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 VIEW CustomerAccounts AS SELECT c.CustomerID, c.Name, a.AcctNo, a.Balance, b.BranchName FROM Customer c JOIN Account a ON c.CustomerID = a.CustomerID JOIN Branch b ON a.BranchID = b.BranchID;

CREATE VIEW CustomerLoans AS SELECT c.CustomerID, c.Name, l.LoanID, l.LoanAmount, b.BranchName FROM Customer c JOIN Loan l ON c.CustomerID = l.CustomerID JOIN Branch b ON l.BranchID = b.BranchID;

CREATE INDEX CustomerId ON Customer (Name ASC); CREATE SEQUENCE AcctNo_seq START WITH 1000 INCREMENT BY 1 NOCACHE;

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 VIEW EmployeesInNY AS SELECT e.EmpID, e.Name, d.DeptName FROM Employee e JOIN Department d ON e.DeptID = d.DeptID WHERE d.Location = 'New York';
CREATE VIEW HighBudgetProjects AS SELECT p.ProjID, p.ProjName, p.Budget, d.DeptName FROM Project p JOIN Department d ON p.DeptID = d.DeptID WHERE p.Budget > 1000000;
CREATE INDEX EmployeeId ON Employee (DeptID ASC, EmpID ASC);
CREATE SEQUENCE EmpID_seq START WITH 1001 INCREMENT BY 1 NOCACHE;

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.

```
DELIMITER //

CREATE TRIGGER trg_update_noc
AFTER INSERT ON Library
FOR EACH ROW
BEGIN
   DECLARE v_noc INT DEFAULT 0;

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

   INSERT INTO Library_Audit (bid, bname, noc, timestampofquery, fine)
   VALUES (NEW.bid, NEW.bname, v_noc, NOW(), 0);
END;
//

DELIMITER //

CREATE TRIGGER trg_check_fine
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
   DECLARE days_diff INT DEFAULT 0;
   DECLARE fine_amount INT DEFAULT 0;

   IF NEW.doi IS NOT NULL THEN
      SET days_diff = DATEDIFF(CURDATE(), NEW.doi);

      IF days_diff > 20 THEN
         SET fine_amount = days_diff * 10;

         -- Insert into Library_Audit with fine
         INSERT INTO Library_Audit (bid, bname, noc, timestampofquery, fine)
         VALUES (NEW.bid, NEW.bname, 0, NOW(), fine_amount);
      END IF;
   END IF;
END;
//

DELIMITER ;
```

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.

```
DELIMITER //

CREATE TRIGGER trg_after_update_library AFTER UPDATE ON Library FOR EACH ROW
 BEGIN
                INSERT INTO Library_Audit (bid, bname, doi, status, action_type)VALUES (OLD.bid, OLD.bname, OLD.doi, OLD.status, 'UPDATE');
 END;
//
```

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

use analyticsDB

db.sites.insertMany([{ url: "https://example.com", dateofaccess: ISODate("2024-03-12T10:00:00Z") }, { url: "https://example.com", dateofaccess: ISODate("2024-03-18T12:30:00Z") }, { url: "https://test.com", dateofaccess: ISODate("2024-03-20T14:00:00Z") }, { url: "https://example.com", dateofaccess: ISODate("2024-04-01T09:15:00Z") }, { url: "https://test.com", dateofaccess: ISODate("2024-04-02T10:15:00Z") }
]);

db.sites.mapReduce(function () { const month = this.dateofaccess.getMonth() + 1; const year = this.dateofaccess.getFullYear(); emit({ url: this.url, month: month, year: year }, 1); }, function (key, values) { return Array.sum(values); }, { out: "monthly_site_accesses" });

db.monthly_site_accesses.find().pretty();

6 Create tables CitiesIndia(<u>pincode</u>,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), -- in square km population INT, avgrainfall DECIMAL(10,2)); CREATE TABLE Categories (Type VARCHAR(20),pincode INT, FOREIGN KEY (pincode) REFERENCES CitiesIndia(pincode));

CREATE FUNCTION get_population_density(pop INT, ar DECIMAL(10,2))RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN IF pop < 10 OR pop > 25718 THEN Return null; END IF; RETURN pop / ar; END;

CREATE PROCEDURE CategorizeCities() BEGIN DECLARE done INT DEFAULT FALSE; DECLARE city_pincode INT; DECLARE city_population INT; DECLARE city_area DECIMAL(10,2); DECLARE density DECIMAL(10,2); DECLARE city_cursor CURSOR FOR SELECT pincode, population, area FROM CitiesIndia; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; OPEN city_cursor; read_loop: LOOP FETCH city_cursor INTO city_pincode, city_population, city_area; IF done THEN LEAVE read_loop; END IF; SET density = get_population_density(city_population, city_area); IF density > 3000 THEN INSERT INTO Categories(Type, pincode) VALUES ('High Density', city_pincode); ELSEIF density BETWEEN 1000 AND 2999 THEN INSERT INTO Categories(Type, pincode) VALUES ('Moderate', city_pincode); ELSE INSERT INTO Categories(Type, pincode) VALUES ('Low Density', city_pincode); END IF; END LOOP; CLOSE city_cursor; END;

CALL CategorizeCities();

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 PROCEDURE process_student_result(IN p_rollno INT) BEGIN DECLARE m1, m2, m3, m4, m5, total INT; DECLARE class VARCHAR(30); DECLARE invalid_marks CONDITION FOR SQLSTATE '45000'; -- Fetch student marks SELECT Marks1, Marks2, Marks3, Marks4, Marks5 INTO m1, m2, m3, m4, m5 FROM Student WHERE rollno = p_rollno; -- Calculate total SET total = m1 + m2 + m3 + m4 + m5; -- Determine class IF total BETWEEN 990 AND 1499 THEN SET class = 'Distinction'; ELSEIF total BETWEEN 900 AND 989 THEN SET class = 'First Class'; ELSEIF total BETWEEN 825 AND 899 THEN SET class = 'Higher Second'; ELSEIF total BETWEEN 750 AND 824 THEN SET class = 'Second Pass'; ELSE SET class = 'Fail'; END IF; -- Insert result INSERT INTO Result (rollno, total_marks, class) VALUES (p_rollno, total, class); END;

INSERT INTO Student VALUES (1, 'Amit', 95, 99, 88, 90, 97); -- Call procedure CALL process_student_result(1); -- View results 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.

CREATE VIEW BookAuthorView AS SELECT b.BookID, b.Title, a.AuthorName, b.Genre, b.Year FROM Books b JOIN Authors a ON b.AuthorID = a.AuthorID;
CREATE VIEW MemberTransactionView AS SELECT m.MemberName, t.TxnID, b.Title, t.IssueDate, t.ReturnDate, t.Status FROM Transactions t JOIN Members m ON t.MemberID = m.MemberID JOIN Books b ON t.BookID = b.BookID;

CREATE INDEX Lib_Index1 ON Authors (AuthorName ASC);
CREATE TABLE BookSeq (BookID INT AUTO_INCREMENT PRIMARY KEY, Dummy CHAR(1) DEFAULT 'X');

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.

```
CREATE PROCEDURE handle_fine(IN i_roll_no INT, IN name_of_book VARCHAR(100))
BEGIN
   DECLARE no_of_days INT DEFAULT 0;
   DECLARE return_date DATE DEFAULT CURDATE();
   DECLARE temp INT DEFAULT 0;
   DECLARE doi DATE;
   DECLARE fine DECIMAL(10,2) DEFAULT 0;
```

```
    -- Get Date of Issue from borrower table
    SELECT dateofissue INTO doi
    FROM borrower
    WHERE rollin = i_roll_no AND nameofbook = name_of_book;

    -- Calculate number of days from issue to return
    SET no_of_days = DATEDIFF(return_date, doi);

    -- Calculate fine
    IF no_of_days > 15 AND no_of_days <= 30 THEN
        SET fine = 5 * no_of_days;
    ELSEIF no_of_days > 30 THEN
        SET temp = no_of_days - 30;
        SET fine = 150 + temp * 50;
    END IF;

    -- Insert into fine table
    IF fine > 0 THEN
        INSERT INTO fine (roll_no, date, amt)
        VALUES (i_roll_no, return_date, fine);
    END IF;

    -- Update status in borrower table
    UPDATE borrower
    SET status = 'RETURNED'
    WHERE rollin = i_roll_no AND nameofbook = name_of_book;
END;
```

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 (  Rollin INT PRIMARY KEY,Name VARCHAR(100) NOT NULL, DateofIssue DATE NOT NULL, NameofBook VARCHAR(100) NOT NULL, Status ENUM('ISSUED', 'RETURNED') DEFAULT 'ISSUED');

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

CREATE INDEX idx_bookname ON Borrower(NameofBook);

CREATE SEQUENCE Rollin_Seq START WITH 1001 INCREMENT BY 1;

CREATE SYNONYM LibBorrower FOR Borrower;
```

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

- **Borrower**(Rollin, Name, DateofIssue, NameofBook, Status)
- **Fine**(Roll_no, Date, Amt)
- **Library**(bid, bname, doi, status, noc)
- **Transaction**(tid, bid, bname, status)

```
SELECT b.Name, b.NameofBook, b.DateofIssue FROM Borrower b INNER JOIN Library l ON b.NameofBook = l.bname
WHERE b.Status = 'I';
SELECT l.bname, b.Name, b.Status FROM Library l LEFT JOIN Borrower b ON l.bname = b.NameofBook AND b.Status = 'I';
SELECT Name, Rollin FROM Borrower WHERE Rollin IN (SELECT Roll_no FROM Fine WHERE Amt > 100);
SELECT b.Name, b.Rollin, (SELECT SUM(Amt) FROM Fine f WHERE f.Roll_no = b.Rollin) AS TotalFine FROM Borrower b;
CREATE OR REPLACE VIEW CurrentlyIssuedBooks AS SELECT b.Rollin, b.Name, b.NameofBook, b.DateofIssue FROM Borrower b WHERE b.Status = 'I'; SELECT Name, NameofBook, DateofIssue FROM Borrower b WHERE DateofIssue = (SELECT MAX(doi) FROM Library WHERE bname = b.NameofBook);
SELECT b.Name, COUNT(t.tid) AS TotalTransactions FROM Borrower b JOIN transaction t ON b.NameofBook = t.bname GROUP BY b.Name;
SELECT b.Name, b.NameofBook, l.bname, l.status FROM Borrower b FULL OUTER JOIN Library l ON b.NameofBook = l.bname;
```

12 **Implement Indexing and** querying with MongoDB using following example. Students(stud_id, stud_name,stud_addr,stud_marks)

```
use your_database;

// Insert sample documents
db.Students.insertMany([
  { stud_id: 1, stud_name: "Alice", stud_addr: "New York", stud_marks: 85 },
  { stud_id: 2, stud_name: "Bob", stud_addr: "California", stud_marks: 92 },
  { stud_id: 3, stud_name: "Charlie", stud_addr: "New York", stud_marks: 78 },
  { stud_id: 4, stud_name: "David", stud_addr: "Texas", stud_marks: 88 },
  { stud_id: 5, stud_name: "Eve", stud_addr: "California", stud_marks: 95 }
]);
 db.Students.createIndex({ stud_name: 1 });   db.Students.createIndex({ stud_addr: 1, stud_marks: -1 });   db.Students.createIndex({ stud_id: 1 }, { unique: true });   db.Students.find({ stud_name: "Alice" });   db.Students.find({ stud_addr: "California", stud_marks: { $gt: 90 } });    db.Students.find().sort({ stud_marks: -1 });     db.Students.find({ stud_name: "Alice" }).explain("executionStats");
```

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.

```
SELECT p.Pnumber, p.Dno, e.Lname, 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';
SELECT DISTINCT p.Pnumber FROM PROJECT p JOIN WORKS_ON w ON p.Pnumber = w.Pno JOIN EMPLOYEE e ON w.Essn = e.Ssn WHERE e.Lname = 'Smith' UNION
SELECT p.Pnumber FROM PROJECT p JOIN DEPARTMENT d ON p.Dno = d.Dno JOIN EMPLOYEE m ON d.Mgr_ssn = m.Ssn WHERE m.Lname = 'Smith';
SELECT * FROM EMPLOYEE WHERE Address LIKE '%Houston, TX%';
SELECT e.Ssn, e.Fname, e.Lname, e.Salary, ROUND(e.Salary * 1.10, 2) 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.

```
SELECT Acc_no, branch_name, balance, ROUND(balance * 1.10, 2) AS increased_balance FROM Account WHERE balance > 5500;
SELECT cust_name, cust_city FROM customer WHERE cust_city = 'New York' OR cust_city = 'Chicago';
SELECT branch_name, branch_city, assets FROM branch WHERE assets BETWEEN 1000000 AND 1500000;
SELECT cust_name FROM customer WHERE cust_name LIKE 'A%';
UPDATE Account SET balance = balance * 1.05 WHERE balance < 6000
DELETE FROM customer WHERE cust_city = 'Chicago';
SELECT cust_name FROM Depositor UNION SELECT cust_name FROM Borrower;
SELECT d.cust_name FROM Depositor d JOIN Borrower b ON d.cust_name = b.cust_name;
SELECT d.cust_name FROM Depositor d LEFT JOIN Borrower b ON d.cust_name = b.cust_name WHERE b.cust_name IS NULL;
SELECT branch_city, SUM(assets) AS total_assets FROM branch WHERE branch_city = 'New York' GROUP BY branch_city;

SELECT branch_name, AVG(balance) AS average_balance FROM Account GROUP BY branch_name;
SELECT A.branch_name, COUNT(DISTINCT D.cust_name) AS num_depositors FROM Depositor D JOIN Account A ON D.acc_no = A.Acc_no GROUP BY A.branch_name;
SELECT branch_name FROM Account GROUP BY branch_name HAVING AVG(balance) > 12000;
SELECT COUNT(*) AS total_customers FROM Customer;
```

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.

```
SELECT DISTINCT branch_name FROM Loan;
SELECT loan_no FROM Loan WHERE branch_name = 'Akurdi' AND amount > 12000;
SELECT DISTINCT cust_name FROM Borrower;
SELECT B.cust_name, L.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)

```
db.Students.insertMany([
  { stud_id: 1, stud_name: "Alice",   stud_addr: "Pune",   stud_marks: 85 },
  { stud_id: 2, stud_name: "Bob",     stud_addr: "Mumbai", stud_marks: 92 },
  { stud_id: 3, stud_name: "Charlie", stud_addr: "Pune",   stud_marks: 78 }
]);
// Map function: emit address and marks
var mapFn = function() {   emit(this.stud_addr, this.stud_marks); }; var reduceFn = function(key, values) {   return Array.sum(values) / values.length; }; db.Students.mapReduce(    mapFn, reduceFn, { out: "avg_marks_per_city" } );
db.avg_marks_per_city.find().pretty();
```

AND

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

```
CREATE PROCEDURE calculate_total_percentage( IN sub1 INT, IN sub2 INT,  IN sub3 INT, IN sub4 INT)BEGIN  DECLARE total INT; DECLARE percentage DECIMAL(5,2);
SET total = sub1 + sub2 + sub3 + sub4;    SET percentage = (total / 400) * 100;  SELECT total AS 'Total Marks', percentage AS 'Percentage'; END //
CALL calculate_total_percentage(85, 90, 75, 80);
```

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

use salesDB; db.Orders.insertMany([{ cust_id: 101, amount: 2500, status: "Pending" }, { cust_id: 102, amount: 4300, status: "Shipped" }, { cust_id: 103, amount: 1200, status: "Cancelled" }]);

db.Orders.find(); db.Orders.find({ status: "Pending" }); db.Orders.find({ amount: { $gt: 2000 } }); db.Orders.updateOne({ cust_id: 101 }, { $set: { status: "Shipped" } }); db.Orders.updateMany({ status: "Shipped" },{ $inc: { amount: 500 } }); db.Orders.deleteOne({ cust_id: 103 }); db.Orders.deleteMany({ status: "Cancelled" }); db.Orders.find().pretty();

18 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 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.

SELECT Acc_no, branch_name, balance, ROUND(balance * 1.10, 2) AS increased_balance FROM Account WHERE balance > 5500;
SELECT cust_name, cust_city FROM customer WHERE cust_city = 'New York' OR cust_city = 'Chicago';
SELECT branch_name, branch_city, assets FROM branch WHERE assets BETWEEN 1000000 AND 1500000;
SELECT cust_name FROM customer WHERE cust_name LIKE 'A%';
UPDATE Account SET balance = balance * 1.05 WHERE balance < 6000
DELETE FROM customer WHERE cust_city = 'Chicago';
SELECT cust_name FROM Depositor UNION SELECT cust_name FROM Borrower;
SELECT d.cust_name FROM Depositor d JOIN Borrower b ON d.cust_name = b.cust_name;
SELECT d.cust_name FROM Depositor d LEFT JOIN Borrower b ON d.cust_name = b.cust_name WHERE b.cust_name IS NULL;
SELECT branch_city, SUM(assets) AS total_assets FROM branch WHERE branch_city = 'New York' GROUP BY branch_city;

SELECT DISTINCT cust_name FROM Depositor UNION SELECT DISTINCT cust_name FROM Borrower;
SELECT DISTINCT d.cust_name FROM Depositor d INNER JOIN Borrower b ON d.cust_name = b.cust_name;
SELECT DISTINCT d.cust_name FROM Depositor d WHERE d.cust_name NOT IN (SELECT cust_name FROM Borrower);
SELECT AVG(balance) AS average_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.

Select sum(amount) as Total_loan_amount from loan;
Delete from loan where amount between 1300 and 1500;

SELECT branch_name FROM Branch WHERE branch_city = 'Nigdi'; DELETE FROM Borrower WHERE loan_no IN (SELECT loan_no FROM Loan WHERE branch_name IN (SELECT branch_name FROM Branch WHERE branch_city = 'Nigdi'));
DELETE FROM Loan WHERE branch_name IN (SELECT branch_name FROM Branch WHERE branch_city = 'Nigdi');
DELETE FROM Depositor WHERE acc_no IN (SELECT acc_no FROM Account WHERE branch_name IN (SELECT branch_name FROM Branch WHERE branch_city = 'Nigdi'));
DELETE FROM Account WHERE branch_name IN (SELECT branch_name FROM Branch WHERE branch_city = 'Nigdi');
DELETE FROM Branch WHERE branch_city = '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

SELECT adate FROM Deposit WHERE cname = 'ABC';
ALTER TABLE Deposit MODIFY amount DECIMAL(12,2);
Select cname from customers where city=”pune”;
Select city from branch where bname=”OBC”;
Select count(*) as total_customers from customer;

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.

SELECT DISTINCT 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 DISTINCT 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 DISTINCT d.cname FROM Deposit d JOIN Borrow b ON d.cname = b.cname JOIN Customers c ON d.cname = c.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'

1.Select loanno ,amount from borrow where branch=(select bname from borrow where cname="sunil"); or SELECT loanno, amount FROM Borrow WHERE bname IN (SELECT bname FROM Borrow WHERE cname = 'Sunil');
2.SELECT d.cname, d.actno, d.bname, d.amount AS deposit_amount, d.adate FROM Deposit d JOIN Customers c ON d.cname = c.cname WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Pramod') UNION SELECT b.cname, NULL, b.bname, b.amount AS loan_amount, NULL FROM Borrow b JOIN Customers c ON b.cname = c.cname WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Pramod');
3.SELECT DISTINCT b.cname FROM Borrow b JOIN Deposit d ON b.cname = d.cname JOIN Customers c1 ON b.cname = c1.cname WHERE d.amount > 1000 AND c1.city = (SELECT city FROM Customers WHERE cname = 'Pramod');
4. SELECT d.bname AS branch_name, c.city AS living_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)
{ "emp_id": 101, "emp_name": "Alice", "emp_dept": "IT", "salary": 75000 }
db.Employee.aggregate([{ $group: {_id: "$emp_dept", avg_salary: { $avg: "$salary" } } }]);
db.Employee.aggregate([{ $group: {_id: "$emp_dept", count: { $sum: 1 } } }]);
db.Employee.aggregate([{ $group: {_id: "$emp_dept", max_salary: { $max: "$salary" }, min_salary: { $min: "$salary" } } }]);
db.Employee.aggregate([{ $group: {_id: "$emp_dept",total_salary: { $sum: "$salary" } } }]);
db.Employee.aggregate([{ $sort: { salary: -1 } }]);
db.Employee.createIndex({ salary: 1 });
db.Employee.createIndex({ emp_dept: 1, salary: -1 });
db.Employee.createIndex({ emp_id: 1 }, { unique: true });

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.

SELECT d.cname, d.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.

db.games.insertMany([{ name: "Racing Rivals", gametype: "Racing", score: 92, achievements: ["Speed Demon", "Nitro King"] },]);
db.games.find({});
db.games.find().sort({ score: -1 }).limit(3);
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)

CREATE TABLE Employee (emp_no INT PRIMARY KEY, emp_name VARCHAR(100)); CREATE TABLE Employee_salary (emp_no INT,basic DECIMAL(10,2), HRA DECIMAL(10,2), DA DECIMAL(10,2), Total_deduction DECIMAL(10,2),net_salary DECIMAL(10,2),gross_salary DECIMAL(10,2),FOREIGN KEY (emp_no) REFERENCES Employee(emp_no));

CREATE PROCEDURE CalculateTax(IN p_empno INT) BEGIN DECLARE v_name VARCHAR(100); DECLARE v_gross DECIMAL(10,2);DECLARE v_tax DECIMAL(10,2); -- Get employee name and gross salary SELECT e.emp_name, es.gross_salary INTO v_name, v_gross FROM Employee e JOIN Employee_salary es ON e.emp_no = es.emp_no WHERE e.emp_no = p_empno; -- Tax calculation IF v_gross <= 250000 THEN SET v_tax = 0; ELSEIF v_gross <= 500000 THEN SET v_tax = v_gross * 0.05; ELSEIF v_gross <= 1000000 THEN SET v_tax = v_gross * 0.20; ELSE SET v_tax = v_gross * 0.30;END IF;-- Output SELECT v_name AS Employee_Name, v_tax AS Tax_Amount;

END //

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 N_RollCall ( RollNo INT PRIMARY KEY,  StudentName VARCHAR(100), AttendanceDate DATE );
CREATE TABLE O_RollCall ( RollNo INT PRIMARY KEY,  StudentName VARCHAR(100),  AttendanceDate DATE);


CREATE PROCEDURE MergeRollCall() BEGIN DECLARE done INT DEFAULT 0; DECLARE v_RollNo INT; DECLARE v_StudentName VARCHAR(100);  DECLARE v_AttendanceDate DATE;  -- Cursor to iterate through N_RollCall  DECLARE cur CURSOR FOR   SELECT RollNo, StudentName, AttendanceDate FROM N_RollCall; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1; OPEN cur; read_loop: LOOP FETCH cur INTO v_RollNo, v_StudentName, v_AttendanceDate; IF done THEN  LEAVE read_loop; END IF;   -- Check if RollNo already exists in O_RollCall  IF NOT EXISTS (SELECT 1 FROM O_RollCall WHERE RollNo = v_RollNo) THEN  INSERT INTO O_RollCall (RollNo, StudentName, AttendanceDate) VALUES (v_RollNo, v_StudentName, v_AttendanceDate);  END IF; END LOOP; CLOSE cur;  SELECT 'Merge operation completed.' AS status; END //
```

29 Writ a PL/SQL procedure to find the number of students ranging from 100-70%, 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_course ( roll_no INT,  course VARCHAR(50),  course_cod VARCHAR(10),sem INT,total_marks INT,  percentage DECIMAL(5,2));

CREATE PROCEDURE Count_Student_Per_Percentage(IN p_course VARCHAR(50)) BEGIN SELECT   course,  SUM(CASE  WHEN percentage BETWEEN 70 AND 100 THEN 1    ELSE 0   END) AS '70-100%', SUM(CASE   WHEN percentage BETWEEN 60 AND 69 THEN 1  ELSE 0  END) AS '60-69%',  SUM(CASE WHEN percentage BETWEEN 50 AND 59 THEN 1    ELSE 0  END) AS '50-59%', SUM(CASE    WHEN percentage < 50 THEN 1     ELSE 0   END) AS 'Below 50%' FROM student_course WHERE course = p_course GROUP BY course; END //

CALL Count_Student_Per_Percentage('Computer Science');
```

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 PROCEDURE proc_Grade()BEGIN  DECLARE done INT DEFAULT 0;  DECLARE v_name VARCHAR(50);  DECLARE v_total INT;  DECLARE v_class VARCHAR(30);  -- Cursor for iterating over student marks  DECLARE stud_cursor CURSOR FOR  SELECT name, total_marks FROM Stud_Marks; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1; OPEN stud_cursor;  read_loop: LOOP    FETCH stud_cursor INTO v_name, v_total;  IF done THEN  LEAVE read_loop; END IF; -- Categorization Logic    IF v_total BETWEEN 990 AND 1500 THEN  SET v_class = 'Distinction'; ELSEIF v_total BETWEEN 900 AND 989 THEN   SET v_class = 'First Class';  ELSEIF v_total BETWEEN 825 AND 899 THEN   SET v_class = 'Higher Second Class'; ELSE    SET v_class = 'Fail/No Class';  END IF;   -- Insert into Result table  INSERT INTO Result(name, class) VALUES(v_name, v_class); END LOOP;  CLOSE stud_cursor; END //

CALL proc_Grade();
```

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.

```
db.Citydetails.insertMany([
  { _id: 1,  name: "Pune",  area: 450, population: {  total: 5000000,  adults: 3500000,   seniorcitizens: 600000, sexratio: 920  }, geography: { avgtemp: 28,     avgrainfall: 900,  longitude: 73.8567, latitude: 18.5204  }},
db.Citydetails.find(  { name: "Pune" }, { _id: 0, name: 1, "population.total": 1 });
db.Citydetails.find(  { "population.total": { $gt: 10000000 } }, { _id: 0, name: 1, "population.total": 1 });
db.Citydetails.aggregate([{  $project: {  name: 1,average_population: { $avg: ["$population.total", "$population.adults",   "$population.seniorcitizens" ]  }   } }]);
db.Citydetails.aggregate([{    $sort: { "population.total": 1 } },{   $group: {    _id: null,   minCity: { $first: "$name" },     minPopulation: { $first: "$population.total" },      maxCity: { $last: "$name" },  maxPopulation: { $last: "$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.

```
db.Citydetails.aggregate([  {   $project: {_id: 0,  name: 1,     area: 1,total_population: "$population.total" } }, {   $sort: { area: 1 }  }]);
db.Citydetails.find( { "geography.avgrainfall": { $gt: 60 } },{ _id: 0, name: 1, area: 1 });
db.Citydetails.createIndex({ name: 1, area: 1 });  db.Citydetails.find( { name: "Mumbai" }, { _id: 0, name: 1, "population.total": 1 }).sort({ "population.total": -1 }).limit(1);
db.Citydetails.createIndex({ name: 1 });
```