1

For the given Library database

BOOK (Book_ISBN [PK], Title[Not Null], Publisher_ Name, price[Check Price>0], Date_Of_Publication,Book_Copy),

BOOK_AUTHORS (Book_ISBN [PK,FK]Author_Name [PK], Author_City)

a) Create tables BOOK & BOOK_AUTHORS with all constraints.(As per Library Database)

```
CREATE TABLE BOOK (
  Book_ISBN INT PRIMARY KEY,
  Title VARCHAR(255) NOT NULL,
  Publisher_Name VARCHAR(255),
  Price DECIMAL(10, 2) CHECK (Price > 0),
  Date_Of_Publication DATE,
  Book_Copy INT
);

CREATE TABLE BOOK_AUTHORS (
  Book_ISBN INT,
  Author_Name VARCHAR(255),
  Author_City VARCHAR(255),
  PRIMARY KEY (Book_ISBN, Author_Name),
  FOREIGN KEY (Book_ISBN) REFERENCES BOOK(Book_ISBN)
);
```

b) Create view BOOK_AUTHOR_INFO consisting Book_ISBN, Title from BOOK Table and Author_Name from BOOK_AUTHORS Table in ascending order of ISBN no.

```
CREATE VIEW BOOK_AUTHOR_INFO AS

SELECT B.Book_ISBN, B.Title, BA.Author_Name

FROM BOOK B

JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN

ORDER BY B.Book_ISBN ASC;
```

c) Create an index on Book_Author on table on attribute “Author_Name”.

```
CREATE INDEX idx_Book_Author ON BOOK_AUTHORS (Author_Name);
```

d) Create table Book_Auto_Increment (BookID int Auto_increament=100, Book Name) insert five records in table.

```
CREATE TABLE Book_Auto_Increment (
  BookID INT AUTO_INCREMENT = 100,
  Book_Name VARCHAR(255),
  PRIMARY KEY (BookID)
);

INSERT INTO Book_Auto_Increment (Book_Name) VALUES ('Book 1');
INSERT INTO Book_Auto_Increment (Book_Name) VALUES ('Book 2');
INSERT INTO Book_Auto_Increment (Book_Name) VALUES ('Book 3');
INSERT INTO Book_Auto_Increment (Book_Name) VALUES ('Book 4');
INSERT INTO Book_Auto_Increment (Book_Name) VALUES ('Book 5');
```

e) Create Synonyms for your Database (E.g. Library Database as LibDB) and view all

```
CREATE SYNONYM LibDB FOR YourDatabaseName; -- Replace YourDatabaseName with your actual database name.

-- View all synonyms:
SELECT * FROM INFORMATION_SCHEMA.SYNONYMS WHERE TABLE_SCHEMA = 'YourDatabaseName'; -- Replace YourDatabaseName with your actual database name.
```

Note: Synonyms are not available in MySQL. The above syntax is provided based on the requirement, but it may not be executable in MySQL. Synonyms are commonly used in other database systems like Oracle

2

For the given Library database

BOOK (Book_ISBN [PK], Title[Not Null], Publisher_ Name, price[Check Price>0], Date_Of_Publication,Book_Copy),

BOOK_AUTHORS (Book_ISBN [PK,FK]Author_Name [PK], Author_City)

i) Insert at least five records in each table.

```
-- Inserting records in the BOOK table
INSERT INTO BOOK (Book_ISBN, Title, Publisher_Name, Price, Date_Of_Publication, Book_Copy)
VALUES (1, 'Book 1', 'Publisher 1', 19.99, '2022-01-01', 5);

INSERT INTO BOOK (Book_ISBN, Title, Publisher_Name, Price, Date_Of_Publication, Book_Copy)
VALUES (2, 'Book 2', 'Publisher 2', 24.99, '2022-02-01', 8);

INSERT INTO BOOK (Book_ISBN, Title, Publisher_Name, Price, Date_Of_Publication, Book_Copy)
VALUES (3, 'Book 3', 'Publisher 1', 14.99, '2022-03-01', 12);

INSERT INTO BOOK (Book_ISBN, Title, Publisher_Name, Price, Date_Of_Publication, Book_Copy)
VALUES (4, 'Book 4', 'Publisher 3', 29.99, '2022-04-01', 15);

INSERT INTO BOOK (Book_ISBN, Title, Publisher_Name, Price, Date_Of_Publication, Book_Copy)
VALUES (5, 'Book 5', 'Publisher 2', 9.99, '2022-05-01', 20);

-- Inserting records in the BOOK_AUTHORS table
INSERT INTO BOOK_AUTHORS (Book_ISBN, Author_Name, Author_City)
VALUES (1, 'Author 1', 'Pune');

INSERT INTO BOOK_AUTHORS (Book_ISBN, Author_Name, Author_City)
VALUES (2, 'Author 2', 'Mumbai');

INSERT INTO BOOK_AUTHORS (Book_ISBN, Author_Name, Author_City)
```

```
VALUES (3, 'Author 3', 'Pune');

INSERT INTO BOOK_AUTHORS (Book_ISBN, Author_Name, Author_City)
VALUES (4, 'Author 1', 'Delhi');

INSERT INTO BOOK_AUTHORS (Book_ISBN, Author_Name, Author_City)
VALUES (5, 'Author 2', 'Pune');
```

ii) Select details of Books whose Author lives in "Pune".

```
SELECT B.Book_ISBN, B.Title, B.Publisher_Name, B.Price, B.Date_Of_Publication, B.Book_Copy
FROM BOOK B
JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN
WHERE BA.Author_City = 'Pune';
```

iii) Select Book Names from table Book whose copies are in between 10 to 15.

```
SELECT Title
FROM BOOK
WHERE Book_Copy BETWEEN 10 AND 15;
```

iv) Update Book Copies as "10" whose Book Publisher is "Tata MacGraw Hill".

```
UPDATE BOOK
SET Book_Copy = 10
WHERE Publisher_Name = 'Tata MacGraw Hill';
```

v) Select the Name of Publisher who supplied maximum books.

```
SELECT Publisher_Name
FROM BOOK
GROUP BY Publisher_Name
HAVING COUNT(*) = (
  SELECT MAX(Book_Count)
```

```
  FROM (
    SELECT COUNT(*) AS Book_Count
    FROM BOOK
    GROUP BY Publisher_Name
  ) AS Book_Counts
);
```

3

For the given Library database

BOOK (Book_ISBN [PK], Title[Not Null], Publisher_ Name, price[Check Price>0], Date_Of_Publication,Book_Copy),

BOOK_AUTHORS (Book_ISBN [PK,FK]Author_Name [PK], Author_City)

i) Display name of publishers as per no of books published by them in ascending order.

```
SELECT Publisher_Name, COUNT(*) AS Book_Count
FROM BOOK
GROUP BY Publisher_Name
ORDER BY Book_Count ASC;
```

ii) Get publisher names who published at least one book written by author name like 'K%'.

```
SELECT DISTINCT Publisher_Name
FROM BOOK
WHERE Book_ISBN IN (
  SELECT Book_ISBN
  FROM BOOK_AUTHORS
  WHERE Author_Name LIKE 'K%'
);
```

iii) Get book name and Authors names where book written by maximum authors.

```
SELECT B.Title, BA.Author_Name
FROM BOOK B
JOIN (
  SELECT Book_ISBN, COUNT(*) AS Author_Count
  FROM BOOK_AUTHORS
  GROUP BY Book_ISBN
  HAVING Author_Count = (
```

```
   SELECT MAX(Author_Count)
   FROM (
    SELECT COUNT(*) AS Author_Count
    FROM BOOK_AUTHORS
    GROUP BY Book_ISBN
   ) AS Author_Counts
 )
) AS Max_Authors ON B.Book_ISBN = Max_Authors.Book_ISBN
JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN;
```

iv) Get publisher names accordingly books published alphabetically

```
SELECT Publisher_Name
FROM BOOK
ORDER BY Title ASC;
```

v) Find the no of books published in 01 Jan 2014 to till date.

```
SELECT COUNT(*) AS Total_Books
FROM BOOK
WHERE Date_Of_Publication >= '2014-01-01' AND Date_Of_Publication <= CURDATE();
```

vi) Delete the book from Book table written by Author ‘Korth’.

```
DELETE FROM BOOK
WHERE Book_ISBN IN (
 SELECT Book_ISBN
 FROM BOOK_AUTHORS
 WHERE Author_Name = 'Korth'
);
```

4

Aim: Design at 10 SQL queries for suitable database applications using SQL DML statements: all types of join, sub-query and view.

For the given Library database

BOOK (Book_ISBN [PK], Title[Not Null], Publisher_ Name, price[Check Price>0], Date_Of_Publication,Book_Copy),

BOOK_AUTHORS (Book_ISBN [PK]Author_Name [PK], Author_City)

a) Select Book_ISBN, Title, Author_Name from relations Book and Book_Authors INNER JOIN on attribute Book_ISBN.

```
SELECT B.Book_ISBN, B.Title, BA.Author_Name
FROM BOOK B
INNER JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN;
```

b) Select Book_ISBN, Title, Publisher, Author_Name from relations Book and Book_Authors LEFT OUTER JOIN on attribute Book_ISBN.

```
SELECT B.Book_ISBN, B.Title, B.Publisher_Name, BA.Author_Name
FROM BOOK B
LEFT JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN;
```

c) Select Book_ISBN, Title, Publisher, Author_Name from relations Book and Book_Authors RIGHT OUTER JOIN on attribute Book_ISBN.

```
SELECT B.Book_ISBN, B.Title, B.Publisher_Name, BA.Author_Name
FROM BOOK B
RIGHT JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN;
```

d)Select Book_ISBN, Title from relation Book whose author is living in City =’Pune’

```
SELECT B.Book_ISBN, B.Title
FROM BOOK B
INNER JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN
WHERE BA.Author_City = 'Pune';
```

5

Write a PL/SQL block of code for the following requirements:-

Schema:

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

Fine(Roll_no, Date, Amt)

a. Accept roll_no & name of book from user.

b. Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.

c. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.

d. After submitting the book, status will change from I to R.

e. If condition of fine is true, then details will be stored into fine table.

```
DECLARE
 v_roll_no  Borrower.Rollin%TYPE;
 v_book_name  Borrower.NameofBook%TYPE;
 v_date_of_issue  Borrower.DateofIssue%TYPE;
 v_status  Borrower.Status%TYPE;
 v_fine_amt  Fine.Amt%TYPE;
 v_fine_days  NUMBER;
BEGIN
 -- Step 1: Accept roll_no and book_name from the user
 v_roll_no := &enter_roll_no;
 v_book_name := '&enter_book_name';

 -- Step 2: Retrieve the borrower details
 SELECT DateofIssue, Status INTO v_date_of_issue, v_status
 FROM Borrower
 WHERE Rollin = v_roll_no AND NameofBook = v_book_name;

 -- Step 3: Calculate the number of fine days
```

```
v_fine_days := TRUNC(SYSDATE) - v_date_of_issue;

-- Step 4: Check the fine conditions and calculate the fine amount
IF v_fine_days > 30 THEN
  v_fine_amt := v_fine_days * 50;
ELSE
  v_fine_amt := v_fine_days * 5;
END IF;

-- Step 5: Update the status of the book to 'R' (Returned)
UPDATE Borrower
SET Status = 'R'
WHERE Rollin = v_roll_no AND NameofBook = v_book_name;

-- Step 6: Check if fine is applicable and insert the details into the fine table
IF v_fine_days > 15 THEN
  INSERT INTO Fine (Roll_no, Date, Amt)
  VALUES (v_roll_no, SYSDATE, v_fine_amt);
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Fine of Rs ' || v_fine_amt || ' has been applied.');
ELSE
  DBMS_OUTPUT.PUT_LINE('No fine is applicable.');
END IF;

-- Step 7: Display the updated borrower details
SELECT Rollin, Name, DateofIssue, NameofBook, Status
INTO v_roll_no, v_book_name, v_date_of_issue, v_book_name, v_status
FROM Borrower
WHERE Rollin = v_roll_no AND NameofBook = v_book_name;

DBMS_OUTPUT.PUT_LINE('Roll No: ' || v_roll_no);
```

```
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_book_name);
  DBMS_OUTPUT.PUT_LINE('Date of Issue: ' || v_date_of_issue);
  DBMS_OUTPUT.PUT_LINE('Name of Book: ' || v_book_name);
  DBMS_OUTPUT.PUT_LINE('Status: ' || v_status);
END;
/
```

6

Write a PL/SQL block of code using parameterized Cursor

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.

```
DECLARE
  CURSOR c_rollcall (p_roll_no N_RollCall.Roll_No%TYPE) IS
    SELECT *
    FROM N_RollCall
    WHERE Roll_No = p_roll_no;

  v_roll_no N_RollCall.Roll_No%TYPE;
  v_name N_RollCall.Name%TYPE;
  v_status N_RollCall.Status%TYPE;
BEGIN
  OPEN c_rollcall(123); -- Provide the desired roll_no

  LOOP
    FETCH c_rollcall INTO v_roll_no, v_name, v_status;
    EXIT WHEN c_rollcall%NOTFOUND;

    -- Check if the data already exists in the O_RollCall table
    SELECT COUNT(*)
    INTO v_count
    FROM O_RollCall
    WHERE Roll_No = v_roll_no;

    -- If the data does not exist, merge it into the O_RollCall table
    IF v_count = 0 THEN
      INSERT INTO O_RollCall (Roll_No, Name, Status)
```

```
      VALUES (v_roll_no, v_name, v_status);
    END IF;
  END LOOP;

  CLOSE c_rollcall;
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Data merged successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
/
```

7

Write a PL/SQL block of Stored Procedure and Stored Function proc_Grade for following problem statement.

Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 category is first class, if marks 899 and 825 category is Higher Second Class.

```
CREATE OR REPLACE PROCEDURE proc_Grade (
  p_marks IN NUMBER,
  p_category OUT VARCHAR2
)
IS
BEGIN
  IF p_marks <= 1500 AND p_marks >= 990 THEN
    p_category := 'Distinction';
  ELSIF p_marks >= 900 AND p_marks <= 989 THEN
    p_category := 'First Class';
  ELSIF p_marks >= 825 AND p_marks <= 899 THEN
    p_category := 'Higher Second Class';
  ELSE
    p_category := 'No Category';
  END IF;
END;
/
```

To use and execute the procedure

```
DECLARE
  v_marks NUMBER := 950;
  v_category VARCHAR2(20);
BEGIN
  proc_Grade(v_marks, v_category);
```

```
  DBMS_OUTPUT.PUT_LINE('Category: ' || v_category);
END;
/
```

8

Aim: Design and develop MongoDB queries using CRUD operations.

Problem Statement: Design and Develop database using MongoDB Client- MongoDB Data sever to Implement all basic CRUD operations and administration commands using two tier architecture. (Command Prompt)

a. Create a collection named Book. Add 5 documents in the collection with keys (book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline, mobile]), publisher_city, price,copies)

```
use YourDatabaseName

db.Book.insertMany([
  {
    "book_isbn": "ISBN001",
    "title": "Book 1",
    "publisher_name": "Publisher 1",
    "author": {
      "name": "Author 1",
      "address": "Address 1",
      "phone_no": {
        "landline": "1234567890",
        "mobile": "9876543210"
      }
    },
    "publisher_city": "Pune",
    "price": 9.99,
    "copies": 10
  },
  {
    "book_isbn": "ISBN002",
    "title": "Book 2",
    "publisher_name": "Publisher 2",
    "author": {
      "name": "Author 2",
```

```
      "address": "Address 2",
      "phone_no": {
        "landline": "1234567890",
        "mobile": "9876543210"
      }
    },
    "publisher_city": "Mumbai",
    "price": 14.99,
    "copies": 5
  },
  // Add three more documents following the same structure
]);
```

b. Give details of Books whose Publisher lives in “Pune”.

```
db.Book.find({ "publisher_city": "Pune" });
```

c. Delete name Book from Book whose name start with “D”

```
db.Book.deleteMany({ "title": /^D/ });
```

d. Change the city of publisher “Pearson” to “Pune”.

```
db.Book.updateMany({ "publisher_name": "Pearson" }, { $set: { "publisher_city": "Pune" } });
```

e. Find the details of publisher named “Pearson”.

```
db.Book.find({ "publisher_name": "Pearson" });
```

9

Design and Implement any 5 query using MongoDB.

Problem Statement: With reference to Book collection in Assignment B-1 write MongoDB Queries for

(book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline, mobile]), publisher_city, price,copies

a. Count the number of documents in the collection.

```
db.Book.countDocuments();
```

b. Arrange the documents in descending order of book_isbn.

```
db.Book.find().sort({ "book_isbn": -1 });
```

c. Select Book Names whose title is ”DBMS” .

```
db.Book.find({ "title": "DBMS" }, { "title": 1 });
```

d. Update Book Copies as “10” whose Book Publisher is “Tata MacGraw Hill”.

```
db.Book.updateMany({ "publisher_name": "Tata MacGraw Hill" }, { $set: { "copies": 10 } });
```

10

Aim: Design and Implement any 5 query using MongoDB.

Problem Statement: With reference to Book collection in Assignment B-1 write MongoDB Queries for

(book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline, mobile]), publisher_city, price,copies

a. Update Book Copies as "10" whose Book Publisher is "Tata MacGraw Hill".

```
db.Book.updateMany({ "publisher_name": "Tata MacGraw Hill" }, { $set: { "copies": 10 } });
```

b. Display name of publishers as per no of books published by them in ascending order.

```
db.Book.aggregate([
 { $group: { _id: "$publisher_name", count: { $sum: 1 } } },
 { $sort: { count: 1 } },
 { $project: { _id: 0, publisher_name: "$_id", count: 1 } }
]);
```

c. Get publisher names who published at least one book written by author name like 'K%'.

```
db.Book.distinct("publisher_name", { "author.name": /^K/ });
```

d. Delete the book from Book table written by Author 'Korth'.

```
db.Book.deleteMany({ "author.name": "Korth" });
```

11

Aim: Implement aggregation and indexing with suitable example using MongoDB

Problem statement: Create a collection named "ORDERS" that contain documents of the following prototype and solve the following queries:

```
{
 cust_id: "abc123",
 ord_date: new Date("Oct 04, 2012"),
 status: 'A',
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
        { sku: "yyy", qty: 25, price: 1 } ]
     }
```

a. Count all records from orders

```
db.ORDERS.count();
```

b. Sum the price field from orders

```
db.ORDERS.aggregate([
 { $group: { _id: null, totalPrice: { $sum: "$price" } } }
]);
```

c. For each unique cust_id, sum the price field.

```
db.ORDERS.aggregate([
 { $group: { _id: "$cust_id", totalPrice: { $sum: "$price" } } }
]);
```

d. For each unique cust_id, sum the price field, results sorted by sum

```
db.ORDERS.aggregate([
 { $group: { _id: "$cust_id", totalPrice: { $sum: "$price" } } },
 { $sort: { totalPrice: 1 } }
]);
```

12

Aim: Implement aggregation and indexing with suitable example using MongoDB

Problem statement: Create a collection named "ORDERS" that contain documents of the following prototype and solve the following queries:

```
{
 cust_id: "abc123",
 ord_date: new Date("Oct 04, 2012"),
 status: 'A',
 price: 50,
 items: [ { sku: "xxx", qty: 25, price: 1 },
        { sku: "yyy", qty: 25, price: 1 } ]
}
```

1.For each unique cust_id, sum the price field, results sorted by sum.

```
db.ORDERS.aggregate([
 { $group: { _id: "$cust_id", totalPrice: { $sum: "$price" } } },
 { $sort: { totalPrice: 1 } }
]);
```

2.For each unique cust_id, ord_date grouping, sum the price field.

```
db.ORDERS.aggregate([
 { $group: { _id: { cust_id: "$cust_id", ord_date: "$ord_date" }, totalPrice: { $sum: "$price" } } }
]);
```

3.For cust_id with multiple records, return the cust_id and the corresponding record count.

```
db.ORDERS.aggregate([
 { $group: { _id: "$cust_id", count: { $sum: 1 } } },
 { $match: { count: { $gt: 1 } } },
 { $project: { _id: 1, count: 1 } }
]);
```

4.For each unique cust_id with status A, sum the price field.

```
db.ORDERS.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", totalPrice: { $sum: "$price" } } }
]);
```

13

Write a PL/SQL program to check whether a date falls on weekend i.e. SATURDAY or SUNDAY.

```
DECLARE
  v_date DATE := TO_DATE('2023-06-04', 'YYYY-MM-DD'); -- Replace with the desired date

  v_day VARCHAR2(20);
BEGIN
  SELECT TO_CHAR(v_date, 'Day') INTO v_day FROM DUAL;

  IF v_day = 'Saturday' OR v_day = 'Sunday' THEN
    DBMS_OUTPUT.PUT_LINE('The date ' || v_date || ' falls on a weekend.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('The date ' || v_date || ' does not fall on a weekend.');
  END IF;
END;
/
```

14

Consider Dept table

DEPTNO	DNAME	LOC

1. Rename the table dept as department

ALTER TABLE dept RENAME TO department;

2. Add a new column PINCODE with not null constraints to the existing table DEPT

ALTER TABLE department ADD (PINCODE NUMBER NOT NULL);

4. Rename the column DNAME to DEPT_NAME in dept table

ALTER TABLE department RENAME COLUMN DNAME TO DEPT_NAME;

5. Change the data type of column loc as CHAR with size 10

ALTER TABLE department MODIFY (LOC CHAR(10));

6. Delete table

DROP TABLE department;

15

Consider Employee table

EMPNO	EMP_NAME	DEPT	SALARY	DOJ	BRANCH
E101	Amit	oduction	45000	12-Mar-00	Bangalore
E102	Amit	HR	70000	03-Jul-02	Bangalore
E103	sunita	anagemer	120000	11-Jan-01	mysore
E105	sunita	IT	67000	01-Aug-01	mysore
E106	mahesh	Civil	145000	20-Sep-03	Mumbai

Perform the following

1. Display all the fields of employee table

SELECT * FROM Employee;

2. Retrieve employee number and their salary

SELECT EmployeeNumber, Salary FROM Employee;

3. Retrieve average salary of all employee

SELECT AVG(Salary) AS AverageSalary FROM Employee;

4. Retrieve number of employee

SELECT COUNT(*) AS NumberOfEmployees FROM Employee;

5. Retrieve distinct number of employee

SELECT COUNT(DISTINCT EmployeeNumber) AS NumberOfDistinctEmployees FROM Employee;

6. Retrieve total salary of employee group by employee name and count similar names

SELECT EmployeeName, SUM(Salary) AS TotalSalary, COUNT(EmployeeName) AS NameCount

FROM Employee

GROUP BY EmployeeName;

7. Retrieve total salary of employee which is greater than >120000

```
SELECT SUM(Salary) AS TotalSalary
FROM Employee
WHERE Salary > 120000;
```

8. Display name of employee in descending order

```
SELECT EmployeeName FROM Employee ORDER BY EmployeeName DESC;
```

9. Display details of employee whose name is AMIT and salary greater than 50000;

```
SELECT * FROM Employee WHERE EmployeeName = 'AMIT' AND Salary > 50000;
```

16

Write a Pl/SQL program to print integers from 1 to 10 by using PL/SQL FOR loop

```
DECLARE
 i NUMBER;
BEGIN
 FOR i IN 1..10 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
 END LOOP;
END;
/
```

17

Given the table EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID) write a cursor to select the five highest paid employees from the table.

EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)

```
DECLARE
  CURSOR highest_paid_cur IS
    SELECT EmpNo, Name, Salary, Designation, DeptID
    FROM EMPLOYEE
    ORDER BY Salary DESC
    FETCH FIRST 5 ROWS ONLY;

  emp_record EMPLOYEE%ROWTYPE;
BEGIN
  OPEN highest_paid_cur;
  LOOP
    FETCH highest_paid_cur INTO emp_record;
    EXIT WHEN highest_paid_cur%NOTFOUND;
    -- Process the fetched employee record here
    DBMS_OUTPUT.PUT_LINE(emp_record.EmpNo || ', ' || emp_record.Name || ', ' ||
emp_record.Salary);
  END LOOP;
  CLOSE highest_paid_cur;
END;
/
```

18

Consider the schema for Company Database:

EMPLOYEE(SSN, Name, Address, Sex, Salary, SuperSSN, DNo)

DEPARTMENT(DNo, DName, MgrSSN, MgrStartDate)

DLOCATION(DNo,DLoc)

PROJECT(PNo, PName, PLocation, DNo)

WORKS_ON(SSN, PNo, Hours)

1.For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

```
SELECT DNo, COUNT(*) AS NumEmployees
FROM EMPLOYEE
WHERE Salary > 600000
GROUP BY DNo
HAVING COUNT(*) > 5;
```

2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.

```
UPDATE EMPLOYEE
SET Salary = Salary * 1.1
WHERE SSN IN (
  SELECT SSN
  FROM WORKS_ON
  WHERE PNo = (SELECT PNo FROM PROJECT WHERE PName = 'IoT')
);
```

19

Consider the schema for Company Database:

EMPLOYEE(SSN, Name, Address, Sex, Salary, SuperSSN, DNo)

DEPARTMENT(DNo, DName, MgrSSN, MgrStartDate)

DLOCATION(DNo,DLoc)

PROJECT(PNo, PName, PLocation, DNo)

WORKS_ON(SSN, PNo, Hours)

1. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
SELECT SUM(Salary) AS TotalSalary, MAX(Salary) AS MaxSalary, MIN(Salary) AS MinSalary,
AVG(Salary) AS AvgSalary

FROM EMPLOYEE

WHERE DNo = (SELECT DNo FROM DEPARTMENT WHERE DName = 'Accounts');
```

2. Retrieve the name of each employee who works on all the projects controlled by department number 5

```
SELECT E.Name

FROM EMPLOYEE E

WHERE NOT EXISTS (

 SELECT PNo

 FROM PROJECT

 WHERE DNo = 5

 MINUS

 SELECT PNo

 FROM WORKS_ON W

 WHERE E.SSN = W.SSN

);
```

20

Consider the schema for Movie Database:

ACTOR(Act_id, Act_Name, Act_Gender)

DIRECTOR(Dir_id, Dir_Name, Dir_Phone)

MOVIES(Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)

MOVIE_CAST(Act_id, Mov_id, Role)

RATING(Mov_id, Rev_Stars)

Write Mongodb queries to

1. List the titles of all movies directed by 'Hitchcock'.

```
db.MOVIES.find({ "Dir_id": { $in: db.DIRECTOR.find({ "Dir_Name": "Hitchcock" }, { "Dir_id": 1 }) } }, { "Mov_Title": 1 })
```

2. Find the movie names where one or more actors acted in two or more movies.

```
db.MOVIE_CAST.aggregate([
 { $group: { _id: "$Act_id", count: { $sum: 1 } } },
 { $match: { count: { $gte: 2 } } },
 { $lookup: { from: "MOVIES", localField: "Mov_id", foreignField: "Mov_id", as: "movies" } },
 { $project: { _id: 0, "movies.Mov_Title": 1 } }
])
```

3. List all actors who acted in a movie before 2000 and in a movie after 2015

```
db.MOVIE_CAST.aggregate([
 { $lookup: { from: "MOVIES", localField: "Mov_id", foreignField: "Mov_id", as: "movies" } },
 {
   $project: {
     Act_id: 1,
     movies: {
       $filter: {
         input: "$movies",
```

```
        as: "movie",
        cond: {
          $or: [
            { $lt: ["$$movie.Mov_Year", 2000] },
            { $gt: ["$$movie.Mov_Year", 2015] }
          ]
        }
       }
      }
    }
  },
  { $match: { movies: { $ne: [] } } },
  { $project: { _id: 0, Act_id: 1 } }
])
```