Collection “city” which contains the documents given as below(Perform on Mongo Terminal)

{

city: “pune”,

type: “urban”,

state: “MH”,

population: 5600000

}

-using mapreduce, find statewise population

-using mapreduce, find citywise population

-using mapreduce, find typewise population

Sample Documents

```
db.city.insertMany([
  { city: "pune", type: "urban", state: "MH", population: 5600000 },
  { city: "nagpur", type: "urban", state: "MH", population: 3500000 },
  { city: "nashik", type: "urban", state: "MH", population: 2200000 },
  { city: "surat", type: "urban", state: "GJ", population: 6000000 },
  { city: "rajkot", type: "urban", state: "GJ", population: 1800000 },
  { city: "latur", type: "rural", state: "MH", population: 1200000 }
]);
```

1️⃣ **Find State-wise PopulationMap Function**

```
var mapState = function() {
  emit(this.state, this.population);
};
```

- **Reduce Function**

```
var reduceState = function(key, values) {
  return Array.sum(values);
};
```

- **Run MapReduce**

```
db.city.mapReduce(
  mapState,
  reduceState,
  { out: "statewise_population" }
);
```

- **View Result**

```
db.statewise_population.find().pretty();
```

2️⃣ Find City-wise Population

- **Map Function**

```
var mapCity = function() {
  emit(this.city, this.population);
};
```

- **Reduce Function**

```
var reduceCity = function(key, values) {
  return Array.sum(values);
};
```

- **Run MapReduce**

```
db.city.mapReduce(
  mapCity,
  reduceCity,
  { out: "citywise_population" }
);
```

- **View Result**

```
db.citywise_population.find().pretty();
```

3️⃣ Find Type-wise Population

- **Map Function**

```
var  mapType = function() {
  emit(this.type, this.population);
};
```

- **Reduce Function**

```
var reduceType = function(key, values) {
  return Array.sum(values);
};
```

- **Run MapReduce**

```
 db.city.mapReduce(
  mapType,
  reduceType,
  { out: "typewise_population" }
);
```

- **View Result**

```
db.typewise_population.find().pretty();
```

Collection “books” which contains the documents given as below(Perform on Mongo Terminal)

{
name: “understanding JAVA”
pages:400
}

-using MapReduce, categerize books into small books and big books, based on number of pages display and categorize total number of books.

```
db.books.insertMany([
  { name: "Understanding JAVA", pages: 400 },
  { name: "Learn MongoDB", pages: 150 },
  { name: "Data Structures", pages: 550 },
  { name: "HTML Basics", pages: 100 },
  { name: "Advanced Python", pages: 700 }
]);
```

Step 1: Define the Map Function

```
var mapBooks = function() {
  if (this.pages <= 300)
    emit("Small Books", 1);
  else
    emit("Big Books", 1);
};
```

Step 2: Define the Reduce Function

```
var reduceBooks = function(key, values) {
  return Array.sum(values);
};
```

Step 3: Run the MapReduce Command

```
db.books.mapReduce(
  mapBooks,
  reduceBooks,
  { out: "book_category_count" }
);
```

Step 4: View the Output

```
db.book_category_count.find().pretty();
```

Collection “movies” which contains the documents given as below(Perform on Mongo Terminal)

{

name: “Movie1” type: “action” budget:1000000 producer:{

name: “producer1” address: “PUNE”

}

}

i. Find the name of the movie having budget greater than 1,00,000.

ii. Find the name of producer who lives in Pune

iii. Update the type of movie “action” to “horror”

iv. Find all the documents produced by name “producer1” with their address

```
db.movies.insertMany([
 {
  name: "Movie1",
  type: "action",
  budget: 1000000,
  producer: { name: "producer1", address: "PUNE" }
 },
 {
  name: "Movie2",
  type: "comedy",
  budget: 80000,
  producer: { name: "producer2", address: "MUMBAI" }
 },
 {
  name: "Movie3",
  type: "action",
```

```
    budget: 5000000,
    producer: { name: "producer1", address: "PUNE" }
  }
]);
```

i. Find the name of the movie having budget greater than 1,00,000

```
db.movies.find(
  { budget: { $gt: 100000 } },
  { name: 1, _id: 0 }
);
```

ii. Find the name of producer who lives in PUNE

```
db.movies.find(
  { "producer.address": "PUNE" },
  { "producer.name": 1, _id: 0 }
);
```

iii. Update the type of movie “action” to “horror”

```
db.movies.updateMany(
  { type: "action" },
  { $set: { type: "horror" } }
);
```

✅ To verify:

```
db.movies.find({}, { name: 1, type: 1, _id: 0 });
```

iv. Find all the documents produced by producer1 with their address

```
db.movies.find(
  { "producer.name": "producer1" },
  { name: 1, "producer.address": 1, _id: 0 }
);
```

Collection “orderinfo” which contains the documents given as below(Perform on Mongo Terminal)

{

cust_id:123 cust_name: “abc”, status: “A”,

price: 250

}

i. Find the total price for each customer and display in the order of total price.

ii. Find the distinct customer names

iii. Display the “price” of customers whose status is 'A'

iv. Delete the customers whose status is 'A'

```
db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
  { cust_id: 124, cust_name: "xyz", status: "B", price: 450 },
  { cust_id: 125, cust_name: "abc", status: "A", price: 350 },
  { cust_id: 126, cust_name: "pqr", status: "A", price: 150 },
  { cust_id: 127, cust_name: "xyz", status: "B", price: 300 }
]);
```

i. Find the total price for each customer and display in the order of total price

```
db.orderinfo.aggregate([
  { $group: { _id: "$cust_name", total_price: { $sum: "$price" } } },
  { $sort: { total_price: 1 } } // 1 for ascending, -1 for descending
]);
```

ii. Find the distinct customer names

```
db.orderinfo.distinct("cust_name");
```

iii. Display the “price” of customers whose status is 'A'

```
db.orderinfo.find(
  { status: "A" },
  { price: 1, _id: 0 }
);
```

iv. Delete the customers whose status is 'A'

```
db.orderinfo.deleteMany({ status: "A" });
```

✅ To verify deletion:

```
db.orderinfo.find();
```

Collection “orderinfo” which contains the documents given as below(Perform on Mongo Terminal)

{

cust_id:123 cust_name:”abc” status:”A” price:250

}

i. find the average price for each customers having status 'A'

ii. Display the status of the customers whose amount/price lie between 100 and 1000

iii. Display the customers information without “_id” .

iv. create a simple index on onderinfo collection and fire the queries. Also explain the impact of index before and after

```
db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
  { cust_id: 124, cust_name: "xyz", status: "B", price: 600 },
  { cust_id: 125, cust_name: "abc", status: "A", price: 800 },
  { cust_id: 126, cust_name: "pqr", status: "A", price: 150 },
  { cust_id: 127, cust_name: "xyz", status: "C", price: 1100 }
]);
```

i. Find the average price for each customer having status 'A'

```
db.orderinfo.aggregate([
  { $match: { status: "A" } },                    // only 'A' customers
  { $group: { _id: "$cust_name", avg_price: { $avg: "$price" } } }
]);
```

ii. Display the status of customers whose price lies between 100 and 1000

```
db.orderinfo.find(
  { price: { $gte: 100, $lte: 1000 } },
  { status: 1, _id: 0 }
);
```

iii. Display the customers information without _id

db.orderinfo.find({}, { _id: 0 });

iv. Create a Simple Index on the orderinfo collection and check its impact

db.orderinfo.createIndex({ price: 1 });

🔍 View Existing Indexes:

db.orderinfo.getIndexes();

Check Query Performance (Before and After Index)

- **Before creating the index**

  db.orderinfo.find({ price: { $gt: 500 } }).explain("executionStats");

- **After creating the index**
  db.orderinfo.find({ price: { $gt: 500 } }).explain("executionStats");

Collection “orderinfo” which contains the documents given as below(Perform on Mongo Terminal)

{

cust_id:123 cust_name:”abc” status:”A” price:250

}

i. Add “Age” field to the orderinfo collection
ii. Create the complex index on orderinfo collection and fire the queries and drop the duplicates.
iii. Display the average price for each customer group by status
iv. Change the customer‟s name whose status is “B”

```
db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
  { cust_id: 124, cust_name: "xyz", status: "B", price: 600 },
  { cust_id: 125, cust_name: "abc", status: "A", price: 800 },
  { cust_id: 126, cust_name: "pqr", status: "C", price: 150 },
  { cust_id: 127, cust_name: "xyz", status: "B", price: 300 }
]);
```

i. Add “Age” field to the orderinfo collection

```
db.orderinfo.updateMany({}, { $set: { Age: 30 } });
```

✅ **Verify the update:**

```
db.orderinfo.find({}, { _id: 0 });
```

ii. Create a complex (compound) index, run queries, and drop duplicates

```
db.orderinfo.createIndex({ cust_name: 1, status: 1 });
```

- **Run a query using that index:**

```
db.orderinfo.find({ cust_name: "abc", status: "A" });
```

- **To verify the index is being used:**

```
db.orderinfo.find({ cust_name: "abc", status: "A" }).explain("executionStats");
```

- **Drop duplicate documents (if any):**

```
db.orderinfo.aggregate([
  { $group: {
      _id: { cust_id: "$cust_id", cust_name: "$cust_name", status: "$status", price: "$price" },
      dups: { $addToSet: "$_id" },
```

```
    count: { $sum: 1 }
  }},
  { $match: { count: { $gt: 1 } } }
]).forEach(function(doc) {
  doc.dups.shift(); // keep one
  db.orderinfo.deleteMany({ _id: { $in: doc.dups } });
});
```

iii. Display the average price for each customer grouped by status

```
db.orderinfo.aggregate([
  { $group: { _id: "$status", avg_price: { $avg: "$price" } } }
]);
```

iv. Change the customer's name whose status = "B"

```
db.orderinfo.updateMany(
  { status: "B" },
  { $set: { cust_name: "updated_name" } }
);
```

✅ **Verify the update:**

```
db.orderinfo.find({}, { _id: 0 });
```

Collection “orderinfo” which contains the documents given as below(Perform on Mongo Terminal)

{

cust_id:123 cust_name:”abc” status:”A” price:250

}

i. Display the name of the customer having the price between 250 and 450

ii. Increment the price by 10 for cust_id: 123 and decrement the price by 5 for cust_id: 124

iii. Remove any one of the field from the orderinfo collection.

iv. Find the name of the customer whose status is either A or price is 250 or both

```
use salesDB;

db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
  { cust_id: 124, cust_name: "xyz", status: "B", price: 400 },
  { cust_id: 125, cust_name: "pqr", status: "C", price: 500 }
]);
```

i. Display the name of the customer having price between 250 and 450

```
db.orderinfo.find(
  { price: { $gte: 250, $lte: 450 } },
  { cust_name: 1, _id: 0 }
);
```

ii. Increment and Decrement the Price

```
db.orderinfo.updateOne(
  { cust_id: 123 },
  { $inc: { price: 10 } }
```

);

db.orderinfo.updateOne(

{ cust_id: 124 },

{ $inc: { price: -5 } }

);

✅ **Check the results:**

db.orderinfo.find({}, { _id: 0 });

iii. Remove any one of the fields from all documents

db.orderinfo.updateMany({}, { $unset: { status: "" } });

✅ **Check:**

db.orderinfo.find({}, { _id: 0 });

iv. Find the name of customers whose status is “A” OR price is 250 (or both)

db.orderinfo.find(

{ $or: [{ status: "A" }, { price: 250 }] },

{ cust_name: 1, _id: 0 }

);

Collection “orderinfo” which contains the documents given as below(Perform on Mongo Terminal)

{

cust_id:123 cust_name:”abc” status:”A” price:250

}

i. Find the total price for each customer and display in the order of total price.

ii. Display the average price for each customer group by status

iii. Create simple index on orderinfo collection.

iv. Create the complex index on orderinfo collection and fire the queries and drop the duplicates.

```
db.orderinfo.insertMany([
 { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
 { cust_id: 124, cust_name: "xyz", status: "B", price: 300 },
 { cust_id: 125, cust_name: "abc", status: "A", price: 150 },
 { cust_id: 126, cust_name: "pqr", status: "C", price: 400 },
 { cust_id: 127, cust_name: "xyz", status: "B", price: 500 }
])
```

i. Find total price for each customer and display in order of total price

```
db.orderinfo.aggregate([
 {
  $group: {
   _id: "$cust_name",
   total_price: { $sum: "$price" }
  }
 },
 {
  $sort: { total_price: 1 }   // ascending order
```

```
 }
])
```

ii. Display the average price for each status

```
db.orderinfo.aggregate([
 {
  $group: {
   _id: "$status",
   avg_price: { $avg: "$price" }
  }
 }
])
```

iii. Create a simple index on cust_name

```
db.orderinfo.createIndex({ cust_name: 1 })
```

Verify created indexes:

```
db.orderinfo.getIndexes()
```

iv. Create a complex (compound) index on multiple fields

```
db.orderinfo.createIndex({ cust_name: 1, status: 1, price: -1 })
```

Check indexes again:

```
db.orderinfo.getIndexes()
```

🔍 **Fire a query using that complex index**

```
db.orderinfo.find({ cust_name: "xyz", status: "B" })
```

Drop duplicate entries (if any)

```
var ids = db.orderinfo.aggregate([
 { $group: { _id: "$cust_id", dups: { $addToSet: "$_id" }, count: { $sum: 1 } } },
```

```
  { $match: { count: { $gt: 1 } } }
]).toArray();

ids.forEach(function(doc) {
  doc.dups.shift();  // keep one
  db.orderinfo.deleteMany({ _id: { $in: doc.dups } });
});
```

To drop an index later:

```
db.orderinfo.dropIndex({ cust_name: 1, status: 1, price: -1 })
```

(Perform on MYSQL Terminal)

teaches(T_ID, course_id, sec_id, semester, year) student(S_ID, name, dept_name, tot_cred) instructor(T_ID, name, dept_name, salary) course(course_id, title, dept_name, credits)

i. Find the names of the instructor in the university who have taught the courses semester wise.

ii. Create View on single table which retrieves student details.

iii. Rename the column of table student from dept_name to department_name

iv. Delete student name whose department is NULL

```
CREATE DATABASE universityDB;
USE universityDB;

CREATE TABLE instructor (
  T_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  salary DECIMAL(10,2)
);

CREATE TABLE course (
  course_id INT PRIMARY KEY,
  title VARCHAR(100),
  dept_name VARCHAR(50),
  credits INT
);

CREATE TABLE teaches (
  T_ID INT,
```

```
  course_id INT,
  sec_id INT,
  semester VARCHAR(20),
  year INT,
  FOREIGN KEY (T_ID) REFERENCES instructor(T_ID),
  FOREIGN KEY (course_id) REFERENCES course(course_id)
);

CREATE TABLE student (
  S_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  tot_cred INT
);



INSERT INTO instructor VALUES
(1, 'John Smith', 'Computer Science', 75000.00),
(2, 'Alice Johnson', 'Information Technology', 68000.00),
(3, 'Mark Brown', 'Electronics', 72000.00);

INSERT INTO course VALUES
(101, 'Database Systems', 'Computer Science', 4),
(102, 'Operating Systems', 'Computer Science', 4),
(103, 'Data Structures', 'Information Technology', 3);

INSERT INTO teaches VALUES
(1, 101, 1, 'Fall', 2023),
```

(2, 103, 1, 'Spring', 2023),

(1, 102, 2, 'Fall', 2024),

(3, 103, 2, 'Summer', 2024);

INSERT INTO student VALUES

(201, 'Riya', 'Computer Science', 120),

(202, 'Amit', NULL, 100),

(203, 'Sneha', 'Information Technology', 90);

i. Instructor names semester-wise

```
SELECT DISTINCT
   i.name AS Instructor_Name,
   t.semester,
   t.year
FROM instructor i
JOIN teaches t ON i.T_ID = t.T_ID
ORDER BY t.year, t.semester;
```

ii. Create a view for student details

```
CREATE VIEW student_view AS
SELECT S_ID, name, dept_name, tot_cred
FROM student;
SELECT * FROM student_view;
```

iii. Rename column dept_name → department_name

```
ALTER TABLE student CHANGE dept_name department_name VARCHAR(50);
```

Check:

```
DESC student;
```

iv. Delete students whose department is NULL

```
DELETE FROM student

WHERE department_name IS NULL;
```

Check remaining records:

```
SELECT * FROM student;
```

(Perform on MYSQL Terminal)

teaches(T_ID, course_id, sec_id, semester, year) student(S_ID, name, dept_name, tot_cred) instructor(T_ID, name, dept_name, salary) course(course_id, title, dept_name, credits)

i. Find the names of all instructor whose salary is greater than 25000.

ii. Update Instructor dept_name from CSE to IT

iii. Create Simple Index on course table on the attribute dept_name and find out department wise courses.

iv. Create a View to retrieve data of instructor with details of course, section, semester and year he conducted lectures on.

```
CREATE DATABASE universityDB;
USE universityDB;

CREATE TABLE instructor (
  T_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  salary DECIMAL(10,2)
);

CREATE TABLE course (
  course_id INT PRIMARY KEY,
  title VARCHAR(100),
  dept_name VARCHAR(50),
  credits INT
);

CREATE TABLE teaches (
```

```
  T_ID INT,
  course_id INT,
  sec_id INT,
  semester VARCHAR(20),
  year INT,
  FOREIGN KEY (T_ID) REFERENCES instructor(T_ID),
  FOREIGN KEY (course_id) REFERENCES course(course_id)
);

CREATE TABLE student (
  S_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  tot_cred INT
);

INSERT INTO instructor VALUES
(1, 'John', 'CSE', 30000),
(2, 'Alice', 'IT', 28000),
(3, 'Mark', 'CSE', 24000),
(4, 'Riya', 'ECE', 40000);

INSERT INTO course VALUES
(101, 'Database Systems', 'CSE', 4),
(102, 'Operating Systems', 'CSE', 3),
(103, 'Networking', 'IT', 3),
(104, 'Microprocessors', 'ECE', 4);
```

```
INSERT INTO teaches VALUES
(1, 101, 1, 'Fall', 2023),
(1, 102, 2, 'Spring', 2024),
(2, 103, 1, 'Winter', 2023),
(4, 104, 1, 'Summer', 2024);
```

i. Find the names of all instructors whose salary > 25000

```
SELECT name, dept_name, salary
FROM instructor
WHERE salary > 25000;
```

ii. Update instructor dept_name from CSE → IT

```
UPDATE instructor
SET dept_name = 'IT'
WHERE dept_name = 'CSE';
```

Verify:

```
SELECT * FROM instructor;
```

iii. Create a simple index on course table for dept_name and find department-wise courses

```
CREATE INDEX idx_deptname ON course(dept_name);
```

Check index:

```
SHOW INDEX FROM course;
```

Find department-wise courses:

```
SELECT dept_name, COUNT(course_id) AS total_courses
FROM course
GROUP BY dept_name;
```

iv. Create a View to retrieve instructor details with course, section, semester, and year

```
CREATE VIEW instructor_course_details AS
SELECT
    i.T_ID,
    i.name AS Instructor_Name,
    i.dept_name AS Instructor_Department,
    c.title AS Course_Title,
    c.dept_name AS Course_Department,
    t.sec_id,
    t.semester,
    t.year
FROM instructor i
JOIN teaches t ON i.T_ID = t.T_ID
JOIN course c ON t.course_id = c.course_id;
```

To view the data:

```
SELECT * FROM instructor_course_details;
```

(Perform on MYSQL Terminal) teaches(T_ID,course_id,sec_id,semester,year) section(course_id,sec_id,semester,year,building,room_no) instructor(T_ID,name, dept_name,salary) course(course_id,title,dept_name,credits)

i. Display the name of the instructor whose salary is between 30000 and 60000 in descending order.

ii. Find the average, minimum, maximum salary in each department.

iii. Display the name of the instructor, his department name and salary who teaches the course_id: 101

iv. Find the name of the instructor whose average salary in greater then 60000.

```
CREATE DATABASE universityDB;

USE universityDB;

CREATE TABLE instructor (
  T_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  salary DECIMAL(10,2)
);

CREATE TABLE course (
  course_id INT PRIMARY KEY,
  title VARCHAR(100),
  dept_name VARCHAR(50),
  credits INT
);

CREATE TABLE teaches (
  T_ID INT,
  course_id INT,
```

```
  sec_id INT,
  semester VARCHAR(20),
  year INT,
  FOREIGN KEY (T_ID) REFERENCES instructor(T_ID),
  FOREIGN KEY (course_id) REFERENCES course(course_id)
);

CREATE TABLE section (
  course_id INT,
  sec_id INT,
  semester VARCHAR(20),
  year INT,
  building VARCHAR(50),
  room_no INT,
  FOREIGN KEY (course_id) REFERENCES course(course_id)
);

INSERT INTO instructor VALUES
(1, 'John', 'CSE', 55000),
(2, 'Alice', 'IT', 65000),
(3, 'Mark', 'ECE', 45000),
(4, 'Riya', 'CSE', 30000),
(5, 'Neha', 'IT', 72000);

INSERT INTO course VALUES
(101, 'Database Systems', 'CSE', 4),
(102, 'Operating Systems', 'CSE', 3),
(103, 'Networking', 'IT', 3),
```

(104, 'Microprocessors', 'ECE', 4);

INSERT INTO teaches VALUES

(1, 101, 1, 'Fall', 2023),

(2, 103, 1, 'Spring', 2023),

(3, 104, 1, 'Winter', 2024),

(4, 102, 1, 'Fall', 2024);

INSERT INTO section VALUES

(101, 1, 'Fall', 2023, 'Main Building', 101),

(102, 1, 'Fall', 2024, 'Tech Block', 202),

(103, 1, 'Spring', 2023, 'IT Wing', 303),

(104, 1, 'Winter', 2024, 'ECE Block', 404);

i. Display the name of the instructor whose salary is between 30000 and 60000, in descending order

```
SELECT name, dept_name, salary
FROM instructor
WHERE salary BETWEEN 30000 AND 60000
ORDER BY salary DESC;
```

ii. Find the average, minimum, and maximum salary in each department

```
SELECT dept_name,
    AVG(salary) AS Avg_Salary,
    MIN(salary) AS Min_Salary,
    MAX(salary) AS Max_Salary
FROM instructor
GROUP BY dept_name;
```

iii. Display the name, department, and salary of instructors who teach course_id = 101

```
SELECT i.name, i.dept_name, i.salary
FROM instructor i
JOIN teaches t ON i.T_ID = t.T_ID
WHERE t.course_id = 101;
```

iv. Find the name of instructors whose average salary is greater than 60000

```
SELECT name, dept_name, AVG(salary) AS avg_salary
FROM instructor
GROUP BY name, dept_name
HAVING AVG(salary) > 60000;
```

(Perform on MYSQL Terminal) student(S_ID,name,dept_name,tot_cred) instructor(T_ID,name,dept_name,salary) course(course_id,title,dept_name,credits)

i. Find the average salary of instructor in those departments where the average salary is more than Rs. 42000/-.

ii. Increase the salary of each instructor in the computer department by 10%.

iii. Find the names of instructors whose names are neither „Amol" nor „Amit".

iv. Find the names of student which contains „am" as its substring.

v. Find the name of students from computer department that "DBMS" courses they take.

```
CREATE DATABASE collegeDB;
USE collegeDB;

CREATE TABLE student (
  S_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  tot_cred INT
);

CREATE TABLE instructor (
  T_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  salary DECIMAL(10,2)
);

CREATE TABLE course (
  course_id INT PRIMARY KEY,
  title VARCHAR(100),
```

```
  dept_name VARCHAR(50),
  credits INT
);

INSERT INTO instructor VALUES
(1, 'Amol', 'Computer', 40000),
(2, 'Amit', 'IT', 45000),
(3, 'Riya', 'Computer', 50000),
(4, 'John', 'Mechanical', 60000),
(5, 'Neha', 'IT', 47000);

INSERT INTO student VALUES
(1, 'Ram', 'Computer', 20),
(2, 'Amita', 'IT', 18),
(3, 'Sam', 'Computer', 22),
(4, 'Kamal', 'Mechanical', 24),
(5, 'Anamika', 'Computer', 21);

INSERT INTO course VALUES
(101, 'DBMS', 'Computer', 4),
(102, 'Networking', 'IT', 3),
(103, 'Thermodynamics', 'Mechanical', 4),
(104, 'OS', 'Computer', 4);
```

i. Find the average salary of instructors in those departments where the average salary > 42000

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM instructor
```

```
GROUP BY dept_name
HAVING AVG(salary) > 42000;
```

ii. Increase the salary of each instructor in the Computer department by 10%

```
UPDATE instructor
SET salary = salary * 1.10
WHERE dept_name = 'Computer';
```

To **verify**:

```
SELECT * FROM instructor;
```

iii. Find the names of instructors whose names are neither 'Amol' nor 'Amit'

```
SELECT name
FROM instructor
WHERE name NOT IN ('Amol', 'Amit');
```

iv. Find the names of students whose names contain 'am' as a substring

```
SELECT name
FROM student
WHERE name LIKE '%am%';
```

v. Find the names of students from the Computer department who take ‘DBMS’ course

```
SELECT s.name
FROM student s
JOIN course c ON s.dept_name = c.dept_name
WHERE c.title = 'DBMS' AND s.dept_name = 'Computer';
```

(Perform on MYSQL Terminal) teaches(T_ID,course_id,sec_id,semester,year) student(S_ID,name,dept_name,tot_cred) instructor(T_ID,name,dept_name,salary) course(course_id,title,dept_name,credits)

i. Find the total number of instructor who teach a course in the First semester 2010.

ii. Find the names of all instructors from computer department whose name includes the substring „shree".

iii. Find the names of all instructor that have a salary greater than average salary of the Civil Department.

iv. Find the set of all courses taught either in Fifth semester 2014 or in First semester 2010 or both.

```
CREATE DATABASE collegeDB2;

USE collegeDB2;

CREATE TABLE instructor (
  T_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  salary DECIMAL(10,2)
);


CREATE TABLE student (
  S_ID INT PRIMARY KEY,
  name VARCHAR(50),
  dept_name VARCHAR(50),
  tot_cred INT
);


CREATE TABLE course (
  course_id INT PRIMARY KEY,
  title VARCHAR(100),
```

```
  dept_name VARCHAR(50),
  credits INT
);

CREATE TABLE teaches (
  T_ID INT,
  course_id INT,
  sec_id INT,
  semester VARCHAR(20),
  year INT,
  FOREIGN KEY (T_ID) REFERENCES instructor(T_ID),
  FOREIGN KEY (course_id) REFERENCES course(course_id)
);

INSERT INTO instructor VALUES
(1, 'Shreekant', 'Computer', 55000),
(2, 'Shreeja', 'Computer', 48000),
(3, 'Amit', 'Civil', 42000),
(4, 'Neha', 'IT', 46000),
(5, 'Rohan', 'Civil', 40000),
(6, 'Anil', 'Computer', 53000);

INSERT INTO course VALUES
(101, 'DBMS', 'Computer', 4),
(102, 'Networking', 'IT', 3),
(103, 'Thermodynamics', 'Mechanical', 4),
(104, 'Structural Analysis', 'Civil', 3),
(105, 'Programming', 'Computer', 4);
```

```
INSERT INTO teaches VALUES
(1, 101, 1, 'First', 2010),
(2, 105, 2, 'Fifth', 2014),
(3, 104, 1, 'First', 2010),
(4, 102, 2, 'Fifth', 2014),
(5, 104, 1, 'Fifth', 2014),
(6, 105, 1, 'First', 2010);
```

i. Find total number of instructors who teach a course in the First semester 2010

```
SELECT COUNT(DISTINCT T_ID) AS total_instructors
FROM teaches
WHERE semester = 'First' AND year = 2010;
```

ii. Find the names of all instructors from Computer department whose name includes substring 'shree'

```
SELECT name
FROM instructor
WHERE dept_name = 'Computer'
  AND name LIKE '%shree%';
```

iii. Find names of all instructors who have salary greater than the average salary of the Civil department

```
SELECT name, salary
FROM instructor
WHERE salary > (
  SELECT AVG(salary)
  FROM instructor
  WHERE dept_name = 'Civil'
```

);

iv. Find the set of all courses taught either in Fifth semester 2014 or First semester 2010 or both

```
SELECT DISTINCT course_id

FROM teaches

WHERE (semester = 'Fifth' AND year = 2014)

  OR (semester = 'First' AND year = 2010);
```

(Perform on MYSQL Terminal) Emp(emp_id,ename, street, city) works(emp_id,c_id,ename, cname, sal) Company(c_id,cname, city) Manager(mgr_id, mgrname)

i. Modify the database so that a particular company (eg. ABC) now in Pune

ii. Give all managers of Mbank a 10% raise. If salary is >20,000, give only 3% raise.

iii. Find out the names of all the employees who works in „Bosch" company in city Pune

iv. Delete all records in the works table for employees of a particular company (Eg, SBC Company) whose salary>50,000.

```
CREATE DATABASE companyDB;
USE companyDB;

CREATE TABLE Emp (
  emp_id INT PRIMARY KEY,
  ename VARCHAR(50),
  street VARCHAR(50),
  city VARCHAR(50)
);

CREATE TABLE Company (
  c_id INT PRIMARY KEY,
  cname VARCHAR(50),
  city VARCHAR(50)
);

CREATE TABLE Works (
  emp_id INT,
  c_id INT,
  ename VARCHAR(50),
```

```
  cname VARCHAR(50),
  sal DECIMAL(10,2),
  FOREIGN KEY (emp_id) REFERENCES Emp(emp_id),
  FOREIGN KEY (c_id) REFERENCES Company(c_id)
);


INSERT INTO Emp VALUES
(1, 'Ravi', 'MG Road', 'Pune'),
(2, 'Neha', 'FC Road', 'Mumbai'),
(3, 'Amit', 'JM Road', 'Pune'),
(4, 'Rohit', 'Baner', 'Pune'),
(5, 'Priya', 'Karve Road', 'Nagpur');

INSERT INTO Company VALUES
(101, 'ABC', 'Mumbai'),
(102, 'Mbank', 'Delhi'),
(103, 'Bosch', 'Pune'),
(104, 'SBC', 'Chennai');

INSERT INTO Works VALUES
(1, 101, 'Ravi', 'ABC', 18000),
(2, 102, 'Neha', 'Mbank', 22000),
(3, 103, 'Amit', 'Bosch', 30000),
(4, 104, 'Rohit', 'SBC', 55000),
(5, 102, 'Priya', 'Mbank', 19000);

INSERT INTO Manager VALUES
```

```
(201, 'Arun'),
(202, 'Kiran'),
(203, 'Meera');

CREATE TABLE Manager (
  mgr_id INT PRIMARY KEY,
  mgrname VARCHAR(50)
);
```

i. Modify the database so that a particular company (e.g. ABC) is now in Pune

```
UPDATE Company
SET city = 'Pune'
WHERE cname = 'ABC';
```

check the update:

```
SELECT * FROM Company;
```

ii. Give all managers of Mbank a raise:

- If salary ≤ 20,000 → **10% raise**
- If salary > 20,000 → **3% raise**

```
UPDATE Works
SET sal =
  CASE
    WHEN sal <= 20000 THEN sal * 1.10
    ELSE sal * 1.03
  END
WHERE cname = 'Mbank';
```

check after update:

```
SELECT * FROM Works WHERE cname = 'Mbank';
```

iii. Find names of all employees who work in ‘Bosch’ company located in Pune

```
SELECT w.ename
FROM Works w
JOIN Company c ON w.c_id = c.c_id
WHERE c.cname = 'Bosch' AND c.city = 'Pune';
```

iv. Delete all records in Works table for employees of a particular company (e.g., SBC) whose salary > 50,000

```
DELETE FROM Works
WHERE cname = 'SBC' AND sal > 50000;
```

To verify deletion:

```
SELECT * FROM Works;
```

(Perform on MYSQL Terminal) Empl(e_no, e_name, post, pay_rate) Position(pos_no, post)

Duty-alloc (pos_no, e_no, month,year, shift)

Implement the following SQL queries

i. Get duty allocation details for e_no 123 for the first shift in the month of April 2003

ii. Get the employees whose rate of pay is > or equal rate of pay of employees 'Sachin'.

iii. Create a view for displaying minimum, maximum and average salary for all the posts.

iv. Get a count of different employees on each shift having post „manager".

```
CREATE DATABASE dutyDB;
USE dutyDB;
CREATE TABLE Empl (
  e_no INT PRIMARY KEY,
  e_name VARCHAR(50),
  post VARCHAR(50),
  pay_rate DECIMAL(10,2)
);

CREATE TABLE Position (
  pos_no INT PRIMARY KEY,
  post VARCHAR(50)
);

CREATE TABLE Duty_alloc (
  pos_no INT,
  e_no INT,
  month VARCHAR(20),
  year INT,
```

```
  shift VARCHAR(20),
  FOREIGN KEY (pos_no) REFERENCES Position(pos_no),
  FOREIGN KEY (e_no) REFERENCES Empl(e_no)
);

INSERT INTO Empl VALUES
(101, 'Sachin', 'Manager', 50000),
(102, 'Ravi', 'Clerk', 30000),
(103, 'Neha', 'Manager', 52000),
(104, 'Anil', 'Supervisor', 40000),
(105, 'Kiran', 'Manager', 55000),
(123, 'Suresh', 'Clerk', 28000);

INSERT INTO Position VALUES
(1, 'Manager'),
(2, 'Clerk'),
(3, 'Supervisor');

INSERT INTO Duty_alloc VALUES
(1, 101, 'April', 2003, 'First'),
(2, 123, 'April', 2003, 'First'),
(3, 104, 'April', 2003, 'Second'),
(1, 103, 'May', 2003, 'First'),
(2, 102, 'April', 2003, 'Third'),
(1, 105, 'April', 2003, 'First');
```

i. Get duty allocation details for employee e_no 123 for first shift in April 2003

```
SELECT *
```

```
FROM Duty_alloc
WHERE e_no = 123
  AND shift = 'First'
  AND month = 'April'
  AND year = 2003;
```

ii. Get employees whose pay_rate ≥ pay_rate of employee 'Sachin'

```
SELECT *
FROM Empl
WHERE pay_rate >= (
  SELECT pay_rate FROM Empl WHERE e_name = 'Sachin'
);
```

iii. Create a view to display minimum, maximum, and average salary for all posts

```
CREATE VIEW post_salary_stats AS
SELECT
  post,
  MIN(pay_rate) AS min_salary,
  MAX(pay_rate) AS max_salary,
  AVG(pay_rate) AS avg_salary
FROM Empl
GROUP BY post;
```

To view results:

```
SELECT * FROM post_salary_stats;
```

iv. Get count of different employees on each shift having post 'Manager'

```
SELECT d.shift, COUNT(DISTINCT d.e_no) AS total_managers
```

```
FROM Duty_alloc d

JOIN Empl e ON d.e_no = e.e_no

WHERE e.post = 'Manager'

GROUP BY d.shift;
```

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)

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

```
CREATE DATABASE IF NOT EXISTS librarydb;

USE librarydb;



CREATE TABLE Borrower (
  Rollin INT NOT NULL,
  Name VARCHAR(100),
  DateofIssue DATE,
  NameofBook VARCHAR(200),
  Status CHAR(1),
  PRIMARY KEY (Rollin, NameofBook)
);

CREATE TABLE Fine (
  id INT AUTO_INCREMENT PRIMARY KEY,
  Roll_no INT,
  Date_of_Return DATE,
```

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

INSERT INTO Borrower (Rollin, Name, DateofIssue, NameofBook, Status) VALUES
(101, 'Ramesh', '2025-10-01', 'DBMS', 'I'),    -- 41 days
(102, 'Sita', '2025-11-01', 'Networks', 'I'),  -- 10 days
(123, 'Vijay', '2025-10-25', 'Java', 'I');     -- 17 days
```

Step 4: Create Stored Procedure

```
DELIMITER $$

CREATE PROCEDURE sp_return_book(
    IN p_roll INT,
    IN p_book VARCHAR(200)
)
BEGIN
    DECLARE v_date_issue DATE;
    DECLARE v_days INT;
    DECLARE v_fine DECIMAL(10,2) DEFAULT 0;

    -- get issue date
    SELECT DateofIssue INTO v_date_issue
    FROM Borrower
    WHERE Rollin = p_roll
      AND NameofBook = p_book
      AND Status = 'I'
    LIMIT 1;
```

```
-- calculate days
SET v_days = DATEDIFF(CURDATE(), v_date_issue);

-- calculate fine
IF v_days <= 15 THEN
   SET v_fine = 0;
ELSE
   SET v_fine = (LEAST(v_days,30) - 15)*5;
   IF v_days > 30 THEN
      SET v_fine = v_fine + (v_days - 30)*50;
   END IF;
END IF;

-- update borrower status
UPDATE Borrower
SET Status = 'R'
WHERE Rollin = p_roll
  AND NameofBook = p_book;

-- insert fine if applicable
IF v_fine > 0 THEN
   INSERT INTO Fine (Roll_no, Date_of_Return, Amt)
   VALUES (p_roll, CURDATE(), v_fine);
END IF;

-- display result
SELECT p_roll AS Roll_No,
```

```
        p_book AS Book_Name,
        v_date_issue AS Date_of_Issue,
        v_days AS Days_Kept,
        v_fine AS Fine_Amount,
        'Status set to R' AS Note;
END $$

DELIMITER ;
```

Step 5: Call Procedure for Each Borrower

(a) Vijay (17 days)

```
CALL sp_return_book(123, 'Java');
```

(b) Sita (10 days → no fine)

```
CALL sp_return_book(102, 'Networks');
```

(c) Ramesh (41 days)

```
CALL sp_return_book(101, 'DBMS');
```

Check updated Borrower table:

```
SELECT * FROM Borrower;
```

Check Fine table:

```
SELECT * FROM Fine;
```

Write a PL/SQL block to implement all type of cursor(Implicit,Explicit, Cursor FOR Loop,Parameterized Cursor). Also 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 the old table
CREATE TABLE O_RollCall (
    RollNo NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Department VARCHAR2(50)
);

-- Create the new table
CREATE TABLE N_RollCall (
    RollNo NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Department VARCHAR2(50)
);

-- Insert sample data
INSERT INTO O_RollCall VALUES (1, 'Amit', 'CSE');
INSERT INTO O_RollCall VALUES (2, 'Sneha', 'IT');
INSERT INTO O_RollCall VALUES (3, 'Raj', 'ECE');

INSERT INTO N_RollCall VALUES (2, 'Sneha', 'IT');
INSERT INTO N_RollCall VALUES (4, 'Neha', 'CSE');
INSERT INTO N_RollCall VALUES (5, 'Pooja', 'CSE');
```

```
COMMIT;
```

(a) Implicit Cursor Example

```
BEGIN
   UPDATE O_RollCall
   SET Department = 'Computer Science'
   WHERE Department = 'CSE';

   DBMS_OUTPUT.PUT_LINE('Rows updated = ' || SQL%ROWCOUNT);
END;
/
```

(b) Explicit Cursor Example

```
 DECLARE
   CURSOR c1 IS SELECT RollNo, Name FROM O_RollCall;
   v_roll O_RollCall.RollNo%TYPE;
   v_name O_RollCall.Name%TYPE;
BEGIN
   OPEN c1;
   LOOP
      FETCH c1 INTO v_roll, v_name;
      EXIT WHEN c1%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('RollNo: ' || v_roll || '  Name: ' || v_name);
   END LOOP;
   CLOSE c1;
END;
/
```

(c) Cursor FOR Loop Example

```
BEGIN
   FOR rec IN (SELECT * FROM O_RollCall) LOOP
      DBMS_OUTPUT.PUT_LINE('FOR LOOP -> ' || rec.RollNo || ' : ' || rec.Name);
   END LOOP;
END;
/
```

(d) Parameterized Cursor Example (Merge Logic)

```
DECLARE
   -- Parameterized cursor takes RollNo as input
   CURSOR c_new(p_roll N_RollCall.RollNo%TYPE) IS
      SELECT RollNo, Name, Department
      FROM N_RollCall
      WHERE RollNo = p_roll;

   v_roll N_RollCall.RollNo%TYPE;
   v_name N_RollCall.Name%TYPE;
   v_dept N_RollCall.Department%TYPE;
   v_exists NUMBER;
BEGIN
   -- Loop through all records in N_RollCall
   FOR rec IN (SELECT RollNo FROM N_RollCall) LOOP

      -- Open parameterized cursor for each RollNo
      OPEN c_new(rec.RollNo);
      FETCH c_new INTO v_roll, v_name, v_dept;
      CLOSE c_new;
```

```
    -- Check if RollNo already exists in O_RollCall
    SELECT COUNT(*) INTO v_exists FROM O_RollCall WHERE RollNo = v_roll;

    IF v_exists = 0 THEN
      INSERT INTO O_RollCall VALUES (v_roll, v_name, v_dept);
      DBMS_OUTPUT.PUT_LINE('Inserted: ' || v_name);
    ELSE
      DBMS_OUTPUT.PUT_LINE('Skipped (Already Exists): ' || v_name);
    END IF;

  END LOOP;

  COMMIT;
END;
/
```

now:

```
SELECT * FROM O_RollCall;
```

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. Write a PL/SQL block for using procedure created with above requirement.

Stud_Marks(rollno, name, total_marks) Result(Roll,Name, Class)

```
CREATE TABLE Stud_Marks (
  rollno INT PRIMARY KEY,
  name VARCHAR(50),
  total_marks INT
);

CREATE TABLE Result (
  rollno INT,
  name VARCHAR(50),
  class VARCHAR(30)
);

INSERT INTO Stud_Marks VALUES (1, 'Amit', 1450);
INSERT INTO Stud_Marks VALUES (2, 'Neha', 960);
INSERT INTO Stud_Marks VALUES (3, 'Raj', 875);
INSERT INTO Stud_Marks VALUES (4, 'Sneha', 780);

COMMIT;
```

Step 3: Create Stored Procedure proc_Grade

```
CREATE OR REPLACE PROCEDURE proc_Grade (
    p_roll Stud_Marks.rollno%TYPE,
    p_name Stud_Marks.name%TYPE,
```

```
    p_marks Stud_Marks.total_marks%TYPE
)
IS
    v_class VARCHAR2(30);
BEGIN
    IF p_marks BETWEEN 990 AND 1500 THEN
        v_class := 'Distinction';
    ELSIF p_marks BETWEEN 900 AND 989 THEN
        v_class := 'First Class';
    ELSIF p_marks BETWEEN 825 AND 899 THEN
        v_class := 'Higher Second Class';
    ELSE
        v_class := 'Fail';
    END IF;

    INSERT INTO Result (rollno, name, class)
    VALUES (p_roll, p_name, v_class);

    DBMS_OUTPUT.PUT_LINE('Student: ' || p_name || ' → ' || v_class);
END;
/
```

Step 4: PL/SQL Block to Use proc_Grade

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR c1 IS SELECT * FROM Stud_Marks;
```

```
BEGIN
   FOR rec IN c1 LOOP
      proc_Grade(rec.rollno, rec.name, rec.total_marks);
   END LOOP;

   DBMS_OUTPUT.PUT_LINE('All students processed successfully.');
END;
/
```

Verify the Results

```
SELECT * FROM Result;
```

Write a PL/SQL block to implement database trigger on Library Table.

```
CREATE TABLE Library (
   book_id NUMBER PRIMARY KEY,
   book_name VARCHAR2(100),
   author VARCHAR2(100),
   price NUMBER
);

CREATE TABLE Library_Log (
   log_id NUMBER GENERATED BY DEFAULT AS IDENTITY,
   action_type VARCHAR2(20),
   book_id NUMBER,
   book_name VARCHAR2(100),
   author VARCHAR2(100),
   price NUMBER,
   action_date DATE
);
```

Step 2: Create Trigger on Library Table

```
CREATE OR REPLACE TRIGGER trg_library_log
AFTER INSERT OR UPDATE OR DELETE
ON Library
FOR EACH ROW
BEGIN
   IF INSERTING THEN
      INSERT INTO Library_Log (action_type, book_id, book_name, author, price, action_date)
```

```
    VALUES ('INSERT', :NEW.book_id, :NEW.book_name, :NEW.author, :NEW.price,
SYSDATE);

  ELSIF UPDATING THEN
    INSERT INTO Library_Log (action_type, book_id, book_name, author, price, action_date)
    VALUES ('UPDATE', :NEW.book_id, :NEW.book_name, :NEW.author, :NEW.price,
SYSDATE);

  ELSIF DELETING THEN
    INSERT INTO Library_Log (action_type, book_id, book_name, author, price, action_date)
    VALUES ('DELETE', :OLD.book_id, :OLD.book_name, :OLD.author, :OLD.price, SYSDATE);
  END IF;
END;
/
```

Step 3: Test the Trigger

```
-- Insert
INSERT INTO Library VALUES (1, 'Database System Concepts', 'Silberschatz', 750);
INSERT INTO Library VALUES (2, 'Let Us C', 'Yashwant Kanetkar', 450);

-- Update
UPDATE Library SET price = 800 WHERE book_id = 1;

-- Delete
DELETE FROM Library WHERE book_id = 2;
COMMIT;
```

4.Check the Log Table:-

```
SELECT * FROM Library_Log ORDER BY log_id;
```

Using MySQL and JAVA connectivity (Two Tier) perform the following queries

i. Create a table of employee details , Employee(SSN, Ename,state, salary)

ii. Insert Records into Employee table

iii. Retrieve the details based on Social Security Number(SSN).

iv. Update the employee state from 'MH' to 'TN'

v. Delete all the employees from „Gujrat"

```
CREATE DATABASE companyDB;
USE companyDB;

CREATE TABLE Employee (
  SSN INT PRIMARY KEY,
  Ename VARCHAR(50),
  state VARCHAR(50),
  salary DECIMAL(10,2)
);
```

✅ **Step 2: Download MySQL JDBC Driver**

Download the MySQL Connector/J:
👉 https://dev.mysql.com/downloads/connector/j/

Then add the .jar file to your Java classpath.
(If using VS Code or IntelliJ, add it as an external library.)

Step 3: Java Program (Two-Tier JDBC Application)

```
import java.sql.*;
import java.util.Scanner;

public class EmployeeJDBC {
    public static void main(String[] args) {
```

```
String url = "jdbc:mysql://localhost:3306/companyDB";
String user = "root";
String password = "your_mysql_password";

try (Connection con = DriverManager.getConnection(url, user, password);
     Scanner sc = new Scanner(System.in)) {

   System.out.println("Connected to MySQL successfully!");

   // i. Create table (only run once)
   Statement st = con.createStatement();
   st.executeUpdate("CREATE TABLE IF NOT EXISTS Employee (" +
         "SSN INT PRIMARY KEY, " +
         "Ename VARCHAR(50), " +
         "state VARCHAR(50), " +
         "salary DECIMAL(10,2))");
   System.out.println("Employee table ready!");

   // ii. Insert records
   System.out.println("Enter Employee details (SSN, Name, State, Salary):");
   int ssn = sc.nextInt();
   sc.nextLine();
   String name = sc.nextLine();
   String state = sc.nextLine();
   double salary = sc.nextDouble();

   PreparedStatement ps = con.prepareStatement("INSERT INTO Employee VALUES
(?,?,?,?)");
```

```
        ps.setInt(1, ssn);
        ps.setString(2, name);
        ps.setString(3, state);
        ps.setDouble(4, salary);
        ps.executeUpdate();
        System.out.println("Record inserted!");

        // iii. Retrieve details based on SSN
        System.out.print("Enter SSN to search: ");
        int searchSSN = sc.nextInt();
        PreparedStatement ps2 = con.prepareStatement("SELECT * FROM Employee WHERE
SSN=?");
        ps2.setInt(1, searchSSN);
        ResultSet rs = ps2.executeQuery();
        while (rs.next()) {
           System.out.println("SSN: " + rs.getInt("SSN") +
                 ", Name: " + rs.getString("Ename") +
                 ", State: " + rs.getString("state") +
                 ", Salary: " + rs.getDouble("salary"));
        }

        // iv. Update employee state from 'MH' to 'TN'
        int updated = st.executeUpdate("UPDATE Employee SET state='TN' WHERE
state='MH'");
        System.out.println(updated + " record(s) updated from MH to TN");

        // v. Delete employees from 'Gujrat'
        int deleted = st.executeUpdate("DELETE FROM Employee WHERE state='Gujrat'");
        System.out.println(deleted + " record(s) deleted from Gujrat.");
```

```
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Step 4: Compile and Run

In your terminal (same folder as the .java file):

```
javac -cp .:mysql-connector-j-9.1.0.jar EmployeeJDBC.java
java -cp .:mysql-connector-j-9.1.0.jar EmployeeJDBC
```