DBMS", "TOC"] } },

{ Name: 1, _id: 0 }

)

Query 2: Roll numbers & class of students having marks > 50 or class = "TE"

db.Student.find(

{ \$or: [{ Marks: { \$gt: 50 } }, { Class: "TE" }] },

{ Roll_No: 1, Class: 1, _id: 0 }

)

Query 3: Update entire record of Roll_No = "A10"

```
db.Student.updateOne(
  { Roll_No: "A10" },
  {
    $set: {
      Name: "Rohan Patil",
      Class: "TE",
      Marks: 75,
      Address: "Pune",
      Enrolled_Courses: ["DBMS", "TOC"]
    }
  }
)
```

Query 4: Display students having 3rd & 4th highest marks

```
db.Student.find({}, { Name: 1, Marks: 1, _id: 0 })
  .sort({ Marks: -1 })
  .skip(2)
  .limit(2)
```

Query 5: Delete students having marks less than 20

```
db.Student.deleteMany({ Marks: { $lt: 20 } })
```

Query 6: Delete only first record from collection

```
db.Student.deleteOne({})
```

Problem Statement 2 (DDL USING MYSQL)

Create following tables using a given schema and insert appropriate data into the same:

Customer (CustID, Name, Cust_Address, Phone_no, Email_ID, Age)

Branch (Branch ID, Branch_Name, Address)

Account (Account_no, Branch ID, CustID, open_date, Account_type, Balance)

1. Create the tables with referential integrity.
2. Draw the ER diagram for the same.
3. Create a View as Saving account displaying the customer details with the open date as 16/8/2018.
4. Update the View with Cust_Address as Pune for CustID =103.
5. Create a View as Loan account displaying the customer details with the open date as 16/2/2018.
6. Create an Index on primary key column of table Customer.
7. Create an Index on primary key column of table Branch.
8. Create a sequence on Customer Table.
9. Create synonym 'Cust_info' for branch table.

Tables

```
CREATE DATABASE BankDB;
USE BankDB;
```

```
CREATE TABLE Customer (
  CustID INT PRIMARY KEY,
  Name VARCHAR(50),
  Cust_Address VARCHAR(100),
  Phone_no VARCHAR(15),
  Email_ID VARCHAR(50),
  Age INT
);
```

```
CREATE TABLE Branch (
  Branch_ID INT PRIMARY KEY,
  Branch_Name VARCHAR(50),
  Address VARCHAR(100)
);
```

```
CREATE TABLE Account (
  Account_no INT PRIMARY KEY,
  Branch_ID INT,
  CustID INT,
  open_date DATE,
  Account_type VARCHAR(20),
  Balance DECIMAL(10,2),
  FOREIGN KEY (Branch_ID) REFERENCES Branch(Branch_ID),
  FOREIGN KEY (CustID) REFERENCES Customer(CustID)
);
```

```
INSERT INTO Customer VALUES
(101, 'Riya', 'Mumbai', '9876543210', 'riya@gmail.com', 22),
(102, 'Raj', 'Pune', '9988776655', 'raj@gmail.com', 25),
(103, 'Tina', 'Delhi', '9123456789', 'tina@gmail.com', 24);
```

```
INSERT INTO Branch VALUES
(1, 'Main', 'Mumbai'),
(2, 'Camp', 'Pune'),
(3, 'Connaught', 'Delhi');
```

```
INSERT INTO Account VALUES
(5001, 1, 101, '2018-08-16', 'Saving', 15000),
(5002, 2, 102, '2018-02-16', 'Loan', 25000),
(5003, 3, 103, '2018-08-16', 'Saving', 30000);
```

Queries

3. View: Saving accounts opened on 16/8/2018

```
CREATE VIEW Saving_Account AS
SELECT c.*, a.open_date
```

```
FROM Customer c
JOIN Account a ON c.CustID = a.CustID
WHERE a.Account_type = 'Saving' AND a.open_date = '2018-08-16';
```

4. Update view

```
UPDATE Saving_Account
SET Cust_Address = 'Pune'
WHERE CustID = 103;
```

5. View: Loan account opened on 16/2/2018

```
CREATE VIEW Loan_Account AS
SELECT c.*, a.open_date
FROM Customer c
JOIN Account a ON c.CustID = a.CustID
WHERE a.Account_type = 'Loan' AND a.open_date = '2018-02-16';
```

6. Index on Customer primary key

```
CREATE INDEX cust_index ON Customer(CustID);
```

7. Index on Branch primary key

```
CREATE INDEX branch_index ON Branch(Branch_ID);
```

8. Sequence on Customer table (for auto CustID)

```
CREATE SEQUENCE cust_seq START WITH 101 INCREMENT BY 1;
```

9. Synonym for Branch table

```
CREATE SYNONYM Cust_info FOR Branch;
```

Problem Statement 3 (Unnamed Block)

Employee(emp_id, dept_id, emp_name, DoJ, salary, commission, job_title)

Salary_Increment(emp_id, new_salary)

Consider the schema given above. Write a PLSQL Unnamed Block of code to increase the salary of employee

115 based on the following conditions:

Accept emp_id from user. If experience of employee is more than 10 years, increase salary by 20%. If experience

is greater than 5 years, increase salary by 10% Otherwise 5%. (Hint: Find the years of experience from Date of

Joining (DoJ)). Store the incremented salary in Salary_Increment table.

Also handle the exception by named exception handler or user defined exception handler.

Tables

Employee(emp_id, dept_id, emp_name, DoJ, salary, commission, job_title)

Salary_Increment(emp_id, new_salary)

PL/SQL Code

```
DECLARE
```

```

v_emp_id Employee.emp_id%TYPE := &emp_id;
v_salary Employee.salary%TYPE;
v_doj Employee.DoJ%TYPE;
v_years NUMBER;
v_new_salary NUMBER;
e_invalid EXCEPTION;
BEGIN
    SELECT salary, DoJ INTO v_salary, v_doj
    FROM Employee WHERE emp_id = v_emp_id;

    v_years := FLOOR(MONTHS_BETWEEN(SYSDATE, v_doj)/12);

    IF v_years > 10 THEN
        v_new_salary := v_salary + v_salary*0.20;
    ELSIF v_years > 5 THEN
        v_new_salary := v_salary + v_salary*0.10;
    ELSE
        v_new_salary := v_salary + v_salary*0.05;
    END IF;

    INSERT INTO Salary_Increment VALUES (v_emp_id, v_new_salary);
    DBMS_OUTPUT.PUT_LINE('New Salary: ' || v_new_salary);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee not found');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred');
END;
/

```

Problem Statement 4 (AGGREGATION & INDEXING USING MONGODB)

Create the Collection Student_Data(Student_ID, Student_Name, Department, Marks)and solve the following:

1. Display all Students based on their departments along with an average Marks of a particular department.
2. Displays the number of Students associated along with a particular department.
3. Display list of Students with the highest Marks in each Department in descending order of Marks.
4. Create an index on field Student_ID.
5. Create an index on fields "Student_Name" and "Department".
6. Drop an index on field Student_ID.
7. Drop an index on fields "Student_Name" and "Department".

```
db.Student_Data.insertMany([
  { Student_ID: 1, Student_Name: "Riya", Department: "CSE", Marks: 80 },
  { Student_ID: 2, Student_Name: "Amit", Department: "CSE", Marks: 75 },
  { Student_ID: 3, Student_Name: "Sneha", Department: "IT", Marks: 65 },
  { Student_ID: 4, Student_Name: "Karan", Department: "IT", Marks: 90 }
])
```

Queries:

1. Average marks per department

```
db.Student_Data.aggregate([
  { $group: { _id: "$Department", avgMarks: { $avg: "$Marks" } } }
])
```

2. Count of students per department

```
db.Student_Data.aggregate([
  { $group: { _id: "$Department", count: { $sum: 1 } } }
])
```

3. Highest marks in each department (descending)

```
db.Student_Data.aggregate([
  { $sort: { Marks: -1 } },
  { $group: { _id: "$Department", Topper: { $first: "$Student_Name" }, MaxMarks: { $first: "$Marks" } } },
  { $sort: { MaxMarks: -1 } }
])
```

4. Create index on Student_ID

```
db.Student_Data.createIndex({ Student_ID: 1 })
```

5. Create index on Student_Name & Department

```
db.Student_Data.createIndex({ Student_Name: 1, Department: 1 })
```

6. Drop index on Student_ID

```
db.Student_Data.dropIndex({ Student_ID: 1 })
```

7. Drop index on Student_Name & Department

```
db.Student_Data.dropIndex({ Student_Name: 1, Department: 1 })
```

Problem Statement 5 (DML USING MYSQL)

Create following tables using a given schema and insert appropriate data into the same:

Customer (CustID, Name, Cust_Address, Phone_no, Age)

Branch (Branch ID, Branch_Name, Address)

Account (Account_no, Branch ID, CustID, date_open, Account_type, Balance)

1. Add the column "Email_Address" in Customer table.

2. Change the name of column "Email_Address" to "Email_ID" in Customer table.

3. Display the customer details with highest balance in the account.
4. Display the customer details with lowest balance for account type= "Saving Account".
5. Display the customer details that live in Pune and have age greater than 35.
6. Display the Cust_ID, Name and Age of the customer in ascending order of their age.
7. Display the Name and Branch ID of the customer group by the Account_type.

Tables

```
CREATE TABLE Customer (  
  CustID INT PRIMARY KEY,  
  Name VARCHAR(50),  
  Cust_Address VARCHAR(100),  
  Phone_no VARCHAR(15),  
  Email_ID VARCHAR(50),  
  Age INT  
);
```

```
CREATE TABLE Branch (  
  Branch_ID INT PRIMARY KEY,  
  Branch_Name VARCHAR(50),  
  Address VARCHAR(100)  
);
```

```
CREATE TABLE Account (  
  Account_no INT PRIMARY KEY,  
  Branch_ID INT,  
  CustID INT,  
  open_date DATE,  
  Account_type VARCHAR(20),  
  Balance DECIMAL(10,2),  
  FOREIGN KEY (Branch_ID) REFERENCES Branch(Branch_ID),  
  FOREIGN KEY (CustID) REFERENCES Customer(CustID)  
);
```

```
INSERT INTO Customer VALUES  
(101, 'Riya', 'Pune', '9876543210', 'riya@gmail.com', 28),  
(102, 'Raj', 'Mumbai', '9988776655', 'raj@gmail.com', 40),  
(103, 'Tina', 'Pune', '9123456789', 'tina@gmail.com', 37),  
(104, 'Amit', 'Delhi', '8765432109', 'amit@gmail.com', 30);
```

```
INSERT INTO Branch VALUES  
(1, 'Main', 'Pune'),  
(2, 'South', 'Mumbai'),  
(3, 'North', 'Delhi');
```

```
INSERT INTO Account VALUES  
(5001, 1, 101, '2018-08-16', 'Saving Account', 45000),  
(5002, 2, 102, '2018-02-16', 'Current Account', 78000),  
(5003, 3, 103, '2019-01-05', 'Saving Account', 12000),
```

(5004, 1, 104, '2020-05-20', 'Saving Account', 52000);

****Queries

1. Add column

ALTER TABLE Customer ADD Email_Address VARCHAR(50);

2. Rename column

ALTER TABLE Customer CHANGE Email_Address Email_ID VARCHAR(50);

3. Customer with highest balance

**SELECT c.*
FROM Customer c
JOIN Account a ON c.CustID = a.CustID
WHERE a.Balance = (SELECT MAX(Balance) FROM Account);**

4. Lowest balance for Saving Account

**SELECT c.*
FROM Customer c
JOIN Account a ON c.CustID = a.CustID
WHERE a.Account_type='Saving' AND a.Balance=(SELECT MIN(Balance) FROM Account WHERE
Account_type='Saving');**

5. Customers from Pune & age > 35

SELECT * FROM Customer WHERE Cust_Address='Pune' AND Age>35;

6. Cust_ID, Name, Age (ascending by age)

SELECT CustID, Name, Age FROM Customer ORDER BY Age ASC;

7. Name & Branch ID grouped by account type

**SELECT c.Name, a.Branch_ID, a.Account_type
FROM Customer c
JOIN Account a ON c.CustID=a.CustID
GROUP BY a.Account_type, c.Name, a.Branch_ID;**

Problem Statement 6 (Procedures / Functions)

Employee(emp_id, dept_id emp_name, DoJ, salary, commission, job_title)

1. Consider the schema given above. Keep the commission column empty initially. Write a Stored Procedure to

record the employee commission based on following conditions. If salary is more than 10000 then commission

is 0.4%, if Salary is less than 10000 but experience is more than 10 years then 0.35%, if salary is less than 3000

then commission is 0.25%. In the remaining cases commission is 0.15%.

2. Write a PLSQL Function that takes department ID and returns the name of the manager of the department.

1. Stored Procedure – Calculate Commission(MySQL)

```
DELIMITER $$
CREATE PROCEDURE update_commission()
BEGIN
    DECLARE exp_years INT;
    DECLARE doj DATE;

    -- Update each employee one by one
    UPDATE Employee
    SET commission =
        CASE
            WHEN salary > 10000 THEN salary * 0.004
            WHEN salary < 10000 AND TIMESTAMPDIF(YEAR, DoJ, CURDATE()) > 10 THEN salary * 0.0035
            WHEN salary < 3000 THEN salary * 0.0025
            ELSE salary * 0.0015
        END;
END$$
DELIMITER ;
```

****To run it:**

```
CALL update_commission();
```

2. Function – Get Manager Name by Department ID(PLSQL)

```
CREATE OR REPLACE FUNCTION get_manager_name(p_dept_id IN NUMBER)
RETURN VARCHAR2 IS
    v_manager_name VARCHAR2(50);
BEGIN
    SELECT emp_name INTO v_manager_name
    FROM Employee
    WHERE dept_id = p_dept_id AND job_title = 'Manager';

    RETURN v_manager_name;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'No Manager Found';
END;
/
```

Problem Statement 7 (Map Reduce)

Create Book Collection with (Title, Author_name, Borrowed_status) as fields. Write Map Reduce Functions for following requirements.

1. Display Author wise list of books.
2. Display Author wise list of books having Borrowed status as “True”.
3. Display Author wise list of books having price greater than 300.

Collection: Book(Title, Author_name, Borrowed_status, Price)

Step 1: Sample Documents

```
db.Book.insertMany([
  { Title: "DBMS Simplified", Author_name: "Navathe", Borrowed_status: true, Price: 450 },
  { Title: "AI Basics", Author_name: "Russell", Borrowed_status: false, Price: 300 },
  { Title: "TOC", Author_name: "Hopcroft", Borrowed_status: true, Price: 400 },
  { Title: "CN", Author_name: "Forouzan", Borrowed_status: true, Price: 350 },
  { Title: "ML Guide", Author_name: "Russell", Borrowed_status: true, Price: 600 }
])
```

1. Author-wise list of books

```
db.Book.mapReduce(
  function() { emit(this.Author_name, this.Title); },
  function(key, values) { return values; },
  { out: "AuthorWiseBooks" }
)
```

2. Author-wise list of borrowed books (Borrowed_status = true)

```
db.Book.mapReduce(
  function() { if (this.Borrowed_status) emit(this.Author_name, this.Title); },
  function(key, values) { return values; },
  { out: "BorrowedBooks" }
)
```

3. Author-wise list of books with price > 300

```
db.Book.mapReduce(
  function() { if (this.Price > 300) emit(this.Author_name, this.Title); },
  function(key, values) { return values; },
  { out: "ExpensiveBooks" }
)
```

Problem Statement 8 (JOINS & SUBQUERIES USING MYSQL)

Consider Following Schema

Employee (Employee_id, First_name, last_name , hire_date, salary, Job_title, manager_id, department_id)

Departments(Department_id, Department_name, Manager_id, Location_id)

Locations(location_id ,street_address ,postal_code, city, state, country_id)

Manager(Manager_id, Manager_name)

Create the tables with referential integrity. Solve following queries using joins and subqueries.

1. Write a query to find the names (first_name, last_name) and the salaries of the employees who have a

higher salary than the employee whose last_name="Singh".

2. Write a query to find the names (first_name, last_name) of the employees who have a manager and

work for a department based in the United States.

3. Write a query to find the names (first_name, last_name), the salary of the employees whose salary is

greater than the average salary.

4. Write a query to find the employee id, name (last_name) along with their manager_id, manager name

(last_name).

5. Find the names and hire date of the employees who were hired after 'Jones'.

Tables:

```
CREATE TABLE Manager (  
  Manager_id INT PRIMARY KEY,  
  Manager_name VARCHAR(50)  
);
```

```
CREATE TABLE Locations (  
  Location_id INT PRIMARY KEY,  
  street_address VARCHAR(100),  
  postal_code VARCHAR(20),  
  city VARCHAR(50),  
  state VARCHAR(50),  
  country_id VARCHAR(10)  
);
```

```
CREATE TABLE Departments (  
  Department_id INT PRIMARY KEY,  
  Department_name VARCHAR(50),  
  Manager_id INT,  
  Location_id INT,  
  FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id),  
  FOREIGN KEY (Location_id) REFERENCES Locations(Location_id)  
);
```

```
CREATE TABLE Employee (  
  Employee_id INT PRIMARY KEY,  
  First_name VARCHAR(50),  
  Last_name VARCHAR(50),  
  hire_date DATE,  
  salary DECIMAL(10,2),  
  Job_title VARCHAR(50),  
  manager_id INT,  
  department_id INT,  
  FOREIGN KEY (manager_id) REFERENCES Manager(Manager_id),  
  FOREIGN KEY (department_id) REFERENCES Departments(Department_id)
```

);

Table 1: Manager

INSERT INTO Manager (Manager_id, Manager_name) VALUES

(1, 'Kumar'),

(2, 'Patil'),

(3, 'Deshmukh'),

(4, 'Sharma');

Table 2: Locations

INSERT INTO Locations (Location_id, street_address, postal_code, city, state, country_id) VALUES

(101, 'MG Road', '411001', 'Pune', 'Maharashtra', 'IN'),

(102, 'Wall Street', '10005', 'New York', 'NY', 'US'),

(103, 'Silicon Ave', '94016', 'San Francisco', 'CA', 'US'),

(104, 'Link Road', '400001', 'Mumbai', 'Maharashtra', 'IN');

Table 3: Departments

INSERT INTO Departments (Department_id, Department_name, Manager_id, Location_id) VALUES

(10, 'HR', 1, 101),

(20, 'Finance', 2, 102),

(30, 'IT', 3, 103),

(40, 'Sales', 4, 104);

Table 4: Employee

INSERT INTO Employee (Employee_id, First_name, Last_name, hire_date, salary, Job_title, manager_id, department_id) VALUES

(1001, 'Rohit', 'Singh', '2018-02-16', 50000, 'Analyst', 1, 10),

(1002, 'Payal', 'Sharma', '2019-03-10', 60000, 'Developer', 3, 30),

(1003, 'Sneha', 'Patil', '2020-07-12', 45000, 'Clerk', 2, 20),

(1004, 'Ramesh', 'Jones', '2017-05-09', 40000, 'Tester', 3, 30),

(1005, 'Priya', 'Kadam', '2021-01-15', 70000, 'Manager', 4, 40),

(1006, 'Karan', 'More', '2019-12-11', 30000, 'Assistant', 1, 10),

(1007, 'Sakshi', 'Pawar', '2022-08-21', 55000, 'Developer', 3, 30);

Queries:

1. Employees with higher salary than “Singh”

SELECT first_name, last_name, salary

FROM Employee

WHERE salary > (

SELECT salary FROM Employee WHERE last_name = 'Singh'

);

2. Employees who have a manager and work in the USA

SELECT e.first_name, e.last_name

FROM Employee e

JOIN Departments d ON e.department_id = d.Department_id

JOIN Locations l ON d.Location_id = l.location_id

WHERE e.manager_id IS NOT NULL AND l.country_id = 'US';

3. Employees whose salary > average salary

```
SELECT first_name, last_name, salary
FROM Employee
WHERE salary > (SELECT AVG(salary) FROM Employee);
```

4. Employee with manager details

```
SELECT e.Employee_id, e.last_name AS Employee_Name,
       m.manager_id, m.manager_name AS Manager_Name
FROM Employee e
JOIN Manager m ON e.manager_id = m.manager_id;
```

5. Employees hired after "Jones"

```
SELECT first_name, last_name, hire_date
FROM Employee
WHERE hire_date > (
    SELECT hire_date FROM Employee WHERE last_name = 'Jones'
);
```

Problem Statement 9 (Cursors)

Consider a table Employee with schema as Employee (Emp_id, Emp_Name,Salary).

1. Write an explicit cursor to display records of all employees with salary greater than 50,000.
2. Write a PL/SQL block of code using Implicit Cursor that will display total number of tuples in Employee table.
3. Write a PL/SQL block of code using Parameterized Cursor that will display salary of employee id entered by the user.

Table:

```
CREATE TABLE Employee (
    Emp_id NUMBER PRIMARY KEY,
    Emp_Name VARCHAR2(50),
    Salary NUMBER
);
```

```
INSERT INTO Employee VALUES (101, 'Rohit', 48000);
INSERT INTO Employee VALUES (102, 'Payal', 52000);
INSERT INTO Employee VALUES (103, 'Sneha', 60000);
INSERT INTO Employee VALUES (104, 'Karan', 75000);
INSERT INTO Employee VALUES (105, 'Sakshi', 49000);
```

1. Explicit Cursor (salary > 50000)

```

DECLARE
  CURSOR emp_cur IS SELECT Emp_Name, Salary FROM Employee WHERE Salary > 50000;
  v_name Employee.Emp_Name%TYPE;
  v_salary Employee.Salary%TYPE;
BEGIN
  OPEN emp_cur;
  LOOP
    FETCH emp_cur INTO v_name, v_salary;
    EXIT WHEN emp_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_name || ' Salary: ' || v_salary);
  END LOOP;
  CLOSE emp_cur;
END;
/

```

2. Implicit Cursor – count total employees

```

DECLARE
  v_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_count FROM Employee;
  DBMS_OUTPUT.PUT_LINE('Total Employees: ' || v_count);
END;
/

```

3. Parameterized Cursor – salary of given employee id

```

DECLARE
  CURSOR emp_cur(p_id NUMBER) IS SELECT Salary FROM Employee WHERE Emp_id = p_id;
  v_sal Employee.Salary%TYPE;
  v_id NUMBER := &Emp_id;
BEGIN
  OPEN emp_cur(v_id);
  FETCH emp_cur INTO v_sal;
  IF emp_cur%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_sal);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No employee found');
  END IF;
  CLOSE emp_cur;
END;
/

```

Problem Statement 10 (CRUD)

Create a collection Social_Media having fields as User_Id, User_Name, No_of_Posts, No_of_Friends, Friends_List, Interests. (Hint: Friends_List and Interests can be of array type)

Insert 20 documents in the collection Social_Media. Write queries for following.

1. List all the users from collection Social_Media in formatted manner.
2. Find all users having number of posts greater than 100.
3. List the user names and their respective Friends_List
4. Display the user ids and Friends list of users who have more than 5 friends.
5. Display all users with no of posts in descending order.

```
db.Social_Media.insertMany([
  { User_Id: 1, User_Name: "Riya", No_of_Posts: 120, No_of_Friends: 10, Friends_List:
["Amit","Sneha"], Interests: ["Music","Travel"] },
  { User_Id: 2, User_Name: "Amit", No_of_Posts: 80, No_of_Friends: 5, Friends_List:
["Riya","Karan"], Interests: ["Gaming","Movies"] },
  { User_Id: 3, User_Name: "Sneha", No_of_Posts: 150, No_of_Friends: 12, Friends_List:
["Riya","Tina"], Interests: ["Dance","Reading"] },
  { User_Id: 4, User_Name: "Tina", No_of_Posts: 30, No_of_Friends: 3, Friends_List: ["Karan"],
Interests: ["Cooking"] },
  { User_Id: 5, User_Name: "Karan", No_of_Posts: 200, No_of_Friends: 15, Friends_List:
["Amit","Sneha","Riya"], Interests: ["Music","Art"] }
])
```

1. List all users (formatted)

```
db.Social_Media.find().pretty()
```

2. Users with posts > 100

```
db.Social_Media.find({ No_of_Posts: { $gt: 100 } }, { User_Name: 1, No_of_Posts: 1, _id: 0 })
```

3. Show user names with their friends list

```
db.Social_Media.find({}, { User_Name: 1, Friends_List: 1, _id: 0 })
```

4. User ids & friends list where No_of_Friends > 5

```
db.Social_Media.find(
  { No_of_Friends: { $gt: 5 } },
  { User_Id: 1, Friends_List: 1, _id: 0 }
)
```

5. All users sorted by posts (descending)

```
db.Social_Media.find().sort({ No_of_Posts: -1 })
```

Problem Statement 11 (DDL USING MYSQL)

Create following tables using a given schema and insert appropriate data into the same:

Customer (CustID, Name, Cust_Address, Phone_no, Email_ID, Age)

Branch (Branch ID, Branch_Name, Address)

Account (Account_no, Branch ID, CustID, date_open, Account_type, Balance)

1. Create the tables with referential integrity.
2. Draw the ER diagram for the same.
3. Create an Index on primary key column of table Account

4. Create the view as Customer_Info displaying the customer details for age less than 45.
5. Update the View with open date as 16/4/2017
6. Create a sequence on Branch able.
7. Create synonym 'Branch_info' for branch table.

1. Create tables with referential integrity

```
CREATE TABLE Customer (  
    CustID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Cust_Address VARCHAR(100),  
    Phone_no VARCHAR(15),  
    Email_ID VARCHAR(50),  
    Age INT  
);
```

```
CREATE TABLE Branch (  
    Branch_ID INT PRIMARY KEY,  
    Branch_Name VARCHAR(50),  
    Address VARCHAR(100)  
);
```

```
CREATE TABLE Account (  
    Account_no INT PRIMARY KEY,  
    Branch_ID INT,  
    CustID INT,  
    date_open DATE,  
    Account_type VARCHAR(20),  
    Balance DECIMAL(12,2),  
    FOREIGN KEY (Branch_ID) REFERENCES Branch(Branch_ID),  
    FOREIGN KEY (CustID) REFERENCES Customer(CustID)  
);
```

INSERT INTO Customer VALUES

```
(101, 'Payal', 'Pune', '9991112222', 'payal@gmail.com', 25),  
(102, 'Sneha', 'Mumbai', '9993334444', 'sneha@gmail.com', 30),  
(103, 'Ramesh', 'Delhi', '9995556666', 'ramesh@gmail.com', 48),  
(104, 'Priya', 'Pune', '9997778888', 'priya@gmail.com', 40);
```

INSERT INTO Branch VALUES

```
(1, 'Main Branch', 'Pune'),  
(2, 'City Branch', 'Mumbai');
```

INSERT INTO Account VALUES

```
(2001, 1, 101, '2018-04-16', 'Saving', 50000.00),  
(2002, 2, 102, '2019-05-21', 'Current', 80000.00),  
(2003, 1, 103, '2017-04-16', 'Saving', 30000.00),  
(2004, 2, 104, '2020-10-11', 'Saving', 25000.00);
```

2.ER diagram (text form)

3. Create index on primary key of Account

```
CREATE INDEX idx_account_pk ON Account(Account_no);
```

4.Create view Customer_Info for age < 45

```
CREATE VIEW Customer_Info AS  
SELECT * FROM Customer WHERE Age < 45;
```

5.Update view with open date = '2017-04-16'

```
UPDATE Account a  
JOIN Customer c ON a.CustID = c.CustID  
SET a.date_open = '2017-04-16'  
WHERE c.Age < 45;
```

6.Create sequence on Branch table (Oracle-style)

```
CREATE SEQUENCE branch_seq START WITH 1 INCREMENT BY 1;
```

7.Create synonym Branch_info for Branch table (Oracle-style)

```
CREATE SYNONYM Branch_info FOR Branch;
```

Problem Statement 12 (Triggers)

Employee(emp_id, dept_id emp_name, DoJ, salary, commission, job_title)

Consider the schema given above for Write a PLSQL Program to

1. Create a Trigger to ensure the salary of the employee is not decreased.
2. Whenever the job title is changed for an employee write the following details into job_history table. Employee ID, old job title, old department ID, DoJ for start date, system date for end date.

1. Trigger to prevent salary decrease (before update)

```
CREATE OR REPLACE TRIGGER trg_no_salary_decrease  
BEFORE UPDATE OF salary ON Employee  
FOR EACH ROW  
BEGIN  
    IF :NEW.salary < :OLD.salary THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Salary decrease not allowed');  
    END IF;  
END;  
/
```

2. Trigger to log job changes into job_history

```
CREATE OR REPLACE TRIGGER trg_job_change  
AFTER UPDATE OF job_title, dept_id ON Employee  
FOR EACH ROW
```

```

BEGIN
  IF :OLD.job_title <> :NEW.job_title OR :OLD.dept_id <> :NEW.dept_id THEN
    INSERT INTO job_history(emp_id, old_job_title, old_dept_id, start_date, end_date)
    VALUES(:OLD.emp_id, :OLD.job_title, :OLD.dept_id, :OLD.DoJ, SYSDATE);
  END IF;
END;
/

```

Problem Statement 13 (Map Reduce)

Create collection for Student{roll_no, name, class, dept, aggregate_marks}. Write Map Reduce Functions for following requirements.

1. Finding the total marks of students of "TE" class department-wise.
2. Finding the highest marks of students of "SE" class department-wise.
3. Find Average marks of students of "BE" class department-wise.

ANS:

```

db.Student.insertMany([
  { roll_no: 1, name: "Payal", class: "TE", dept: "Computer", aggregate_marks: 78 },
  { roll_no: 2, name: "Amit", class: "TE", dept: "IT", aggregate_marks: 85 },
  { roll_no: 3, name: "Riya", class: "TE", dept: "Computer", aggregate_marks: 90 },
  { roll_no: 4, name: "Sagar", class: "SE", dept: "IT", aggregate_marks: 76 },
  { roll_no: 5, name: "Neha", class: "SE", dept: "Computer", aggregate_marks: 82 },
]);

```

1. Total Marks of TE Class — Department-wise

```

db.Student.mapReduce(
  function() { if (this.class === "TE") emit(this.dept, this.aggregate_marks); },
  function(key, values) { return Array.sum(values); },
  { out: "TE_total_by_dept" }
);

```

2. Highest Marks of SE Class — Department-wise

```

db.Student.mapReduce(
  function() { if (this.class === "SE") emit(this.dept, this.aggregate_marks); },
  function(key, values) { return Array.max(values); },
  { out: "SE_max_by_dept" }
);

```

3. Average Marks of BE Class — Department-wise

```

db.Student.mapReduce(
  function() { if (this.class === "BE") emit(this.dept, { sum: this.aggregate_marks, count: 1 }); },
  function(key, values) {

```

```

    return values.reduce((r, v) => ({ sum: r.sum + (v.sum || v), count: r.count + (v.count || 1) }), { sum:0,
count:0 });
},
{
    out: "BE_avg_by_dept",
    finalize: function(key, reduced) { return reduced.count ? reduced.sum / reduced.count : 0; }
}
);

```

Problem Statement 14 (DML USING MYSQL)

Create following tables using a given schema and insert appropriate data into the same:

Customer (CustID, Name, Cust_Address, Phone_no, Email_ID, Age)

Branch (Branch ID, Branch_Name, Address)

Account (Account_no, Branch ID, CustID, date_open, Account_type, Balance)

1. Modify the size of column "Email_Address" to 20 in Customer table.
2. Change the column "Email_Address" to Not Null in Customer table.
3. Display the total customers with the balance >50, 000 Rs.
4. Display average balance for account type="Saving Account".
5. Display the customer details that lives in Pune or name starts with 'A'.
6. Create a table Saving_Account with (Account_no, Branch ID, CustID, date_open, Balance) using Account Table.
7. Display the customer details Age wise with balance>=20,000Rs

1. Modify size of column Email_Address to 20

```
ALTER TABLE Customer MODIFY Email_Address VARCHAR(20);
```

2. Change Email_Address to NOT NULL

```
ALTER TABLE Customer MODIFY Email_Address VARCHAR(20) NOT NULL;
```

3. Total customers with balance > 50,000

```

SELECT COUNT(DISTINCT c.CustID) AS total_customers
FROM Customer c
JOIN Account a ON c.CustID = a.CustID
WHERE a.Balance > 50000;

```

4. Average balance for account_type = "Saving Account"

```

SELECT AVG(Balance) AS avg_saving_balance
FROM Account
WHERE Account_type = 'Saving Account';

```

5. Customers who live in Pune OR name starts with 'A'

```

SELECT * FROM Customer
WHERE Cust_Address = 'Pune' OR Name LIKE 'A%';

```

6. Create table Saving_Account using Account table (only Saving accounts)

```
CREATE TABLE Saving_Account AS
SELECT Account_no, Branch_ID, CustID, date_open, Balance
FROM Account
WHERE Account_type = 'Saving Account';
```

7. Display customer details age-wise with balance >= 20,000

```
SELECT c.*
FROM Customer c
JOIN Account a ON c.CustID = a.CustID
WHERE a.Balance >= 20000
ORDER BY c.Age;
```

Problem Statement 15 (Cursors)

Consider the following schema for Products table.

Products(Product_id, Product_Name, Product_Type, Price)

1. Write a parameterized cursor to display all products in the given price range of price and type 'Apparel'.

Hint: Take the user input for minimum and maximum price for price range.

2. Write an explicit cursor to display information of all products with Price greater than 5000.

3. Write an implicit cursor to display the number of records affected by the update operation incrementing Price of all products by 1000.

1. Parameterized cursor for given price range & type = 'Apparel'

```
DECLARE
  p_min NUMBER := &min_price;
  p_max NUMBER := &max_price;
  CURSOR cur_range IS
    SELECT Product_id, Product_Name, Price
    FROM Products
    WHERE Product_Type = 'Apparel' AND Price BETWEEN p_min AND p_max;
  v_id Products.Product_id%TYPE;
  v_name Products.Product_Name%TYPE;
  v_price Products.Price%TYPE;
BEGIN
  OPEN cur_range;
  LOOP
    FETCH cur_range INTO v_id, v_name, v_price;
    EXIT WHEN cur_range%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_id || ' ' || v_name || ' ' || v_price);
  END LOOP;
  CLOSE cur_range;
END;
/
```

2. Explicit cursor for Price > 5000

```
DECLARE
  CURSOR cur_exp IS SELECT Product_id, Product_Name, Price FROM Products WHERE Price > 5000;
  v_id Products.Product_id%TYPE;
  v_name Products.Product_Name%TYPE;
  v_price Products.Price%TYPE;
BEGIN
  OPEN cur_exp;
  LOOP
    FETCH cur_exp INTO v_id, v_name, v_price;
    EXIT WHEN cur_exp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_id || ' ' || v_name || ' ' || v_price);
  END LOOP;
  CLOSE cur_exp;
END;
/
```

3. Implicit cursor to show number of rows affected by update (increase price by 1000)

```
BEGIN
  UPDATE Products SET Price = Price + 1000;
  DBMS_OUTPUT.PUT_LINE('Rows affected: ' || SQL%ROWCOUNT);
END;
/
```

Problem Statement 16

Implement MongoDB database connectivity with Java.

1. Add dependency (Maven)(in lib folder)(lib folder inside MongoJava folder)

```
mongodb-driver-sync-4.9.0.jar
mongodb-driver-core-4.9.0.jar
bson-4.9.0.jar
(INSTALL)
```

** Then compile with:

```
javac -cp ".:lib/*" App.java
```

** Then run with:

```
java -cp ".:lib/*" App
```

2. Simple Java code

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import org.bson.Document;
```

```

public class App {
    public static void main(String[] args) {
        String uri = "mongodb://localhost:27017";

        try (MongoClient mongoClient = MongoClient.create(uri)) {
            MongoDBDatabase db = mongoClient.getDatabase("yourDB");
            MongoCollection<Document> col = db.getCollection("Student");

            Document doc = new Document("Roll_No", "A01").append("Name", "Riya");
            col.insertOne(doc);

            System.out.println("Connected & Inserted!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Problem Statement 17 (JOINS & SUBQUERIES USING MYSQL)

Consider Following Schema

Employee (Employee_id, First_name, Last_name , Hire_date, Salary, Job_title, Manager_id, department_id)

Departments (Department_id, Department_name, Manager_id, Location_id)

Locations (Location_id , Street_address , Postal_code, city, state, Country_id)

Manager (Manager_id, Manager_name)

Create the tables with referential integrity. Solve following queries using joins and subqueries.

1. Write a query to find the names (first_name, last_name), the salary of the employees who earn more than

the average salary and who works in any of the IT departments.

2. Write a query to find the names (first_name, last_name), the salary of the employees who earn the same

salary as the minimum salary for all departments.

3. Write a query to display the employee ID, first name, last names, salary of all employees whose salary is

above average for their departments.

4. Write a query to display the department name, manager name, and city.

5. Write a query to display the name (first_name, last_name), hire date, salary of all managers whose experience is more than 15 years.

Table:

```

CREATE TABLE Manager (
    Manager_id INT PRIMARY KEY,
    Manager_name VARCHAR(50)

```

```
);
```

```
CREATE TABLE Locations (  
    Location_id INT PRIMARY KEY,  
    Street_address VARCHAR(100),  
    Postal_code VARCHAR(10),  
    City VARCHAR(50),  
    State VARCHAR(50),  
    Country_id VARCHAR(10)  
);
```

```
CREATE TABLE Departments (  
    Department_id INT PRIMARY KEY,  
    Department_name VARCHAR(50),  
    Manager_id INT,  
    Location_id INT,  
    FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id),  
    FOREIGN KEY (Location_id) REFERENCES Locations(Location_id)  
);
```

```
CREATE TABLE Employee (  
    Employee_id INT PRIMARY KEY,  
    First_name VARCHAR(50),  
    Last_name VARCHAR(50),  
    Hire_date DATE,  
    Salary DECIMAL(10,2),  
    Job_title VARCHAR(50),  
    Manager_id INT,  
    Department_id INT,  
    FOREIGN KEY (Manager_id) REFERENCES Manager(Manager_id),  
    FOREIGN KEY (Department_id) REFERENCES Departments(Department_id)  
);
```

Sample Data

```
INSERT INTO Manager VALUES  
(1, 'Anita'), (2, 'Rajesh'), (3, 'Sneha');
```

```
INSERT INTO Locations VALUES  
(101, 'MG Road', '411001', 'Pune', 'MH', 'IN'),  
(102, 'Andheri', '400053', 'Mumbai', 'MH', 'IN');
```

```
INSERT INTO Departments VALUES  
(10, 'IT', 1, 101),  
(20, 'HR', 2, 102),  
(30, 'Finance', 3, 101);
```

```
INSERT INTO Employee VALUES
```

(1, 'Payal', 'Jadhav', '2010-05-10', 75000, 'Developer', 1, 10),
(2, 'Raj', 'Patil', '2012-03-20', 50000, 'Tester', 1, 10),
(3, 'Sneha', 'More', '2005-07-15', 90000, 'Manager', 2, 20),
(4, 'Riya', 'Shah', '2018-09-01', 40000, 'HR Assistant', 2, 20),
(5, 'Amit', 'Deshmukh', '2011-01-10', 60000, 'Accountant', 3, 30);

1. Employees who earn > average and work in any IT departments

```
SELECT e.first_name, e.last_name, e.salary
FROM Employee e
JOIN Departments d ON e.department_id = d.Department_id
WHERE e.salary > (SELECT AVG(salary) FROM Employee)
AND d.Department_name LIKE '%IT%';
```

2. Employees who earn same salary as the minimum salary for all departments

```
SELECT first_name, last_name, salary
FROM Employee
WHERE salary = (SELECT MIN(salary) FROM Employee);
```

3. Employees whose salary is above average for their department

```
SELECT e.Employee_id, e.First_name, e.Last_name, e.Salary
FROM Employee e
WHERE e.Salary > (
    SELECT AVG(salary) FROM Employee WHERE department_id = e.department_id
);
```

4. Display department name, manager name, and city

```
SELECT d.Department_name, m.Manager_name, l.city
FROM Departments d
JOIN Manager m ON d.Manager_id = m.Manager_id
JOIN Locations l ON d.Location_id = l.Location_id;
```

5. Managers with experience > 15 years (based on Hire_date)

```
SELECT e.First_name, e.Last_name, e.Hire_date, e.Salary
FROM Employee e
WHERE e.job_title = 'Manager'
AND (FLOOR(MONTHS_BETWEEN(SYSDATE, e.Hire_date)/12) > 15);
```

Problem Statement 18 (Procedures / Functions)

Consider following schema for Bank database.

Account(Account_No, Cust_Name, Balance, NoOfYears)

Earned_Interest(Account_No, Interest_Amt)

1. Write a PL/SQL procedure for following requirement. Take as input Account_No and Interest Rate from User.

Calculate the Interest_Amt as simple interest for the given Account_No and store it in Earned_Interest table.

Display all the details of Earned_Interest Table.

2. Write a PLSQL function to display all records from Account table having Balance greater than 50,000.

1. Procedure: compute simple interest and store

```
CREATE OR REPLACE PROCEDURE calc_interest(p_acc_no IN NUMBER, p_rate IN NUMBER)
IS
    v_balance NUMBER;
    v_years NUMBER;
    v_interest NUMBER;
BEGIN
    -- Get balance and number of years
    SELECT Balance, NoOfYears INTO v_balance, v_years
    FROM Account
    WHERE Account_No = p_acc_no;

    -- Calculate simple interest
    v_interest := (v_balance * p_rate * v_years) / 100;

    -- Insert into Earned_Interest
    INSERT INTO Earned_Interest(Account_No, Interest_Amt)
    VALUES (p_acc_no, v_interest);

    COMMIT;

    -- Display all Earned_Interest records
    FOR rec IN (SELECT * FROM Earned_Interest) LOOP
        DBMS_OUTPUT.PUT_LINE('Account: ' || rec.Account_No || ', Interest: ' || rec.Interest_Amt);
    END LOOP;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Account not found');
END;
/
```

2. Function: return records from Account having Balance > 50000

(PL/SQL function cannot directly return a resultset easily; example returns SYS_REFCURSOR)

```
CREATE OR REPLACE FUNCTION get_high_balance_accounts
RETURN SYS_REFCURSOR
IS
    rc SYS_REFCURSOR;
BEGIN
    OPEN rc FOR SELECT * FROM Account WHERE Balance > 50000;
    RETURN rc;
END;
```

/

Problem Statement 19 (Aggregation & Indexing)

Create the Collection Movies_Data(Movie_ID, Movie_Name, Director, Genre, BoxOfficeCollection) and solve the following:

1. Display a list stating how many Movies are directed by each "Director".
2. Display list of Movies with the highest BoxOfficeCollection in each Genre.
3. Display list of Movies with the highest BoxOfficeCollection in each Genre in ascending order of BoxOfficeCollection.
4. Create an index on field Movie_ID.
5. Create an index on fields " Movie_Name" and " Director".
6. Drop an index on field Movie_ID.
7. Drop an index on fields " Movie_Name" and " Director".

Table:

```
db.createCollection("Movies_Data")
```

```
db.Movies_Data.insertMany([
  { Movie_ID: 1, Movie_Name: "Inception", Director: "Christopher Nolan", Genre: "Sci-Fi",
    BoxOfficeCollection: 850 },
  { Movie_ID: 8, Movie_Name: "The Avengers", Director: "Joss Whedon", Genre: "Action",
    BoxOfficeCollection: 1500 }
])
```

1. How many movies directed by each Director

```
db.Movies_Data.aggregate([
  { $group: { _id: "$Director", count: { $sum: 1 } } }
])
```

2. Movies with highest BoxOfficeCollection in each Genre

```
db.Movies_Data.aggregate([
  { $sort: { BoxOfficeCollection: -1 } },
  { $group: { _id: "$Genre", TopMovie: { $first: "$Movie_Name" }, MaxCollection: { $first:
    "$BoxOfficeCollection" } } }
])
```

3. Highest BoxOffice in each Genre in ascending order

```
db.Movies_Data.aggregate([
  { $sort: { BoxOfficeCollection: -1 } },
  { $group: { _id: "$Genre", TopMovie: { $first: "$Movie_Name" }, MaxCollection: { $first:
    "$BoxOfficeCollection" } } },
  { $sort: { MaxCollection: 1 } }
])
```

4. Create index on Movie_ID

```
db.Movies_Data.createIndex({ Movie_ID: 1 })
```

5. Create index on Movie_Name and Director

```
db.Movies_Data.createIndex({ Movie_Name: 1, Director: 1 })
```

6. Drop index on Movie_ID

```
db.Movies_Data.dropIndex({ Movie_ID: 1 })
```

7. Drop index on Movie_Name and Director

```
db.Movies_Data.dropIndex({ Movie_Name: 1, Director: 1 })
```

Problem Statement 20

Implement MYSQL database connectivity with PHP.

To check php installed or not:

Just run this single command in your Ubuntu terminal:

```
php -v
```

```
<?php
// 1. Connect to MySQL
$servername = "localhost";
$username = "root";
$password = "password";
$dbname = "bankdb";

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

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

// 2. Fetch data from Customer table
$sql = "SELECT * FROM Customer";
$result = $conn->query($sql);

// 3. Display output
if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo $row["CustID"] . " - " . $row["Name"] . "<br>";
    }
} else {
    echo "0 results";
}
```

```
// 4. Close connection
$conn->close();
?>
```

Problem Statement 21 (Triggers)

Employee(emp_id, emp_name, salary, designation)

Salary_Backup(emp_id, old_salary, new_salary, salary_difference)

Create a Trigger to record salary change of the employee. Whenever salary is updated insert the details in

Salary_Backup table.

Create a Trigger that will prevent deleting the employee record having designation as CEO.

1. Trigger to record salary change

```
CREATE OR REPLACE TRIGGER trg_salary_change
```

```
AFTER UPDATE OF salary ON Employee
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO Salary_Backup(emp_id, old_salary, new_salary, salary_difference)
```

```
    VALUES(:OLD.emp_id, :OLD.salary, :NEW.salary, :NEW.salary - :OLD.salary);
```

```
END;
```

```
/
```

2. Trigger to prevent deleting CEO

```
CREATE OR REPLACE TRIGGER trg_prevent_ceo_delete
```

```
BEFORE DELETE ON Employee
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    IF :OLD.designation = 'CEO' THEN
```

```
        RAISE_APPLICATION_ERROR(-20002, 'Cannot delete employee with designation CEO');
```

```
    END IF;
```

```
END;
```

```
/
```