Slip 1

Q1) 1)List the department and the total salary for each department, sorted by total

salary in descending order:

```
->CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;
CREATE TABLE Worker (
WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
FIRST_NAME CHAR(25),
LAST_NAME CHAR(25),
SALARY INT(15),
JOINING_DATE DATETIME,
DEPARTMENT CHAR(25));
INSERT INTO Worker
(WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT) VALUES
(001, &#39;Monika&#39;, &#39;Arora&#39;, 100000, &#39;21-02-20 09.00.00&#39;, &#39;HR&#39;),
(002, &#39;Niharika&#39;, &#39;Verma&#39;, 80000, &#39;21-06-11 09.00.00&#39;, &#39;Admin&#39);
```

```
SELECT department, SUM(salary) AS total_salary
FROM Worker
GROUP BY department
ORDER BY total_salary DESC;
```

2)Find the average salary for each department and display only those
departments where the average salary is above $100,000, sorted by average
salary in descending order
=>

```
SELECT Department, AVG(Salary) AS avg_salary FROM Worker GROUP BY Department HAVING
AVG(Salary) > 100000 ORDER BY avg_salary DESC;
```

Q.2)Model the following Property system as a document database. Consider a set of
Property, Owner. One owner can buy many properties.
1. Assume appropriate attributes and collections as per the query
requirements.
2. Insert at least 05 documents in each collection. [3]

```
use MScDS
db.createCollection('Property')
db. Property.insertMany {
    "_id": ObjectId("property_id"),
    "address": "123 Main St",
    "type": "Residential",
    "size": "2000 sq ft",
    "price": 250000,
    "owner_id": ObjectId("owner_id")}
```

```
Owner Collection:
1. {
    "_id": ObjectId("609ed2511d5bc60e46b1dcaa"),
    "name": "Mr. Patil",
    "contact": "patil@example.com",
    "properties_owned": [
      {
        "property_id": ObjectId("609ed20a1d5bc60e46b1dca1"),
        "purchase_date": ISODate("2024-03-29"),
        "price_paid": 250000
      },
      {
        "property_id": ObjectId("609ed20a1d5bc60e46b1dca4"),
        "purchase_date": ISODate("2023-06-15"),
        "price_paid": 600000
      }
    ]
}
```

3. Answer the following Queries

a)Display area wise property details. [3]

```
db.Property.aggregate([
  {
    $group: {
      _id: "$area",
      properties: { $push: "$$ROOT" }
    }
  }
])
```

b)Display property owned by &#39;Mr.Patil&#39; having minimum rate [3]

```
db.Owner.aggregate([
  {
    $match: { name: "Mr. Patil" }
  },
  {
    $unwind: "$properties_owned"
  },
  {
    $lookup: {
      from: "Property",
      localField: "properties_owned.property_id",
      foreignField: "_id",
      as: "property_details"
    }
  },
  {
    $unwind: "$property_details"
  },
  {
    $sort: { "property_details.price": 1 }
  },
  {
    $limit: 1
  }
])
```

c)Give the details of owner whose property is at “Nashik”. [4]

```
db.Property.aggregate([
   {
     $match: { area: "Nashik" }
   },
   {
     $lookup: {
       from: "Owner",
       localField: "_id",
       foreignField: "properties_owned.property_id",
       as: "owners"
     }
   },
   {
     $unwind: "$owners"
   }
])
```

d)Display area of property whose rate is less than 100000. [4]

```
db.Property.find({ price: { $lt: 100000 } }, { area: 1 })
```

Slip 2

Q1)Consider the Employee table

Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)
Write a query to insert 5 rows in it that as per the query requirements.

```
 CREATE TABLE Employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(255),
    job_name VARCHAR(255),
    manager_id INT,
    hire_date DATE,
    salary DECIMAL(10, 2)
);
INSERT INTO Employee (emp_id, emp_name, job_name, manager_id, hire_date, salary)
VALUES
  (1, 'John Doe', 'Manager', NULL, '2022-01-05', 2500),
  (2, 'Jane Smith', 'Engineer', 1, '2022-02-10', 2200),
  (3, 'Michael Johnson', 'Engineer', 1, '2022-03-15', 2300),
  (4, 'Emily Brown', 'Analyst', 2, '2022-04-20', 2100),
  (5, 'David Lee', 'Analyst', 3, '2022-05-25', 2400);
```

1)List the average salary of each job title, sorted in descending order of
average salary:
=>
SELECT job_name, AVG(salary) AS avg_salary FROM Employee GROUP BY job_name ORDER BY
avg_salary DESC
2)List the department and the maximum salary among employees who joinedafter 1995, sorted by
maximum salary in descending order

```
SELECT department, MAX(salary) AS max_salary
FROM employees
WHERE join_date > '1995-01-01'
GROUP BY department
ORDER BY max_salary DESC;
```

Q.2)Model the following system as a document database.Consider a database of newspaper, publisher, and city.Different publisher publishes various newspapers in different cities
2. Assume appropriate attributes and collections as per the query
requirements. [3]
3. Insert at least 5 documents in each collection. [3]

```
db.Newspaper.insertMany[(
 {
    "_id": ObjectId("..."),
    "name": "The Times",
    "language": "English",
    "publisher_id": ObjectId("..."),
    "city_id": ObjectId("...")
}])
db.Publisher.insertMany([{
    "_id": ObjectId("..."),
    "name": "ABC Publications",
    "state": "Maharashtra"
}])
db.City.insertMany([{
    "_id": ObjectId("..."),
    "name": "Nashik",
    "state": "Maharashtra"
}
```

4. Answer the following Queries.
a. List all newspapers available “NASHIK” city [3]

```
db.Newspaper.aggregate([
  {
    $lookup: {
      from: "City",
      localField: "city_id",
      foreignField: "_id",
      as: "city"
    }
  },
  {
    $match: { "city.name": "Nashik" }
  },
  {
    $project: { name: 1, language: 1 }
  }
])
```
b. List all the newspaper of “Marathi” language [3]
```
db.Newspaper.find({ language: "Marathi" }, { name: 1 })
```
c. Count no. of publishers of “Gujrat” state [4]
```
db.Publisher.count({ state: "Gujarat" })
```
d. Write a cursor to show newspapers with highest sale in Maharashtra state[4]
```
var cursor = db.Newspaper.aggregate([
  {
    $lookup: {
      from: "City",
      localField: "city_id",
      foreignField: "_id",
      as: "city"
    }
  },
  {
    $match: { "city.state": "Maharashtra" }
  },
  {
    $group: {
      _id: "$name",
      total_sales: { $sum: 1 }
    }
  },
  {
    $sort: { total_sales: -1 }
  }
]);

while (cursor.hasNext()) {
  printjson(cursor.next());
}
```

Slip 3

Q1)Consider the Worker table
Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARTMENT)
CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;
CREATE TABLE Worker (

```
WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
FIRST_NAME CHAR(25),
LAST_NAME CHAR(25),
SALARY INT(15),
JOINING_DATE DATETIME,
DEPARTMENT CHAR(25));
INSERT INTO Worker
(WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT) VALUES
(001, &#39;Monika"&#39;Arora&#39;, 100000, &#39;21-02-20 09.00.00&#39;, &#39;HR&#39;),
(002, &#39;Niharika&#39;, &#39;Verma&#39;, 80000, &#39;21-06-11 09.00.00&#39;, &#39;Admin&#39);
```

Write a query to insert 5 rows in it as per the query requirements.
1)List the number of workers in each department who joined after January 1, 2021, sorted by department name:
2)Find the department with the highest number of workers earning a salary greater than $90,000, sorted by department name:
➔

```
CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;
CREATE TABLE Worker (
WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
FIRST_NAME CHAR(25),
LAST_NAME CHAR(25),
SALARY INT(15),
JOINING_DATE DATETIME,
DEPARTMENT CHAR(25)
);
INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
VALUES
  (1, 'John', 'Doe', 95000, '2021-03-15', 'Finance'),
  (2, 'Jane', 'Smith', 87000, '2021-05-20', 'IT'),
  (3, 'Michael', 'Johnson', 105000, '2021-02-10', 'Finance'),
  (4, 'Emily', 'Brown', 92000, '2021-07-08', 'Marketing'),
  (5, 'David', 'Lee', 98000, '2021-04-25', 'IT');
SELECT Department, COUNT(*) AS num_workers FROM Worker WHERE JOINING_DATE > '2021-01-01'
GROUP BY Department ORDER BY Department;
SELECT DEPARTMENT, COUNT(*) AS NumOfWorkers
FROM Worker
WHERE SALARY > 90000
GROUP BY DEPARTMENT
ORDER BY NumOfWorkers DESC, DEPARTMENT;
```

Q2) 1. Model the following system as a document database. Consider employee and department's information.

2. Assume appropriate attributes and collections as per the query requirements.
[3]
3. Insert at least 5 documents in each collection. [3]

```
db.Employee.insertMany([
{
    "_id": ObjectId("..."),
    "first_name": "John",
    "last_name": "Doe",
    "salary": 75000,
    "department_id": ObjectId("...")
}])
db.Department.insertMany([
 {
    "_id": ObjectId("..."),
    "name": "Finance",
    "location": "New York"
}])
```

4. Answer the following Queries.
a. Display name of employee who has highest salary [3]

```
db.Employee.find().sort({ salary: -1 }).limit(1).project({ first_name: 1, last_name: 1 })
```

b. Display biggest department with max. no. of employees [3]

```
     db.Employee.aggregate([
 {
  $group: {
   _id: "$department_id",
   count: { $sum: 1 }
  }
 },
 {
  $sort: { count: -1 }
 },
 {
  $lookup: {
   from: "Department",
   localField: "_id",
   foreignField: "_id",
   as: "department"
  }
 },
 {
  $limit: 1
 },
 {
  $project: {
      department_name: { $arrayElemAt: ["$department.name", 0] },
      num_employees: "$count"
    }
  }
])
```

c. Write a cursor which shows department wise employee information [4]

```
      var cursor = db.Department.find();

while (cursor.hasNext()) {
  var department = cursor.next();
  var departmentEmployees = db.Employee.find({ department_id: department._id }).toArray();
  print("Department: " + department.name);
  departmentEmployees.forEach(function(employee) {
    printjson(employee);
  });
}
```

d. List all the employees who work in Sales dept and salary &gt; 50000 [4]

```
     db.Employee.find({ department_id: ObjectId("sales_department_id"), salary: { $gt: 50000 } })
```

Slip 4

Q1)Consider the Patient table

Patient(PatientID,Name,DateOfBirth,Gender,admit_date,ward_no,City)

Write a query to insert 5 rows in it that as per the query requirements.

1)List the number of patients admitted to each ward, sorted by ward number:

2)Find the average age of patients admitted to each ward, and display only those wards where the average age is below 40, sorted by average age in descending order

=>

```
CREATE TABLE Patient (
    PatientID INT PRIMARY KEY,
    Name VARCHAR(255),
    DateOfBirth DATE,
    Gender VARCHAR(10),
    admit_date DATE,
    ward_no INT,
    City VARCHAR(255)
);
INSERT INTO Patient (PatientID, Name, DateOfBirth, Gender, admit_date, ward_no, City)
VALUES
  (1, 'John Doe', '1990-05-15', 'Male', '2022-03-10', 101, 'New York'),
  (2, 'Jane Smith', '1985-08-20', 'Female', '2022-03-12', 102, 'Los Angeles'),
  (3, 'Michael Johnson', '1982-11-30', 'Male', '2022-03-14', 103, 'Chicago'),
  (4, 'Emily Brown', '1978-04-25', 'Female', '2022-03-16', 101, 'San Francisco'),
  (5, 'David Lee', '1995-09-10', 'Male', '2022-03-18', 102, 'Boston');

SELECT ward_no, COUNT(*) AS num_patients
FROM Patient
GROUP BY ward_no
ORDER BY ward_no;

SELECT ward_no,
    AVG(DATEDIFF(CURRENT_DATE(), DateOfBirth) / 365) AS avg_age
FROM Patient
GROUP BY ward_no
HAVING avg_age < 40
ORDER BY avg_age DESC;
```

Q2). Model the following information system as a document database.
Consider hospitals around Nashik. Each hospital may have one or more specializations like Pediatric, Gynaec, Orthopedic, etc. A person can recommend/provide review for a hospital. A doctor can give service to one or more hospitals.

2. Assume appropriate attributes and collections as per the query requirements. [3]

3. Insert at least 10 documents in each collection. [3]

```
                    Hospital.insertMany([{

    "_id": ObjectId("..."),
    "name": "City Hospital",
    "city": "Nashik",
    "specializations": ["Pediatric", "Gynaec", "Orthopedic"],
    "rating": 4.5
}])
Doctor.insertMany([{
    "_id": ObjectId("..."),
    "name": "Dr. Deshmukh",
    "hospitals": [ObjectId("..."), ObjectId("...")]
}])
```

4. Answer the following Queries

a. List the names of hospitals with……….. specialization. [3]

```
            db.Hospital.find({ specializations: "Pediatric" }, { name: 1 })
```

b. List the Names of all hospital located in ……. city [3]

```
      db.Hospital.find({ city: "Nashik" }, { name: 1 })
```

c. List the names of hospitals where Dr. Deshmukh visits [4]

```
db.Hospital.find({ name: { $in: db.Person.findOne({ name: "Dr. Deshmukh" }).hospitals_visited } }, {
name: 1 })
```

d. List the names of hospitals whose rating &gt;=4 [4]

```
        db.Hospital.find({ rating: { $gte: 4 } }, { name: 1 })
```

Slip 5

Q1)Consider the Patient table
Patient(PatientID,Name,DateOfBirth,Gender,admit_date,ward_no,City)
Write a query to insert 5 rows in it that as per the query requirements.
1)List the number of male and female patients admitted to each ward, sorted by
ward number and gender:
2)Find the ward with the highest number of patients admitted, and display the
top 3 wards with the highest number of patients, sorted by the number of
patients in descending order

```
CREATE TABLE Patient (
   PatientID INT PRIMARY KEY,
   Name VARCHAR(255),
   DateOfBirth DATE,
   Gender VARCHAR(10),
   admit_date DATE,
   ward_no INT,
   City VARCHAR(255)
);
INSERT INTO Patient (PatientID, Name, DateOfBirth, Gender, admit_date, ward_no, City)
VALUES
  (1, 'John Doe', '1990-05-15', 'Male', '2022-03-10', 101, 'New York'),
  (2, 'Jane Smith', '1985-08-20', 'Female', '2022-03-12', 102, 'Los Angeles'),
  (3, 'Michael Johnson', '1982-11-30', 'Male', '2022-03-14', 103, 'Chicago'),
  (4, 'Emily Brown', '1978-04-25', 'Female', '2022-03-16', 101, 'San Francisco'),
  (5, 'David Lee', '1995-09-10', 'Male', '2022-03-18', 102, 'Boston');
SELECT ward_no, Gender, COUNT(*) AS num_patients
FROM Patient
GROUP BY ward_no, Gender
ORDER BY ward_no, Gender;
SELECT ward_no, COUNT(*) AS num_patients
FROM Patient
GROUP BY ward_no
ORDER BY num_patients DESC
LIMIT 3;
```

Q2). Model the following database. Many employees working on one project. A

company has various ongoing projects.
2. Assume appropriate attributes and collections as per the query requirements.
[3]
3. Insert at least 5 documents in each collection. [3]

Employee Collection:
```
1. {
   "_id": ObjectId("..."),
   "name": "John Doe",
   "employee_id": "EMP001",
   "projects": ["ProjectA", "ProjectB"]
}
2. {
   "_id": ObjectId("..."),
   "name": "Jane Smith",
   "employee_id": "EMP002",
   "projects": ["ProjectA", "ProjectC"]
}
3. {
   "_id": ObjectId("..."),
   "name": "Michael Johnson",
   "employee_id": "EMP003",
   "projects": ["ProjectB"]
}
4. {
   "_id": ObjectId("..."),
   "name": "Emily Brown",
   "employee_id": "EMP004",
   "projects": ["ProjectC", "ProjectD"]
}
5. {
   "_id": ObjectId("..."),
   "name": "David Lee",
   "employee_id": "EMP005",
   "projects": ["ProjectD"]
}
```

Project Collection:
```
1. {
   "_id": ObjectId("..."),
   "name": "ProjectA",
   "project_type": "TypeA",
   "duration_months": 5
}
2. {
   "_id": ObjectId("..."),
   "name": "ProjectB",
   "project_type": "TypeB",
   "duration_months": 4
}
3. {
   "_id": ObjectId("..."),
   "name": "ProjectC",
   "project_type": "TypeA",
   "duration_months": 6
}
4. {
   "_id": ObjectId("..."),
   "name": "ProjectD",
   "project_type": "TypeC",
   "duration_months": 3
}
5. {
   "_id": ObjectId("..."),
   "name": "ProjectE",
   "project_type": "TypeB",
   "duration_months": 2
}
```

4. Answer the following Queries

a. List all names of projects where Project_type =….. [3]

```
db.Project.find({ project_type: "TypeA" }, { name: 1 })
```

b. List all the projects with duration greater than 3 months [3]

```
db.Project.find({ duration_months: { $gt: 3 } }, { name: 1 })
```

c. Count no. of employees working on ……..project [4]

```
db.Employee.aggregate([
 { $match: { projects: "ProjectA" } },
 { $group: { _id: null, count: { $sum: 1 } } }
])
```

d. List the names of projects on which Mr. Patil is working [4]

```
db.Project.find({ name: { $in: db.Employee.findOne({ name: "Mr. Patil" }).projects } }, { name: 1 })
```

Slip 6

Q1)Consider the Worker table

Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_D

ATE,DEPARTMENT)
Write a query to insert 5 rows in it as per the query requirements.
1)List the department and the number of workers who joined in February 2021
and have a salary greater than $80,000, sorted by department name:
2)Find the department with the highest average salary among departments with
at least 2 workers, sorted by average salary in descending order:

```
  CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;
CREATE TABLE Worker (
WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
FIRST_NAME CHAR(25),
LAST_NAME CHAR(25),
SALARY INT(15),
JOINING_DATE DATETIME,
DEPARTMENT CHAR(25)
);
INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
VALUES
  (1, 'John', 'Doe', 95000, '2021-03-15', 'Finance'),
  (2, 'Jane', 'Smith', 87000, '2021-05-20', 'IT'),
  (3, 'Michael', 'Johnson', 105000, '2021-02-10', 'Finance'),
  (4, 'Emily', 'Brown', 92000, '2021-07-08', 'Marketing'),
  (5, 'David', 'Lee', 98000, '2021-04-25', 'IT');
```

Q2)Model the following information as a Mongodb database in mongoshell. A customer can take different policies and get the benefit. There are different types of policies provided by various companies. Assume appropriate attributes and collections as per the query requirements. Insert at least 5 documents in each collection.

```
db.createCollection("customers")
db.customers.insertMany([ { name: "Amaan", policy: "Komal Jeevan", premium_amount: 200,type:
"Yearly",company:"Life Insurers Ltd" }, { name: "Vanshay", policy: "Health Guard", premium_amount:
250,type: "Monthly",company:"Health Insurers Inc"}, { name: "Adityya", policy: "Retirement Plus",
premium_amount: 300,type: "Half Yearly",company:"Retirement Solutions" }, { name: "Rushikesh",
policy: "Komal Jeevan", premium_amount: 220,type: "Yearly",company:"Life Insurers Ltd"}, { name:
"Sneha", policy: "Accident Protector", premium_amount: 270,type: "Monthly",company:"Health
Insurers Inc"} ])
```

List the details of customers who have taken "Komal Jeevan" Policy

```
        db.customers.find({ policy: "Komal Jeevan" })
```

Display average premium amount

```
      db.customers.aggregate([{ $group: { _id: null, avg_premium: { $avg: "$premium_amount" } } }])
```

Increase the premium amount by 5% for policy type="Monthly"

```
db.policies.updateMany({ type: "Monthly" }, { $mul: { premium_amount: 1.05 } })
```

Count the number of customers who have taken policy type "half yearly

```
db.customers.find({ "type": "Half Yearly" }).count()
```

Slip 7

Q1)Consider the Employee table

Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)

Write a query to insert 5 rows in it that as per the query requirements.

```
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(255),
   job_name VARCHAR(255),
   manager_id INT,
   hire_date DATE,
   salary DECIMAL(10, 2)
);
INSERT INTO Employee (emp_id, emp_name, job_name, manager_id, hire_date, salary)
        VALUES
          (1, 'John Doe', 'Manager', NULL, '2022-01-05', 2500),
          (2, 'Jane Smith', 'Engineer', 1, '2022-02-10', 2200),
          (3, 'Michael Johnson', 'Engineer', 1, '2022-03-15', 2300),
          (4, 'Emily Brown', 'Analyst', 2, '2022-04-20', 2100),
          (5, 'David Lee', 'Analyst', 3, '2022-05-25', 2400);
```

1)Find the total number of employees in each department whose salary is above $2000, sorted by department name:

```
          SELECT job_name, COUNT(*) AS num_employees
FROM Employee
WHERE salary > 2000
GROUP BY job_name
ORDER BY job_name;
```

2)List the manager_id and the number of employees managed by each manager who manages more than one employee, sorted by manager_id:

```
             SELECT manager_id, COUNT(emp_id) AS num_managed_employees
FROM Employee
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING COUNT(emp_id) > 1
ORDER BY manager_id;
```

Q2) 1. Model the following information as a Mongodb database in mongo shell. A customer operates his bank account, does various transactions and get the banking services Assume appropriate attributes and collections as per the query requirements.
Insert at least 5 documents in each collection.

db.createCollection("customers")
db.createCollection("transactions")
db.customers.insertMany([{ firstName: "Amaan", acctype: "Saving", branch: "A" }, { firstName: "Sara", acctype: "Loan", branch: "B" }, { firstName: "Sam", acctype: "Saving", branch: "A" }, { firstName: "Denzil", acctype: "Loan", branch: "C" }, { firstName: "Merlin", acctype: "Saving", branch: "B" }])

db.transactions.insertMany([{ customerId: 1, date: "2020-01-01" }, { customerId: 2, date: "2020-01-01" }, { customerId: 3, date: "2021-03-15" }, { customerId: 4, date: "2020-01-01" }, { customerId: 5, date: "2020-01-01" }]) List names of all customers whose first name starts with a "S" db.customers.find({ firstName: /^S/ }, { _id: 0, firstName: 1 }) List the names customers where acctype="Saving" db.customers.find({ acctype: "Saving" }, { _id: 0, firstName: 1 })

List names of all customers whose first name starts with a "S"
```
        db.customers.find({ firstName: /^S/ }, { _id: 0, firstName: 1 })
```
List the names customers where acctype="Saving"
```
        db.customers.find({ acctype: "Saving" }, { _id: 0, firstName: 1 })
```
Count the total number of loan account holders of a specific branch
```
         db.customers.count({ acctype: "Loan", branch: "A" })
         db.customers.count({ acctype: "Loan", branch: "C" })
```

Slip 8

Consider the Employee table
Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)
Write a query to insert 5 rows in it that as per the query requirements.
```
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(255),
   job_name VARCHAR(255),
   manager_id INT,
   hire_date DATE,
   salary DECIMAL(10, 2)
);
INSERT INTO Employee (emp_id, emp_name, job_name, manager_id, hire_date, salary)
     VALUES
       (1, 'John Doe', 'Manager', NULL, '2022-01-05', 2500),
       (2, 'Jane Smith', 'Engineer', 1, '2022-02-10', 2200),
       (3, 'Michael Johnson', 'Engineer', 1, '2022-03-15', 2300),
       (4, 'Emily Brown', 'Analyst', 2, '2022-04-20', 2100),
       (5, 'David Lee', 'Analyst', 3, '2022-05-25', 2400);
```

1)Find the department with the highest average salary among departments with at least 3 employees:

```
    SELECT job_name AS department, AVG(salary) AS avg_salary
FROM Employee
GROUP BY job_name
HAVING COUNT(*) >= 3
ORDER BY avg_salary DESC
LIMIT 1;
```

2)Find the top 3 departments with the highest total salary expenditure, sorted by total salary expenditure in descending order

```
        SELECT job_name AS department, SUM(salary) AS total_salary_expenditure
FROM Employee
GROUP BY job_name
ORDER BY total_salary_expenditure DESC
LIMIT 3;
```

Q1)Model the following inventory information as a Mongodb database in mongoshell. The inventory keeps track of various items. The items are tagged in various categories. Items may be kept in various warehouses and each warehouse keeps track of the quantity of the item. Assume appropriate attributes and collections as per the query requirements.

```
 db.createCollection("items")
 db.createCollection("inventories")
 db.items.insertMany([ { name: "Laptop", tags: ["electronics",
"laptop","Gadgets","Tech","Office","Gaming"], status: "A", height:10 }, { name: "T-shirt", tags:
["clothing", "t-shirt","Fashion","Fabric"], status: "B", height: 5 }, { name: "Notebook", tags: ["office
supplies", "notebook","Stationary","School","Official"], status: "C", height: 8 }, { name: "Chair", tags:
["furniture", "chair","Home","Decor"], status: "A", height: 12 }, { name: "Basketball", tags: ["sports
equipment", "basketball"], status: "B", height: 10 } ]);
 db.inventories.insertMany([ { item: "Laptop", warehouse: "Warehouse A", quantity: 500 }, { item: "T-
shirt", warehouse: "Warehouse B", quantity: 100 }, { item: "Notebook", warehouse: "Warehouse C",
quantity: 200 }, { item: "Chair", warehouse: "Warehouse A", quantity: 50 }, { item: "Basketball",
warehouse: "Warehouse B", quantity: 400 } ]);
 db.inventories.insertOne({ item: "Planner", warehouse: "Warehouse B", quantity: 10 })
db.items.insertOne({ name: "Planner", tags: ["office supplies",
"notebook","Stationary","School","Official"], status: "C", height: 8 })
```

List all the items where quantity is greater than 300

```
        db.inventories.find({ quantity: { $gt: 300 } }, { _id: 0, item: 1, quantity: 1 })
```

List all items which have less than 5 tags

```
                db.items.find({ $expr: { $lt: [{ $size: "$tags" }, 5] } })
```
List all items having status equal to "B" or having quantity less than 50 and height of the product greater than 8
```
                db.items.find({ $or: [ { status: "B" },{ $and: [{ quantity: { $lt: 50 } }, { height: { $gt: 8 } }] } ]});
```
Find all warehouses that keep the item "Planner" and have in-stock quantity less than 20
```
               db.inventories.find({ item: "Planner", quantity: { $lt: 20 } }, { _id: 0, warehouse: 1 })
```

Slip 9

Consider the Worker table

Worker(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_D
ATE,DEPARTMENT)

Write a query to insert 5 rows in it that as per the query requirements.
```
        CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;
CREATE TABLE Worker (
WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
FIRST_NAME CHAR(25),
LAST_NAME CHAR(25),
SALARY INT(15),
JOINING_DATE DATETIME,
DEPARTMENT CHAR(25)
);
INSERT INTO Worker (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
VALUES
  (1, 'John', 'Doe', 95000, '2021-03-15', 'Finance'),
  (2, 'Jane', 'Smith', 87000, '2021-05-20', 'IT'),
  (3, 'Michael', 'Johnson', 105000, '2021-02-10', 'Finance'),
  (4, 'Emily', 'Brown', 92000, '2021-07-08', 'Marketing'),
  (5, 'David', 'Lee', 98000, '2021-04-25', 'IT');
```

1)List the departments where the total salary expenditure is less than $300,000, sorted by total salary expenditure in ascending order
```
                        SELECT DEPARTMENT, SUM(SALARY) AS total_salary_expenditure
FROM Worker
GROUP BY DEPARTMENT
HAVING total_salary_expenditure < 300000
ORDER BY total_salary_expenditure ASC;
```

2)Find the worker with the highest salary in each department, sorted by department name:
```
                SELECT w1.*FROM Worker w1 JOIN (SELECT DEPARTMENT, MAX(SALARY) AS max_salary FROM Worker
   GROUP BY DEPARTMENT) w2 ON w1.DEPARTMENT = w2.DEPARTMENT AND w1.SALARY = w2.max_salary
ORDER BY w1.DEPARTMENT;
```

Model the following Customer Loan information as a document database. Consider Customer Loan information system where the customer can take many types of loans. Assume appropriate attributes and collections as per the query requirements
db.createCollection("customers");
db.createCollection("loans")
db.customers.insertMany([{ name: "David", address: "Vallab Nagar", city: "Pimpri" }, { name: "Amaan", address: "st Andrews Bandra", city: "Mumbai" }, { name: "Vanshay", address: "Chinchwad", city: "Pune" }, { name: "Dharam", address: "Rawat", city: "Pimpri" }, { name: "Mr.Patil", address: "Andheri", city: "Mumbai" }, { name: "Derek", address: "Viman Nagar", city: "Pune" }, { name: "Dinesh", address: "kasarwadi", city: "Pimpri" }, { name: "Rushikesh", address: "Churchgate", city: "Mumbai" }, { name: "Aditya", address: "Dadar", city: "Pune" }, { name: "Duncan", address: "777 Elm St", city: "Pimpri" }])

db.loans.insertMany([{ customer_id: ObjectId("65fc8c748b4816bd1a1d4983"), loan_type: "Personal", loan_amount: 10000 }, { customer_id: ObjectId("65fc8c748b4816bd1a1d4984"), loan_type: "Home", loan_amount: 200000 }, { customer_id: ObjectId("65fc8c748b4816bd1a1d4985"), loan_type: "Car", loan_amount: 50000 }, { customer_id: ObjectId("65fc8c748b4816bd1a1d4986"), loan_type: "Education", loan_amount: 30000}, { customer_id: ObjectId("65fc8c748b4816bd1a1d4987"), loan_type: "Personal", loan_amount: 150000}, { customer_id: ObjectId("65fc8c748b4816bd1a1d4988"), loan_type: "Home", loan_amount: 250000 }, { customer_id: ObjectId("65fc8c748b4816bd1a1d4989"), loan_type: "Car", loan_amount: 60000 }, { customer_id: ObjectId("65fc8c748b4816bd1a1d498a"), loan_type: "Education", loan_amount: 40000}, { customer_id: ObjectId("65fc8c748b4816bd1a1d498b"), loan_type: "Personal", loan_amount: 200000}, { customer_id: ObjectId("65fc8c748b4816bd1a1d498c"), loan_type: "Home", loan_amount: 300000 }])

List all customers whose name starts with 'D' character.
    db.customers.find({ name: /^D/ })
List the names of customers in descending order who have taken a loan from Pimpri city.
    db.loans.find({ loan_amount: { $gt: 100000 } })
Display customer details having the maximum loan amount.
    db.loans.aggregate([{ $group: { _id: "$customer_id", maxLoanAmount: { $max: "$loan_amount" } } }, { $lookup: { from: "customers", localField: "_id", foreignField: "_id", as: "customerDetails" } }, { $unwind: "$customerDetails" }, { $sort: { maxLoanAmount: -1 } }, { $limit: 1 }, { $project: { _id: 0, "Customer Name": "$customerDetails.name", "Address": "$customerDetails.address", "City": "$customerDetails.city", "Max Loan Amount": "$maxLoanAmount" } }])

Update the address of the customer whose name is "Mr. Patil" and loan amount is greater than 100000
    db.customers.updateOne({ name: "Mr.Patil", }, { $set: { address: "Dahanu" } })

Slip 10

Consider the Employee table
Employee(emp_id,emp_name,job_name,manager_id,hire_date,salary)

```
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(255),
   job_name VARCHAR(255),
   manager_id INT,
   hire_date DATE,
   salary DECIMAL(10, 2)
);
INSERT INTO Employee (emp_id, emp_name, job_name, manager_id, hire_date, salary)
      VALUES
        (1, 'John Doe', 'Manager', NULL, '2022-01-05', 2500),
        (2, 'Jane Smith', 'Engineer', 1, '2022-02-10', 2200),
        (3, 'Michael Johnson', 'Engineer', 1, '2022-03-15', 2300),
        (4, 'Emily Brown', 'Analyst', 2, '2022-04-20', 2100),
        (5, 'David Lee', 'Analyst', 3, '2022-05-25', 2400);
```

1)List the manager_id and the average salary of employees managed by each manager who manages at least 2 employees, sorted by average salary in descending order:

```
            SELECT manager_id, AVG(salary) AS avg_salary
FROM Employee
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING COUNT(emp_id) >= 2
ORDER BY avg_salary DESC;
```

2)Find the departments where the average salary of employees is greater than the average salary of all employees, sorted by department name

```
              SELECT job_name, AVG(salary) AS dept_avg_salary
FROM Employee
GROUP BY job_name
HAVING AVG(salary) > (SELECT AVG(salary) FROM Employee)
ORDER BY job_name;
```

Q2)Model the following Online shopping information as a document database. Consider online shopping where the customer can get different products from different brands. Customers can rate the brands and products Assume appropriate attributes and collections as per the query requirements

```
db.createCollection("customers")
db.createCollection("products")
db.createCollection("brands")
 db.customers.insertMany([ { name: "Alice", city: "New York", purchase_date: "15/08/2023",
bill_amount: 60000 }, { name: "Bob", city: "Los Angeles", purchase_date: "15/08/2023", bill_amount:
```

```
70000 }, { name: "Charlie", city: "Chicago", purchase_date: "16/08/2023", bill_amount: 45000 }, { name: "David", city: "New York", purchase_date: "15/08/2023", bill_amount: 80000 }, { name: "Eve", city: "San Francisco", purchase_date: "16/08/2023", bill_amount: 55000 } ])

 db.products.insertMany([ { name: "Laptop", brand: "Dell", warranty_period: "1 year" }, { name: "Smartphone", brand: "Apple", warranty_period: "2 years" }, { name: "TV", brand: "Samsung", warranty_period: "1 year" }, { name: "Watch", brand: "Fossil", warranty_period: "1 year" }, { name: "Headphones", brand: "Sony", warranty_period: "2 years" } ]) db.brands.insertMany([ { name: "Dell", rating: 4.5 }, { name: "Apple", rating: 4.8 }, { name: "Samsung", rating: 4.3 }, { name: "Fossil", rating: 4.2 }, { name: "Sony", rating: 4.4 } ])
```

List the names of product whose warranty period is one year

```
db.products.find({ warranty_period: "1 year" }, { _id: 0, name: 1 })
```

List the customers has done purchase on “15/08/2023”

```
db.customers.find({ purchase_date: "15/08/2023" })
```

Display the names of products with brand which have highest rating

```
db.brands.find().sort({ rating: -1 }).limit(1)
db.products.find({ brand: "Apple" }, { _id: 0, name: 1 })
```

Display customers who stay in …… city and billamt >50000

```
db.customers.find({ city: "New York", bill_amount: { $gt: 50000 } })
```