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)
Solve the following

- **a) 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.**
- **b) Create an index on Book_Author on table on attribute "Author_Name".**
- **c) Create table Book_Auto_Increment (BookID int Auto_increament=100, Book Name) insert five records in table.**
- **d) Delete the book from Book table written by Author 'Korth'.**
- **e) Select Book Names from table Book whose copies are in between 10 to 15.**

```
CREATE DATABASE Library;

 use Library;

Database changed

CREATE TABLE BOOK(

   Book_ISBN VARCHAR(13) PRIMARY KEY,

   Title VARCHAR(100) NOT NULL,

   Publisher_Name VARCHAR(200),

   Price DECIMAL(10,2) CHECK (Price > 0),

   Date_Of_Publication DATE,

   Book_Copy INT

    );


 CREATE TABLE BOOK_AUTHOR(

    Book_ISBN VARCHAR(13),
  Author_Name VARCHAR(20),

  Author_City VARCHAR(20),

  PRIMARY KEY(Book_ISBN, Author_Name),

  FOREIGN KEY(Book_ISBN) REFERENCES BOOK(Book_ISBN)

);
```

a) 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, A.Author_Name

FROM BOOK B

JOIN BOOK_AUTHORS A ON B.Book_ISBN = A.Book_ISBN

ORDER BY B.Book_ISBN ASC;
```

b) Create an index on Book_Author on table on attribute "Author_Name".

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

c) 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 PRIMARY KEY,
   Book_Name VARCHAR(255)
) AUTO_INCREMENT = 100;
```

Insert Values

```
INSERT INTO Book_Auto_Increment(Book_Name)VALUES('Clean Code'),(Introduction
to Algorithm'),('Database System Concept'),('Design Pattern'),('The Pragmatic
Programmer');

mysql> SELECT *FROM Book_Auto_Increment;
```

```
+--------+--------------------------------------+
| BookID | Book_Name                            |
+--------+--------------------------------------+
|    100 | Clean Code),(Introduction to Algorithm |
|    101 | Database System Concept              |
|    102 | Design Pattern                       |
|    103 | The Pragmatic Programmer             |
+--------+--------------------------------------+
```

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

```
INSERT INTO BOOK (Book_ISBN, Title, Publisher_Name, Price, Date_Of_Publication, Book_Copy) VALUES

('9780073523323', 'Database System Concepts', 'McGraw-Hill', 55.99, '2019-04-15', 12),

('9780262033848', 'Introduction to Algorithms', 'MIT Press', 75.50, '2009-07-31', 15),

('9780201616224', 'The Pragmatic Programmer', 'Addison-Wesley', 42.99, '1999-10-30', 10),

('9780132350884', 'Clean Code', 'Prentice Hall', 39.99, '2008-08-01', 8),

('9780201633610', 'Design Patterns: Elements of Reusable Object-Oriented Software', 'Addison-Wesley', 49.99, '1994-10-31', 14);

INSERT INTO BOOK_AUTHOR (Book_ISBN, Author_Name, Author_City) VALUES ('9780131873254', 'Henry F. Korth', 'USA');
```

DELETE

```
DELETE FROM BOOK

WHERE Book_ISBN IN (

   SELECT Book_ISBN FROM BOOK_AUTHOR  WHERE Author_Name = 'Korth'

);
```

e) 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;
```

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)
Solve the following :

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

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

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

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

e) **Select Book_ISBN, Title from relation Book, which written by more than 2**

Authors.

a) **Select Book_ISBN, Title, and Author_Name with an INNER JOIN on Book_ISBN**

This query joins the BOOK and BOOK_AUTHORS tables on Book_ISBN and selects Book_ISBN, Title, and Author_Name.

```
SELECT B.Book_ISBN, B.Title, A.Author_Name

FROM BOOK B

INNER JOIN BOOK_AUTHOR A ON B.Book_ISBN = A.Book_ISBN;
```

b) **Select Book_ISBN, Title, Publisher_Name, and Author_Name with a LEFT OUTER JOIN on Book_ISBN**

This query retrieves all records from BOOK and matching records from BOOK_AUTHORS. If there's no match in BOOK_AUTHORS, Author_Name will be NULL.

```
SELECT B.Book_ISBN, B.Title, B.Publisher_Name, A.Author_Name
```

```
FROM BOOK B
LEFT OUTER JOIN BOOK_AUTHORS A 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.**

This query retrieves all records from BOOK_AUTHORS and matching records from BOOK. If there's no match in BOOK, fields from BOOK will be NULL.

```
SELECT B.Book_ISBN, B.Title, B.Publisher_Name, A.Author_Name

FROM BOOK B

RIGHT OUTER 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

JOIN BOOK_AUTHORS A ON B.Book_ISBN = A.Book_ISBN

WHERE A.Author_City = 'Pune';
```

e) **Select Book_ISBN, Title from relation Book, which written by more than 2 Authors.**

```
SELECT B.Book_ISBN, B.Title

FROM BOOK B

JOIN BOOK_AUTHORS A ON B.Book_ISBN = A.Book_ISBN

GROUP BY B.Book_ISBN, B.Title

HAVING COUNT(A.Author_Name) > 2;
```

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)
Solve the following

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

- **ii) Get publisher names who published at least one book written by author name like ‘K%’.**

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

- **iv) Get publisher names accordingly books published alphabetically**

- **v) Find the no of books published in 01 Jan 2014 to till date.**
- **vi) Display name of publishers as per no of books published by them in ascending order**

```
SELECT Publisher_Name, COUNT(*) AS Number_of_Books

FROM BOOK

GROUP BY Publisher_Name

ORDER BY Number_of_Books ASC;
```

- **vii) Get publisher names who published at least one book written by author name like ‘K%’.**

```
SELECT DISTINCT B.Publisher_Name

FROM BOOK B

JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN

WHERE BA.Author_Name LIKE 'K%';
```

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

```
SELECT B.Title, BA.Author_Name
FROM BOOK B
JOIN BOOK_AUTHORS BA ON B.Book_ISBN = BA.Book_ISBN
WHERE B.Book_ISBN = (
   SELECT Book_ISBN
   FROM BOOK_AUTHORS
   GROUP BY Book_ISBN
   ORDER BY COUNT(Author_Name) DESC
   LIMIT 1
);
```

ix) Get publisher names accordingly books published alphabetically

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

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

```
SELECT COUNT(*) AS Number_of_Books
FROM BOOK
WHERE Date_Of_Publication >= '2014-01-01';
```

4. **Consider insurance database with following schema :**

   person(driver-id, name, address)

   car(license, model, year)

   accident (report - no, date, location)

   owns(driver-id,license)

   participated(driver-id,car,report-no,damage-amount)

Write a query in SQL for following requirements :

i) Find the total no. of people who owned cars that were involved in accidents in 2016.

ii) Retrieve the name of person whose address contains Pune.

iii) Find the name of persons having more than two cars.

```
CREATE TABLE person (
    driver_id INT PRIMARY KEY,
    name VARCHAR(50),
    address VARCHAR(100)
);

CREATE TABLE car (
    license VARCHAR(20) PRIMARY KEY,
    model VARCHAR(50),
    year INT
);

CREATE TABLE accident (
    report_no INT PRIMARY KEY,
    date DATE,
    location VARCHAR(100)
);

CREATE TABLE owns (
    driver_id INT,
    license VARCHAR(20),
    FOREIGN KEY (driver_id) REFERENCES person(driver_id),
    FOREIGN KEY (license) REFERENCES car(license)
);

CREATE TABLE participated (
    driver_id INT,
    car VARCHAR(20),
    report_no INT,
    damage_amount DECIMAL(10, 2),
    FOREIGN KEY (driver_id) REFERENCES person(driver_id),
    FOREIGN KEY (car) REFERENCES car(license),
    FOREIGN KEY (report_no) REFERENCES accident(report_no)
);
```

Insert Data

```
INSERT INTO person (driver_id, name, address) VALUES
(1, 'Alice', '123 Main St, Pune'),
(2, 'Bob', '456 Elm St, Mumbai'),
(3, 'Charlie', '789 Oak St, Pune'),
(4, 'David', '101 Maple St, Delhi');

INSERT INTO car (license, model, year) VALUES
('ABC123', 'Toyota Corolla', 2015),
('XYZ456', 'Honda Civic', 2016),
('LMN789', 'Ford Focus', 2014),
('PQR678', 'Chevrolet Malibu', 2016),
('UVW123', 'Tesla Model 3', 2020);

-- Inserting data into accident table
INSERT INTO accident (report_no, date, location) VALUES
(1, '2016-03-15', 'Pune'),
(2, '2016-07-22', 'Mumbai'),
(3, '2015-11-05', 'Delhi');

-- Inserting data into owns table
INSERT INTO owns (driver_id, license) VALUES
(1, 'ABC123'),
(1, 'XYZ456'),
(2, 'LMN789'),
(3, 'PQR678'),
(3, 'UVW123');

-- Inserting data into participated table
INSERT INTO participated (driver_id, car, report_no, damage_amount) VALUES
(1, 'ABC123', 1, 5000),
(1, 'XYZ456', 2, 3000),
(2, 'LMN789', 2, 4500),
(3, 'PQR678', 1, 6000);
```

1) **Find the total number of people who owned cars that were involved in accidents in 2016**

```
SELECT COUNT(DISTINCT o.driver_id) AS Total_Owners
FROM owns o
JOIN participated p ON o.license = p.car
JOIN accident a ON p.report_no = a.report_no
WHERE YEAR(a.date) = 2016;
```

2) Retrieve the name of persons whose address contains "Pune"

```
SELECT name

FROM person

WHERE address LIKE '%Pune%';
```

3. Find the name of persons having more than two cars

```
SELECT p.name

FROM person p

JOIN owns o ON p.driver_id = o.driver_id

GROUP BY p.driver_id, p.name

HAVING COUNT(o.license) > 2;
```

5 Implement MySQL database connectivity with Java Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC.

```
import java.sql.*;

import java.util.Scanner;

public class MySql {

    public static void main(String[] args) throws
Exception {        int n, sno;
        String name, telephone, gender;
        Scanner in = new Scanner(System.in);

        // Establishing a connection
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/student", "root",
"Tanuja1006");
        Statement stmt = con.createStatement();

        // Prepared statements for inserting, deleting, and updating records
        PreparedStatement pstm = con.prepareStatement("INSERT INTO Personal
VALUES(?,?,?,?)");        PreparedStatement pstm1 = con.prepareStatement("DELETE
FROM Personal WHERE sno=?");        PreparedStatement pstm2 =
con.prepareStatement("UPDATE Personal SET name=?, age=?, gender=? WHERE
sno=?");

        // Inserting records
        System.out.print("Enter the number of records you want
to insert: ");        n = in.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("\nData " + (i + 1) +
"\nEnter Sno: ");            sno = in.nextInt();
pstm.setInt(1, sno);
            System.out.print("Enter Name: ");
            name = in.next();
pstm.setString(2, name);
System.out.print("Enter Age: ");
telephone = in.next();
pstm.setString(3, telephone);
System.out.print("Enter Gender: ");
```

```
            gender = in.next();
pstm.setString(4, gender);
            pstm.executeUpdate();
        }

        // Display records after insertion
        ResultSet rs = stmt.executeQuery("SELECT * FROM Personal;");
        System.out.println("\nAfter Insertion");

System.out.println("Sno\tName\tAge\tGender");
while (rs.next()) {
            System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t" + rs.getString(3) + "\t"
+ rs.getString(4));
        }

        // Updating records
        System.out.print("Enter the number of records you want
to update: ");        n = in.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("\nData " + (i + 1) + "\nEnter Sno to
update: ");           sno = in.nextInt();
            System.out.print("Enter New Name: ");
            name = in.next();
pstm2.setString(1, name);
System.out.print("Enter New Age:
");
            telephone = in.next();
pstm2.setString(2, telephone);
            System.out.print("Enter New Gender: ");
            gender = in.next();
pstm2.setString(3, gender);
pstm2.setInt(4, sno);
            pstm2.executeUpdate();
        }

        // Display records after update
        ResultSet rs2 = stmt.executeQuery("SELECT * FROM Personal;");
        System.out.println("\nAfter Update");
        System.out.println("Sno\tName\tAge\tGender");
        while (rs2.next()) {
            System.out.println(rs2.getInt(1) + "\t" + rs2.getString(2) + "\t" + rs2.getString(3) +
"\t" + rs2.getString(4));
        }

        // Deleting records
        System.out.print("Enter the number of records you want
to delete: ");        n = in.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("\nData " + (i + 1) +
"\nEnter Sno: ");            sno = in.nextInt();
pstm1.setInt(1, sno);
pstm1.executeUpdate();

        }
        // Display records after deletion
        ResultSet rs1 = stmt.executeQuery("SELECT * FROM Personal;");
        System.out.println("\nAfter Deletion");
        System.out.println("Sno\tName\tAge\tGender");

    while (rs1.next()) {

            System.out.println(rs1.getInt(1) + "\t" + rs1.getString(2) + "\t" + rs1.getString(3) +
"\t" + rs1.getString(4));
        }

        // Closing resources
con.close();
    }
}
```

6 For the given Employee database

EmployeeInfo(EmpID[PK],EmpFname,EmpLname,Department,Project,Address,DOB,Gender)

EmployeePosition(EmpID[FK],EmpPosition,DateOfJoining,Salary)

i. **Write a query to fetch the EmpFname from the EmployeeInfo table in the upper case and use the ALIAS name as EmpName.**

ii. **Write a query to fetch the number of employees working in the department 'HR'.**

iii. **Write q query to find all the employees whose salary is between 50000 to 100000**

iv. **Write a query to find the names of employees that begin with 'S'**

v. **Write a query to fetch top N records.**

```
CREATE TABLE EmployeeInfo (
   EmpID INT PRIMARY KEY,
   EmpFname VARCHAR(50),
   EmpLname VARCHAR(50),
   Department VARCHAR(50),
   Project VARCHAR(50),
   Address VARCHAR(100),
   DOB DATE,
   Gender CHAR(1)
);
CREATE TABLE EmployeePosition (
   EmpID INT,
   EmpPosition VARCHAR(50),
   DateOfJoining DATE,
Salary DECIMAL(10, 2),

FOREIGN KEY (EmpID) REFERENCES EmployeeInfo(EmpID));
```

i. **Write a query to fetch the EmpFname from the EmployeeInfo table in the upper case and use the ALIAS name as EmpName.**

```
SELECT UPPER(EmpFname) AS EmpName

FROM EmployeeInfo;
```

ii. **Write a query to fetch the number of employees working in the department ‘HR’.**

```
SELECT COUNT(*) AS HR_Employees

FROM EmployeeInfo

WHERE Department = 'HR';
```

iii. **Write q query to find all the employees whose salary is between 50000 to 100000**

```
SELECT EI.EmpID, EI.EmpFname, EI.EmpLname, EP.Salary

FROM EmployeeInfo EI

JOIN EmployeePosition EP ON EI.EmpID = EP.EmpID

WHERE EP.Salary BETWEEN 50000 AND 100000;
```

iv. **Write a query to find the names of employees that begin with ‘S’**

```
SELECT EmpFname

FROM EmployeeInfo

WHERE EmpFname LIKE 'S%';
```

v. Write a query to fetch top N records.

```
SELECT *

FROM EmployeeInfo

LIMIT 3;
```

7. **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.2.

```
DROP TABLE IF EXISTS N_RollCall;
DROP TABLE IF EXISTS O_RollCall;
```

STEP 1: Create table **N_RollCall**

```
CREATE TABLE N_RollCall (
   RollCallID      INT PRIMARY KEY,
   StudentID       INT NOT NULL,
   AttendanceDate  DATE NOT NULL
);
```

STEP 2: Create table **O_RollCall**

```
CREATE TABLE O_RollCall (
   RollCallID      INT PRIMARY KEY,
   StudentID       INT NOT NULL,
   AttendanceDate  DATE NOT NULL
);
```

STEP 3: Insert values into tables

```
INSERT INTO N_RollCall (RollCallID, StudentID, AttendanceDate) VALUES
(1, 1001, '2024-09-20'),
(2, 1002, '2024-09-20'),
(3, 1003, '2024-09-21');

DROP PROCEDURE IF EXISTS MergeRollCallData;
```

STEP 4: PL/SQL Code

```
DELIMITER //

CREATE PROCEDURE MergeRollCallData()
BEGIN
   -- Declare variables
   DECLARE v_RollCallID INT;
   DECLARE v_StudentID INT;
   DECLARE v_AttendanceDate DATE;
   DECLARE v_count INT;
   DECLARE done INT DEFAULT 0;

   -- Declare a cursor to select data from N_RollCall
```

```
  DECLARE c_RollCall CURSOR FOR
    SELECT RollCallID, StudentID, AttendanceDate
    FROM N_RollCall;

  -- Declare a CONTINUE HANDLER for NOT FOUND
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  -- Open the cursor
  OPEN c_RollCall;

  -- Loop through each row in the cursor
  FETCH_LOOP: LOOP
    FETCH c_RollCall INTO v_RollCallID, v_StudentID, v_AttendanceDate;

    IF done THEN
      LEAVE FETCH_LOOP;
    END IF;

    -- Check if the data already exists in O_RollCall
    SELECT COUNT(*)
    INTO v_count
    FROM O_RollCall
    WHERE RollCallID = v_RollCallID
     AND StudentID = v_StudentID
     AND AttendanceDate = v_AttendanceDate;

    -- If the data does not exist, insert it into O_RollCall
    IF v_count = 0 THEN
      INSERT INTO O_RollCall (RollCallID, StudentID, AttendanceDate)
      VALUES (v_RollCallID, v_StudentID, v_AttendanceDate);
    END IF;
  END LOOP FETCH_LOOP;

  -- Close the cursor
  CLOSE c_RollCall;
END;
//

DELIMITER ;
--call procedure
CALL MergeRollCallData();
```

8. **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.

```
SQL> CREATE TABLE Stud_Marks (
  name VARCHAR2(50),
  total_marks NUMBER
  );
Table created.

SQL> CREATE TABLE Result (
  Roll NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  Name VARCHAR2(50),
  Class VARCHAR2(20)
  );
Table created.
```

STEP – 2: Create the **proc_Grade** Stored Procedure

```
SQL> CREATE OR REPLACE PROCEDURE proc_Grade (
  p_total_marks IN NUMBER
  )
  IS
  BEGIN
  -- Determine the class based on the marks
  IF p_total_marks >= 990 AND p_total_marks <= 1500 THEN
 v_class := 'Distinction';
 ELSIF p_total_marks >= 900 AND p_total_marks <= 989 THEN
 v_class := 'First Class';
 ELSIF p_total_marks >= 825 AND p_total_marks <= 899 THEN
 v_class := 'Higher Second Class';
 ELSE
 v_class := 'No Class'; -- For marks outside the specified ranges
 END IF;


 INSERT INTO Result (Name, Class)
 VALUES (p_name, v_class);
 COMMIT;
 END proc_Grade;
 /
Procedure created.
```

STEP – 3: Insert record into Stud_Marks

```
INSERT INTO Stud_Marks (name, total_marks) VALUES (Rajnandini, 1000);
INSERT INTO Stud_Marks (name, total_marks) VALUES (Vanita, 920);
INSERT INTO Stud_Marks (name, total_marks) VALUES (Tanuja, 860);
```

STEP – 4: Create a PL/SQL Block to Use the **proc_Grade** Procedure

```
SQL> DECLARE
  2     v_name Stud_Marks.name%TYPE;
  3     v_total_marks Stud_Marks.total_marks%TYPE;
  4     BEGIN
  5
  6     FOR rec IN (SELECT name, total_marks FROM Stud_Marks) LOOP
  7         v_name := rec.name;
  8         v_total_marks := rec.total_marks;
  9
 10          -- Call the proc_Grade procedure
 11          proc_Grade(v_name, v_total_marks);
 12      END LOOP;
 13  END;
 14  /

PL/SQL procedure successfully completed.
```

9. **Write a database trigger: Row level and Statement level triggers, Before and After delete or update of database.**

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.

STEP - 1 : Create **Library** and **Library_Audit** tables

```
SQL> CREATE TABLE Library (
  BookID NUMBER PRIMARY KEY,
  BookTitle VARCHAR2(100),
  Author VARCHAR2(100),
  Category VARCHAR2(50)
  );
Table created.

SQL> CREATE TABLE Library_Audit (
  AuditID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  Operation VARCHAR2(10),  -- Stores 'UPDATE' or 'DELETE'
  BookID NUMBER,
  BookTitle VARCHAR2(100),
  Author VARCHAR2(100),
  Category VARCHAR2(50),
  AuditTimestamp DATE DEFAULT SYSDATE -- Stores the date and time of the operation
  );
Table created
```

STEP – 2: Create the Trigger

```
SQL> CREATE OR REPLACE TRIGGER trg1_Library_Audit BEFORE UPDATE OR
DELETE ON Library
  FOR EACH ROW
  BEGIN
  IF UPDATING THEN
          INSERT INTO Library_Audit (Operation, BookID, BookTitle, Author, Category)
  VALUES ('UPDATE', :OLD.BookID, :OLD.BookTitle, :OLD.Author, :OLD.Category);
  END IF;

 IF DELETING THEN
 INSERT INTO Library_Audit (Operation, BookID, BookTitle, Author, Category)
 VALUES ('DELETE', :OLD.BookID, :OLD.BookTitle, :OLD.Author, :OLD.Category);
 END IF;
 END trg_Library_Audit;
 /

Trigger created.
```

OUTPUT:
STEP 3 – Test the trigger

Insert Sample Data

```
SQL> INSERT INTO Library (BookID, BookTitle, Author, Category)
  2  VALUES (1, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
1 row created.

SQL> INSERT INTO Library (BookID, BookTitle, Author, Category)
  2  VALUES (2, '1984', 'George Orwell', 'Dystopian');
1 row created.
```

Update a record in the Library table

```
SQL> UPDATE Library SET BookTitle = 'Brave New World' WHERE BookID = 1;
1 row updated.
```

Delete a record from the Library table

```
SQL> DELETE FROM Library WHERE BookID = 2;
1 row deleted.
```

10. 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.

```
CREATE TABLE Borrower(
   Roll_no INT PRIMARY KEY,
   Name VARCHAR(100),
   DateofIssue DATE,
   NameofBook VARCHAR(100),
   Status CHAR(1) -- 'I' for Issued, 'R' for Returned
);
CREATE TABLE Fine (
   Roll_no INT,
   Date DATE,
   Amt DECIMAL(10, 2)
);
```

STEP 3: Insert values into tables

```
INSERT INTO Borrower (Roll_no, Name, DateofIssue, NameofBook, Status) VALUES
(1, 'Alice', '2023-08-01', 'Harry Potter', 'I'),
(2, 'Bob', '2023-08-15', 'The Hobbit', 'I'),
(3, 'Charlie', '2023-09-10', '1984', 'I');


-- Define the stored procedure
DELIMITER $$

CREATE PROCEDURE ManageBorrowerFine(
   IN p_Roll_no INT,
   IN p_NameofBook VARCHAR(100)
)
BEGIN
   DECLARE v_DateofIssue DATE;
   DECLARE v_FineAmount DECIMAL(10, 2) DEFAULT 0;
   DECLARE v_Days INT;
   DECLARE v_Status CHAR(1);

   -- Fetch borrower details
   SELECT DateofIssue, Status INTO v_DateofIssue, v_Status
   FROM Borrower
   WHERE Roll_no = p_Roll_no AND NameofBook = p_NameofBook;

   -- Check if the record exists
   IF v_DateofIssue IS NULL THEN
       SELECT 'No record found for the given Roll No and Book Name.' AS Message;
   ELSE
       -- Check the status of the book
       IF v_Status != 'I' THEN
           SELECT 'The book has not been issued or has already been returned.' AS Message;
       ELSE
           -- Calculate days since the date of issue
           SET v_Days = DATEDIFF(CURDATE(), v_DateofIssue);

           -- Determine fine based on the number of days
           IF v_Days > 30 THEN
               SET v_FineAmount = v_Days * 50;
           ELSEIF v_Days BETWEEN 15 AND 30 THEN
               SET v_FineAmount = v_Days * 5;
           ELSE
               SET v_FineAmount = 0; -- No fine for less than 15 days
           END IF;

           -- Update status to 'R' (Returned)
           UPDATE Borrower SET Status = 'R' WHERE Roll_no = p_Roll_no AND NameofBook =
p_NameofBook;

           -- If fine is applicable, insert into Fine table
           IF v_FineAmount > 0 THEN
               INSERT INTO Fine (Roll_no, Date, Amt) VALUES (p_Roll_no, CURDATE(),
v_FineAmount);
           END IF;

           COMMIT; -- Commit the changes
           SELECT 'Book returned successfully.' AS Message;
       END IF;
    END IF;
END $$
DELIMITER ;
-- Call the stored procedure with test values
CALL ManageBorrowerFine(1, 'Harry Potter');
-- Check the results
```

11. **The organization has decided to increase the salary of employees by 10% of existing salary, whose existing salary is less than Rs. 10000/-**

Write a PL/SQ block to update the salary as per above requirement, display an appropriate message based on the no. of rows affected by this update (using implicit cursor status variables).

```
CREATE TABLE EmployeePosition (
    EmpID INT PRIMARY KEY,
    EmpPosition VARCHAR(50),
    DateOfJoining DATE,
    Salary NUMBER(10, 2)
);

INSERT INTO EmployeePosition (EmpID, EmpPosition, DateOfJoining, Salary) VALUES
(1, 'Manager', TO_DATE('2015-06-15', 'YYYY-MM-DD'), 9500),
(2, 'Analyst', TO_DATE('2018-09-10', 'YYYY-MM-DD'), 10500),
(3, 'Developer', TO_DATE('2019-01-20', 'YYYY-MM-DD'), 8000),
(4, 'Clerk', TO_DATE('2021-04-25', 'YYYY-MM-DD'), 7000),
(5, 'Assistant', TO_DATE('2020-11-15', 'YYYY-MM-DD'), 12000);

SET SERVEROUTPUT ON;
DECLARE
    v_rows_updated NUMBER;
BEGIN
    UPDATE EmployeePosition
    SET Salary = Salary * 1.1
    WHERE Salary < 10000;
    v_rows_updated := SQL%ROWCOUNT;


    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE(v_rows_updated || ' employee(s) had their salary increased by 10%.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No employees found with a salary less than Rs. 10000.');
    END IF;
END;
/
```

12. Create The following two tables :

College-info

Faculty-info

College-info consists of fields : college-code, college-name, address

Faculty-info consists of fields : college-code, faculty-code, faculty-name, qualification, experience-in-no-of-years, address.

The field college-code is foreign key.

Generate queries to do the following :

a) **Retrieve all records from the College_info table**
b) **Retrieve all records from the Faculty_info table**
c) **List the faculty members along with the college name they are associated with**
d) **List all those faculty members whose experience is greater than or equal to 10 years and have M. Tech degree.**
e) **List all those faculty members, who have at least 10 years of experience but do not have M. Tech degree**

```
CREATE TABLE College_info (
    college_code INT PRIMARY KEY,
    college_name VARCHAR(100),
    address VARCHAR(255)
);


CREATE TABLE Faculty_info (
    faculty_code INT PRIMARY KEY,
    faculty_name VARCHAR(100),
    qualification VARCHAR(50),
    experience_in_no_of_years INT,

    address VARCHAR(255),

    college_code INT,

    FOREIGN KEY (college_code) REFERENCES College_info(college_code)

);
```

f) Retrieve all records from the College_info table

```
SELECT * FROM College_info;
```

g) Retrieve all records from the Faculty_info table

```
SELECT * FROM Faculty_info;
```

h) List the faculty members along with the college name they are associated with

```
SELECT FI.faculty_name, CI.college_name

FROM Faculty_info FI

JOIN College_info CI ON FI.college_code = CI.college_code;
```

i) List all those faculty members whose experience is greater than or equal to 10 years and have M. Tech degree.

```
SELECT faculty_name, qualification, experience_in_no_of_years

FROM Faculty_info

WHERE experience_in_no_of_years >= 10 AND qualification = 'M. Tech';
```

j) List all those faculty members, who have at least 10 years of experience but do not have M. Tech degree

```
SELECT faculty_name, qualification, experience_in_no_of_years

FROM Faculty_info

WHERE experience_in_no_of_years >= 10 AND qualification <> 'M. Tech';
```

13. Create the following table :

Student (roll-no, name, subject-name, subject-opted)

Subject(faculty-code, faculty-name, specialization)

Generate queries to do the following :

```
CREATE TABLE Student (

    roll_no INT PRIMARY KEY,

    name VARCHAR(100),

    subject_name VARCHAR(100),

    subject_opted BOOLEAN

);


CREATE TABLE Subject (

    faculty_code INT PRIMARY KEY,

    faculty_name VARCHAR(100),

    specialization VARCHAR(100)  );
```

i) **Retrieve all records from the Student table**

```
SELECT *FROM Student;
```

ii)Retrieve all records from the Subject table

```
SELECT *FROM Subject;
```

(iii) Find the number of students who have enrolled for the subject "DBMS".

```
SELECT COUNT(*) AS NumberOfStudents
```

```
FROM Student

WHERE subject_name = 'DBMS';
```

(iv) Find all those faculty members who have not offered any subject.

```
SELECT S.faculty_code, S.faculty_name, S.specialization

FROM Subject S

LEFT JOIN Student ST ON S.specialization = ST.subject_name

WHERE ST.subject_name IS NULL;
```

v) Find all subjects enrolled by Roll-no 1101.

```
SELECT subject_name

FROM Student

WHERE roll_no = 1101;
```

14. Create the following table :

Item (item-code, item-name, qty-in-stock, reorder-level)

Supplier (supplier-code, supplier-name, address)

Can-supply(supplier-code, item-code)

Generate queries to do the following :

(i) Retrieve all records from the Item table

ii)Retrieve all records from the Supplier table

iii) Display all Items supplied by all suppliers.

iv) Retrieve items where the quantity in stock is below the reorder level

v)List all those suppliers who can supply the given item.

vi) List all those items which cannot be supplied by given company

```
CREATE TABLE Item (
   item_code INT PRIMARY KEY,
   item_name VARCHAR(50),
   qty_in_stock INT,
   reorder_level INT
);

CREATE TABLE Supplier (
   supplier_code INT PRIMARY KEY,
   supplier_name VARCHAR(50),
   address VARCHAR(100)
);
```

```
CREATE TABLE Can_supply (
    supplier_code INT,
    item_code INT,
    PRIMARY KEY (supplier_code, item_code),
    FOREIGN KEY (supplier_code) REFERENCES Supplier(supplier_code),
    FOREIGN KEY (item_code) REFERENCES Item(item_code)
);
```

Generate queries to do the following :

(i) **Retrieve all records from the Item table**

```
SELECT  *FROM Item;
```

ii)Retrieve all records from the Supplier table

```
SELECT *FROM Supplier;
```

ii) **Display all Items supplied by all suppliers.**

```
SELECT I.item_name, S.supplier_name
FROM Item I
JOIN Can_supply CS ON I.item_code = CS.item_code
JOIN Supplier S ON CS.supplier_code = S.supplier_code;
```

iii) **Retrieve items where the quantity in stock is below the reorder level**

```
  SELECT item_name
  FROM Item
  WHERE qty_in_stock < reorder_level;
```

v)List all those suppliers who can supply the given item.

```
SELECT S.supplier_name
FROM Supplier S
JOIN Can_supply CS ON S.supplier_code = CS.supplier_code
WHERE CS.item_code = 105;
```

vi) List all those items which cannot be supplied by given company

```
SELECT item_name
FROM Item
WHERE item_code NOT IN (
    SELECT item_code
    FROM Can_supply
    WHERE supplier_code = 201  );
```

15. Create the following tables:

Student (roll-no, marks, category, district, state)

Student-rank(roll-no, marks, rank)

Generate queries to do the following :

i) Retrieve all records from the Student table

ii) Retrieve all records from the Student-rank table

iii)display each student's details along with their rank

ivi) List all those students who have come from Tamilnadu state and secured a rank above 100.

v) List all those students who come from Andhra Pradesh state and belong to given category who have secured a rank above 100

```
CREATE TABLE Student ( roll_no INT PRIMARY KEY, marks INT, category VARCHAR(50),
district VARCHAR(50), state VARCHAR(50) );
```

```
CREATE TABLE Student_rank (

   roll_no INT PRIMARY KEY,

   marks INT,

   student_rank INT,

   FOREIGN KEY (roll_no) REFERENCES Student(roll_no) ON DELETE CASCADE

);
```

Generate queries to do the following :

i) **Retrieve all records from the Student table**

```
SELECT * FROM Student;
```

ii) Retrieve all records from the Student-rank table

```
SELECT * FROM Student_rank;
```

iii)display each student’s details along with their rank

```
SELECT s.roll_no, s.marks, s.category, s.district, s.state, sr.rank

FROM Student s

JOIN Student_rank sr ON s.roll_no = sr.roll_no;
```

ivi) List all those students who have come from Tamilnadu state and secured a rank above 100.

```
SELECT S.roll_no, S.marks, S.category, S.district, S.state, R.student_rank

FROM Student S

JOIN Student_rank R ON S.roll_no = R.roll_no

WHERE S.state = 'Tamilnadu' AND R.student_rank > 100;
```

v) List all those students who come from Andhra Pradesh state and belong to given category who have secured a rank above 100

```
SELECT S.roll_no, S.marks, S.category, S.district, S.state, R.student_rank

FROM Student S

JOIN Student_rank R ON S.roll_no = R.roll_no

WHERE S.state = 'Andhra Pradesh' AND S.category = 'OBC' AND R.student_rank > 100;
```

MONGODB

16. Create a collection named Book. (book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline, mobile]), publisher_city, price,copies)

a. Add 5 documents in the collection with keys

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

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

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

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

I. Add 5 documents in the collection with keys

```
// Switch to the database (use your database name here)
use library; // or any name you'd prefer for the database

// Insert 5 sample documents into the "Book" collection
db.Book.insertMany([
  {
    book_isbn: "123-456-789",
    title: "Data Structures",
    publisher_name: "Pearson",
    publisher_city: "Mumbai",
    price: 500,
    copies: 10,
    author: { name: "Alice", address: "123 Street A", phone: { landline: "020-12345678",
mobile: "9123456780" } }
  },
  {
    book_isbn: "234-567-890",
    title: "Database Systems",
    publisher_name: "McGraw Hill",
        publisher_city: "Pune",
        price: 700,
        copies: 5,
        author: { name: "Bob", address: "456 Street B", phone: { landline: "020-23456789",
mobile: "9876543210" } }
    },
    {
        book_isbn: "345-678-901",
        title: "Introduction to Algorithms",
        publisher_name: "O'Reilly",
        publisher_city: "Delhi",
        price: 1200,
        copies: 8,
        author: { name: "Charlie", address: "789 Street C", phone: { landline: "020-34567890",
mobile: "8765432109" } }
    },
    {
        book_isbn: "456-789-012",
        title: "Design Patterns",
        publisher_name: "Pearson",
        publisher_city: "Bangalore",
        price: 950,
        copies: 4,
        author: { name: "Dave", address: "101 Street D", phone: { landline: "020-45678901",
mobile: "7654321098" } }
    },
    {
        book_isbn: "567-890-123",
        title: "Distributed Systems",
        publisher_name: "Oxford",
        publisher_city: "Pune",
        price: 800,
        copies: 6,
        author: { name: "Eve", address: "202 Street E", phone: { landline: "020-56789012",
mobile: "6543210987" } }
    }
]);
```

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

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

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

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

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

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

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

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

17. **Create a collection named Book. (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.**

    b. **Arrange the documents in descending order of book_isbn.**

    c. **Select Book Names whose title is "DBMS" .**

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

        e. **Display name of publishers as per no of books published by them in ascending order.**
        f. **Count the number of documents in the collection**

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

        g. **Arrange the documents in descending order of book_isbn.**

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

        h. **Select Book Names whose title is "DBMS" .**

```
db.Book.find({ title: "DBMS" }, { title: 1, _id: 0 });
```

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

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

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

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

18. **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**

b. **Sum the price field from orders**

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

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

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

```
use salesDB; // Or any name you'd like


// Insert sample documents into the "ORDERS" collection
db.ORDERS.insertMany([
  {
    cust_id: "abc123",
    ord_date: new Date("2012-10-04"),
    status: "A",
    price: 50,
    },
        items: [
            { sku: "xxx", qty: 25, price: 1 },
            { sku: "yyy", qty: 25, price: 1 }
        ]
    },
    {
        cust_id: "def456",
        ord_date: new Date("2012-10-05"),
        status: "B",
        price: 100,
        items: [
            { sku: "zzz", qty: 50, price: 2 },
            { sku: "aaa", qty: 25, price: 1 }
        ]
    },
    {
        cust_id: "abc123",
        ord_date: new Date("2012-11-04"),
        status: "A",
        price: 75,
        items: [
            { sku: "bbb", qty: 30, price: 2.5 },
            { sku: "ccc", qty: 15, price: 1 }
        ]
    },
    {
        cust_id: "xyz789",
        ord_date: new Date("2012-11-10"),
        status: "A",
        price: 150,
        items: [
            { sku: "ddd", qty: 20, price: 3 },
            { sku: "eee", qty: 40, price: 2.5 }
        ]
    },
    {
        cust_id: "def456",
        ord_date: new Date("2012-11-15"),
        status: "A",
        price: 200,
        items: [
            { sku: "fff", qty: 100, price: 2 },
            { sku: "ggg", qty: 50, price: 2.5 }
        ]
    }
]);
```

a. Count all records from orders

```
db.ORDERS.countDocuments();
```

b. Sum the price field from orders

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

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

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

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

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

```
db.ORDERS.aggregate([
    { $group: { _id: "$cust_id", total_price: { $sum: "$price" } } },
    { $sort: { total_price: 1 } } // Ascending order by total_price
]);
```

19. Create a collection named rating that contain 5 documents of the following prototype and solve the following Queries.

```
{
  movie_id: 123,
  user_id: 12,
  title: Toy Story(1995),
  status: 'A'
}
```

a) **Creating an index on movie_id and sorts the keys in the index in ascending order. Verify the query plan**

b) **Show various indexes created on movie collection.**

c) **Sort movie_id in descending order.**

d) **Create a descending order index on movie_id to get ratings related to "Toy Story (1995)" verify the query plan.**

e) **Limit the number of items in the result of above query.**

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

f) **Creating an index on movie_id and sorts the keys in the index in ascending order. Verify the query plan**

```
db.rating.createIndex({ movie_id: 1 });
```

g) **Show various indexes created on movie collection**.

```
db.rating.getIndexes();
```

h) Sort movie_id in descending order.

```
db.rating.find().sort({ movie_id: -1 });
```

i) Create a descending order index on movie_id to get ratings related to “Toy Story (1995)” verify the query plan.

```
db.rating.createIndex({ movie_id: -1 });
```

```
db.rating.find({ title: "Toy Story (1995)" }).sort({ movie_id: -1
}).explain("executionStats");
```

j) Limit the number of items in the result of above query.

```
db.rating.find({ title: "Toy Story (1995)" }).sort({ movie_id: -1 }).limit(1);
```

20. **Design a map-reduce operations on a collection “orders” that contains documents of the following prototype. Solve the following .**

```
{
    cust_id: "abc123",
    ord_date: new Date("Oct 04, 2012"),
    status: 'A',
    price: 25,
    gender :’F’,
    rating: 1

  }
```

```
db.orders.insertMany([
   {
      cust_id: "abc123",
      ord_date: new Date("Oct 04, 2012"),
      status: 'A',
      price: 25,
      gender: 'F',
      rating: 1
   },
   {
      cust_id: "abc124",
      ord_date: new Date("Oct 05, 2012"),
      status: 'A',
      price: 30,
    gender: 'M',
    rating: 2
},
{
    cust_id: "abc125",
    ord_date: new Date("Oct 06, 2012"),
    status: 'B,
price: 15,
    gender: 'F',
    rating: 3
},
{
    cust_id: "abc126",
    ord_date: new Date("Oct 07, 2012"),
    status: 'A',
    price: 20,
    gender: 'M',
    rating: 4
},
{
    cust_id: "abc127",
    ord_date: new Date("Oct 08, 2012"),
    status: 'A',
    price: 45,
    gender: 'F',
        rating: 5
    }
]);
```

a) Count the number of female (F) and male (M) respondents in the orders collection

```
var mapGenderCount = function() {
    emit(this.gender, 1); // Emit gender as the key and 1 as the value
};

var reduceGenderCount = function(key, values) {
    return Array.sum(values); // Sum up all values (1s) for each gender
};
db.orders.mapReduce(
    mapGenderCount,
    reduceGenderCount,
    {
        out: "gender_count"
    }
)
```

b) Count the number of each type of rating (1, 2, 3, 4 or 5) for each orders

```
var mapRatingCount = function() {
    emit(this.rating, 1); // Emit rating as the key and 1 as the value
};
```

```
var reduceRatingCount = function(key, values) {
    return Array.sum(values); // Sum up all values (1s) for each rating
};

db.orders.mapReduce(
    mapRatingCount,
    reduceRatingCount,
    {
        out: "rating_count"
    }
)
```

21. **Create a collection named rating that contain 5 documents of the following prototype and solve the following Queries.**

{

movie_id: 123,

user_id: 12,

title: Toy Story(1995),

status: 'A'

}

a) Get ratings for the movie “ICE AGE(2005)” using the descending ordered index on

movie_id and explain.

b) Rebuild all indexes for the ratings collection.

c) Drop index on rating collection.

d) Create an index on movie_id and ratings fields together with movie_id (ascending order

sorted) and rating (descending order sorted).

e) Create a descending order index on movie_id to get ratings related to “Toy Story (1995)” verify the query plan.

k) Get ratings for the movie “ICE AGE(2005)” using the descending ordered index on

movie_id and explain.

```
db.rating.createIndex({ movie_id: -1 });


db.rating.find({ title: "ICE AGE (2005)" }).sort({ movie_id: -1 }).explain("executionStats");
```

g). Rebuild all indexes for the ratings collection.

```
db.rating.reIndex();
```

b) **Drop index on rating collection.**

```
db.rating.getIndexes();

db.rating.dropIndex("index_name_here");

db.rating.dropIndexes();
```

d) Create an index on movie_id and ratings fields together with movie_id (ascending order sorted) and rating (descending order sorted).

```
db.rating.createIndex({ movie_id: 1, status: -1 });
```

e) Create a descending order index on movie_id to get ratings related to "Toy Story (1995)" verify the query plan.

```
db.rating.createIndex({ movie_id: -1 });

db.rating.find({ title: "Toy Story (1995)" }).sort({ movie_id: -1 }).explain("executionStats");
```

22.

23. 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)

```
db.createCollection("Book");

db.Book.insertMany([
   {
      book_isbn: "978-0136006633",
      title: "DBMS",
      publisher_name: "Tata McGraw Hill",
      author: {
         Name: "Alice Smith",
         Address: "123 Main St, Cityville",
         Phone_No: { landline: "123-456-7890", mobile: "987-654-3210" }
      },
      publisher_city: "New York",
      price: 500,
      copies: 5
   },
   {
      book_isbn: "978-0201633610",
      title: "Algorithms",
      publisher_name: "Springer",
      author: {
         Name: "Bob Johnson",
         Address: "456 Maple Rd, Townsville",
         Phone_No: { landline: "234-567-8901", mobile: "876-543-2109" }
      },
      publisher_city: "Berlin",
      price: 600,
    copies: 3
},
{
    book_isbn: "978-0132103123",
    title: "DBMS Concepts",
    publisher_name: "Tata McGraw Hill",
    author: {
        Name: "Carol White",
        Address: "789 Oak Ave, Villageplace",
        Phone_No: { landline: "345-678-9012", mobile: "765-432-1098" }
    },
    publisher_city: "Los Angeles",
    price: 450,
    copies: 2
},
{
    book_isbn: "978-1119473666",
    title: "Networking",
    publisher_name: "O'Reilly Media",
    author: {
        Name: "David Brown",
        Address: "321 Pine St, Hamlet",
        Phone_No: { landline: "456-789-0123", mobile: "654-321-0987" }
    },
    publisher_city: "San Francisco",
    price: 700,
    copies: 4
},
{
    book_isbn: "978-0135161100",
    title: "DBMS",
            publisher_name: "McGraw Hill Education",
            author: {
                Name: "Korth",
                Address: "123 Elm St, Citycenter",
                Phone_No: { landline: "567-890-1234", mobile: "543-210-9876" }
            },
            publisher_city: "Chicago",
            price: 800,
            copies: 1
        }
    ]);
```

a) Select Book Names whose title is ”DBMS” .

```
db.Book.find(
   { title: "DBMS" },
   { title: 1, _id: 0 } // Only return the title field
);
```

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

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

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

```
db.Book.aggregate([
    {
        $group: {
            _id: "$publisher_name", // Group by publisher name
            count: { $sum: 1 }      // Count the number of books for each publisher
        }
    },
    {
        $sort: { count: 1 } // Sort by count in ascending order
    },
    {
        $project: {
            publisher_name: "$_id", // Rename _id to publisher_name
            _id: 0,
            count: 1
        }
    }
]);
```

d) Get publisher names who published at least one book written by author name like ‘K%’.

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

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

```
db.Book.deleteOne(
   { "author.Name": "Korth" }
);
```

24. **Consider following structure for MongoDB collections and write a query for following requirements in MongoDB**

Teachers(Tname, dno, experience, salary, date_of joining)

Students(Sname, roll_no, class)

i) Write a MongoDB query to create above collections & for insertion

of some sample documents.

```
// Create Teachers collection and insert sample documents

db.createCollection("Teachers");


db.Teachers.insertMany([

  {

    Tname: "John Doe",

    dno: 1,

    experience: 12,

    salary: 15000,

    date_of_joining: new Date("2010-05-15")

  },

  {

    Tname: "Jane Smith",

    dno: 2,

    experience: 8,

    salary: 9500,

    date_of_joining: new Date("2014-09-01")

  },
    {
        Tname: "Emily Johnson",
        dno: 2,
        experience: 15,
        salary: 20000,
        date_of_joining: new Date("2007-03-22")
    }
]);

// Create Students collection and insert sample documents
db.createCollection("Students");

db.Students.insertMany([
    {
        Sname: "Anil",
        roll_no: 1,
        class: "TE"
    },
    {
        Sname: "Ravi",
        roll_no: 2,
        class: "TE"
    },
    {
        Sname: "Priya",
        roll_no: 3,
        class: "BE"
    }
]);
```

ii) Find the information about all teachers of dno = 2 and having salary greater than or equal to 10,000/-

```
db.Teachers.find(
    { dno: 2, salary: { $gte: 10000 } }
);
```

iii) Find the student information having roll_no = 2 or Sname = Anil

```
db.Students.find(
    { $or: [ { roll_no: 2 }, { Sname: "Anil" } ] }
);
```

iv) Display Total no of Students of TE Class

```
db.Students.countDocuments(
    { class: "TE" }
);
```

V) update salary as 5% increment of teacher whose experience is >10 years.

```
db.Teachers.updateMany(
    { experience: { $gt: 10 } },
    { $mul: { salary: 1.05 } } // Increase salary by 5%
);
```

25.

26. Create the following table with the fields given below : PRODUCT (P_ID, Model, Price, Name, Date_of Manufacture, Date_of Expiry)

```
CREATE TABLE PRODUCT (
   P_ID INT PRIMARY KEY,
   Model VARCHAR(50),
   Price DECIMAL(10, 2),
   Name VARCHAR(255),
   Date_of_Manufacture DATE,
   Date_of_Expiry DATE
);
```

(a) Display name and date_of expiry of all the products whose price is more than 500.

```
SELECT Name, Date_of_Expiry
FROM PRODUCT
WHERE Price > 500;
```

(b) Display name, product_ID and price of all the products whose date_of manufacture is after "01-01-2018".

```
SELECT Name, P_ID, Price
FROM PRODUCT
WHERE Date_of_Manufacture > TO_DATE('2018-01-01', 'YYYY-MM-DD');
```

(c) Display name and date_of manufacture and date- of expiry of all the products whose price is between 5,000 and 10,000.

SELECT Name, Date_of_Manufacture, Date_of_Expiry

FROM PRODUCT

WHERE Price BETWEEN 5000 AND 10000;

(d) Display name, product_ID and model of all the products which are going to expire after two months from today.

SELECT Name, P_ID, Model

FROM PRODUCT

WHERE Date_of_Expiry > ADD_MONTHS(SYSDATE, 2);

27. **Create a table named STUDENT with the following fields : 20 (FIRST NAME, MIDDLE NAME, LAST NAME, STUDENT_ENRLNO, DATE_OF_BIRTH, CLASS, SECTION, GENDER, YEAR_OF JOIN, ADMISSION_NO, ADDRESS1, ADDRESS2, CITY, STATE, RESPHONE, PIN_CODE)**

```
CREATE TABLE STUDENT (
   FIRST_NAME VARCHAR(50),
   MIDDLE_NAME VARCHAR(50),
   LAST_NAME VARCHAR(50),
   STUDENT_ENRLNO INT PRIMARY KEY,
   DATE_OF_BIRTH DATE,
   CLASS INT,
   SECTION CHAR(1),
   GENDER CHAR(1), -- Assuming 'M' for male and 'F' for female
   YEAR_OF_JOIN INT,
   ADMISSION_NO INT,
   ADDRESS1 VARCHAR(255),
   ADDRESS2 VARCHAR(255),
   CITY VARCHAR(50),
   STATE VARCHAR(50),
   RESPHONE VARCHAR(15),
   PIN_CODE VARCHAR(10)
);
```

(a) Display all the list of students who are in class - 6, section - A.

```
SELECT *
FROM STUDENT
```

```
WHERE CLASS = 6 AND SECTION = 'A';
```

(b) To display all the students list whose first name starts with "A".

```
SELECT *

FROM STUDENT

WHERE FIRST_NAME LIKE 'A%';
```

(c) To display all the students list who are girls.

```
SELECT *

FROM STUDENT

WHERE GENDER = 'F';
```

(d) To display all the students whose YEAR-OF-JOIN is 2000.

```
SELECT *

FROM STUDENT

WHERE YEAR_OF_JOIN = 2000;
```

(e) Sort the records of students with respect to their ADMISSION_NO, in ascending order.

```
SELECT *

FROM STUDENT

ORDER BY ADMISSION_NO ASC;
```

28. **Create the following table CATALOG with the following fields : (BOOK ID, BOOK TITLE, AUTHOR, AUTHOR_ID, PUBLISHER_ID, CATEGORY_ID, YEAR, ISBN, PRICE)**

```
CREATE TABLE CATALOG (

   BOOK_ID INT PRIMARY KEY,

   BOOK_TITLE VARCHAR(255),

   AUTHOR VARCHAR(255),

   AUTHOR_ID VARCHAR(50),

   PUBLISHER_ID VARCHAR(100),

   CATEGORY_ID VARCHAR(100),

   YEAR INT,

   ISBN VARCHAR(20),

   PRICE DECIMAL(10, 2)

);
```

(a) To display all the books of the CATEGORY_ID : "COMPUTERS".

```
SELECT *

FROM CATALOG

WHERE CATEGORY_ID = 'COMPUTERS';
```

(b) List all the books whose PRICE is greater than or equal to 1000/-.

```
SELECT *

FROM CATALOG

WHERE PRICE >= 1000;
```

(c) List all the books whose PUBLISHER_ID is "Tata McGraw-Hill".

```
SELECT *

FROM CATALOG

WHERE PUBLISHER_ID = 'Tata McGraw-Hill';
```

(d) List all the books whose YEAR of publication is 2013.

```
SELECT *

FROM CATALOG

WHERE YEAR = 2013;
```

(e) List all the BOOK_TITLEs whose AUTHOR_ID is "123".

```
SELECT BOOK_TITLE

FROM CATALOG

WHERE AUTHOR_ID = '123';
```

30 Create the following tables :Student(roll-no, name, date-of-birth, course-id)Course (Course-id, name, fee, duration, status)

Write PL/SQL procedure to do the following :Set the status of course to "offered" in which the number of candidates is atleast 10 otherwise set it to "not offered"

```
CREATE TABLE Course (

   Course_id INT PRIMARY KEY,

   name VARCHAR(100),

   fee DECIMAL(10, 2),

   duration VARCHAR(50),

   status VARCHAR(20)

);
```

```
CREATE TABLE Student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(100),
    date_of_birth DATE,
    course_id INT,
    FOREIGN KEY (course_id) REFERENCES Course(Course_id)
);

CREATE OR REPLACE PROCEDURE UpdateCourseStatus AS
BEGIN
    FOR rec IN (
        SELECT Course_id
        FROM Course c
    )
    LOOP
        DECLARE
            student_count INT;
        BEGIN
            -- Count the number of students enrolled in the course
            SELECT COUNT(*)
            INTO student_count
            FROM Student s
            WHERE s.course_id = rec.Course_id;
            IF student_count >= 10 THEN
                UPDATE Course
            SET status = 'offered'
            WHERE Course_id = rec.Course_id;
          ELSE
            UPDATE Course
            SET status = 'not offered'
            WHERE Course_id = rec.Course_id;
          END IF;
        END;
    END LOOP;
    COMMIT;
END;
/
BEGIN
    UpdateCourseStatus;
END;
/
```

29. Create the following tables :Student(roll-no, name, date-of-birth, course-id)Course (Course-id, name, fee, duration, status) Write PL/SQL procedure to do the following :Set the status of course to "not offered" in which the number of candidates is less than 5

```
CREATE TABLE Course (

   Course_id INT PRIMARY KEY,

   name VARCHAR(100),

   fee DECIMAL(10, 2),

   duration VARCHAR(50),

   status VARCHAR(20)

);


CREATE TABLE Student (

   roll_no INT PRIMARY KEY,

   name VARCHAR(100),

   date_of_birth DATE,

   course_id INT,

   FOREIGN KEY (course_id) REFERENCES Course(Course_id)

);


CREATE OR REPLACE PROCEDURE UpdateCourseStatus AS

BEGIN

   FOR rec IN (

      SELECT Course_id

      FROM Course

   )
```

```
    LOOP
        DECLARE
            student_count INT;
        BEGIN
            -- Count the number of students enrolled in the course
            SELECT COUNT(*)
            INTO student_count
            FROM Student
            WHERE course_id = rec.Course_id;

            -- Update the course status if student count is less than 5
            IF student_count < 5 THEN
                UPDATE Course
                SET status = 'not offered'
                WHERE Course_id = rec.Course_id;
            END IF;
        END;
    END LOOP;

    COMMIT;
END;
/

BEGIN
    UpdateCourseStatus;
```

```
END;

/
```