1: Draw an ER for Bank database with atleast 5 entities and convert them into tables.

Perform DDL on above converted tables.

1. Create tables with all constraints

2. Create views on any two tables using join conditions

3. Create index called CustomerId. Entries should be in ascending order by customer name.

4. Create sequence on Acctno.

```
CREATE TABLE Customer (
   CustomerId INT PRIMARY KEY,
   Name VARCHAR(50) NOT NULL,
   Address VARCHAR(100),
   PhoneNo VARCHAR(15) UNIQUE
);

CREATE TABLE Branch (
   BranchId INT PRIMARY KEY,
   BranchName VARCHAR(50),
   BranchCity VARCHAR(50)
);

CREATE TABLE Account (
   AcctNo INT AUTO_INCREMENT PRIMARY KEY,
   CustomerId INT,
   BranchId INT,
   Balance DECIMAL(10,2),
   FOREIGN KEY (CustomerId) REFERENCES Customer(CustomerId),
   FOREIGN KEY (BranchId) REFERENCES Branch(BranchId)
);

CREATE TABLE Loan (
   LoanId INT PRIMARY KEY,
   CustomerId INT,
   BranchId INT,
   Amount DECIMAL(10,2),
   FOREIGN KEY (CustomerId) REFERENCES Customer(CustomerId),
   FOREIGN KEY (BranchId) REFERENCES Branch(BranchId)
);

CREATE TABLE Transaction (
   TxnId INT PRIMARY KEY,
   AcctNo INT,
   TxnDate DATE,
   Amount DECIMAL(10,2),
   Type VARCHAR(10), -- e.g., Deposit/Withdraw
   FOREIGN KEY (AcctNo) REFERENCES Account(AcctNo)
);
INSERT INTO Customer (CustomerId, Name, Address, PhoneNo) VALUES
```

```
(1, 'Isha Sharma', 'Pune', '9876543210'),
(2, 'Raj Patel', 'Mumbai', '9123456780'),
(3, 'Anita Desai', 'Nagpur', '9988776655');

INSERT INTO Branch (BranchId, BranchName, BranchCity) VALUES
(101, 'Main Branch', 'Pune'),
(102, 'Central Branch', 'Mumbai'),
(103, 'East Branch', 'Nagpur');

INSERT INTO Account (CustomerId, BranchId, Balance) VALUES
(1, 101, 10000.00),
(2, 102, 20000.00),
(3, 103, 15000.00);

INSERT INTO Loan (LoanId, CustomerId, BranchId, Amount) VALUES
(201, 1, 101, 500000.00),
(202, 2, 102, 300000.00);

INSERT INTO Transaction (TxnId, AcctNo, TxnDate, Amount, Type) VALUES
(301, 1, '2024-05-01', 2000.00, 'Deposit'),
(302, 2, '2024-05-02', 1500.00, 'Withdraw'),
(303, 3, '2024-05-03', 500.00, 'Deposit');

CREATE VIEW Customer_Accounts AS

SELECT c.CustomerId, c.Name, a.AcctNo, a.Balance

FROM Customer c
JOIN Account a ON c.CustomerId = a.CustomerId;

CREATE VIEW Loan_Branch AS
SELECT l.LoanId, l.Amount, b.BranchName, b.BranchCity
FROM Loan l
JOIN Branch b ON l.BranchId = b.BranchId;


CREATE INDEX idx_CustomerName
ON Customer (Name ASC);
EXPLAIN SELECT * FROM Customer WHERE Name = 'Raj Patel';
SHOW INDEX FROM Customer;


--For sequence:
Set session auto_increment_increment=10;
Alter Account modify AccNo int auto_increment;
```

2: Draw an ER for Company database with atleast 4 entities and convert them into tables.

Perform DDL on Above converted tables.

1. Create tables with all constraints

2. create views on any two tables using conditions

3. create index called EmployeeId for the department table. Entries should be in ascending order by department id and then by employee id within each department.

4. create sequence on Employee id.

```
CREATE TABLE Department (
   DeptID INT PRIMARY KEY,
   DeptName VARCHAR(50) NOT NULL
);

CREATE TABLE Employee (
   EmpID INT AUTO_INCREMENT PRIMARY KEY,
   EmpName VARCHAR(50) NOT NULL,
   Gender VARCHAR(10),
   Salary DECIMAL(10, 2) CHECK (Salary > 0),
   DeptID INT,
   FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);

CREATE TABLE Project (
   ProjID INT PRIMARY KEY,
   ProjName VARCHAR(100) NOT NULL,
   Budget DECIMAL(12, 2) CHECK (Budget >= 0)
);

CREATE TABLE Works_On (
   EmpID INT,
   ProjID INT,
   HoursWorked DECIMAL(5,2) CHECK (HoursWorked >= 0),
   PRIMARY KEY (EmpID, ProjID),
   FOREIGN KEY (EmpID) REFERENCES Employee(EmpID),
   FOREIGN KEY (ProjID) REFERENCES Project(ProjID)
);

-- Departments
INSERT INTO Department (DeptID, DeptName) VALUES
(1, 'HR'),
(2, 'IT'),
(3, 'Finance');

-- Employees
INSERT INTO Employee (EmpName, Gender, Salary, DeptID) VALUES
('Isha', 'Female', 60000, 1),
('Raj', 'Male', 75000, 2),
('Amit', 'Male', 45000, 1),
('Sneha', 'Female', 90000, 3);
```

```
-- Projects
INSERT INTO Project (ProjID, ProjName, Budget) VALUES
(101, 'ERP System', 200000), (102,
'Mobile App', 50000),
(103, 'AI Research', 150000);

-- Works_On (Employees working on Projects)
INSERT INTO Works_On (EmpID, ProjID, HoursWorked) VALUES
(1, 101, 40),
(2, 101, 35),
(2, 103, 20),
(4, 103, 30);

CREATE VIEW HighSalaryEmployees AS
SELECT e.EmpID, e.EmpName, e.Salary, d.DeptName
FROM Employee e
JOIN Department d ON e.DeptID = d.DeptID
WHERE e.Salary > 50000;

CREATE VIEW ProjectAssignments AS
SELECT p.ProjName, e.EmpName, w.HoursWorked
FROM Works_On w
JOIN Employee e ON w.EmpID = e.EmpID
JOIN Project p ON w.ProjID = p.ProjID;

CREATE INDEX EmployeeId
ON Employee (DeptID ASC, EmpID ASC);

SELECT * FROM Employee;
SELECT * FROM Department;
SELECT * FROM HighSalaryEmployees;
SELECT * FROM BigBudgetProjects;
```

3: write a trigger for Library (bid, bname, doi, status) to update the number of copies (noc) according to ISSUE & RETURN status on update or insert query. Increase the noc if status is RETURN, Decrease noc if status is ISSUE in Library_Audit table(bid,bname,noc,timestampofquery). Write a trigger after update on Library such that if doi is more than 20 days ago then status should be FINE and in the Library_Audit table fine should be equal to no. of days * 10.

```
-- Main Library Table
CREATE TABLE Library (
bid INT PRIMARY KEY,
bname VARCHAR(100),
doi DATE,
  status VARCHAR(10)
);

-- Audit Table
CREATE TABLE Library_Audit (
  bid INT,
  bname VARCHAR(100),
  noc INT,
  timestampofquery TIMESTAMP DEFAULT CURRENT_TIMESTAMP,    fine
INT DEFAULT 0
);

DELIMITER //

CREATE TRIGGER trg_Library_Insert
AFTER INSERT ON Library
FOR EACH ROW
BEGIN
  DECLARE new_noc INT;

  SELECT noc INTO new_noc FROM Library_Audit WHERE bid = NEW.bid LIMIT 1;

  IF NEW.status = 'ISSUE' THEN
    SET new_noc = new_noc - 1;
  ELSEIF NEW.status = 'RETURN' THEN
    SET new_noc = new_noc + 1;
  END IF;

  UPDATE Library_Audit
  SET noc = new_noc,
    timestampofquery = NOW(),
    fine = 0
  WHERE bid = NEW.bid;
END;
//

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg_Library_AdjustStock_AfterUpdate
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
  DECLARE current_noc INT;

  SELECT noc INTO current_noc FROM Library_Audit WHERE bid = NEW.bid LIMIT 1;
```

```
    IF NEW.status = 'ISSUE' AND OLD.status != 'ISSUE' THEN
        SET current_noc = current_noc - 1;
    ELSEIF NEW.status = 'RETURN' AND OLD.status != 'RETURN' THEN
        SET current_noc = current_noc + 1;
    END IF;

    UPDATE Library_Audit
    SET noc = current_noc,
        timestampofquery = NOW()
    WHERE bid = NEW.bid;
END;
//

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg_Library_CheckFine
after UPDATE ON Library
FOR EACH ROW
BEGIN
    DECLARE days_elapsed INT;
    DECLARE fine_amount INT;

    SET days_elapsed = DATEDIFF(CURDATE(), NEW.doi);

    IF days_elapsed > 20 THEN
        SET fine_amount = (days_elapsed - 20) * 10;

        -- Update Library_Audit with fine
        UPDATE Library_Audit
        SET fine = fine_amount,
            timestampofquery = NOW()
        WHERE bid = NEW.bid;

        -- Update status to FINE
        SET new.status = 'FINE';
    END IF;
END;
//

DELIMITER ;

-- Insert a sample book with 5 copies in audit
INSERT INTO Library_Audit (bid, noc, fine) VALUES (101, 5, 0);

-- Insert a new issue record
INSERT INTO Library (bid, status, doi) VALUES (101, 'ISSUE', CURDATE());

-- Update the status to RETURN
UPDATE Library SET status = 'RETURN' WHERE bid = 101;
```

```
-- Simulate an overdue return (set old date of issue)
UPDATE Library SET doi = '2023-01-01' WHERE bid = 101;

-- Then update status to RETURN to invoke the trigger
UPDATE Library SET status = 'RETURN', doi = '2025-05-05' WHERE bid = 101;

SELECT * FROM Library;

SELECT * FROM Library_Audit;
```

4: Write a database 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 (
bid INT PRIMARY KEY,
   bname VARCHAR(100),
   doi DATE,          -- date of issue or relevant date
   status VARCHAR(20)
);

CREATE TABLE Library_Audit (
   audit_id INT AUTO_INCREMENT PRIMARY KEY,
   bid INT,
   bname VARCHAR(100),
   doi DATE,
   status VARCHAR(20),
   action VARCHAR(10),        -- 'UPDATE' or 'DELETE'
   timestampofquery TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);



DELIMITER $$
-- Trigger for UPDATE
CREATE TRIGGER trg_library_update
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
   INSERT INTO Library_Audit (bid, bname, doi, status, action)
   VALUES (OLD.bid, OLD.bname, OLD.doi, OLD.status, 'UPDATE');
END$$

-- Trigger for DELETE
CREATE TRIGGER trg_library_delete
AFTER DELETE ON Library
FOR EACH ROW
BEGIN
```

```
    INSERT INTO Library_Audit (bid, bname, doi, status, action)
    VALUES (OLD.bid, OLD.bname, OLD.doi, OLD.status, 'DELETE');
END$$
DELIMITER ;



-- Insert sample data
INSERT INTO Library (bid, bname, doi, status) VALUES (1, 'Database Concepts', CURDATE(), 'ISSUE');
INSERT INTO Library (bid, bname, doi, status) VALUES (2, 'Operating Systems', CURDATE(), 'ISSUE');

-- Update a record
UPDATE Library SET status = 'RETURN' WHERE bid = 1;

-- Delete a record
DELETE FROM Library WHERE bid = 2;

-- Check audit logs
SELECT * FROM Library_Audit;
```

5 Create a collection sites(url,dateofaccess). Write a MapReduce function to find the no. of times a site was accessed in a month.

```
db.sites.insertMany([

  { url: "https://example.com", dateofaccess: new Date("2024-05-01") },

  { url: "https://example.com", dateofaccess: new Date("2024-05-03") },

  { url: "https://example.com", dateofaccess: new Date("2024-06-04") },

  { url: "https://abc.com", dateofaccess: new Date("2024-05-04") },

  { url: "https://abc.com", dateofaccess: new Date("2024-05-05") },

  { url: "https://abc.com", dateofaccess: new Date("2024-06-07") }

]);

var mapFunction = function() {

  var month = this.dateofaccess.getMonth() + 1; // 0-based month

  var year = this.dateofaccess.getFullYear();
```

```
  var key = this.url + "-" + year + "-" + (month < 10 ? "0" + month : month);
  emit(key, 1);
};
var reduceFunction = function(key, values) {
  return Array.sum(values);
};
db.sites.mapReduce(
  mapFunction,
  reduceFunction,
  {
    out: "site_access_count"
  }
);
db.site_access_count.find().pretty();
```

6 Create tables CitiesIndia(pincode,nameofcity,earliername,area,population,avgrainfall) Categories(Type,pincode) Note:- Enter data only in CitiesIndia Write PL/SQL Procedure & function to find the population density of the cities. If the population density is above 3000 then Type of city must be entered as High Density in Category table. Between 2999 to 1000 as Moderate and below 999 as Low Density. Error must be displayed for population less than 10 or greater than 25718.

```
CREATE TABLE CitiesIndia (
    pincode INT PRIMARY KEY,
    nameofcity VARCHAR(100),
    earliername VARCHAR(100),
    area DECIMAL(10,2),
    population INT,        -- population count
    avgrainfall DECIMAL(6,2)  -- average rainfall in mm
);
CREATE TABLE Categories (
```

```
    Type VARCHAR(20),
    pincode INT,
    FOREIGN KEY (pincode) REFERENCES CitiesIndia(pincode)
);
INSERT INTO CitiesIndia VALUES (110001, 'New Delhi', 'Delhi', 1484.0, 25000, 800);
INSERT INTO CitiesIndia VALUES (560001, 'Bangalore', 'Bengaluru', 709.5, 8500, 970);
INSERT INTO CitiesIndia VALUES (400001, 'Mumbai', 'Bombay', 603.4, 12000, 2100);
INSERT INTO CitiesIndia VALUES (700001, 'Kolkata', 'Calcutta', 185.0, 300, 1500);
DELIMITER $$
CREATE FUNCTION get_population_density(p_pincode INT) RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE v_area DECIMAL(10,2);
    DECLARE v_population INT;
    DECLARE v_density DECIMAL(10,2);
    SELECT area, population INTO v_area, v_population FROM CitiesIndia WHERE pincode = p_pincode;
    IF v_population < 10 OR v_population > 25718 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Population is out of allowed range (10-25718)';
    END IF;
    SET v_density = v_population / v_area;
    RETURN v_density;
END$$
CREATE PROCEDURE update_city_category(IN p_pincode INT)
BEGIN
    DECLARE v_density DECIMAL(10,2);
    DECLARE v_type VARCHAR(20);
```

```
    SET v_density = get_population_density(p_pincode);

    IF v_density > 3000 THEN

        SET v_type = 'High Density';

    ELSEIF v_density BETWEEN 1000 AND 2999 THEN

        SET v_type = 'Moderate Density';

    ELSE

        SET v_type = 'Low Density';

    END IF;

    INSERT INTO Categories(Type, pincode) VALUES (v_type, p_pincode);

END$$

DELIMITER ;

CALL update_city_category(110001);

CALL update_city_category(560001);

CALL update_city_category(400001);

CALL update_city_category(700001);

SELECT * FROM Categories;
```

7) Write PL/SQL Procedure & function to find class [Distinction (Total marks from 1499 to 990) ,First Class(899 to 900) Higher Second (899 to 825) ,Second,Pass (824 to 750)] of a student based on total marks from table Student (rollno, name, Marks1, Marks2, Marks3, Marks4, Marks5).

Use exception handling when negative marks are entered by user(Marks<0) or Marks more than 100 are entered by user.. Store the result into Result table recording RollNo,total marks, and class for each student .

```
CREATE TABLE Student (
    rollno INT PRIMARY KEY,
    name VARCHAR(100),
    Marks1 INT,
    Marks2 INT,
    Marks3 INT,
    Marks4 INT,
```

```
    Marks5 INT
);
CREATE TABLE Result (
    rollno INT,
    total_marks INT,
    class VARCHAR(30)
);
DELIMITER $$
CREATE FUNCTION get_class(total INT) RETURNS VARCHAR(30)
DETERMINISTIC
BEGIN
    DECLARE result VARCHAR(30);
    IF total BETWEEN 990 AND 1499 THEN
        SET result = 'Distinction';
    ELSEIF total BETWEEN 900 AND 989 THEN
        SET result = 'First Class';
    ELSEIF total BETWEEN 825 AND 899 THEN
        SET result = 'Higher Second';
    ELSEIF total BETWEEN 750 AND 824 THEN
        SET result = 'Second, Pass';
    ELSE
        SET result = 'Fail';
    END IF;
    RETURN result;
END$$
CREATE PROCEDURE classify_students()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE rno INT;
    DECLARE m1 INT;
    DECLARE m2 INT;
    DECLARE m3 INT;
    DECLARE m4 INT;
    DECLARE m5 INT;
    DECLARE tname VARCHAR(100);
    DECLARE total INT;
    DECLARE class_type VARCHAR(30);
    DECLARE cur CURSOR FOR SELECT rollno, name, Marks1, Marks2, Marks3, Marks4, Marks5 FROM
Student;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    -- Clear previous results to avoid duplicates
    TRUNCATE TABLE Result;
    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO rno, tname, m1, m2, m3, m4, m5;
        IF done THEN
            LEAVE read_loop;
        END IF;
```

```
        -- Exception Handling: Check for invalid marks
        IF m1 < 0 OR m1 > 100 OR m2 < 0 OR m2 > 100 OR m3 < 0 OR m3 > 100 OR m4 < 0 OR m4 > 100 OR m5
< 0 OR m5 > 100 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid marks: Marks should be between 0 and
100';
        END IF;
        SET total = m1 + m2 + m3 + m4 + m5;
        SET class_type = get_class(total);
        INSERT INTO Result (rollno, total_marks, class) VALUES (rno, total, class_type);
    END LOOP;
    CLOSE cur;
END$$
DELIMITER ;
-- Sample Data
INSERT INTO Student VALUES (1, 'Amit', 98, 96, 92, 90, 89);
INSERT INTO Student VALUES (2, 'Sneha', 100, 99, 98, 97, 96);
INSERT INTO Student VALUES (3, 'Rahul', 90, 85, 88, 84, 80);
INSERT INTO Student VALUES (4, 'Riya', 100, 100, 100, 100, 100);
CALL classify_students();
SELECT * FROM Result;
```

8 Draw ER for Library database with atleast 5 entities and convert them into tables. Perform DDL on above converted tables.

1. **Create tables with all constraints (Based on ERD cardinalities)**
2. **Create views on any two tables using join condition**
3. **Create index called Lib_Index1. Entries should be in ascending order by Author name.**
4. **Create sequence on Bookid.**

```
-- Author Table
CREATE TABLE Author (
  AuthorID INT AUTO_INCREMENT PRIMARY KEY,
  AuthorName VARCHAR(100)
);

-- Publisher Table
CREATE TABLE Publisher (
  PublisherID INT AUTO_INCREMENT PRIMARY KEY,
  PublisherName VARCHAR(100)
);

-- Category Table
CREATE TABLE Category (
  CategoryID INT AUTO_INCREMENT PRIMARY KEY,
  CategoryName VARCHAR(50)
);
```

```
-- Book Table
CREATE TABLE Book (
  BookID INT AUTO_INCREMENT PRIMARY KEY,
  Title VARCHAR(100),
  AuthorID INT,
  PublisherID INT,
  CategoryID INT,
  FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID),
  FOREIGN KEY (PublisherID) REFERENCES Publisher(PublisherID),
  FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);

-- Member Table
CREATE TABLE Member (
  MemberID INT AUTO_INCREMENT PRIMARY KEY,
  MemberName VARCHAR(100)
);

INSERT INTO Author (AuthorName) VALUES ('Kiran'), ('Rahul');
INSERT INTO Publisher (PublisherName) VALUES ('Penguin'), ('Scholastic'); INSERT
INTO Category (CategoryName) VALUES ('Fiction'), ('Science'); INSERT INTO Book
(Title, AuthorID, PublisherID, CategoryID)
VALUES
('Book A', 1, 1, 1),
('Book B', 2, 2, 2);

INSERT INTO Member (MemberName) VALUES ('Amit'), ('Sneha');

-- View 1: Book with Author
CREATE VIEW BookAuthorView AS
SELECT B.Title, A.AuthorName
FROM Book B
JOIN Author A ON B.AuthorID = A.AuthorID;

-- View 2: Book with Category
CREATE VIEW BookCategoryView AS
SELECT B.Title, C.CategoryName
FROM Book B
JOIN Category C ON B.CategoryID = C.CategoryID;


CREATE INDEX Lib_Index1 ON Author (AuthorName ASC);

INSERT INTO Book (Title, AuthorID, PublisherID, CategoryID)
VALUES ('Book C', 1, 1, 1);

SELECT LAST_INSERT_ID();

SELECT * FROM BookAuthorView;
```

```
CREATE SEQUENCE Book_Seq

START WITH 1001

INCREMENT BY 1;
```

9.PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-

Schema:

1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll_no,Date,Amt)
3. Library (bid, bname, doi, status,noc) 4. transaction (tid,bid, bname, status)

1. **Accept roll_no & name of book from user.**
2. **Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.**
3. **If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.**
4. **After submitting the book, status will change from I to R.**
5. **Update the noc in library according to the transaction made. Increase the noc if status is RETURN, Decrease noc if status is ISSUE.**
6. **If condition of fine is true, then details will be stored into fine table.**

```
-- Step 1: Create Tables

CREATE TABLE Library (
   bid INT PRIMARY KEY,
   bname VARCHAR(100),
   doi DATE,
   status CHAR(1),
   noc INT
);

CREATE TABLE Borrower (
   rollno INT,
   name VARCHAR(100),
   dateofissue DATE,
   nameofbook VARCHAR(100),
   status CHAR(1)
);

CREATE TABLE Fine (
   rollno INT,
   date DATE,
```

```
    amt DECIMAL(10,2)
);

CREATE TABLE TransactionTable (
    tid INT AUTO_INCREMENT PRIMARY KEY,
    bid INT,
    bname VARCHAR(100),
    status CHAR(1)
);

-- Step 2: Insert Sample Data

INSERT INTO Library (bid, bname, doi, status, noc) VALUES
(1, 'Data Science Handbook', '2024-04-15', 'I', 3),
(2, 'Database Systems', '2024-04-20', 'I', 5),
(3, 'Operating Systems', '2024-04-10', 'R', 2);

INSERT INTO Borrower (rollno, name, dateofissue, nameofbook, status) VALUES
(101, 'Isha', '2024-04-15', 'Data Science Handbook', 'I'),
(102, 'Ravi', '2024-04-01', 'Operating Systems', 'I'),
(103, 'Neha', '2024-04-25', 'Database Systems', 'I');

-- Step 3: Create Procedure with Control Structures and Exception Handling

DELIMITER $$

CREATE PROCEDURE HandleReturn (
    IN p_rollno INT,
    IN p_bookname VARCHAR(100)
)
BEGIN
    DECLARE v_doi DATE;
    DECLARE v_days INT;
    DECLARE v_amt DECIMAL(10,2);
    DECLARE v_bid INT;

    -- Exception handler
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT 'An error occurred during the return process.' AS ErrorMessage;
    END;

    -- Get date of issue
    SELECT dateofissue INTO v_doi
    FROM Borrower
    WHERE rollno = p_rollno AND nameofbook = p_bookname;

    -- Calculate number of days since issue
    SET v_days = DATEDIFF(CURDATE(), v_doi);

    -- Calculate fine based on conditions
```

```
    IF v_days > 30 THEN
        SET v_amt = v_days * 50;
    ELSEIF v_days >= 15 THEN
        SET v_amt = v_days * 5;
    ELSE
        SET v_amt = 0;
    END IF;

    -- Update Borrower status to 'R' (Returned)
    UPDATE Borrower
    SET status = 'R'
    WHERE rollno = p_rollno AND nameofbook = p_bookname;

    -- Get Book ID from Library
    SELECT bid INTO v_bid
    FROM Library
    WHERE bname = p_bookname;

    -- Update Library: Increase number of copies and set status to 'R'
    UPDATE Library
    SET noc = noc + 1, status = 'R'
    WHERE bid = v_bid;

    -- Insert into TransactionTable
    INSERT INTO TransactionTable (bid, bname, status)
    VALUES (v_bid, p_bookname, 'R');

    -- Insert into Fine table if fine exists
    IF v_amt > 0 THEN
        INSERT INTO Fine (rollno, date, amt)
        VALUES (p_rollno, CURDATE(), v_amt);
    END IF;

END$$
DELIMITER ;

-- Step 4: Execute the Procedure
CALL HandleReturn(102, 'Operating Systems');

-- Step 5: View Results
SELECT * FROM Borrower;
SELECT * FROM Library;
SELECT * FROM Fine;
SELECT * FROM TransactionTable;
```

10 Implement SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym for following relational schema:

Borrower(Rollin, Name, DateofIssue, NameofBook, Status)

```
CREATE TABLE Borrower (
   Rollno INT PRIMARY KEY,
   Name VARCHAR(100),
   DateofIssue DATE,
   NameofBook VARCHAR(100),
   Status CHAR(1) -- 'I' for Issued, 'R' for Returned
);

INSERT INTO Borrower (Rollno, Name, DateofIssue, NameofBook, Status) VALUES
(101, 'Isha', '2024-04-10', 'Data Structures', 'I'),
(102, 'Amit', '2024-04-12', 'Operating Systems', 'R'),
(103, 'Neha', '2024-04-14', 'DBMS', 'I');

CREATE VIEW IssuedBooks AS
SELECT Rollno, Name, NameofBook, DateofIssue
FROM Borrower
WHERE Status = 'I';

SELECT * FROM IssuedBooks;

CREATE INDEX Borrower_Book_Index ON Borrower(NameofBook);

CREATE TABLE BorrowerWithAutoID (
   BorrowerID INT AUTO_INCREMENT PRIMARY KEY,
   Rollno INT,
   Name VARCHAR(100),
   DateofIssue DATE,
   NameofBook VARCHAR(100),
   Status CHAR(1)
);

INSERT INTO BorrowerWithAutoID (Rollno, Name, DateofIssue, NameofBook, Status) VALUES
(104, 'Ravi', '2024-05-01', 'Machine Learning', 'I');

CREATE SYNONYM BorrowerSyn FOR Borrower;
SELECT * FROM BorrowerSyn;

 CREATE VIEW BorrowerSyn AS SELECT * FROM Borrower;
```

11 Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

```
CREATE TABLE Author (
AuthorID INT PRIMARY KEY AUTO_INCREMENT,
AuthorName VARCHAR(100)
);
CREATE TABLE Publisher (
PublisherID INT PRIMARY KEY AUTO_INCREMENT,
```

```
PublisherName VARCHAR(100)
);
CREATE TABLE Category (
CategoryID INT PRIMARY KEY AUTO_INCREMENT,
CategoryName VARCHAR(50)
);
CREATE TABLE Book (
BookID INT PRIMARY KEY AUTO_INCREMENT,
Title VARCHAR(100),
AuthorID INT,
PublisherID INT,
CategoryID INT,
FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID),
FOREIGN KEY (PublisherID) REFERENCES Publisher(PublisherID),
FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);
CREATE TABLE Member (
MemberID INT PRIMARY KEY AUTO_INCREMENT,
MemberName VARCHAR(100)
);
CREATE TABLE Borrow (
BorrowID INT PRIMARY KEY AUTO_INCREMENT,
MemberID INT,
BookID INT,
BorrowDate DATE,
ReturnDate DATE,
FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
FOREIGN KEY (BookID) REFERENCES Book(BookID)
);
```

📄 Step 2: Insert Sample Data

```
-- Authors
INSERT INTO Author (AuthorName) VALUES ('Kiran'), ('Neha'), ('Rahul');
-- Publishers
INSERT INTO Publisher (PublisherName) VALUES ('Penguin'), ('Scholastic');
-- Categories
INSERT INTO Category (CategoryName) VALUES ('Fiction'), ('Science'), ('Technology');
-- Books
INSERT INTO Book (Title, AuthorID, PublisherID, CategoryID) VALUES
('Data Structures', 1, 1, 3),
('Operating Systems', 2, 2, 3),
('Physics Fundamentals', 3, 1, 2),
('Literary Classics', 2, 2, 1);
-- Members
INSERT INTO Member (MemberName) VALUES ('Amit'), ('Isha'), ('Ravi');
-- Borrow Records
INSERT INTO Borrow (MemberID, BookID, BorrowDate, ReturnDate) VALUES
(1, 1, '2024-04-10', '2024-04-20'),
(2, 2, '2024-04-12', NULL),
(3, 3, '2024-04-14', NULL),
(1, 4, '2024-04-18', '2024-04-25');
```

```
INNER JOIN — Book titles and authors
SELECT B.Title, A.AuthorName
FROM Book B
INNER JOIN Author A ON B.AuthorID = A.AuthorID;

LEFT JOIN — All books with publishers (even if some don't have)
SELECT B.Title, P.PublisherName
FROM Book B
LEFT JOIN Publisher P ON B.PublisherID = P.PublisherID;

RIGHT JOIN — All categories and books (including categories without books)
SELECT C.CategoryName, B.Title
FROM Category C
RIGHT JOIN Book B ON C.CategoryID = B.CategoryID;

FULL OUTER JOIN (simulated using UNION)
SELECT A.AuthorName, B.Title
FROM Author A
LEFT JOIN Book B ON A.AuthorID = B.AuthorID
UNION
SELECT A.AuthorName, B.Title
FROM Author A
RIGHT JOIN Book B ON A.AuthorID = B.AuthorID;

Subquery in WHERE — Members who borrowed 'Data Structures'
SELECT MemberName
FROM Member
WHERE MemberID IN (
SELECT MemberID FROM Borrow
WHERE BookID = (SELECT BookID FROM Book WHERE Title = 'Data Structures')
);

Subquery in FROM — Count of books borrowed per member
SELECT M.MemberName, BorrowStats.Total
FROM Member M
JOIN (
SELECT MemberID, COUNT(*) AS Total
FROM Borrow
GROUP BY MemberID
) AS BorrowStats ON M.MemberID = BorrowStats.MemberID;

Subquery in SELECT — Books with times borrowed
SELECT B.Title,
(SELECT COUNT(*) FROM Borrow WHERE BookID = B.BookID) AS TimesBorrowed
FROM Book B;

View — Book details with category and author
CREATE VIEW BookDetailsView AS
SELECT B.Title, C.CategoryName, A.AuthorName
FROM Book B
JOIN Category C ON B.CategoryID = C.CategoryID
JOIN Author A ON B.AuthorID = A.AuthorID;
```

```
Use the view — View science books
SELECT * FROM BookDetailsView
WHERE CategoryName = 'Science';

Complex JOIN — Members and their borrowed books
SELECT M.MemberName, B.Title, BR.BorrowDate, BR.ReturnDate
FROM Borrow BR
JOIN Member M ON BR.MemberID = M.MemberID
JOIN Book B ON BR.BookID = B.BookID;
```

12 Implement Indexing and querying with MongoDB using following example.

Students(stud_id, stud_name,stud_addr,stud_marks)

```
use SchoolDB

db.Students.insertMany([
  { stud_id: 101, stud_name: "Amit", stud_addr: "Pune", stud_marks: 85 },
  { stud_id: 102, stud_name: "Sneha", stud_addr: "Mumbai", stud_marks: 92 },
  { stud_id: 103, stud_name: "Kiran", stud_addr: "Nagpur", stud_marks: 76 },
  { stud_id: 104, stud_name: "Riya", stud_addr: "Pune", stud_marks: 66 },
  { stud_id: 105, stud_name: "Neha", stud_addr: "Delhi", stud_marks: 98 }
])

db.Students.createIndex({ stud_marks: 1 }) // Ascending order

db.Students.getIndexes()

db.Students.find({ stud_marks: { $gt: 80 } })

db.Students.find({ stud_addr: "Pune" })

db.Students.find().sort({ stud_marks: 1 })
```

13 Create the instance of the COMPANY which consists of the following tables:

EMPLOYEE(Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Dno)
DEPARTEMENT(Dname, Dno, Mgr_ssn, Mgr_start_date)
DEPT_LOCATIONS(Dnumber, Dlocation)
PROJECT(Pname, Pnumber, Plocation, Dno)
WORKS_ON(Essn, Pno, Hours)
DEPENDENT(Essn, Dependent_name, Sex, Bdate, Relationship)

Perform following queries

1. **For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name,address, and birth date.**
2. **Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.**
3. **Retrieve all employees whose address is in Houston, Texas.**
4. **Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10 percent raise.**

```
-- EMPLOYEE Table
CREATE TABLE EMPLOYEE (
   Fname VARCHAR(20),
   Minit CHAR(1),
   Lname VARCHAR(20),
   Ssn CHAR(9) PRIMARY KEY,
   Bdate DATE,
   Address VARCHAR(100),
   Sex CHAR(1),
   Salary DECIMAL(10,2),
   Dno INT
);

-- DEPARTMENT Table
CREATE TABLE DEPARTMENT (
   Dname VARCHAR(30),
   Dno INT PRIMARY KEY,
   Mgr_ssn CHAR(9),
   Mgr_start_date DATE
);

-- DEPT_LOCATIONS Table
CREATE TABLE DEPT_LOCATIONS (
   Dnumber INT,
   Dlocation VARCHAR(50)
);

-- PROJECT Table
CREATE TABLE PROJECT (
   Pname VARCHAR(30),
   Pnumber INT PRIMARY KEY,
   Plocation VARCHAR(50),
   Dno INT
);

-- WORKS_ON Table
CREATE TABLE WORKS_ON (
   Essn CHAR(9),
   Pno INT,
```

```
    Hours DECIMAL(4,1)
);

-- DEPENDENT Table
CREATE TABLE DEPENDENT (
    Essn CHAR(9),
    Dependent_name VARCHAR(30),
    Sex CHAR(1),
    Bdate DATE,
    Relationship VARCHAR(20)
);


-- Sample Employees
INSERT INTO EMPLOYEE VALUES
('John', 'B', 'Smith', '123456789', '1980-01-01', 'Houston, Texas', 'M', 50000, 1),
('Anna', 'D', 'Jones', '987654321', '1985-02-02', 'Dallas, Texas', 'F', 60000, 2);

-- Departments
INSERT INTO DEPARTMENT VALUES
('Research', 1, '123456789', '2010-01-01'),
('Development', 2, '987654321', '2012-03-15');

-- Department Locations
INSERT INTO DEPT_LOCATIONS VALUES
(1, 'Houston'),
(2, 'Stafford');

-- Projects
INSERT INTO PROJECT VALUES
('ProductX', 1001, 'Stafford', 1),
('ProductY', 1002, 'Dallas', 2);

-- Works On
INSERT INTO WORKS_ON VALUES
('123456789', 1001, 20),
('987654321', 1002, 25);

-- Dependents
INSERT INTO DEPENDENT VALUES
('123456789', 'Michael', 'M', '2010-10-10', 'Son');


SELECT
    P.Pnumber,
    P.Dno,
    E.Lname AS Manager_LastName,
    E.Address,
    E.Bdate
FROM PROJECT P
JOIN DEPARTMENT D ON P.Dno = D.Dno JOIN
EMPLOYEE E ON D.Mgr_ssn = E.Ssn
```

```
WHERE P.Plocation = 'Stafford';


-- As a worker
SELECT DISTINCT P.Pnumber
FROM EMPLOYEE E
JOIN WORKS_ON W ON E.Ssn = W.Essn JOIN
PROJECT P ON W.Pno = P.Pnumber
WHERE E.Lname = 'Smith'
UNION
-- As a department manager
SELECT DISTINCT P.Pnumber
FROM EMPLOYEE E
JOIN DEPARTMENT D ON E.Ssn = D.Mgr_ssn
JOIN PROJECT P ON D.Dno = P.Dno
WHERE E.Lname = 'Smith';


SELECT *
FROM EMPLOYEE
WHERE Address LIKE '%Houston, Texas%';


SELECT
   E.Fname,
   E.Lname,
   E.Salary,
   (E.Salary * 1.10) AS New_Salary
FROM EMPLOYEE E
JOIN WORKS_ON W ON E.Ssn = W.Essn
JOIN PROJECT P ON W.Pno = P.Pnumber
WHERE P.Pname = 'ProductX';
```

14 Implement all SQL DML opeartions with operators, functions, and set operator for given schema:

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)

Solve following query:

1. **Find the average account balance at each branch**
2. **Find no. of depositors at each branch.**

3. Find the branches where average account balance > 12000.

4. Find number of tuples in customer relation.

```
-- Create Tables

CREATE TABLE Account (
   Acc_no INT PRIMARY KEY,
   branch_name VARCHAR(50),
   balance DECIMAL(10, 2)
);

CREATE TABLE Branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50),
   assets DECIMAL(15, 2)
);

CREATE TABLE Customer (
   cust_name VARCHAR(50),
   cust_street VARCHAR(50),
   cust_city VARCHAR(50)
);

CREATE TABLE Depositor (
   cust_name VARCHAR(50),
   acc_no INT
);

CREATE TABLE Loan (
   loan_no INT PRIMARY KEY,
   branch_name VARCHAR(50),
   amount DECIMAL(10, 2)
);

CREATE TABLE Borrower (
   cust_name VARCHAR(50),
   loan_no INT
);

-- Insert Data

INSERT INTO Branch VALUES
('Camp', 'Pune', 5000000),
('MG Road', 'Mumbai', 3000000);

INSERT INTO Account VALUES
(101, 'Camp', 15000),
(102, 'Camp', 10000),
(103, 'MG Road', 9000);
```

```
INSERT INTO Customer VALUES
('Isha', 'Main Street', 'Pune'),
('Ravi', 'High Street', 'Mumbai'),
('Anu', 'Link Road', 'Pune');

INSERT INTO Depositor VALUES
('Isha', 101),
('Ravi', 102),
('Anu', 103);

INSERT INTO Loan VALUES
(201, 'Camp', 50000),
(202, 'MG Road', 30000);

INSERT INTO Borrower VALUES
('Isha', 201),
('Anu', 202);

-- Queries

-- 1. Find the average account balance at each branch
SELECT branch_name, AVG(balance) AS avg_balance
FROM Account
GROUP BY branch_name;

-- 2. Find number of depositors at each branch (unique customers)
SELECT A.branch_name, COUNT(DISTINCT D.cust_name) AS no_of_depositors
FROM Account A
JOIN Depositor D ON A.Acc_no = D.acc_no
GROUP BY A.branch_name;

-- 3. Find branches where average account balance > 12000
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 12000;

-- 4. Find number of tuples in Customer relation
SELECT COUNT(*) AS total_customers
FROM Customer;

-- Extra: Find customers who are depositors or borrowers (using set operator UNION)
SELECT DISTINCT cust_name FROM Depositor
UNION
SELECT DISTINCT cust_name FROM Borrower;
```

15 Implement all SQL DML opeartions with operators, functions, and set operator for given schema:

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, check constrains, not null etc.

Solve following query:

1. **Find the names of all branches in loan relation.**
2. **Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000.**
3. **Find all customers who have a loan from bank.**
4. **Find their names,loan_no and loan amount.**

```
-- 1. Branch Table
CREATE TABLE Branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50) NOT NULL,
   assets DECIMAL(15,2) CHECK (assets >= 0)
);

-- 2. Account Table
CREATE TABLE Account (
   acc_no INT PRIMARY KEY,
   branch_name VARCHAR(50) NOT NULL,
   balance DECIMAL(10,2) CHECK (balance >= 0),
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

-- 3. Customer Table
CREATE TABLE Customer (
   cust_name VARCHAR(50) PRIMARY KEY,
   cust_street VARCHAR(100) NOT NULL,
   cust_city VARCHAR(50) NOT NULL
);

-- 4. Depositor Table
CREATE TABLE Depositor (
   cust_name VARCHAR(50) NOT NULL,
   acc_no INT NOT NULL,
   PRIMARY KEY (cust_name, acc_no),
   FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
   FOREIGN KEY (acc_no) REFERENCES Account(acc_no)
```

```
);

-- 5. Loan Table
CREATE TABLE Loan (
   loan_no INT PRIMARY KEY,
   branch_name VARCHAR(50) NOT NULL,
   amount DECIMAL(10,2) CHECK (amount > 0),
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

-- 6. Borrower Table
CREATE TABLE Borrower (
   cust_name VARCHAR(50) NOT NULL,
   loan_no INT NOT NULL,
   PRIMARY KEY (cust_name, loan_no),
   FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
   FOREIGN KEY (loan_no) REFERENCES Loan(loan_no)
);

-- Insert data

INSERT INTO Branch VALUES
('Akurdi', 'Pune', 5000000),
('Kothrud', 'Pune', 3000000);

INSERT INTO Account VALUES
(101, 'Akurdi', 15000),
(102, 'Kothrud', 8000);

INSERT INTO Customer VALUES
('Isha', 'Main St', 'Pune'),
('Ravi', 'Ring Rd', 'Pune'),
('Anu', 'FC Rd', 'Mumbai');

INSERT INTO Depositor VALUES
('Isha', 101),
('Ravi', 102);

INSERT INTO Loan VALUES
(201, 'Akurdi', 15000),
(202, 'Kothrud', 11000),
(203, 'Akurdi', 9000);

INSERT INTO Borrower VALUES
('Anu', 201),
('Isha', 202);

-- Queries

-- 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 Akurdi Branch with amount > 12000.
SELECT loan_no
FROM Loan
WHERE branch_name = 'Akurdi' AND amount > 12000;

-- 3. Find all customers who have a loan from the bank.
SELECT DISTINCT cust_name
FROM Borrower;

-- 4. Find their names, loan_no and loan amount.
SELECT B.cust_name, B.loan_no, L.amount
FROM Borrower B
JOIN Loan L ON B.loan_no = L.loan_no;
```

16 Implement Map reduce operation with following example using MongoDB Students(stud_id, stud_name,stud_addr,stud_marks)

AND

Write a PL/SQL code to calculate total and percentage of marks of the students in four subjects.

```
// MongoDB Insert Students Documents
db.Students.insertMany([
{
stud_id: 1,
stud_name: "Isha",
stud_addr: "Pune",
stud_marks: { sub1: 85, sub2: 90, sub3: 75, sub4: 80 }
},
{
stud_id: 2,
stud_name: "Raj",
stud_addr: "Mumbai",
stud_marks: { sub1: 70, sub2: 65, sub3: 60, sub4: 75 }
}
]);
// Map function to sum marks
var mapFunction = function () {
var total = this.stud_marks.sub1 + this.stud_marks.sub2 + this.stud_marks.sub3 + this.stud_marks.sub4;
emit(this.stud_name, total);
};
// Reduce function to sum emitted values
var reduceFunction = function (key, values) {
return Array.sum(values);
};
// Execute mapReduce
db.Students.mapReduce(
mapFunction,
reduceFunction,
{
```

```
out: "student_totals"
}
);
// View results
db.student_totals.find().pretty();
-- Create Students Table
CREATE TABLE Students (
stud_id INT PRIMARY KEY,
stud_name VARCHAR(50),
stud_addr VARCHAR(100),
marks1 INT,
marks2 INT,
marks3 INT,
marks4 INT,
total INT,
percentage DECIMAL(5,2)
);
-- Insert sample data
INSERT INTO Students (stud_id, stud_name, stud_addr, marks1, marks2, marks3, marks4)
VALUES
(1, 'Isha', 'Pune', 85, 90, 88, 92),
(2, 'Raj', 'Mumbai', 75, 70, 65, 80),
(3, 'Sneha', 'Nagpur', 95, 98, 92, 96);
-- Create procedure to update total and percentage
DELIMITER $$
CREATE PROCEDURE UpdateTotalAndPercentage()
BEGIN
UPDATE Students
SET total = marks1 + marks2 + marks3 + marks4,
percentage = (marks1 + marks2 + marks3 + marks4) / 4;
END $$
DELIMITER ;
-- Call procedure to calculate totals and percentages
CALL UpdateTotalAndPercentage();

-- Optional: View updated table
SELECT * FROM Students;
```

17 Create following collection and using MongoDB implement all CRUD operations. Orders(cust_id, amount, status)

```
use shopDB

db.Orders.insertMany([
  { cust_id: 1, amount: 500, status: "pending" },
  { cust_id: 2, amount: 1000, status: "shipped" },
  { cust_id: 3, amount: 750, status: "delivered" } ])

db.Orders.insertOne({ cust_id: 4, amount: 1200, status: "pending" })

db.Orders.insertMany([
```

```
  { cust_id: 5, amount: 900, status: "shipped" },
  { cust_id: 6, amount: 300, status: "cancelled" }
])

db.Orders.find()

db.Orders.find({ status: "pending" })

db.Orders.find({ amount: { $gt: 700 } })

db.Orders.updateOne(

  { cust_id: 1 },
  { $set: { status: "shipped" } }
)

db.Orders.updateMany(
  { status: "pending" },
  { $set: { status: "processing" } }
)

db.Orders.deleteOne({ cust_id: 6 })


db.Orders.deleteMany({ status: "cancelled" })

db.Orders.find().pretty()
```

18 Implement all SQL DML operations with operators, functions, and set operator for given schema:

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, check constrains, not null etc.Solve following query:

1. **Find all customers who have an account or loan or both at bank.**
2. **Find all customers who have both account and loan at bank.**
3. **Find all customer who have account but no loan at the bank.**
4. **Find average account balance at Akurdi branch.**

```
CREATE TABLE Branch (     branch_name
VARCHAR(50) PRIMARY KEY,     branch_city
VARCHAR(50) NOT NULL,     assets INT
CHECK (assets >= 0)
);

CREATE TABLE Account (      acc_no
INT PRIMARY KEY,      branch_name
VARCHAR(50),              balance   INT
CHECK (balance >= 0),
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE  TABLE  Customer  (         cust_name
VARCHAR(50) PRIMARY KEY,
   cust_street VARCHAR(50),
   cust_city VARCHAR(50)
);

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)
);

CREATE TABLE Loan (      loan_no
INT PRIMARY KEY,    branch_name
VARCHAR(50),            amount   INT
CHECK (amount > 0),
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

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)
);


-- Branches
INSERT INTO Branch VALUES ('Akurdi', 'Pune', 1000000);
INSERT INTO Branch VALUES ('Nigdi', 'Pune', 800000);

-- Accounts
INSERT INTO Account VALUES (101, 'Akurdi', 15000);
INSERT INTO Account VALUES (102, 'Nigdi', 5000);

-- Customers
```

```
INSERT INTO Customer VALUES ('Alice', 'MG Road', 'Pune');
INSERT INTO Customer VALUES ('Bob', 'FC Road', 'Pune');
INSERT INTO Customer VALUES ('Charlie', 'JM Road', 'Pune');

-- Depositors
INSERT INTO Depositor VALUES ('Alice', 101);
INSERT INTO Depositor VALUES ('Bob', 102);

-- Loans
INSERT INTO Loan VALUES (201, 'Akurdi', 20000);
INSERT INTO Loan VALUES (202, 'Nigdi', 30000);

-- Borrowers
INSERT INTO Borrower VALUES ('Alice', 201);
INSERT INTO Borrower VALUES ('Charlie', 202);


SELECT cust_name FROM Depositor
UNION
SELECT cust_name FROM Borrower;


SELECT cust_name FROM Depositor
INTERSECT
SELECT cust_name FROM Borrower;

SELECT cust_name FROM Depositor
EXCEPT
SELECT cust_name FROM Borrower;

SELECT AVG(balance) AS avg_balance
FROM Account
WHERE branch_name = 'Akurdi';
```

19 Implement all SQL DML operations with operators, functions, and set operator for given schema:

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)

Solve following query:

1. **Calculate total loan amount given by bank.**
2. **Delete all loans with loan amount between 1300 and 1500.**

3. Delete all tuples at every branch located in Nigdi.

```
-- Table Creation
CREATE TABLE Branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50),
   assets INT CHECK (assets >= 0)
);

CREATE TABLE Account (
   acc_no INT PRIMARY KEY,
   branch_name VARCHAR(50),
   balance INT,
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Customer (
   cust_name VARCHAR(50) PRIMARY KEY,
   cust_street VARCHAR(100),
   cust_city VARCHAR(50)
);

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)
);

CREATE TABLE Loan (
   loan_no INT PRIMARY KEY,
   branch_name VARCHAR(50),
   amount INT,
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

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)
);
```

```
-- Inserts
INSERT INTO Branch VALUES ('Akurdi', 'Pune', 1000000);
INSERT INTO Branch VALUES ('Nigdi', 'Pune', 900000);
INSERT INTO Branch VALUES ('Hadapsar', 'Pune', 800000);

INSERT INTO Loan VALUES (301, 'Akurdi', 1000);
INSERT INTO Loan VALUES (302, 'Nigdi', 1400);
INSERT INTO Loan VALUES (303, 'Nigdi', 1600);
INSERT INTO Loan VALUES (304, 'Hadapsar', 2500);

-- 1. Calculate total loan amount
SELECT SUM(amount) AS total_loan_amount
FROM Loan;

-- 2. Delete loans with amount between 1300 and 1500
DELETE FROM Loan
WHERE amount BETWEEN 1300 AND 1500;

-- 3. Delete all tuples related to branch 'Nigdi'
-- Delete Borrowers related to loans in Nigdi
DELETE FROM Borrower
WHERE loan_no IN (
   SELECT loan_no FROM Loan WHERE branch_name = 'Nigdi'
);

-- Delete Loans in Nigdi
DELETE FROM Loan
WHERE branch_name = 'Nigdi';

-- Delete Depositors who have accounts in Nigdi
DELETE FROM Depositor
WHERE acc_no IN (
   SELECT acc_no FROM Account WHERE branch_name = 'Nigdi'
);

-- Delete Accounts in Nigdi
DELETE FROM Account
WHERE branch_name = 'Nigdi';

-- Finally, delete the branch Nigdi
DELETE FROM Branch
WHERE branch_name = 'Nigdi';
```

20 Create the following tables.

1. Deposit (actno,cname,bname,amount,adate)

2. **Branch (bname,city)**
3. **Customers (cname, city)**
4. **Borrow(loanno,cname,bname, amount)**

Add primary key and foreign key wherever applicable.Insert data into the above created tables.

1. **Display account date of customers “ABC”.**
2. **Modify the size of attribute of amount in deposit**
3. **Display names of customers living in city pune.**
4. **Display name of the city where branch “OBC” is located.**
5. **Find the number of tuples in the *customer* relation**

```
-- 1. Create Tables

CREATE TABLE Branch (
   bname VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50) NOT NULL
);

CREATE TABLE Customers (
   cname VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50) NOT NULL
);

CREATE TABLE Deposit (
   actno INT PRIMARY KEY,
   cname VARCHAR(50),
   bname VARCHAR(50),
   amount DECIMAL(15,2),
   adate DATE,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
   loanno INT PRIMARY KEY,
   cname VARCHAR(50),
   bname VARCHAR(50),
   amount DECIMAL(15,2),
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- 2. Insert Sample Data

INSERT INTO Branch VALUES
('OBC', 'Pune'),
('SBI', 'Mumbai'),
```

```
('HDFC', 'Pune');

INSERT INTO Customers VALUES
('ABC', 'Pune'),
('DEF', 'Mumbai'),
('GHI', 'Pune');

INSERT INTO Deposit VALUES
(1001, 'ABC', 'OBC', 50000.00, '2024-01-15'),
(1002, 'DEF', 'SBI', 75000.00, '2024-02-20'),
(1003, 'GHI', 'HDFC', 30000.00, '2024-03-25');

INSERT INTO Borrow VALUES
(2001, 'ABC', 'OBC', 200000.00),
(2002, 'DEF', 'SBI', 150000.00);

-- 3. Queries

-- 3.1 Display account date of customer "ABC"
SELECT adate FROM Deposit WHERE cname = 'ABC';

-- 3.2 Modify the size of attribute amount in Deposit
-- Assuming you want to increase precision or size of amount column
ALTER TABLE Deposit MODIFY amount DECIMAL(20,2);

-- 3.3 Display names of customers living in city Pune
SELECT cname FROM Customers WHERE city = 'Pune';

-- 3.4 Display name of city where branch "OBC" is located
SELECT city FROM Branch WHERE bname = 'OBC';

-- 3.5 Find number of tuples in customer relation
SELECT COUNT(*) AS total_customers FROM Customers;
```

21 Create following tables:

6. **Deposit (actno,cname,bname,amount,adate)**
7. **Branch (bname,city)**
8. **Customers (cname, city)**
9. **Borrow(loanno,cname,bname, amount)**

Add primary key and foreign key wherever applicable. Insert data into the above created tables.

1. **Display customer name having living city Bombay and branch city Nagpur**
2. **Display customer name having same living city as their branch city**
3. **Display customer name who are borrowers as well as depositors and having living city Nagpur.**

```
CREATE TABLE Branch (          bname
VARCHAR(50) PRIMARY KEY,       city
VARCHAR(50)
);
CREATE TABLE Customers (       cname
VARCHAR(50) PRIMARY KEY,       city
VARCHAR(50)
);
CREATE TABLE Deposit (
actno INT PRIMARY KEY,
cname VARCHAR(50), bname
VARCHAR(50), amount INT,
   adate DATE,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);
CREATE TABLE Borrow (
loanno INT PRIMARY KEY,
cname VARCHAR(50),    bname
VARCHAR(50),    amount INT,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);


-- Branch data
INSERT INTO Branch VALUES ('SBI', 'Pune');
INSERT INTO Branch VALUES ('OBC', 'Nagpur');
INSERT INTO Branch VALUES ('ICICI', 'Mumbai');
INSERT INTO Branch VALUES ('KAROLBAGH', 'Delhi');

-- Customers data
INSERT INTO Customers VALUES ('ABC', 'Pune');
INSERT INTO Customers VALUES ('Sunil', 'Nagpur');
INSERT INTO Customers VALUES ('Pramod', 'Delhi');
INSERT INTO Customers VALUES ('Anil', 'Pune');
```

```
-- Deposit data
INSERT INTO Deposit VALUES (101, 'ABC', 'SBI', 1500, '2000-01-01');
INSERT INTO Deposit VALUES (102, 'Sunil', 'OBC', 900, '1998-05-01');
INSERT INTO Deposit VALUES (103, 'Anil', 'SBI', 1200, '1999-08-01');

-- Borrow data
INSERT INTO Borrow VALUES (201, 'ABC', 'SBI', 2000);
INSERT INTO Borrow VALUES (202, 'Sunil', 'OBC', 1500);
INSERT INTO Borrow VALUES (203, 'Pramod', 'KAROLBAGH', 3000);

SELECT d.cname
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = 'Bombay' AND b.city = 'Nagpur';

SELECT d.cname
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = b.city;


SELECT c.cname
FROM Customers c
JOIN Deposit d ON c.cname = d.cname
JOIN Borrow b ON c.cname = b.cname
WHERE c.city = 'Nagpur';
```

22 Create the following tables.

4. **Deposit (actno,cname,bname,amount,adate)**
5. **Branch (bname,city)**
6. **Customers (cname, city)**
7. **Borrow(loanno,cname,bname, amount)**

Add primary key and foreign key wherever applicable. Insert data into the above created tables.

1. **Display loan no and loan amount of borrowers having the same branch as that of sunil.**
2. **Display deposit and loan details of customers in the city where pramod is living.**

3. Display borrower names having deposit amount greater than 1000 and having the same living city as pramod.

4. Display branch and living city of 'ABC'

```
CREATE TABLE Branch ( bname
VARCHAR(50) PRIMARY KEY, city
VARCHAR(50)
);
CREATE TABLE Customers ( cname
   VARCHAR(50) PRIMARY KEY, city
   VARCHAR(50)
);
CREATE TABLE Deposit (
actno INT PRIMARY KEY,
cname VARCHAR(50),
bname VARCHAR(50),
amount INT, adate DATE,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);
CREATE TABLE Borrow (
loanno INT PRIMARY KEY,
cname VARCHAR(50), bname
VARCHAR(50), amount INT,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);
-- Branch data
INSERT INTO Branch VALUES ('SBI', 'Pune');
INSERT INTO Branch VALUES ('OBC', 'Nagpur');
INSERT INTO Branch VALUES ('ICICI', 'Mumbai');
INSERT INTO Branch VALUES ('KAROLBAGH', 'Delhi');

-- Customers data
INSERT INTO Customers VALUES ('ABC', 'Pune');
INSERT INTO Customers VALUES ('Sunil', 'Nagpur');
INSERT INTO Customers VALUES ('Pramod', 'Delhi');
```

```
INSERT INTO Customers VALUES ('Anil', 'Pune');

-- Deposit data
INSERT INTO Deposit VALUES (101, 'ABC', 'SBI', 1500, '2000-01-01');
INSERT INTO Deposit VALUES (102, 'Sunil', 'OBC', 900, '1998-05-01');
INSERT INTO Deposit VALUES (103, 'Anil', 'SBI', 1200, '1999-08-01'); -- Borrow data
INSERT INTO Borrow VALUES (201, 'ABC', 'SBI', 2000);
INSERT INTO Borrow VALUES (202, 'Sunil', 'OBC', 1500);
INSERT INTO Borrow VALUES (203, 'Pramod', 'KAROLBAGH', 3000);

SELECT loanno, amount
FROM Borrow
WHERE bname = (
   SELECT bname FROM Borrow WHERE cname = 'Sunil'
);

SELECT d.*
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Pramod')
UNION
SELECT b.*
FROM Borrow b
JOIN Customers c ON b.cname = c.cname
WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Pramod');

SELECT DISTINCT b.cname
FROM Borrow b
JOIN Deposit d ON b.cname = d.cname
JOIN Customers c ON b.cname = c.cname
WHERE d.amount > 1000 AND c.city = (SELECT city FROM Customers WHERE cname = 'Pramod');

SELECT d.bname, c.city
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE d.cname = 'ABC';
```

23 Implement all Aggregation operations and types of indexing with following collection using MongoDB.

Employee(emp_id, emp_name,emp_dept,salary)

```
use companyDB

db.Employee.insertMany([
  { emp_id: 1, emp_name: "Isha", emp_dept: "HR", salary: 50000 },
  { emp_id: 2, emp_name: "Raj", emp_dept: "IT", salary: 70000 },
  { emp_id: 3, emp_name: "Sneha", emp_dept: "Finance", salary: 60000 },
  { emp_id: 4, emp_name: "Amit", emp_dept: "IT", salary: 80000 },
  { emp_id: 5, emp_name: "Nina", emp_dept: "HR", salary: 55000 }
])

db.Employee.aggregate([
  { $match: { salary: { $gt: 60000 } } }
])
db.Employee.aggregate([
  { $group: { _id: "$emp_dept", avgSalary: { $avg: "$salary" } } }
])
db.Employee.aggregate([
  { $project: { emp_name: 1, salary: 1, _id: 0 } }
])
db.Employee.aggregate([
  { $sort: { salary: -1 } }
])
db.Employee.aggregate([       {
$count: "TotalEmployees" }
])
db.Employee.aggregate([
  { $sort: { salary: -1 } },
  { $limit: 2 }
])
```

```
db.Employee.aggregate([
  { $group: { _id: null, totalSalary: { $sum: "$salary" } } }
])
db.Employee.aggregate([
  {
    $group:  {  _id:  null,
minSalary:  {  $min:  "$salary"  },
maxSalary: { $max: "$salary" }
    }
  }
])

db.Employee.createIndex({ emp_name: 1 }) db.Employee.createIndex({
emp_dept: 1, salary: -1 }) db.Employee.createIndex({ emp_id: 1 }, {
unique: true }) db.Employee.createIndex({ emp_name: "text" })
db.Employee.createIndex({ emp_id: "hashed" })
db.Employee.getIndexes()
```

24 Create the following tables.

5. **Deposit (actno,cname,bname,amount,adate)**
6. **Branch (bname,city)**
7. **Customers (cname, city)**
8. **Borrow(loanno,cname,bname, amount)**

Add primary key and foreign key wherever applicable. Insert data into the above created tables.

1. **Display amount for depositors living in the city where Anil is living.**
2. **Display total loan and maximum loan taken from KAROLBAGH branch.**
3. **Display total deposit of customers having account date later than '1-jan-98'.**
4. **Display maximum deposit of customers living in PUNE.**

```
CREATE TABLE Branch (          bname
VARCHAR(50) PRIMARY KEY,       city
VARCHAR(50)
);
CREATE TABLE Customers (
```

```
   cname    VARCHAR(50)    PRIMARY    KEY,
city VARCHAR(50)
);
CREATE TABLE Deposit (
actno INT PRIMARY KEY,
cname VARCHAR(50),
bname VARCHAR(50),
amount INT,     adate DATE,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);
CREATE TABLE Borrow (
loanno INT PRIMARY KEY,
cname VARCHAR(50),     bname
VARCHAR(50),     amount INT,
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);
-- Branch data
INSERT INTO Branch VALUES ('SBI', 'Pune');
INSERT INTO Branch VALUES ('OBC', 'Nagpur');
INSERT INTO Branch VALUES ('ICICI', 'Mumbai');
INSERT INTO Branch VALUES ('KAROLBAGH', 'Delhi');

-- Customers data
INSERT INTO Customers VALUES ('ABC', 'Pune');
INSERT INTO Customers VALUES ('Sunil', 'Nagpur');
INSERT INTO Customers VALUES ('Pramod', 'Delhi');
INSERT INTO Customers VALUES ('Anil', 'Pune');

-- Deposit data
INSERT INTO Deposit VALUES (101, 'ABC', 'SBI', 1500, '2000-01-01');
INSERT INTO Deposit VALUES (102, 'Sunil', 'OBC', 900, '1998-05-01');
INSERT INTO Deposit VALUES (103, 'Anil', 'SBI', 1200, '1999-08-01'); -- Borrow data
INSERT INTO Borrow VALUES (201, 'ABC', 'SBI', 2000);
INSERT INTO Borrow VALUES (202, 'Sunil', 'OBC', 1500);
```

```
INSERT INTO Borrow VALUES (203, 'Pramod', 'KAROLBAGH', 3000);

SELECT amount FROM
Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Anil');

SELECT SUM(amount) AS total_loan, MAX(amount) AS max_loan
FROM Borrow
WHERE bname = 'KAROLBAGH';

SELECT SUM(amount) AS total_deposit
FROM Deposit
WHERE adate > '1998-01-01';

SELECT MAX(d.amount) AS max_deposit
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = 'PUNE';
```

25 Design and Implement any 5 query using MongoDB

1. **Create a collection called ‘games’.**
2. **Add 5 games to the database. Give each document the following properties: name, gametype, score (out of 100), achievements**
3. **Write a query that returns all the games**
4. **Write a query that returns the 3 highest scored games.**
5. **Write a query that returns all the games that have both the ‘Game Maser’ and the ‘Speed Demon’ achievements.**

```
// 1. Switch to or create the database
use gamingDB;
// 2. Create the 'games' collection
db.createCollection("games");
// 3. Insert 5 game documents
db.games.insertMany([
  {
    name: "Need for Speed",
gametype: "Racing",
    score: 85,
    achievements: ["Speed Demon", "Track Pro"]
  },
  {
    name: "Call of Duty",
    gametype: "Shooter",
```

```
  score: 92,
  achievements: ["Sharp Shooter", "Game Master"]
 },
 {
  name: "Minecraft",
  gametype: "Sandbox",
  score: 90,
  achievements: ["Builder", "Game Master", "Explorer"]
 },
 {
  name: "Among Us",
  gametype: "Multiplayer",
  score: 75,
  achievements: ["Deceiver", "Speed Demon"]
 },
 {
  name: "Fortnite",
  gametype: "Battle Royale",
  score: 95,
  achievements: ["Victory Royale", "Game Master", "Speed Demon"]
 }
]);
// 4. Query: Return all games
db.games.find();
// 5. Query: Return the 3 highest scored games
db.games.find().sort({ score: -1 }).limit(3);
// 6. Query: Return all games that have both 'Game Master' and 'Speed Demon' achievements
db.games.find({
 achievements: { $all: ["Game Master", "Speed Demon"] }
});
```

26 Write a PL/SQL code to calculate tax for an employee of an organization ABC and to display his/her name & tax, by creating a table under employee database as below:
Employee_salary(emp_no,basic,HRA,DA,Total_deduction,net_salary,gross_Salary)

```
-- Step 1: Create the Employee_salary table

CREATE TABLE Employee_salary (

    emp_no INT PRIMARY KEY,

    emp_name VARCHAR(50),

    basic DECIMAL(10,2),

    HRA DECIMAL(10,2),

    DA DECIMAL(10,2),

    Total_deduction DECIMAL(10,2),
```

```
    gross_salary DECIMAL(10,2),
    net_salary DECIMAL(10,2),
    tax DECIMAL(10,2)
);
-- Step 2: Define the stored procedure
DELIMITER $$
CREATE PROCEDURE CalculateTax (
    IN p_emp_no INT,
    IN p_emp_name VARCHAR(50),
    IN p_basic DECIMAL(10,2),
    IN p_HRA DECIMAL(10,2),
    IN p_DA DECIMAL(10,2),
    IN p_deduction DECIMAL(10,2)
)
BEGIN
    DECLARE v_gross DECIMAL(10,2);
    DECLARE v_net DECIMAL(10,2);
    DECLARE v_tax DECIMAL(10,2);
    -- Calculate gross, net salary and tax
    SET v_gross = p_basic + p_HRA + p_DA;
    SET v_net = v_gross - p_deduction;
    SET v_tax = v_net * 0.10;
    -- Insert calculated values into table
    INSERT INTO Employee_salary (
        emp_no, emp_name, basic, HRA, DA,
        Total_deduction, gross_salary, net_salary, tax
    ) VALUES (
        p_emp_no, p_emp_name, p_basic, p_HRA, p_DA,
```

```
    p_deduction, v_gross, v_net, v_tax
  );
  -- Display employee name and tax
  SELECT CONCAT('Employee: ', p_emp_name, ' | Tax: ₹', FORMAT(v_tax, 2)) AS Tax_Report;
END $$
DELIMITER ;
-- Step 3: Call the procedure with example values
CALL CalculateTax(101, 'Isha', 30000, 5000, 4000, 2000);
-- Step 4: View the table content
SELECT * FROM Employee_salary;
```

27 Create PL/SQL code block: Write a PL/SQL block of code for the following schema:

Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
Fine(Roll_no,Date,Amt)

Solve following queries:

1. **Accept roll_no & name of book from user.**
2. **Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.**
3. **If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.**
4. **After submitting the book, status will change from I to R.**
5. **If condition of fine is true, then details will be stored into fine table.**

Use of Control structure and Exception handling is mandatory.

```
CREATE TABLE Borrower (
  Rollin INT PRIMARY KEY,
  Name VARCHAR(50),
  DateofIssue DATE,
  NameofBook VARCHAR(100),
  Status CHAR(1) CHECK (Status IN ('I', 'R'))
);
```

```
CREATE TABLE Fine (
    Roll_no INT,
    Date DATE,
    Amt INT
);
DELIMITER $$
CREATE PROCEDURE ReturnBookAndCalculateFine(IN in_roll INT, IN in_bookname VARCHAR(100))
BEGIN
    DECLARE v_days INT;
    DECLARE v_fine INT DEFAULT 0;
DECLARE v_dateofissue DATE;
    DECLARE v_status CHAR(1);
    DECLARE book_found INT DEFAULT 0;
    -- Check if record exists
    SELECT COUNT(*) INTO book_found
    FROM Borrower
    WHERE Rollin = in_roll AND NameofBook = in_bookname;
    IF book_found = 0 THEN
        SELECT 'No such book issued to this roll number' AS Message;
    ELSE
        -- Get Date of Issue and Status
        SELECT DateofIssue, Status INTO v_dateofissue, v_status
        FROM Borrower
        WHERE Rollin = in_roll AND NameofBook = in_bookname;
        IF v_status = 'R' THEN
            SELECT 'Book already returned!' AS Message;
        ELSE
            SET v_days = DATEDIFF(CURDATE(), v_dateofissue);
            IF v_days > 30 THEN
```

```
            SET v_fine = (15 * 5) + ((v_days - 30) * 50);
        ELSEIF v_days > 15 THEN
            SET v_fine = (v_days - 15) * 5;
        END IF;
        -- Update status to 'R'
        UPDATE Borrower
        SET Status = 'R'
        WHERE Rollin = in_roll AND NameofBook = in_bookname;
        -- Insert fine record if applicable
        IF v_fine > 0 THEN
            INSERT INTO Fine (Roll_no, Date, Amt)
            VALUES (in_roll, CURDATE(), v_fine);
        END IF;
        SELECT CONCAT('Book returned successfully. Fine: Rs.', v_fine) AS Message;
      END IF;
   END IF;
END$$
DELIMITER ;
INSERT INTO Borrower VALUES (101, 'Isha', CURDATE() - INTERVAL 25 DAY, 'C++ Primer', 'I');
INSERT INTO Borrower VALUES (102, 'Smit', CURDATE() - INTERVAL
25 DAY, 'C++ Primer', 'R');
CALL ReturnBookAndCalculateFine(101, 'C++ Primer');
CALL ReturnBookAndCalculateFine(102, 'PYQ Jee Advanced');
```

28 Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall 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.

```
CREATE     TABLE     O_RollCall     (
roll_no INT PRIMARY KEY,
   name VARCHAR(100)
);

CREATE TABLE N_RollCall (
   roll_no INT,
   name VARCHAR(100)
);
INSERT INTO O_RollCall VALUES (1, 'Isha'), (2, 'Amit');
INSERT INTO N_RollCall VALUES (2, 'Amit'), (3, 'Neha'), (4, 'Rahul');

DELIMITER $$

CREATE PROCEDURE MergeRollCall()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE v_roll INT;
   DECLARE v_name VARCHAR(100);

   -- Cursor to read all records from N_RollCall
   DECLARE cur CURSOR FOR
      SELECT roll_no, name FROM N_RollCall;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   OPEN cur;

   read_loop: LOOP
      FETCH cur INTO v_roll, v_name;

      IF done THEN
         LEAVE read_loop;
      END IF;

      -- Check if record already exists in O_RollCall
      IF NOT EXISTS (
         SELECT 1 FROM O_RollCall WHERE roll_no = v_roll
      ) THEN
         INSERT INTO O_RollCall (roll_no, name)
         VALUES (v_roll, v_name);
      END IF;
   END LOOP;

   CLOSE cur;
END$$
DELIMITER ;
CALL MergeRollCall();
SELECT * FROM O_RollCall;
```

29 Writ a PL/SQL procedure to find the number of students ranging from 10070%, 69-60%, 59-50% & below 49% in each course from the student_course table given by the procedure as parameter.
Schema: Student (ROLL_NO ,COURSE, COURSE_COD ,SEM ,TOTAL_MARKS, PERCENTAGE)

```
CREATE TABLE Student (
   ROLL_NO INT,
   COURSE VARCHAR(100),
   COURSE_COD VARCHAR(10),
   SEM INT,
   TOTAL_MARKS INT,
   PERCENTAGE DECIMAL(5,2)
);

INSERT INTO Student VALUES
(1, 'DBMS', 'DB101', 3, 480, 80.00),
(2, 'DBMS', 'DB101', 3, 420, 70.00),
(3, 'DBMS', 'DB101', 3, 390, 65.00),
(4, 'DBMS', 'DB101', 3, 350, 58.33),
(5, 'DBMS', 'DB101', 3, 300, 48.00),
(6, 'AI', 'AI201', 4, 490, 81.66),
(7, 'AI', 'AI201', 4, 370, 61.66),
(8, 'AI', 'AI201', 4, 310, 51.66),
(9, 'AI', 'AI201', 4, 240, 40.00);

DELIMITER $$

CREATE PROCEDURE Count_Percentage_Ranges(IN course_name VARCHAR(100))
BEGIN
   DECLARE high INT DEFAULT 0;
   DECLARE upper_mid INT DEFAULT 0;
   DECLARE lower_mid INT DEFAULT 0;
   DECLARE low INT DEFAULT 0;

   SELECT COUNT(*) INTO high
   FROM Student
   WHERE COURSE = course_name AND PERCENTAGE BETWEEN 70 AND 100;

   SELECT COUNT(*) INTO upper_mid
   FROM Student
   WHERE COURSE = course_name AND PERCENTAGE BETWEEN 60 AND 69.99;

   SELECT COUNT(*) INTO lower_mid
   FROM Student
   WHERE COURSE = course_name AND PERCENTAGE BETWEEN 50 AND 59.99;

   SELECT COUNT(*) INTO low
   FROM Student
   WHERE COURSE = course_name AND PERCENTAGE < 50;
```

```
    SELECT
        course_name AS Course,
        high AS '70–100%',
        upper_mid AS '60–69%',
        lower_mid AS '50–59%',
        low AS '<50%';
END$$
DELIMITER ;

CALL Count_Percentage_Ranges('DBMS'); CALL
Count_Percentage_Ranges('AI');
```

30 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 and900 category is first class, if marks 899 and 825 category is Higher Second Class .

Consider Schema as Stud_Marks(name, total_marks) and Result(Roll,Name, Class)

```
CREATE TABLE Stud_Marks (
    name                VARCHAR(100),
total_marks INT
);

CREATE TABLE Result (
    Roll INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100),
    Class VARCHAR(50)
);

INSERT INTO Stud_Marks (name, total_marks) VALUES
('Alice', 1450),
('Bob', 980),
('Charlie', 850),
('David', 800);

DELIMITER $$

CREATE PROCEDURE proc_Grade()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE stud_name VARCHAR(100);
    DECLARE marks INT;
    DECLARE grade VARCHAR(50);
    DECLARE cur CURSOR FOR SELECT name, total_marks FROM Stud_Marks;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;
```

```
    read_loop: LOOP
        FETCH cur INTO stud_name, marks;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Determine class
        IF marks BETWEEN 990 AND 1500 THEN
            SET grade = 'Distinction';
        ELSEIF marks BETWEEN 900 AND 989 THEN
            SET grade = 'First Class';
        ELSEIF marks BETWEEN 825 AND 899 THEN
            SET grade = 'Higher Second Class';
        ELSE
            SET grade = 'No Class';
        END IF;

        -- Insert into Result table
        INSERT INTO Result (Name, Class) VALUES (stud_name, grade);
    END LOOP;

    CLOSE cur;
END$$

DELIMITER ;

CALL proc_Grade();
SELECT * FROM Result;
```

31.Create database :Citydetails(_id,name,area,population(total,Adults,seniorcitizens,sexratio), geography(avgtemp, avgrainfall, longitude, latitude))

1. **Find the total population in pune.**
2. **returns all city with total population greater than 10 million**
3. **returns the average populations for each city.**
4. **returns the minimum and maximum cities by population for each city.**

```
use CityDB;

// Insert sample data
db.Citydetails.insertMany([
  {
    _id: 1,
    name: "Pune",
    area: 731,
```

```
    population: {
      total: 6500000,
      Adults: 4200000,
      seniorcitizens: 600000,
      sexratio: 940
    },
    geography: {
      avgtemp: 24,
      avgrainfall: 700,
      longitude: 73.8567,
      latitude: 18.5204
    }
  },
  {
    _id: 2,
    name: "Mumbai",
    area: 603,
    population: {
      total: 12400000,
      Adults: 8000000,
      seniorcitizens: 1000000,
      sexratio: 920
    },
    geography: {
      avgtemp: 27,
      avgrainfall: 2400,
      longitude: 72.8777,
      latitude: 19.0760
    }
  },
  {
    _id: 3,
    name: "Delhi",
    area: 1484,
    population: {
      total: 19000000,
      Adults: 12000000,
      seniorcitizens: 1500000,
      sexratio: 889
    },
    geography: {
      avgtemp: 25,
      avgrainfall: 800,
      longitude: 77.1025,
      latitude: 28.7041
    }
  }
]);

// 1. Find total population in Pune
db.Citydetails.find(
```

```
  { name: "Pune" },
  { _id: 0, name: 1, "population.total": 1 }
);

// 2. Find all cities with total population greater than 10 million
db.Citydetails.find(
  { "population.total": { $gt: 10000000 } },
  { _id: 0, name: 1, "population.total": 1 }
);

// 3. Find average population (total) across all cities
db.Citydetails.aggregate([
  {
    $group: {
      _id: null,
      averagePopulation: { $avg: "$population.total" }
    }
  }
]);

// 4. Find city with minimum population
db.Citydetails.find()
  .sort({ "population.total": 1 })
  .limit(1);

// Find city with maximum population
db.Citydetails.find()
  .sort({ "population.total": -1 })
  .limit(1);

// Alternatively, get min and max population values across cities
db.Citydetails.aggregate([
  {
    $group: {
      _id: null,
      minPopulation: { $min: "$population.total" },
      maxPopulation: { $max: "$population.total" }
    }
  }
]);
```

32.Create database :Citydetails(_id,name,area,population(total,Adults,seniorcitizens,sexratio), geography (avgtemp, avgrainfall, longitude, latitude))

1. **Find area wise total population and sort them in increasing order.**
2. **Retrieve name and area where average rain fall is greater than 60**
3. **Create index on city and area find the max population in Mumbai**
4. **Create index on name.**

```
use CityDB;

// Insert sample data (assuming already done)
db.Citydetails.insertMany([
  {
    _id: 1,
    name: "Pune",
    area: 731,
    population: {
      total: 6500000,
      Adults: 4200000,
      seniorcitizens: 600000,
      sexratio: 940
    },
    geography: {
      avgtemp: 24,
      avgrainfall: 700,
      longitude: 73.8567,
      latitude: 18.5204
    }
  },
  {
    _id: 2,
    name: "Mumbai",
    area: 603,
    population: {
      total: 12400000,
      Adults: 8000000,
      seniorcitizens: 1000000,
      sexratio: 920
    },
    geography: {
      avgtemp: 27,
      avgrainfall: 2400,
      longitude: 72.8777,
      latitude: 19.0760
    }
  },
  {
    _id: 3,
    name: "Delhi",
    area: 1484,
    population: {
      total: 19000000,
      Adults: 12000000,
      seniorcitizens: 1500000,
      sexratio: 889
    },
    geography: {
      avgtemp: 25,
      avgrainfall: 800,
```

```
      longitude: 77.1025,
      latitude: 28.7041
    }
  }
]);

// 1. Find area-wise total population and sort in increasing order
// Assuming "area-wise" means for each city: show area and population total sorted by population ascending
db.Citydetails.aggregate([
  {
    $project: {
      name: 1,
      area: 1,
      totalPopulation: "$population.total"
    }
  },
  {
    $sort: { totalPopulation: 1 }
  }
]);

// 2. Retrieve name and area where average rainfall > 60
db.Citydetails.find(
  { "geography.avgrainfall": { $gt: 60 } },
  { _id: 0, name: 1, area: 1 }
);

// 3. Create compound index on name and area
db.Citydetails.createIndex({ name: 1, area: 1 });

// Find max population in Mumbai (though Mumbai is a single city, so just find its population)
db.Citydetails.find(
  { name: "Mumbai" },
  { _id: 0, name: 1, "population.total": 1 }
);

// 4. Create index on name
db.Citydetails.createIndex({ name: 1 });

// To verify indexes
db.Citydetails.getIndexes();
```