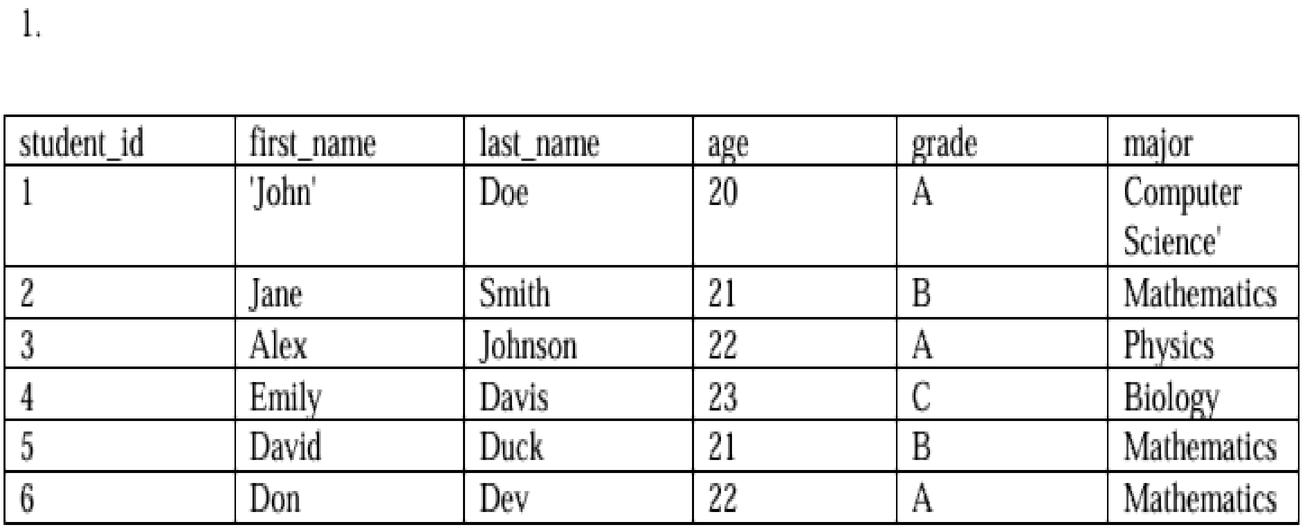


1.

student_id	first_name	last_name	age	grade	major
1	'John'	Doe	20	A	Computer Science'
2	Jane	Smith	21	B	Mathematics
3	Alex	Johnson	22	A	Physics
4	Emily	Davis	23	C	Biology
5	David	Duck	21	B	Mathematics
6	Don	Dev	22	A	Mathematics

1. Create above table Student display the table

```
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    age INT,
    grade CHAR(1),
    major VARCHAR(100)
);
INSERT INTO Student (student_id, first_name, last_name, age, grade, major) VALUES
(1, 'John', 'Doe', 20, 'A', 'Computer Science'),
(2, 'Jane', 'Smith', 21, 'B', 'Mathematics'),
(3, 'Alex', 'Johnson', 22, 'A', 'Physics'),
(4, 'Emily', 'Davis', 23, 'C', 'Biology'),
(5, 'David', 'Duck', 21, 'B', 'Mathematics'),
(6, 'Don', 'Dev', 22, 'A', 'Mathematics');
```

```
SELECT * FROM Student;
```

2. Change the name of student Jane to Jenne.

```
  UPDATE Student
 SET first_name = 'Jenne'
  WHERE first_name = 'Jane';
```

3. Find Students with a Specific Grade A

```
  SELECT * FROM Student
  WHERE grade = 'A';
```

4. Count the Number of Students in Each Major

```
SELECT major, COUNT(*) AS student_count
FROM Student
GROUP BY major;
```

5. Order Students by Age in Ascending Order

```
SELECT * FROM Student
ORDER BY age ASC;
```

6. Find the Oldest Student /Find the youngest Student

- **Oldest Student:**

```
SELECT * FROM Student
ORDER BY age DESC
LIMIT 1;
```

- **Youngest Student:**

```
SELECT * FROM Student
ORDER BY age ASC
LIMIT 1;
```

7. Update a Student's Major of student_id-2

```
UPDATE Student
SET major = 'New Major Name'
WHERE student_id = 2;
```

8. Delete a Student Record of id=6;

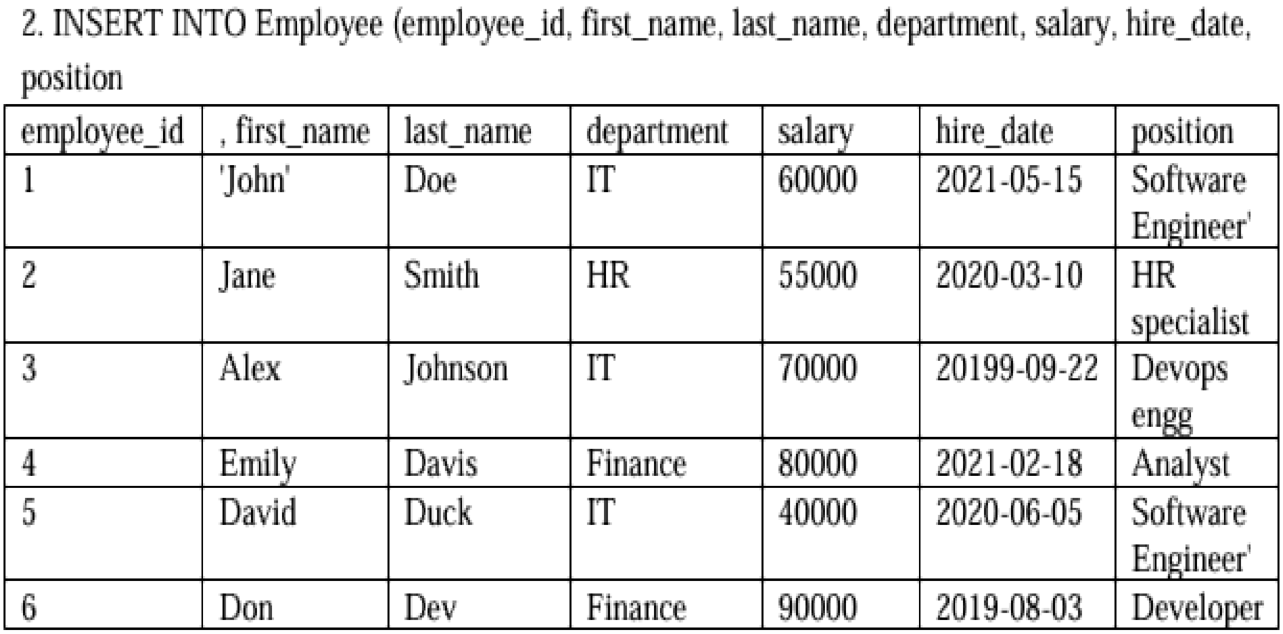
```
DELETE FROM Student

WHERE student_id = 6;
```

9. Count the Number of Students in each Major where grade=”a”

```
SELECT major, COUNT(*) AS student_count

FROM Student

WHERE grade = 'A'

GROUP BY major;
```

10. Count the Number of Students in Each Grade having count greater than 2

```
SELECT grade, COUNT(*) AS student_count

FROM Student

GROUP BY grade

HAVING COUNT(*) > 2;
```



2. INSERT INTO Employee (employee_id, first_name, last_name, department, salary, hire_date, position

employee_id	, first_name	last_name	department	salary	hire_date	position
1	'John'	Doe	IT	60000	2021-05-15	Software Engineer'
2	Jane	Smith	HR	55000	2020-03-10	HR specialist
3	Alex	Johnson	IT	70000	20199-09-22	Devops engg
4	Emily	Davis	Finance	80000	2021-02-18	Analyst
5	David	Duck	IT	40000	2020-06-05	Software Engineer'
6	Don	Dev	Finance	90000	2019-08-03	Developer

1. Select All Data from Employee Table

SELECT * FROM Employee;

2. Select Employees in a Specific Department of IT

SELECT * FROM Employee

WHERE department = 'IT';

3. Count the Number of Employees in Each Department

SELECT department, COUNT(*) AS employee_count

FROM Employee

GROUP BY department;

4. Find the Average Salary in Each Department

SELECT department, AVG(salary) AS average_salary

FROM Employee

GROUP BY department;

5. List Employees Hired After a 1 February 2021

SELECT * FROM Employee

```
WHERE hire_date > '2021-02-01';
```

6. Increase the salary of an Employees of IT department by 5000.

```
UPDATE Employee

SET salary = salary + 5000

WHERE department = 'IT';
```

7. Find the highest salary in each department

```
SELECT department, MAX(salary) AS highest_salary

FROM Employee

GROUP BY department;
```

8. Count the Number of Employees in Each Department Having More Than 1 Employee

```
SELECT department, COUNT(*) AS employee_count

FROM Employee

GROUP BY department

HAVING COUNT(*) > 1;
```

9. Find the employee having Highest / Lowest salary.

- **Highest Salary:**

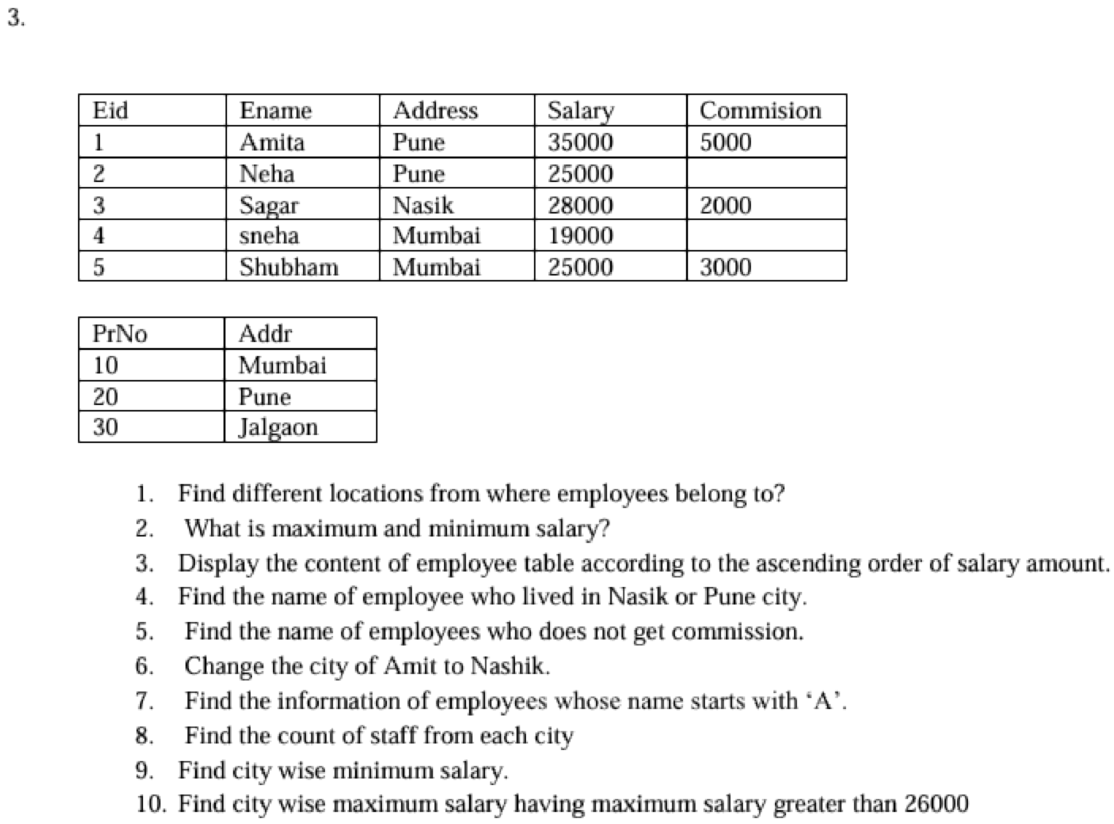
```
SELECT * FROM Employee
WHERE salary = (SELECT MAX(salary) FROM Employee);
```

- **Lowest Salary:**

```
SELECT * FROM Employee
WHERE salary = (SELECT MIN(salary) FROM Employee);
```

10. Delete an Employee Record having last name=Dev

```
DELETE FROM Employee

WHERE last_name = 'Dev';
```



3.

Eid	Ename	Address	Salary	Commision
1	Amita	Pune	35000	5000
2	Neha	Pune	25000	
3	Sagar	Nasik	28000	2000
4	sneha	Mumbai	19000	
5	Shubham	Mumbai	25000	3000

PrNo	Addr
10	Mumbai
20	Pune
30	Jalgaon

1. Find different locations from where employees belong to?
2. What is maximum and minimum salary?
3. Display the content of employee table according to the ascending order of salary amount.
4. Find the name of employee who lived in Nasik or Pune city.
5. Find the name of employees who does not get commission.
6. Change the city of Amit to Nashik.
7. Find the information of employees whose name starts with 'A'.
8. Find the count of staff from each city
9. Find city wise minimum salary.
10. Find city wise maximum salary having maximum salary greater than 26000

Step 1: Create the `Employee` and `Location` Tables and Insert Data

```
CREATE TABLE Employee (
    Eid INT PRIMARY KEY,
    Ename VARCHAR(50),
    Address VARCHAR(50),
    Salary INT,
    Commission INT
);

INSERT INTO Employee (Eid, Ename, Address, Salary, Commission) VALUES
(1, 'Amita', 'Pune', 35000, 5000),
(2, 'Neha', 'Pune', 25000, NULL),
(3, 'Sagar', 'Nasik', 28000, 2000),
(4, 'Sneha', 'Mumbai', 19000, NULL),
(5, 'Shubham', 'Mumbai', 25000, 3000);

CREATE TABLE Location (
    PrNo INT PRIMARY KEY,
    Addr VARCHAR(50)
);

INSERT INTO Location (PrNo, Addr) VALUES
(10, 'Mumbai'),
(20, 'Pune'),
(30, 'Jalgaon');
```

Answers to Questions

1. **Find Different Locations from Where Employees Belong**

```
SELECT DISTINCT Address
FROM Employee;
```

2. **What is the Maximum and Minimum Salary?**

```
SELECT MAX(Salary) AS max_salary, MIN(Salary) AS min_salary
FROM Employee;
```

3. **Display the Content of Employee Table in Ascending Order of Salary**

```
SELECT * FROM Employee
ORDER BY Salary ASC;
```

4. **Find the Name of Employees Who Lived in Nasik or Pune City**

```
SELECT Ename
FROM Employee
WHERE Address IN ('Nasik', 'Pune');
```

5. **Find the Name of Employees Who Do Not Get Commission**

```
SELECT Ename
FROM Employee
WHERE Commission IS NULL;
```

6. **Change the City of Amit to Nashik**

```
UPDATE Employee
SET Address = 'Nashik'
WHERE Ename = 'Amita';
```

7. **Find the Information of Employees Whose Name Starts with ‘A’**

```
SELECT * FROM Employee
WHERE Ename LIKE 'A%';
```

8. **Find the Count of Staff from Each City**

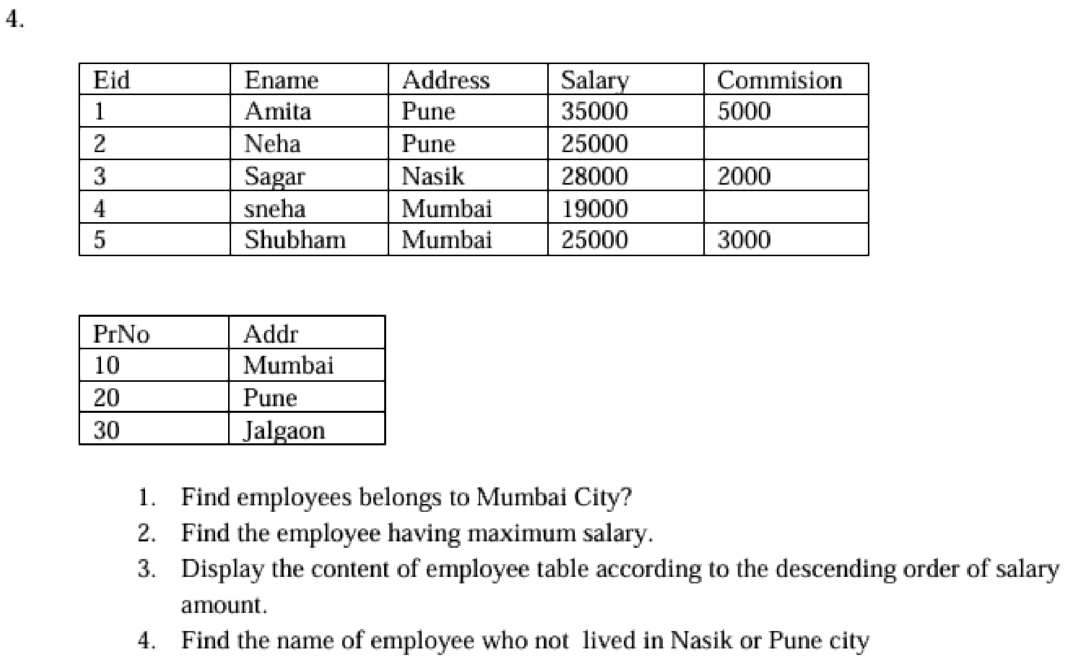
```
SELECT Address, COUNT(*) AS staff_count
FROM Employee
GROUP BY Address;
```

9. **Find City-Wise Minimum Salary**

```
SELECT Address, MIN(Salary) AS min_salary
FROM Employee
GROUP BY Address;
```

10. **Find City-Wise Maximum Salary Having Maximum Salary Greater Than 26000**

```
SELECT Address, MAX(Salary) AS max_salary
FROM Employee
GROUP BY Address
HAVING MAX(Salary) > 26000;
```

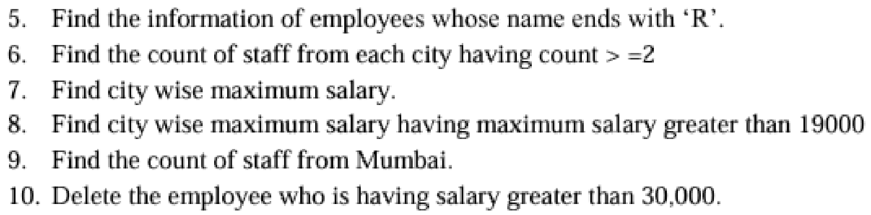


4.

Eid	Ename	Address	Salary	Commision
1	Amita	Pune	35000	5000
2	Neha	Pune	25000	
3	Sagar	Nasik	28000	2000
4	sneha	Mumbai	19000	
5	Shubham	Mumbai	25000	3000

PrNo	Addr
10	Mumbai
20	Pune
30	Jalgaon

1. Find employees belongs to Mumbai City?
2. Find the employee having maximum salary.
3. Display the content of employee table according to the descending order of salary amount.
4. Find the name of employee who not lived in Nasik or Pune city



5. Find the information of employees whose name ends with ‘R’.
6. Find the count of staff from each city having count > =2
7. Find city wise maximum salary.
8. Find city wise maximum salary having maximum salary greater than 19000
9. Find the count of staff from Mumbai.
10. Delete the employee who is having salary greater than 30,000.

1) **Find employees who belong to Mumbai City:**

```
SELECT * FROM Employee WHERE Address = 'Mumbai';
```

2)**Find the employee having the maximum salary:**

```
SELECT * FROM Employee ORDER BY Salary DESC LIMIT 1;
```

3)**Display the content of the employee table according to the descending order of salary amount:**

```
SELECT * FROM Employee ORDER BY Salary DESC;
```

4) **Find the name of employees who do not live in Nasik or Pune city:**

```
SELECT Ename FROM Employee WHERE Address NOT IN ('Nasik', 'Pune');
```

5)**Find the information of employees whose names end with 'R':**

```
SELECT * FROM Employee WHERE Ename LIKE '%R';
```

6) **Find the count of staff from each city having a count greater than 2:**

```
SELECT Address, COUNT(*) AS StaffCount
FROM Employee
GROUP BY Address
HAVING COUNT(*) > 2;
```

7) **Find the city-wise maximum salary:**

```
SELECT Address, MAX(Salary) AS MaxSalary
FROM Employee
GROUP BY Address;
```

8)**Find city-wise maximum salary where the maximum salary is greater than 19000:**

```
SELECT Address, MAX(Salary) AS MaxSalary
FROM Employee
GROUP BY Address
HAVING MAX(Salary) > 19000;
```

9) **Find the count of staff from Mumbai:**

```
SELECT COUNT(*) AS StaffCount
FROM Employee
WHERE Address = 'Mumbai';
```

10) **Delete the employee who is having a salary greater than 30000:**

```
DELETE FROM Employee WHERE Salary > 30000;
```

5. Consider the given database Employee(emp-no,skill,pay-rate) Position(posting-no,skill) Duty-allocation(posting-no,emp-no,day,shift)

1. Find duty allocation details for emp-no 101 for the month of April 2003.

```
SELECT *
FROM Duty_allocation
WHERE emp_no = 101 AND MONTH(day) = 4 AND YEAR(day) = 2003;
```

2. Find the shift details of employee „Bhushan”

```
SELECT d.posting_no, d.day, d.shift
FROM Duty_allocation d
JOIN Employee e ON d.emp_no = e.emp_no
WHERE e.emp_name = 'Bhushan';
```

3. find employees whose rate of pay is more than or equal to the rate of pay of employee „AHIRE‟.

```
SELECT *
FROM Employee
WHERE pay_rate >= (SELECT pay_rate FROM Employee WHERE emp_name = 'AHIRE');
```

4. find the names and pay rates of employee with emp-no less than 1000 whose pay-rate is more than the rate of pay of at least one employee with emp-no greatar than or equal to 1000.

```
SELECT e1.emp_name, e1.pay_rate
FROM Employee e1
WHERE e1.emp_no < 1000
AND e1.pay_rate > (SELECT MIN(e2.pay_rate)
          FROM Employee e2
          WHERE e2.emp_no >= 1000);
```

5. Find the employees with the lowest pay-rate

```
SELECT *
FROM Employee
```

WHERE pay_rate = (SELECT MIN(pay_rate) FROM Employee);

6) Consider the following database. Doctor (Doctor_no, Doctor_name, Address, City). Hospital (Hospital_no, Name, Street, City). Doc_Hosp (Doctor_no, Hospital_no, Date).

1) Find the details of doctors and hospital names to which each doctor has visited:

```
SELECT d.Doctor_no, d.Doctor_name, d.Address, d.City, h.Name AS Hospital_Name
FROM Doctor d
JOIN Doc_Hosp dh ON d.Doctor_no = dh.Doctor_no
JOIN Hospital h ON dh.Hospital_no = h.Hospital_no;
```

2) Find all doctors who have visited a hospital in the same city in which they live:

```
SELECT d.Doctor_no, d.Doctor_name, d.Address, d.City
FROM Doctor d
JOIN Doc_Hosp dh ON d.Doctor_no = dh.Doctor_no
JOIN Hospital h ON dh.Hospital_no = h.Hospital_no
WHERE d.City = h.City;
```

3)Find the hospitals that "Dr. Joshi" has visited:

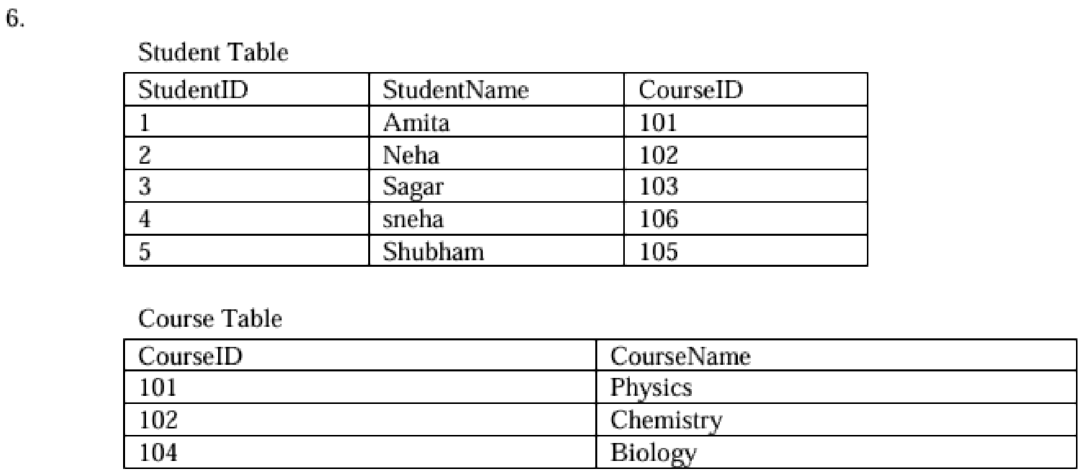
```
SELECT h.Hospital_no, h.Name, h.Street, h.City
FROM Doctor d
JOIN Doc_Hosp dh ON d.Doctor_no = dh.Doctor_no
JOIN Hospital h ON dh.Hospital_no = h.Hospital_no
WHERE d.Doctor_name = 'Dr. Joshi';
```

4)Count the number of doctors who visited "Shree Clinic" on 1st March 2023:

```
SELECT COUNT(*) AS Doctor_Count
FROM Doc_Hosp dh
JOIN Hospital h ON dh.Hospital_no = h.Hospital_no
WHERE h.Name = 'Shree Clinic' AND dh.Date = '2023-03-01';
```

5)Find out how many times "Dr. Joshi" has visited "Shree Clinic":

```
SELECT COUNT(*) AS Visit_Count
FROM Doc_Hosp dh
JOIN Doctor d ON dh.Doctor_no = d.Doctor_no
JOIN Hospital h ON dh.Hospital_no = h.Hospital_no
WHERE d.Doctor_name = 'Dr. Joshi' AND h.Name = 'Shree Clinic';
```



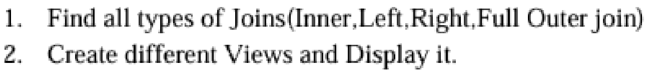
6.

Student Table

StudentID	StudentName	CourseID
1	Amita	101
2	Neha	102
3	Sagar	103
4	sneha	106
5	Shubham	105

Course Table

CourseID	CourseName
101	Physics
102	Chemistry
104	Biology



1. Find all types of Joins(Inner,Left,Right,Full Outer join)
2. Create different Views and Display it.

1. Find all types of Joins (Inner, Left, Right, Full Outer Join)

- **Inner Join** (returns only matching records from both tables):

```
sql
Copy code
SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
FROM Student s
INNER JOIN Course c ON s.CourseID = c.CourseID;
```

- **Left Join** (returns all records from the `Student` table and matching records from the `Course` table):

```
sql
Copy code
SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
FROM Student s
LEFT JOIN Course c ON s.CourseID = c.CourseID;
```

- **Right Join** (returns all records from the `Course` table and matching records from the `Student` table):

  ```
  sql
  Copy code
  SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
  FROM Student s
  RIGHT JOIN Course c ON s.CourseID = c.CourseID;
  ```

- **Full Outer Join** (returns all records when there is a match in either `Student` or `Course` table):

  ```
  sql
  Copy code
  SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
  FROM Student s
  FULL OUTER JOIN Course c ON s.CourseID = c.CourseID;
  ```

  Note: Some SQL databases (like MySQL) do not support full outer join directly. In that case, you can use a `UNION` of `LEFT JOIN` and `RIGHT JOIN` as shown below:

  ```
  sql
  Copy code
  SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
  FROM Student s
  LEFT JOIN Course c ON s.CourseID = c.CourseID
  UNION
  SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
  FROM Student s
  RIGHT JOIN Course c ON s.CourseID = c.CourseID;
  ```

2. Create different Views and Display them

- **View for Students and their Courses (Inner Join)**:

  ```
  sql
  Copy code
  CREATE VIEW StudentCourses AS
  SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
  FROM Student s
  INNER JOIN Course c ON s.CourseID = c.CourseID;

  -- Display the view
  SELECT * FROM StudentCourses;
  ```

- **View for All Students with Course Information (Left Join)**:

  ```
  sql
  Copy code
  CREATE VIEW AllStudentCourses AS
  SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
  FROM Student s
  ```

```
LEFT JOIN Course c ON s.CourseID = c.CourseID;

-- Display the view
SELECT * FROM AllStudentCourses;
```

- **View for All Courses with Enrolled Students (Right Join)**:

```
sql
Copy code
CREATE VIEW CourseStudents AS
SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
FROM Student s
RIGHT JOIN Course c ON s.CourseID = c.CourseID;

-- Display the view
SELECT * FROM CourseStudents;
```

- **View for All Student and Course Information (Full Outer Join)**:

```
sql
Copy code
CREATE VIEW FullStudentCourseInfo AS
SELECT s.StudentID, s.StudentName, s.CourseID, c.CourseName
FROM Student s
FULL OUTER JOIN Course c ON s.CourseID = c.CourseID;

-- Display the view
SELECT * FROM FullStudentCourseInfo;
```

7 .Create Table student(Rollno,Name, Address,Marks)

1. Create different Triggers After Insertion, After Updation,After Deletion,Before Insertion.

2. Create different Views and Display it.

Step 1: Create the `Student` Table

```
sql
Copy code
CREATE TABLE Student (
    Rollno INT PRIMARY KEY,
    Name VARCHAR(50),
    Address VARCHAR(100),
    Marks INT
);
```

1. Create Triggers

- **After Insertion Trigger**: This trigger will execute after a new record is inserted into the `Student` table. For example, it can log the insertion.

  ```
  sql
  Copy code
  CREATE TRIGGER after_student_insert
  AFTER INSERT ON Student
  FOR EACH ROW
  BEGIN
      INSERT INTO Student_Log (Action, Rollno, Name, Address, Marks, 
  ActionDate)
      VALUES ('INSERT', NEW.Rollno, NEW.Name, NEW.Address, NEW.Marks, 
  NOW());
  END;
  ```

- **After Update Trigger**: This trigger will execute after a record is updated in the `Student` table.

  ```
  sql
  Copy code
  CREATE TRIGGER after_student_update
  AFTER UPDATE ON Student
  FOR EACH ROW
  BEGIN
      INSERT INTO Student_Log (Action, Rollno, Name, Address, Marks, 
  ActionDate)
      VALUES ('UPDATE', NEW.Rollno, NEW.Name, NEW.Address, NEW.Marks, 
  NOW());
  END;
  ```

- **After Deletion Trigger**: This trigger will execute after a record is deleted from the `Student` table.

  ```
  sql
  Copy code
  CREATE TRIGGER after_student_delete
  AFTER DELETE ON Student
  FOR EACH ROW
  BEGIN
      INSERT INTO Student_Log (Action, Rollno, Name, Address, Marks, 
  ActionDate)
      VALUES ('DELETE', OLD.Rollno, OLD.Name, OLD.Address, OLD.Marks, 
  NOW());
  END;
  ```

- **Before Insertion Trigger**: This trigger will execute before a new record is inserted into the `Student` table. It can be used, for example, to check data integrity.

  ```
  sql
  Copy code
  CREATE TRIGGER before_student_insert
  BEFORE INSERT ON Student
  FOR EACH ROW
  ```

```
BEGIN
    IF NEW.Marks < 0 OR NEW.Marks > 100 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Marks must be between 0 and 100';
    END IF;
END;
```

Note: The `Student_Log` table should be created beforehand to store the logs from these triggers.

Step 2: Create Views and Display Them

- **View for Students with Passing Marks**: This view shows students who have scored 40 or more marks.

  ```
  sql
  Copy code
  CREATE VIEW Passing_Students AS
  SELECT Rollno, Name, Address, Marks
  FROM Student
  WHERE Marks >= 40;

  -- Display the view
  SELECT * FROM Passing_Students;
  ```

- **View for Students with Marks above 80**: This view shows students with distinction.

  ```
  sql
  Copy code
  CREATE VIEW Distinction_Students AS
  SELECT Rollno, Name, Address, Marks
  FROM Student
  WHERE Marks > 80;

  -- Display the view
  SELECT * FROM Distinction_Students;
  ```

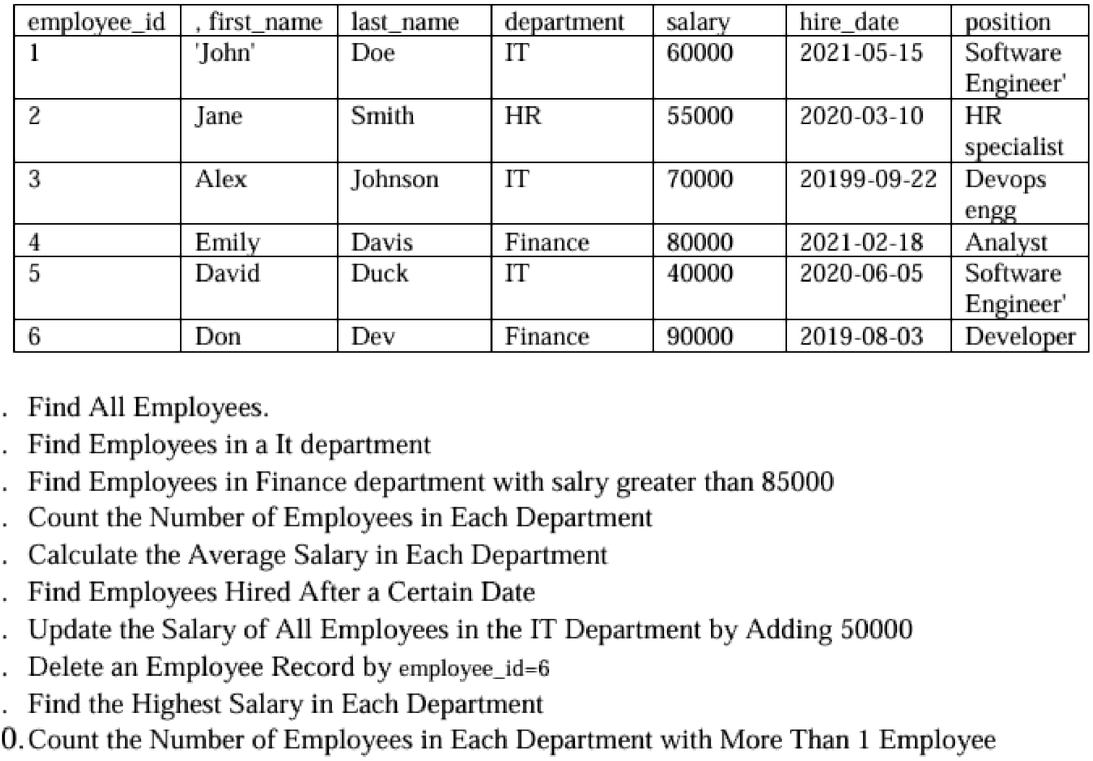
- **View for Students with Marks below 40**: This view shows students who need improvement.

  ```
  sql
  Copy code
  CREATE VIEW Improvement_Students AS
  SELECT Rollno, Name, Address, Marks
  FROM Student
  WHERE Marks < 40;

  -- Display the view
  SELECT * FROM Improvement_Students;
  ```

10. Cursors: 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.

```
DECLARE
   -- Parameterized cursor to fetch data from N_RollCall table
   CURSOR cur_new_rollcall (p_rollno NUMBER) IS
      SELECT RollNo, StudentName, DateOfRollCall
      FROM N_RollCall
      WHERE RollNo = p_rollno;

   v_rollno N_RollCall.RollNo%TYPE;
   v_studentname N_RollCall.StudentName%TYPE;
   v_dateofrollcall N_RollCall.DateOfRollCall%TYPE;

BEGIN
   -- Loop through each record in N_RollCall
   FOR rec IN (SELECT RollNo, StudentName, DateOfRollCall FROM N_RollCall) LOOP
      -- Open the parameterized cursor to check if the record already exists in O_RollCall
      OPEN cur_new_rollcall(rec.RollNo);

      FETCH cur_new_rollcall INTO v_rollno, v_studentname, v_dateofrollcall;

      -- Check if the record already exists in O_RollCall
      IF NOT EXISTS (SELECT 1 FROM O_RollCall WHERE RollNo = rec.RollNo) THEN
      -- If it doesn't exist, insert it into O_RollCall
      INSERT INTO O_RollCall (RollNo, StudentName, DateOfRollCall)
      VALUES (rec.RollNo, rec.StudentName, rec.DateOfRollCall);
    END IF;

    CLOSE cur_new_rollcall;
  END LOOP;

  COMMIT;
END;
```

12.MongoDB:



employee_id	, first_name	last_name	department	salary	hire_date	position
1	'John'	Doe	IT	60000	2021-05-15	Software Engineer'
2	Jane	Smith	HR	55000	2020-03-10	HR specialist
3	Alex	Johnson	IT	70000	20199-09-22	Devops engg
4	Emily	Davis	Finance	80000	2021-02-18	Analyst
5	David	Duck	IT	40000	2020-06-05	Software Engineer'
6	Don	Dev	Finance	90000	2019-08-03	Developer

1. Find All Employees.
2. Find Employees in a It department
3. Find Employees in Finance department with salry greater than 85000
4. Count the Number of Employees in Each Department
5. Calculate the Average Salary in Each Department
6. Find Employees Hired After a Certain Date
7. Update the Salary of All Employees in the IT Department by Adding 50000
8. Delete an Employee Record by employee_id=6
9. Find the Highest Salary in Each Department
10. Count the Number of Employees in Each Department with More Than 1 Employee

1. Find All Employees

```
javascript
Copy code
db.employees.find({})
```

2. Find Employees in the IT Department

```
javascript
Copy code
db.employees.find({ department: "IT" })
```

3. Find Employees in the Finance Department with Salary Greater than 85000

```
javascript
Copy code
db.employees.find({ department: "Finance", salary: { $gt: 85000 } })
```

4. Count the Number of Employees in Each Department

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: "$department", employeeCount: { $sum: 1 } } }
])
```

5. Calculate the Average Salary in Each Department

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: "$department", averageSalary: { $avg: "$salary" } } }
])
```

6. Find Employees Hired After a Certain Date

For example, if the certain date is "2020-01-01":

```
javascript
Copy code
db.employees.find({ hire_date: { $gt: ISODate("2020-01-01") } })
```

7. Update the Salary of All Employees in the IT Department by Adding 50000

```
javascript
Copy code
db.employees.updateMany(
  { department: "IT" },
  { $inc: { salary: 50000 } }
)
```

8. Delete an Employee Record by `employee_id = 6`

```
javascript
Copy code
db.employees.deleteOne({ employee_id: 6 })
```

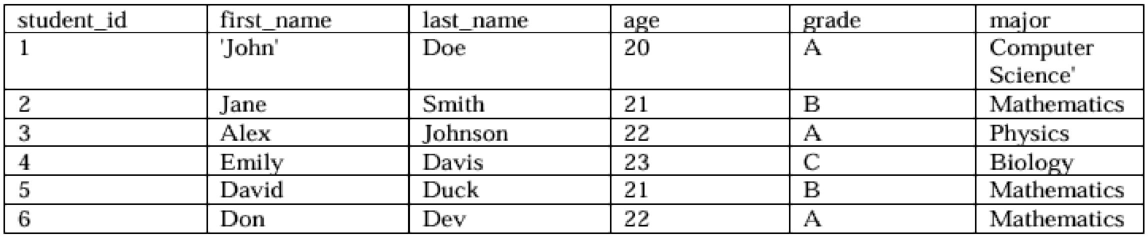
9. Find the Highest Salary in Each Department

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: "$department", maxSalary: { $max: "$salary" } } }
])
```

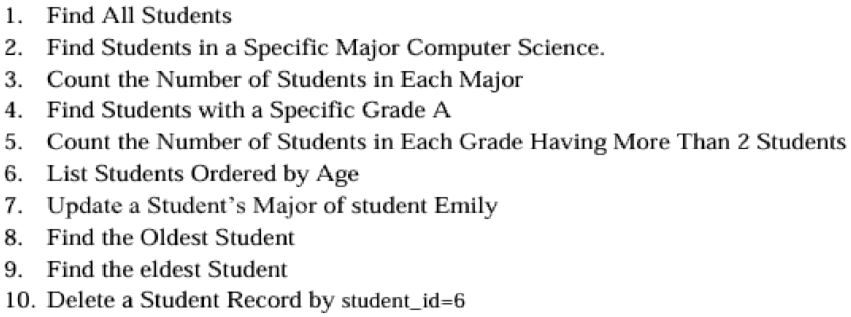
10. Count the Number of Employees in Each Department with More Than 1 Employee

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: "$department", employeeCount: { $sum: 1 } } },
  { $match: { employeeCount: { $gt: 1 } } }
])
```

13. Mongodb



student_id	first_name	last_name	age	grade	major
1	'John'	Doe	20	A	Computer Science'
2	Jane	Smith	21	B	Mathematics
3	Alex	Johnson	22	A	Physics
4	Emily	Davis	23	C	Biology
5	David	Duck	21	B	Mathematics
6	Don	Dev	22	A	Mathematics



1. Find All Students
2. Find Students in a Specific Major Computer Science.
3. Count the Number of Students in Each Major
4. Find Students with a Specific Grade A
5. Count the Number of Students in Each Grade Having More Than 2 Students
6. List Students Ordered by Age
7. Update a Student’s Major of student Emily
8. Find the Oldest Student
9. Find the eldest Student
10. Delete a Student Record by student_id=6

1. Find All Students

```
javascript
Copy code
```

```
db.students.find({})
```

2. Find Students in a Specific Major (e.g., "Computer Science")

```
javascript
Copy code
db.students.find({ major: "Computer Science" })
```

3. Count the Number of Students in Each Major

```
javascript
Copy code
db.students.aggregate([
    { $group: { _id: "$major", count: { $sum: 1 } } }
])
```

4. Find Students with a Specific Grade (e.g., "A")

```
javascript
Copy code
db.students.find({ grade: "A" })
```

5. Count the Number of Students in Each Grade Having More Than 2 Students

```
javascript
Copy code
db.students.aggregate([
    { $group: { _id: "$grade", count: { $sum: 1 } } },
    { $match: { count: { $gt: 2 } } }
])
```

6. List Students Ordered by Age

```
javascript
Copy code
db.students.find({}).sort({ age: 1 }) // Ascending order by age
```

To list in descending order:

```
javascript
Copy code
db.students.find({}).sort({ age: -1 }) // Descending order by age
```

7. Update a Student's Major (e.g., Change Major of Student with `first_name: "Emily"` to "Physics")

```
javascript
Copy code
db.students.updateOne(
    { first_name: "Emily" },
    { $set: { major: "Physics" } }
)
```

8. Find the Oldest Student

```
javascript
Copy code
db.students.find().sort({ age: -1 }).limit(1)
```

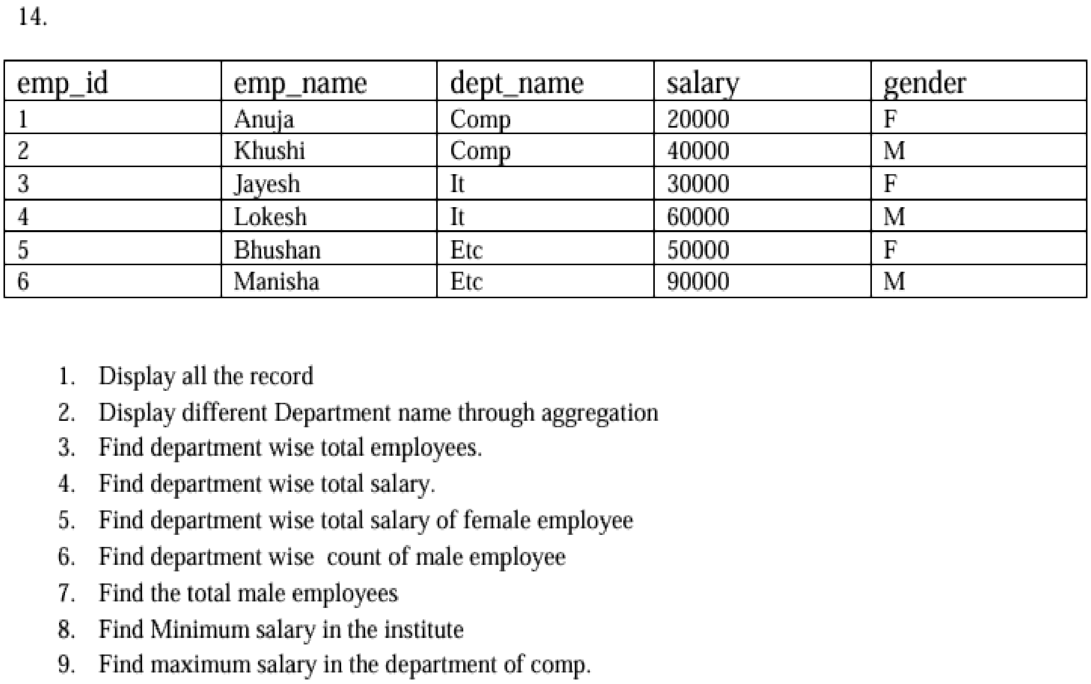
9. Find the Eldest Student

The "eldest" typically means the oldest, so this would be the same as the previous query:

```
javascript
Copy code
db.students.find().sort({ age: -1 }).limit(1)
```

10. Delete a Student Record by `student_id=6`

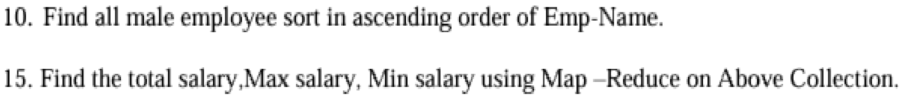
```
javascript
Copy code
db.students.deleteOne({ student_id: 6 })
```



14.

emp_id	emp_name	dept_name	salary	gender
1	Anuja	Comp	20000	F
2	Khushi	Comp	40000	M
3	Jayesh	It	30000	F
4	Lokesh	It	60000	M
5	Bhushan	Etc	50000	F
6	Manisha	Etc	90000	M

1. Display all the record
2. Display different Department name through aggregation
3. Find department wise total employees.
4. Find department wise total salary.
5. Find department wise total salary of female employee
6. Find department wise count of male employee
7. Find the total male employees
8. Find Minimum salary in the institute
9. Find maximum salary in the department of comp.



10. Find all male employee sort in ascending order of Emp-Name.

15. Find the total salary,Max salary, Min salary using Map –Reduce on Above Collection.

1. Display all records

```
javascript
Copy code
db.employees.find({})
```

2. Display different department names through aggregation

```
javascript
```

```
Copy code
db.employees.distinct("dept_name")
```

3. Find department-wise total employees

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: "$dept_name", totalEmployees: { $sum: 1 } } }
])
```

4. Find department-wise total salary

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: "$dept_name", totalSalary: { $sum: "$salary" } } }
])
```

5. Find department-wise total salary of female employees

```
javascript
Copy code
db.employees.aggregate([
  { $match: { gender: "F" } },
  { $group: { _id: "$dept_name", totalFemaleSalary: { $sum: "$salary" } } }
])
```

6. Find department-wise count of male employees

```
javascript
Copy code
db.employees.aggregate([
  { $match: { gender: "M" } },
  { $group: { _id: "$dept_name", maleCount: { $sum: 1 } } }
])
```

7. Find the total male employees

```
javascript
Copy code
db.employees.countDocuments({ gender: "M" })
```

8. Find minimum salary in the institute

```
javascript
Copy code
db.employees.aggregate([
  { $group: { _id: null, minSalary: { $min: "$salary" } } }
])
```

9. Find maximum salary in the department of "Comp"

```
javascript
Copy code
db.employees.aggregate([
  { $match: { dept_name: "Comp" } },
  { $group: { _id: null, maxSalary: { $max: "$salary" } } }
])
```

10. Find all male employees sorted in ascending order of emp_name

```
javascript
Copy code
db.employees.find({ gender: "M" }).sort({ emp_name: 1 })
```

15. Find the total salary, max salary, and min salary using Map-Reduce

```
javascript
Copy code
db.employees.mapReduce(
  function() { emit("salaryStats", { total: this.salary, max: this.salary,
min: this.salary }); },
  function(key, values) {
    return {
      total: Array.sum(values.map(v => v.total)),
      max: Math.max.apply(Math, values.map(v => v.max)),
      min: Math.min.apply(Math, values.map(v => v.min))
    };
  },
  { out: "salary_stats" }
)
```

After running the map-reduce, the results can be viewed in the `salary_stats` collection:

```
javascript
Copy code
db.salary_stats.find()
```