1.Scholars Database
Create a Scholar database with the fields: (URN, sname, program, course Type(fulltime/parttime), researchArea)

```
use ScholarsDB;
```

i.

```
db.scholars.insertMany([
  { URN: "S001", sname: "Alice", program: "Computer Science", courseType: "fulltime", researchArea: "AI" },
  { URN: "S002", sname: "Bob", program: "Mathematics", courseType: "parttime", researchArea: "Algebra" },
  { URN: "S003", sname: "Charlie", program: "Computer Science", courseType: "fulltime", researchArea: "Networks" },
  { URN: "S004", sname: "David", program: "Physics", courseType: "parttime", researchArea: "Quantum Mechanics" },
  { URN: "S005", sname: "Eve", program: "Computer Science", courseType: "parttime", researchArea: "AI" },
  { URN: "S006", sname: "Frank", program: "Mathematics", courseType: "fulltime", researchArea: "Topology" },
  { URN: "S007", sname: "Grace", program: "Chemistry", courseType: "fulltime", researchArea: "Organic Chemistry" },
  { URN: "S008", sname: "Heidi", program: "Computer Science", courseType: "fulltime", researchArea: "AI" },
  { URN: "S009", sname: "Ivan", program: "Mathematics", courseType: "parttime", researchArea: "Statistics" },
  { URN: "S010", sname: "Judy", program: "Computer Science", courseType: "fulltime", researchArea: "Cybersecurity" }
]);
```

2. Queries

ii. Display all the documents

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

iii. Display all the scholars in Computer Science

```
db.scholars.find({ program: "Computer Science" });
```

iv. Display scholars' names alphabetically

```
db.scholars.find({}, { sname: 1, _id: 0 }).sort({ sname: 1 });
```

v. Display the first 5 scholars

```
db.scholars.find().limit(5);
```

vi. Display the number of scholars in Mathematics

```
db.scholars.countDocuments({ program: "Mathematics" });
```

vii. Display all the distinct Programs

```
db.scholars.distinct("program");
```

viii. Display Computer Science scholars doing research full time

```
db.scholars.find({ program: "Computer Science", courseType: "fulltime" });
```

2. Employee Database
Create an employee database with the fields: (eid, ename, dept, desig, salary, yoj, address (dno, street,locality, city]]

i. Insert 10 documents:

```
use EmployeeDB;

db.employees.insertMany([
 {
  eid: "E001", ename: "Alice", dept: "HR", desig: "Manager", salary: 60000, yoj: 2015,
  address: { dno: "12", street: "Maple St", locality: "North", city: "New York" }
 },
 {
  eid: "E002", ename: "Bob", dept: "IT", desig: "Developer", salary: 70000, yoj: 2018,
  address: { dno: "45", street: "Pine St", locality: "East", city: "Los Angeles" }
 },
 {
  eid: "E003", ename: "Charlie", dept: "IT", desig: "Developer", salary: 65000, yoj: 2020,
  address: { dno: "67", street: "Elm St", locality: "South", city: "Chicago" }
 },
 {
  eid: "E004", ename: "David", dept: "Sales", desig: "Executive", salary: 50000, yoj: 2016,
  address: { dno: "89", street: "Oak St", locality: "West", city: "Houston" }
 },
 {
  eid: "E005", ename: "Eve", dept: "Finance", desig: "Analyst", salary: 55000, yoj: 2019,
  address: { dno: "101", street: "Cedar St", locality: "Central", city: "Phoenix" }
```

```
  },
  {
    eid: "E006", ename: "Frank", dept: "HR", desig: "Recruiter", salary: 48000, yoj: 2021,
    address: { dno: "34", street: "Ash St", locality: "North", city: "Dallas" }
  },
  {
    eid: "E007", ename: "Grace", dept: "IT", desig: "Tester", salary: 52000, yoj: 2022,
    address: { dno: "56", street: "Birch St", locality: "East", city: "San Diego" }
  },
  {
    eid: "E008", ename: "Heidi", dept: "Sales", desig: "Executive", salary: 53000, yoj: 2017,
    address: { dno: "78", street: "Spruce St", locality: "West", city: "Austin" }
  },
  {
    eid: "E009", ename: "Ivan", dept: "Finance", desig: "Accountant", salary: 75000, yoj: 2014,
    address: { dno: "90", street: "Palm St", locality: "South", city: "Seattle" }
  },
  {
    eid: "E010", ename: "Judy", dept: "IT", desig: "Developer", salary: 80000, yoj: 2013,
    address: { dno: "102", street: "Willow St", locality: "Central", city: "Boston" }
  }
]);
```

ii. Display all the employees with salary in range (50000, 75000):

```
db.employees.find({ salary: { $gt: 50000, $lt: 75000 } });
```

iii. Display all the employees with designation.

```
db.employees.find({ desig: { $exists: true, $ne: null } });
```

iv. Display the Salary of any one employee

```
db.employees.findOne({}, { ename: 1, salary: 1, _id: 0 });
```

V. Update the salary of developers by 5000 increment

```
db.employees.updateMany(
  { desig: "Developer" },
  { $inc: { salary: 5000 } }
);
```

vi. Add field age to employee "XXX"

```
db.employees.updateOne(
  { ename: "Bob" },
  { $set: { age: 30 } }
);
```

vii. Remove YOJ from "XXX"

```
db.employees.updateOne(
  { ename: "Bob" },
  { $unset: { yoj: "" } }
);
```

viii. Add an array field project to "XXX"

```
db.employees.updateOne(
  { ename: "Bob" },
  { $set: { project: ["p1"] } }
);
```

ix. Add p2 and p3 project to "XXX"

```
db.employees.updateOne(
  { ename: "Bob" },
  { $push: { project: { $each: ["p2", "p3"] } } }
);
```

x. Add a new embedded object "contacts" with "email" and "phone" as array objects to "XXX"

```
db.employees.updateOne(
  { ename: "Bob" },
  {
    $set: {
      contacts: {
        email: ["bob@example.com"],
        phone: []
      }
    }
  }
);
```

xi. Add two phone numbers to "XXX"

```
db.employees.updateOne(
  { ename: "Bob" },
  { $push: { "contacts.phone": { $each: ["1234567890", "9876543210"] } } }
);
```

3. Book Database
Create a book Database with the fields: (isbn, bname, author [], year, publisher, price
i. Insert 5 documents.

```
use BookDB;

db.books.insertMany([
  {
    isbn: "ISBN001",
    bname: "Java Programming",
    author: ["Herbert Schildt"],
    year: 2019,
    publisher: "McGraw-Hill",
    price: 450
  },
  {
    isbn: "ISBN002",
    bname: "Modern Poetry",
    author: ["Kuvempu"],
    year: 2018,
    publisher: "Kannada Prakashana",
    price: 300
  },
  {
    isbn: "ISBN003",
    bname: "Python Essentials",
    author: ["Mark Lutz", "David Ascher"],
    year: 2020,
    publisher: "O'Reilly",
    price: 500
  },
  {
    isbn: "ISBN004",
    bname: "Data Structures",
```

```
    author: ["Seymour Lipschutz"],
    year: 2017,
    publisher: "Schaum's Outline",
    price: 400
  },
  {
    isbn: "ISBN005",
    bname: "Literature Classics",
    author: ["Kuvempu", "Bendre"],
    year: 2021,
    publisher: "Kannada Prakashana",
    price: 350
  }
]);
```

ii. List all the documents,

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

iii. List all book names with author name and isbn.

```
db.books.find({}, { bname: 1, author: 1, isbn: 1, _id: 0 });
```

iv. Display all the books published by "XXXX"

```
db.books.find({ publisher: "Kannada Prakashana" });
```

v. List all the books published in the year 2018-2020.

```
db.books.find({ year: { $gte: 2018, $lte: 2020 } });
```

vi. List the publisher of the book "java".

```
db.books.findOne({ bname: { $regex: /^java/i } }, { publisher: 1, _id: 0 });
```

vii. Display the price of "XXBook" except _ id.

```
db.books.findOne({ bname: "Python Essentials" }, { price: 1, _id: 0 });
```

viii. Sort and display only 3 books in descending order of price.

```
db.books.find().sort({ price: -1 }).limit(3);
```

ix. Display all the books written by herbert and kuvempu.

```
db.books.find({ author: { $all: ["Herbert Schildt", "Kuvempu"] } });
```

x. Display all the books either written by herbert and kuvempu.

```
db.books.find({ author: { $in: ["Herbert Schildt", "Kuvempu"] } });
```

xi. Display all the books where "YYYY" is the first author.

```
db.books.find({ "author.0": "Kuvempu" });
```

xii. Skip the first 2 records and print the remaining.

```
db.books.find().skip(2);
```

4. Implementation of aggregation in university database
Create two collections namely "universities" and "courses" with the following set of data.
universities:
country: 'Spain', city: Salamanen', name : 'USAL', location: (type: Point',
coordinates: [-5.6722512,17, 40.9607792]
},
students:
‹ year: 2014, number: 24774 b (year: 2015, number: 23166 %,
1 year: 2016, number: 21913 %
1 year: 2017, number : 21715}
-
country: 'Spain'. city: "Salamanca',
name: UPSA, location: 1
type: Point
coordinates: (-5.6691191.17, 40.963(732 |
students: (
(year: 2014, number : 4788). (year: 2015, number : 4821 1. {ycr : 2016, number : 6550 1.
(year: 2017, number : 6125)
314
Genenue queries Using $match, Using $project, Using $group, Using $out, Using $unwind, Using $sort, Using $limit.

1. Create the universities collection with documents:

```
db.universities.insertMany([
  {
    country: 'Spain',
    city: 'Salamanca',
```

```
    name: 'USAL',
    location: {
      type: 'Point',
      coordinates: [-5.6722512, 40.9607792]
    },
    students: [
      { year: 2014, number: 24774 },
      { year: 2015, number: 23166 },
      { year: 2016, number: 21913 },
      { year: 2017, number: 21715 }
    ]
  },
  {
    country: 'Spain',
    city: 'Salamanca',
    name: 'UPSA',
    location: {
      type: 'Point',
      coordinates: [-5.6691191, 40.9631732]
    },
    students: [
      { year: 2014, number: 4788 },
      { year: 2015, number: 4821 },
      { year: 2016, number: 6550 },
      { year: 2017, number: 6125 }
    ]
  }
]);
```

---

2. Sample Aggregation Queries:

a. $match – Find universities in Salamanca:

```
db.universities.aggregate([
  { $match: { city: "Salamanca" } }
]);
```

b. $project – Show only name and student count for 2017:

```
db.universities.aggregate([
  {
```

```
    $project: {
      name: 1,
      student2017: {
        $arrayElemAt: [
          {
            $filter: {
              input: "$students",
              as: "student",
              cond: { $eq: ["$$student.year", 2017] }
            }
          },
          0
        ]
      }
    }
  }
]);
```

c. $unwind – Flatten student data:

```
db.universities.aggregate([
  { $unwind: "$students" }
]);
```

d. $group – Total students per year across all universities:

```
db.universities.aggregate([
  { $unwind: "$students" },
  {
    $group: {
      _id: "$students.year",
      totalStudents: { $sum: "$students.number" }
    }
  }
]);
```

e. $sort – Sort universities by name:

```
db.universities.aggregate([
  { $sort: { name: 1 } }
]);
```

f. $limit – Get only 1 university:

```
db.universities.aggregate([
  { $limit: 1 }
]);
```

g. $out – Output student data to a new collection:

```
db.universities.aggregate([
  { $unwind: "$students" },
  { $out: "student_stats" }
]);
```

_____

1. Create the courses Collection

```
db.courses.insertMany([
  {
    university: "USAL",
    course_name: "Computer Science",
    duration_years: 4,
    enrolled: 1200
  },
  {
    university: "USAL",
    course_name: "Law",
    duration_years: 5,
    enrolled: 900
  },
  {
    university: "UPSA",
    course_name: "Psychology",
    duration_years: 4,
    enrolled: 650
  },
  {
    university: "UPSA",
    course_name: "Business Administration",
    duration_years: 4,
    enrolled: 700
  }
]);
```

---

2. Aggregation Queries on courses

a. $match – Courses with more than 800 students:

```
db.courses.aggregate([
  { $match: { enrolled: { $gt: 800 } } }
]);
```

b. $project – Show only course name and duration:

```
db.courses.aggregate([
  {
    $project: {
      _id: 0,
      course_name: 1,
      duration_years: 1
    }
  }
]);
```

c. $group – Total enrollment per university:

```
db.courses.aggregate([
  {
    $group: {
      _id: "$university",
      totalEnrolled: { $sum: "$enrolled" }
    }
  }
]);
```

d. $sort – Sort by number of students descending:

```
db.courses.aggregate([
  { $sort: { enrolled: -1 } }
]);
```

e. $limit – Top 2 most enrolled courses:

```
db.courses.aggregate([
  { $sort: { enrolled: -1 } },
  { $limit: 2 }
]);
```

f. $unwind – Not needed here as data isn't nested; but if we had an array of instructors or modules, we could use it.

g. $out – Store most popular courses in a new collection:

```
db.courses.aggregate([
  { $match: { enrolled: { $gt: 800 } } },
  { $out: "popular_courses" }
]);
```

---

3. BONUS: Join Universities and Courses Using $lookup

```
db.universities.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "name",
      foreignField: "university",
      as: "offered_courses"
    }
  }
]);
```

5. Implement all aggregation functions in the HR management system.

1. Sample Collections & Data

employees collection:

```
db.employees.insertMany([
  {
    emp_id: 101,
    name: "Alice",
    age: 30,
    salary: 50000,
    department: "HR",
    join_year: 2018,
    skills: ["communication", "conflict resolution"]
```

```
  },
  {
    emp_id: 102,
    name: "Bob",
    age: 45,
    salary: 75000,
    department: "Finance",
    join_year: 2015,
    skills: ["accounting", "analysis"]
  },
  {
    emp_id: 103,
    name: "Charlie",
    age: 28,
    salary: 48000,
    department: "IT",
    join_year: 2020,
    skills: ["JavaScript", "MongoDB"]
  },
  {
    emp_id: 104,
    name: "David",
    age: 35,
    salary: 62000,
    department: "IT",
    join_year: 2016,
    skills: ["Java", "Python"]
  }
]);
```

departments collection:

```
db.departments.insertMany([
  { dept_id: 1, name: "HR", location: "New York" },
  { dept_id: 2, name: "Finance", location: "London" },
  { dept_id: 3, name: "IT", location: "Berlin" }
]);
```

---

2. Aggregation Function Examples

a. $match – Employees in IT department

```
db.employees.aggregate([
  { $match: { department: "IT" } }
]);
```

b. $project – Show only names and salaries

```
db.employees.aggregate([
  { $project: { _id: 0, name: 1, salary: 1 } }
]);
```

c. $group – Total salary by department

```
db.employees.aggregate([
  {
    $group: {
      _id: "$department",
      totalSalary: { $sum: "$salary" },
      avgSalary: { $avg: "$salary" }
    }
  }
]);
```

d. $sort – Sort employees by salary descending

```
db.employees.aggregate([
  { $sort: { salary: -1 } }
]);
```

e. $limit – Top 2 highest paid employees

```
db.employees.aggregate([
  { $sort: { salary: -1 } },
  { $limit: 2 }
]);
```

f. $unwind – Unwind skills array

```
db.employees.aggregate([
  { $unwind: "$skills" }
]);
```

g. $out – Store high earners into a new collection

```
db.employees.aggregate([
  { $match: { salary: { $gt: 60000 } } },
  { $out: "high_earners" }
]);
```

h. $lookup – Join with departments to get location info

```
db.employees.aggregate([
  {
    $lookup: {
      from: "departments",
      localField: "department",
      foreignField: "name",
      as: "dept_info"
    }
  }
]);
```

i. $count – Total number of employees

```
db.employees.aggregate([
  { $count: "total_employees" }
]);
```

j. $addFields – Add experience (years since joining)

```
db.employees.aggregate([
  {
    $addFields: {
      experience: { $subtract: [2025, "$join_year"] }
    }
  }
]);
```

6. Create a document with the minimum of 5 records and implement indexing

1. Create a Sample Collection: employees

```
db.employees.insertMany([
  {
    emp_id: 1,
    name: "Alice",
    department: "HR",
```

```
    age: 30,
    salary: 50000
  },
  {
    emp_id: 2,
    name: "Bob",
    department: "Finance",
    age: 45,
    salary: 75000
  },
  {
    emp_id: 3,
    name: "Charlie",
    department: "IT",
    age: 28,
    salary: 48000
  },
  {
    emp_id: 4,
    name: "David",
    department: "IT",
    age: 35,
    salary: 62000
  },
  {
    emp_id: 5,
    name: "Eva",
    department: "HR",
    age: 40,
    salary: 58000
  }
]);
```

---

2. Implement Indexing

a. Create an index on emp_id (unique)

```
db.employees.createIndex({ emp_id: 1 }, { unique: true });
```

b. Create an index on department for faster filtering

db.employees.createIndex({ department: 1 });

c. Create a compound index on department and salary

db.employees.createIndex({ department: 1, salary: -1 });

---

3. Check Indexes

To list all indexes on the employees collection:

db.employees.getIndexes();

---

4. Use Explain to See Index Usage

You can verify index use like this:

db.employees.find({ department: "IT" }).explain("executionStats");

7. Implement Map reduce operation with suitable examples using MongoDB

1. Sample Data

```
db.employees.insertMany([
  { emp_id: 1, name: "Alice", department: "HR", salary: 50000 },
  { emp_id: 2, name: "Bob", department: "Finance", salary: 75000 },
  { emp_id: 3, name: "Charlie", department: "IT", salary: 48000 },
  { emp_id: 4, name: "David", department: "IT", salary: 62000 },
  { emp_id: 5, name: "Eva", department: "HR", salary: 58000 }
]);
```

---

2. Define the Map Function

Emit the department as key and salary as value:

```
var mapFunction = function() {
  emit(this.department, this.salary);
};
```

---

3. Define the Reduce Function

Add up all salaries in each department:

```
var reduceFunction = function(department, salaries) {
  return Array.sum(salaries);
};
```

---

4. Run MapReduce

```
db.employees.mapReduce(
  mapFunction,
  reduceFunction,
  {
    out: "total_salary_per_dept"
  }
);
```

---

5. View Results

```
db.total_salary_per_dept.find();
```

Expected Output:

```
{ "_id": "Finance", "value": 75000 }
{ "_id": "HR", "value": 108000 }
{ "_id": "IT", "value": 110000 }
```