

Chit 01

Here are the MySQL queries based on the provided table structure:

1. Show the **cname, **Acc_Type**, and **Amount** of customers who have a saving account:**

sql

Copy code

```
SELECT C.cname, A.Acc_Type, A.Amount
FROM Customer AS C
JOIN Account AS A
ON C.C_Id = A.C_Id
WHERE A.Acc_Type = 'Saving';
```

2. Display the data using Natural, Left, and Right join:

a) Natural Join:

sql

Copy code

```
SELECT cname, Acc_Type, Amount
FROM Customer
NATURAL JOIN Account
WHERE Acc_Type = 'Saving';
```

b) Left Join:

sql

Copy code

```
SELECT C.cname, A.Acc_Type, A.Amount
FROM Customer AS C
LEFT JOIN Account AS A
ON C.C_Id = A.C_Id;
```

c) Right Join:

sql

Copy code

```
SELECT C.cname, A.Acc_Type, A.Amount  
FROM Customer AS C  
RIGHT JOIN Account AS A  
ON C.C_Id = A.C_Id;
```

3. Display the information of customers living in the same city as 'Pooja':

sql

Copy code

```
SELECT *  
FROM Customer  
WHERE City = (SELECT City FROM Customer WHERE Cname =  
'Pooja');
```

4. Display the information of accounts having less amount than the average amount throughout the bank:

sql

Copy code

```
SELECT *  
FROM Account  
WHERE Amount < (SELECT AVG(Amount) FROM Account);
```

5. Display the C_Id having the maximum amount in account:

sql

Copy code

```
SELECT C_Id  
FROM Account  
WHERE Amount = (SELECT MAX(Amount) FROM Account);
```

6. Display the amount and **Acc_Type of those customers whose amount is the minimum for that **Acc_Type**:**

sql

Copy code

```
SELECT Acc_Type, MIN(Amount) AS Min_Amount  
FROM Account  
GROUP BY Acc_Type;
```

7. Display the amount of those accounts whose amount is higher than any saving account amount:

sql

Copy code

```
SELECT Amount  
FROM Account  
WHERE Amount > (SELECT MIN(Amount) FROM Account WHERE  
Acc_Type = 'Saving');
```

These queries should help you retrieve the required information based on the provided tables and conditions.

Chit 04

Here are the SQL queries to perform the specified tasks on your MySQL database terminal based on the given table structures:

Table Structures:

- **student:** (S_ID, name, dept_name, tot_cred)
- **instructor:** (T_ID, name, dept_name, salary)
- **course:** (course_id, title, dept_name, credits)

SQL Queries

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

sql

Copy code

```
SELECT AVG(salary) AS average_salary
FROM instructor
WHERE dept_name IN (
    SELECT dept_name
    FROM instructor
    GROUP BY dept_name
    HAVING AVG(salary) > 42000
);
```

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

sql

Copy code

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

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

sql

Copy code

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

iv. Find the names of students which contain 'am' as its substring.

sql

Copy code

```
SELECT name
```

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

v. Find the names of students from the computer department that take “DBMS” courses.

sql

Copy code

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

Chit 8

Table Structures:

- **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)

SQL Queries

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

sql

Copy code

```
SELECT i.name, t.semester, t.year, c.title
FROM instructor AS i
JOIN teaches AS t ON i.T_ID = t.T_ID
JOIN course AS c ON t.course_id = c.course_id
ORDER BY t.semester, t.year;
```

ii. Create a view on the student table that retrieves student details.

sql

Copy code

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

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

sql

Copy code

```
ALTER TABLE student  
CHANGE dept_name deptatrment_name VARCHAR(255);
```

iv. Delete student names whose department is NULL.

sql

Copy code

```
DELETE FROM student  
WHERE deptatrment_name IS NULL;
```

Chit 17

Table Structures

- **Employee Table:** (Eid, EName, Address, Salary, Commission)
- **Project Table:** (PNo, Addr)

SQL Queries

Find different locations from where employees belong to?

sql

Copy code

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

1.

What is maximum and minimum salary?

sql

Copy code

```
SELECT MAX(Salary) AS Maximum_Salary, MIN(Salary) AS  
Minimum_Salary  
FROM Employee;
```

2.

Display the content of employee table according to the ascending order of salary amount.

sql

Copy code

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

3.

Find the name of employee who lived in Nasik or Pune city.

sql

Copy code

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

4.

Find the name of employees who do not get a commission.

sql

Copy code

```
SELECT EName  
FROM Employee  
WHERE Commission IS NULL OR Commission = 0;
```

5.

Change the city of Amit to Nashik.

sql

Copy code

```
UPDATE Employee  
SET Address = 'Nashik'  
WHERE EName = 'Amit';
```

6.

Find the information of employees whose name starts with 'A'.

sql

Copy code

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

7.

Find the count of staff from Mumbai.

sql

Copy code

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

8.

Find the count of staff from each city.

sql

Copy code

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

9.

Find the address from where employees are belonging as well as where projects are going on.

sql

Copy code

```
SELECT DISTINCT e.Address  
FROM Employee e  
JOIN Project p ON e.Address = p.Addr;
```

10.

Find city-wise minimum salary.

sql

Copy code

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

11.

Find city-wise maximum salary having maximum salary greater than 26000.

sql

Copy code

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

12.

Delete the employee who is having a salary greater than 30,000.

sql

Copy code

```
DELETE FROM Employee
```

13. `WHERE Salary > 30000;`

Chit 19

Step 1: Create the emp Table

sql

Copy code

```
CREATE TABLE emp (  
    Eno INT PRIMARY KEY AUTO_INCREMENT,  
    Ename VARCHAR(255) NOT NULL,  
    Address VARCHAR(255) DEFAULT 'Nashik',  
    Joindate DATE,  
    Salary DECIMAL(10, 2),  
    Post VARCHAR(255)  
) AUTO_INCREMENT = 101;
```

Step 2: Insert Data into the emp Table

sql

Copy code

```
INSERT INTO emp (Ename, Address, Joindate, Salary,  
Post) VALUES  
( 'Amit', 'Pune', '2023-01-15', 30000.00, 'Manager'),  
( 'Sneha', 'Nashik', '2023-02-20', 25000.00,  
'Developer'),  
( 'Savita', 'Nashik', '2023-03-10', 28000.00,  
'Analyst'),  
( 'Pooja', 'Mumbai', '2023-04-05', 19000.00,  
'Designer'),  
( 'Sagar', 'Mumbai', '2023-05-15', 25000.00,  
'Tester');
```

Step 3: Create Index on the Ename Field

sql

Copy code

```
CREATE INDEX idx_ename ON emp (Ename);
```

Step 4: Create View on Employee Table

sql

Copy code

```
CREATE VIEW emp_view AS  
SELECT Ename, Salary  
FROM emp;
```

Explanation:

1. Table Creation:

- **Eno** is set as the primary key with an auto-increment starting from 101.
- **Ename** has a NOT NULL constraint to ensure every employee has a name.
- **Address** has a default value of 'Nashik'.
- **Joindate** is used to record when the employee joined.
- **Salary** is a decimal type to store salary values.
- **Post** is added as an additional field after the table creation.

2. Inserting Data:

- Five records are inserted into the **emp** table.

3. Creating Index:

- An index named **idx_ename** is created on the **Ename** field to improve query performance.

4. Creating View:

- A view named **emp_view** is created to display only the **Ename** and **Salary** columns from the **emp** table.

You can execute these SQL statements in your MySQL terminal to create the table, insert data, create the index, and create the view successfully.

4o mini

You said:

how to add new col in emp table

ChatGPT said:

ChatGPT

To add a new column to an existing table in MySQL, you can use the **ALTER TABLE** statement. Here's the syntax and an example for adding a new column to the **emp** table.

Syntax

sql

Copy code

```
ALTER TABLE table_name  
ADD column_name column_type [constraints];
```

Example: Add a New Column

Assuming you want to add a new column named **Phone** of type **VARCHAR(15)** to the **emp** table, you would execute the following SQL command:

sql

Copy code

```
ALTER TABLE emp  
ADD Phone VARCHAR(15);
```

Chit 24

Here are the SQL statements to perform the specified operations on the MySQL terminal for the given tables (**Emp**, **works**, **Company**, and **Manager**).

Step 1: Modify the Database for a Particular Company

To update the location of a specific company (e.g., ABC) to Pune, you can use the following SQL statement:

sql

Copy code

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

Step 2: Give All Managers of Mbank a 10% Raise (with conditions)

To give a 10% raise to all managers of Mbank, and a 3% raise if their salary is greater than 20,000, you can use the following statement:

sql

Copy code

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

Step 3: Find Names of All Employees Working in Bosch Company in Pune

To find all employees who work in Bosch and are located in Pune, you can execute the following SQL query:

sql

Copy code

```
SELECT E.ename
FROM Emp AS E
```

```
JOIN works AS W ON E.emp_id = W.emp_id
JOIN Company AS C ON W.c_id = C.c_id
WHERE C.cname = 'Bosch' AND C.city = 'Pune';
```

Step 4: Delete Records in the Works Table for Employees of a Particular Company (e.g., SBC Company) with Salary > 50,000

To delete records from the **works** table for employees of SBC Company whose salary is greater than 50,000, you can use the following SQL command:

sql

Copy code

```
DELETE FROM works
WHERE c_id IN (
    SELECT c_id FROM Company WHERE cname = 'SBC'
) AND sal > 50000;
```

Summary of SQL Statements:

Modify Company Location:

sql

Copy code

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

1.

Give Managers a Raise:

sql

Copy code

```
UPDATE works
SET sal = CASE
    WHEN sal > 20000 THEN sal * 1.03
    ELSE sal * 1.10
```

```
END  
WHERE cname = 'Mbank';
```

2.

Find Employees in Bosch:

sql

Copy code

```
SELECT E.ename  
FROM Emp AS E  
JOIN works AS W ON E.emp_id = W.emp_id  
JOIN Company AS C ON W.c_id = C.c_id  
WHERE C.cname = 'Bosch' AND C.city = 'Pune';
```

3.

Delete High Salary Records:

sql

Copy code

```
DELETE FROM works  
WHERE c_id IN (  
    SELECT c_id FROM Company WHERE cname = 'SBC'  
4. ) AND sal > 50000;
```

Chit25

Here are the SQL queries to implement the requirements using the given tables: **Empl**, **Position**, and **Duty-alloc**.

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

sql

Copy code

```
SELECT *  
FROM Duty-alloc
```

```
WHERE e_no = 123 AND month = 'April' AND year = 2003  
AND shift = 'first';
```

ii. Get the employees whose rate of pay is \geq rate of pay of employees 'Sachin'

sql

Copy code

```
SELECT *  
FROM Emp1  
WHERE pay_rate >= (SELECT pay_rate FROM Emp1 WHERE  
e_name = 'Sachin');
```

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

sql

Copy code

```
CREATE VIEW SalaryStats AS  
SELECT p.post,  
       MIN(e.pay_rate) AS MinSalary,  
       MAX(e.pay_rate) AS MaxSalary,  
       AVG(e.pay_rate) AS AvgSalary  
FROM Emp1 e  
JOIN Position p ON e.post = p.post  
GROUP BY p.post;
```

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

sql

Copy code

```
SELECT da.shift, COUNT(da.e_no) AS EmployeeCount  
FROM Duty-alloc da  
JOIN Emp1 e ON da.e_no = e.e_no
```



```
WHERE e.post = 'manager'  
GROUP BY da.shift;
```

Chit 9

Sample Document Structure

Assuming your documents in the **orderinfo** collection look something like this:

json

Copy code

```
{  
  "cust_id": 123,  
  "cust_name": "abc",  
  "status": "A",  
  "price": 250  
}
```

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

javascript

Copy code

```
db.orderinfo.aggregate([  
  { $match: { status: 'A' } },  
  { $group: { _id: "$cust_id", avgPrice: { $avg:  
"$price" } } }  
]);
```

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

javascript

Copy code

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

iii. Display the customers' information without “_id”

javascript

Copy code

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

iv. Create a simple index on **orderinfo** collection and fire the queries

Create Index

javascript

Copy code

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

Fire the Queries Again After Creating the Index

Find the average price for each customer having status 'A':

javascript

Copy code

```
db.orderinfo.aggregate([
  { $match: { status: 'A' } },
  { $group: { _id: "$cust_id", avgPrice: { $avg:
"$price" } } }
]);
```

1.

Display the status of the customers whose amount/price lies between 100 and 1000:

javascript

Copy code

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

2.

Display the customers' information without “_id”:

javascript

Copy code

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

3.);

Chit 13

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

js

Copy code

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

This query retrieves the **name** of the movies with a budget greater than **1,00,000**.

2. Find the name of the producer who lives in Pune:

js

Copy code

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

This query looks for the documents where the producer's address is **"PUNE"** and returns the **name** of the producer.

3. Update the type of movie from "action" to "horror":

js

Copy code

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

This query updates the **type** field from **"action"** to **"horror"** for all matching documents.

4. Find all the documents produced by "producer1" with their address:

js

Copy code

```
db.movies.find(
  { "producer.name": "producer1" },
```

```
{ name: 1, "producer.address": 1, _id: 0 }  
);
```

This query retrieves all documents where the producer's name is "**producer1**" and includes both the movie name and producer's address.

Chit 14

1. Create collections and insert some sample documents

js

Copy code

```
// Create 'Teachers' collection and insert  
sample documents  
db.Teachers.insertMany([  
  { Tname: "Mr. A", dno: 1, Experience: 5,  
    Salary: 12000, Date_of_Joining: new  
    Date("2018-01-01") },  
  { Tname: "Ms. B", dno: 2, Experience: 7,  
    Salary: 15000, Date_of_Joining: new  
    Date("2017-03-15") },  
  { Tname: "Mr. C", dno: 2, Experience: 4,  
    Salary: 9000, Date_of_Joining: new  
    Date("2019-08-10") }  
]);  
  
// Create 'Department' collection and insert  
sample documents  
db.Department.insertMany([
```

```
{ Dno: 1, Dname: "Computer Science" },
{ Dno: 2, Dname: "Mathematics" },
{ Dno: 3, Dname: "Physics" }
]);

// Create 'Students' collection and insert
sample documents
db.Students.insertMany([
  { Sname: "John", Roll_No: 1, Class: "SE" },
  { Sname: "Jane", Roll_No: 2, Class: "FE" },
  { Sname: "xyz", Roll_No: 3, Class: "TE" },
  { Sname: "Mark", Roll_No: 5, Class: "FE" }
]);
```

2. Find the information about all teachers of Dno=2 and having salary greater than or equal to 10,000

js

Copy code

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

This query retrieves all teachers who belong to **dno = 2** and have a salary greater than or equal to 10,000.

3. Find the student information having Roll_No=2 or Sname='xyz'

js

Copy code

```
db.Students.find(
```

```
    { $or: [{ Roll_No: 2 }, { Sname: "xyz" }] }  
  );
```

This query uses the **\$or** operator to find students whose **Roll_No** is 2 or whose **Sname** is 'xyz'.

4. Update student name whose Roll_No=5

js

Copy code

```
db.Students.updateOne(  
  { Roll_No: 5 },  
  { $set: { Sname: "Updated Name" } }  
);
```

This query updates the **Sname** of the student with **Roll_No** 5.

5. Delete all students whose Class is 'FE'

js

Copy code

```
db.Students.deleteMany(  
  { Class: "FE" }  
);
```

This query deletes all students who belong to the class 'FE'.

6. Apply index on **Students** collection

js

Copy code

```
db.Students.createIndex({ Roll_No: 1 });
```

Chit15

Here's a step-by-step guide with MongoDB queries to achieve the mentioned operations:

1. Create the database **Institute**

js

Copy code

```
use Institute;
```

2. Create the collection **Students**

js

Copy code

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

3. Insert 10 documents with the mentioned structure

js

Copy code

```
db.Students.insertMany([
  { RollNo: 1, StudentName: "Rohan", Age: 16,
    Branch: "Science", Address: { City: "Mumbai",
    State: "Maharashtra" }, Hobbies: ["Cricket",
    "Music"] },
```



```
{ RollNo: 2, StudentName: "Amit", Age: 17,  
Branch: "Commerce", Address: { City: "Pune",  
State: "Maharashtra" }, Hobbies: ["Football",  
"Reading"] },
```

```
{ RollNo: 3, StudentName: "Sneha", Age: 15,  
Branch: "Arts", Address: { City: "Nashik",  
State: "Maharashtra" }, Hobbies: ["Dance",  
"Singing"] },
```

```
{ RollNo: 4, StudentName: "Ankita", Age: 14,  
Branch: "Science", Address: { City: "Pune",  
State: "Maharashtra" }, Hobbies: ["Painting",  
"Drawing"] },
```

```
{ RollNo: 5, StudentName: "Karan", Age: 18,  
Branch: "Science", Address: { City: "Pune",  
State: "Maharashtra" }, Hobbies: ["Swimming",  
"Coding"] },
```

```
{ RollNo: 6, StudentName: "Priya", Age: 16,  
Branch: "Commerce", Address: { City: "Mumbai",  
State: "Maharashtra" }, Hobbies:  
["Photography", "Music"] },
```

```
{ RollNo: 7, StudentName: "Raj", Age: 17,  
Branch: "Arts", Address: { City: "Pune", State:  
"Maharashtra" }, Hobbies: ["Chess", "Cycling"]  
},
```

```
{ RollNo: 8, StudentName: "Anjali", Age: 15,  
Branch: "Science", Address: { City:  
"Aurangabad", State: "Maharashtra" }, Hobbies:  
["Skating", "Music"] },
```

```
    { RollNo: 9, StudentName: "Akash", Age: 18,
      Branch: "Arts", Address: { City: "Pune", State:
        "Maharashtra" }, Hobbies: ["Photography",
        "Cricket"] },
    { RollNo: 10, StudentName: "Neha", Age: 14,
      Branch: "Commerce", Address: { City: "Pune",
        State: "Maharashtra" }, Hobbies: ["Reading",
        "Painting"] }
  ]);
```

4. Display all students' information

js

Copy code

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

5. Display Student information whose age is greater than 15

js

Copy code

```
db.Students.find({ Age: { $gt: 15 }
}).pretty();
```

6. Display Student information sorted on the name field

js

Copy code

```
db.Students.find().sort({ StudentName: 1
}).pretty();
```

7. Update student branch to "Computer" for RollNo 3

js

Copy code

```
db.Students.updateOne(  
  { RollNo: 3 },  
  { $set: { Branch: "Computer" } }  
);
```

8. Remove the document with RollNo 1

js

Copy code

```
db.Students.deleteOne({ RollNo: 1 });
```

9. Display Student information whose name starts with 'A'

js

Copy code

```
db.Students.find({ StudentName: { $regex: '^A'  
} }).pretty();
```

10. Display the total number of documents available in the collection

js

Copy code

```
db.Students.countDocuments();
```

11. Display only the first 5 documents

js

Copy code

```
db.Students.find().limit(5).pretty();
```

12. Display all documents except the first 3

js

Copy code

```
db.Students.find().skip(3).pretty();
```

13. Display the name of students who live in Pune city

js

Copy code

```
db.Students.find({ "Address.City": "Pune" }, {  
  StudentName: 1, _id: 0 }).pretty();
```

14. Display only the names of all students

js

Copy code

```
db.Students.find({}, { StudentName: 1, _id: 0  
}).pretty();
```

15. Drop the **Students** collection

js

Copy code

```
db.Students.drop();
```

Chit 16

1. Create a Database **department**

js

Copy code

```
use department;
```

2. Create a Collection **teacher** and Insert Documents

js

Copy code

```
db.teacher.insertMany([
  { name: "John", department: "Computer",
    experience: 5, salary: 50000 },
  { name: "Amit", department: "Math",
    experience: 10, salary: 60000 },
  { name: "Priya", department: "Computer",
    experience: 3, salary: 45000 },
  { name: "Anil", department: "Physics",
    experience: 7, salary: 55000 },
  { name: "Ravi", department: "Math",
    experience: 12, salary: 65000 },
  { name: "Neha", department: "Physics",
    experience: 8, salary: 58000 },
  { name: "Sneha", department: "Computer",
    experience: 9, salary: 70000 }
]);
```

3. Display Department-Wise Average Salary

You can use the MongoDB aggregation framework to calculate the average salary for each department.

js

Copy code

```
db.teacher.aggregate([
```

```
    { $group: { _id: "$department", avgSalary: {
$avg: "$salary" } } }
]);
```

4. Display the Number of Employees Working in Each Department

This query will count the number of employees (teachers) per department.

js

Copy code

```
db.teacher.aggregate([
  { $group: { _id: "$department",
employeeCount: { $sum: 1 } } }
]);
```

5. Display Department-Wise Minimum Salary

To get the minimum salary for each department:

js

Copy code

```
db.teacher.aggregate([
  { $group: { _id: "$department", minSalary: {
$min: "$salary" } } }
]);
```

6. Apply Index on `department` Field

Creating an index on the `department` field will optimize queries related to department-wise search.

js

Copy code

```
db.teacher.createIndex({ department: 1 });
```

7. Drop the Index

To drop the created index on **department**:

js

Copy code

```
db.teacher.dropIndex({ department: 1 });
```

Chit 18

1. Statewise Population Calculation Using MapReduce

Map Function: This function will map each document by its **state** and emit the population for each state.

js

Copy code

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

Reduce Function: The reduce function will sum up the populations for each state.

js

Copy code

```
var reducePopulation = function(state,
populations) {
    return Array.sum(populations);
};
```

Executing the MapReduce:

js

Copy code

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

// To display the results:

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

2. Citywise Population Calculation Using MapReduce

Map Function: This function maps each document by its **city** and emits the population for each city.

js

Copy code

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


Reduce Function: The reduce function will sum up the populations for each city.

js

Copy code

```
var reducePopulation = function(city,
populations) {
    return Array.sum(populations);
};
```

Executing the MapReduce:

js

Copy code

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

// To display the results:

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

3. Typewise Population Calculation Using MapReduce

Map Function: This function maps each document by its **type** (e.g., urban or rural) and emits the population for each type.

js

Copy code

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

Reduce Function: The reduce function will sum up the populations for each type.

js

Copy code

```
var reducePopulation = function(type,  
populations) {  
    return Array.sum(populations);  
};
```

Executing the MapReduce:

js

Copy code

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

// To display the results:

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