1. & 14.

Implement Map reduces operation using MongoDB.

Problem:A king want to count the total population in his country. He can send one person to count the population. The assigned person will visit every city serially and return with the total population in the country.

```
// Step 1: Create the collection
db.createCollection("MapReduce_King");

// Step 2: Insert multiple documents into the collection at once
db.MapReduce_King.insertMany([
   {City: "Los Angeles", Population: 300000},
   {City: "Texas", Population: 42000},
   {City: "Vegas", Population: 99000},
   {City: "Nashville", Population: 30000},
   {City: "Edinburgh", Population: 900000}
]);

// Step 3: Define Map and Reduce functions for population count
var mapFunction2 = function() {
   emit(null, this.Population);  // Emit with null key as we want a total sum
};

var reduceFunction2 = function(key, values) {
   return Array.sum(values);
};

// Step 4: Execute the MapReduce operation
db.MapReduce_King.mapReduce(
   mapFunction2,
   reduceFunction2,
   { out: "Result" }
);

// Step 5: Query the result
db.Result.find().pretty();
```

2. & 15.

Implement aggregation and indexing with following example using MongoDB
Example: In this Assignment, we are creating Student Database. Which contain the information ofstudent_name,student_rollno,status of a student. Here status is whether student is passed/failed by the university.

5. & 17.

Implement aggregation and indexing (all three) with example using MongoDB

8.

Implement aggregation and indexing with example using MongoDB

```
// Create Collection
db.createCollection("Student");

// Insert Documents
db.Student.insert({Stud_Name: "Aarohi", Stud_Roll_No: 1, Status: "Passed"});
db.Student.insert({Stud_Name: "Vrushali", Stud_Roll_No: 2, Status: "Passed"});
db.Student.insert({Stud_Name: "Monica", Stud_Roll_No: 3, Status: "Passed"});
db.Student.insert({Stud_Name: "Joey", Stud_Roll_No: 4, Status: "Failed"});
db.Student.insert({Stud_Name: "Srinidhi", Stud_Roll_No: 5, Status: "Failed"});

db.Student.find().pretty();

// Create Indexes
db.Student.createIndex({Stud_Roll_No: 1});
db.Student.createIndex({Stud_Roll_No: 4}); // This is redundant as the previous index already covers this

// Get Indexes
db.Student.getIndexes();

// Drop Indexes
db.Student.dropIndexes();
db.Student.getIndexes();
```

```
// Create Index Again
db.Student.createIndex({Stud_Roll_No: 4}); // This is redundant again

// Drop a Specific Index
db.Student.dropIndex("Stud_Roll_No_4"); // Use the index name instead of the field and value

// Aggregation Queries
db.Student.aggregate([{$group: {_id: "$Status", sum: {$sum: "$Stud_Roll_No"}}}]);
db.Student.aggregate([{$group: {_id: "$Stud_Roll_No", avg: {$avg: "$Stud_Roll_No"}}}]);
db.Student.aggregate([{$group: {_id: "$Status", avg: {$avg: "$Stud_Roll_No"}}}]);
db.Student.aggregate([{$group: {_id: null, max: {$max: "$Stud_Roll_No"}}}]);
db.Student.aggregate([{$group: {_id: null, min: {$min: "$Stud_Roll_No"}}}]);
db.Student.aggregate([{$group: {_id: "$Status", name: {$push: "$Stud_Name"}}}]);
db.Student.aggregate([{$group: {_id: "$Status", sum: {$sum: "$Stud_Roll_No"}}}, {$match: {sum: {$lte: 7}}}]);
db.Student.aggregate([{$group: {_id: "$Status", sum: {$sum: "$Stud_Roll_No"}}}, {$match: {sum: {$gte: 7}}}]);
```

3.

Implement queries using MongoDB

Teacher_id	Teacher_Name	Dept_Name,	Salary	Status
Pic001	Ravi	IT	30000	A
Pic002	Mangesh	IT	20000	A
Pic003	Akshay	Comp	25000	N

```
// a. Create Collection, Insert data into collection, Find All, find One (with condition)
db.createCollection("Teacher");

db.Teacher.insert({Teacher_id: "Pic001", Teacher_Name: "Ravi", Dept_Name: "IT", Salary: 30000, Status: "A"});
db.Teacher.insert({Teacher_id: "Pic002", Teacher_Name: "Mangesh", Dept_Name: "IT", Salary: 20000, Status: "A"});
db.Teacher.insert({Teacher_id: "Pic003", Teacher_Name: "Akshay", Dept_Name: "Comp", Salary: 25000, Status: "N"});
```

```
// Find all teachers
db.Teacher.find();

// Find one teacher
db.Teacher.findOne();
db.Teacher.findOne({Teacher_id: "Pic003"});
db.Teacher.findOne({Salary: {$eq: 30000}});
db.Teacher.find({Salary: {$eq: 30000}});

// b. Find teacher with salary greater than 50000 and status is A
db.Teacher.find({$and: [{Salary: {$gt: 50000}}, {Status: {$eq: "A"}}]}).pretty();

// c. Find teacher with salary greater than 50000 OR status is A
db.Teacher.find({$or: [{Salary: {$gt: 50000}}, {Status: {$eq: "A"}}]}).pretty();

// d. Display teacher info in ascending and descending order
db.Teacher.find().sort({Salary: 1}).pretty(); // Ascending order by Salary
db.Teacher.find().sort({Salary: -1}).pretty(); // Descending order by Salary

// e. Find teacher from different departments
db.Teacher.find({Dept_Name: "IT"});
db.Teacher.find({Dept_Name: "Comp"});

// f. Update dept_name to ENTC of teacher_id= Pic002
db.Teacher.update({Teacher_id: "Pic002"}, {$set: {Dept_Name: "ENTC"}});

// g. Increment the salary of teacher by 10000 who is having Status A
db.Teacher.updateMany({Status: "A"}, {$inc: {Salary: 10000}});

// h. Delete teacher of teacher_id=Pic001
db.Teacher.deleteOne({Teacher_id: "Pic001"});
```

4. & 11. & 16.

Student_id	Student_Name	Dept_Name,	Fees	Result
101E	Ravi	IT	30000	Pass
102E	Mangesh	IT	20000	Pass
103F	Akshay	Comp	25000	Fail

```
db.createCollection("Stud")

// Insert one document at a time
db.Stud.insert({Student_id: "101E", Student_Name: "Ravi", Dept_Name: "IT", Fees: 30000, Result: "Pass"});

// Insert Multiple documents using batch insert
var student = [
   {Student_id: "102E", Student_Name: "Mangesh", Dept_Name: "IT", Fees: 20000, Result: "Pass"},
   {Student_id: "103F", Student_Name: "Akshay", Dept_Name: "Comp", Fees: 25000, Result: "Fail"}
];
db.Stud.insertMany(student); // Correctly using insertMany

// Find all documents
db.Stud.find();

// Remove a document using $where
db.Stud.remove({$where: function() { return (this.Student_id == "101E"); }});

// Update a document using $where
db.Stud.updateOne(
   {$where: function() { return this.Student_id === "103F"; }},
   {$set: { "Result": "Pass" }}
);

// Upserting a document using save()
db.Stud.save({
   "Student_id": "104E",
   "Student_Name": "Priya",
   "Dept_Name": "Bio",
   "Fees": 27000,
   "Result": "Pass"
});

db.Stud.insertOne({
   "Student_id": "104E",
   "Student_Name": "Priya",
   "Dept_Name": "Bio",
   "Fees": 27000,
   "Result": "Pass"
```

});

6. & 18.

Execute at least 10 queries on following database that demonstrates following querying
techniques:

Book(Book_id,Book_Name,Author,Price,No_of_Pages)

```
db.createCollection("Book")

// 1. Insert multiple documents
db.Book.insertMany([
   {Book_Id: "Id01" , Book_Name: "Wings of Fire", Author: "ABC", Price: 350, No_of_Pages: 300},
   {Book_Id: "Id02" ,Book_Name: "Hello its Me", Author: "ABCD", Price: 4550, No_of_Pages: 600},
   {Book_Id: "Id03" ,Book_Name: "Aishwarya", Author: "ABCDE", Price: 550, No_of_Pages: 200}
]);

// 2. Display all books
db.Book.find();

// 3. Display one book
db.Book.findOne();
db.Book.findOne({Book_Name: "Aishwarya"});

// 4. Display books with Price > 300
db.Book.find({Price: {$gt: 300}});

// 5. Display books with Price < 300 AND No_of_Pages > 1000
db.Book.find({$and: [{Price: {$lt: 300}}, {No_of_Pages: {$gt: 1000}}]});

// 6. Display books with Price <= 300 OR No_of_Pages >= 1000
db.Book.find({$or: [{Price: {$lte: 300}}, {No_of_Pages: {$gte: 1000}}]}).pretty();

// 7. Display books with Price <= 350 OR No_of_Pages >= 500
db.Book.find({$or: [{Price: {$lte: 350}}, {No_of_Pages: {$gte: 500}}]}).pretty();

// 8. Use $not operator to exclude books with Price <= 400
db.Book.find({Price: {$not: {$lte: 400}}}).pretty();

// 9. Accept a null value in a document
db.Book.insertOne({Book_Name: "Random", Author: "Ariana", Price: 700, No_of_Pages: null});
```

```
// 10. Find books whose name contains 'b'
db.Book.find({ Book_Name: { $regex: /^b/i } });
```

7.

Execute at least 10 queries on any suitable MongoDB database that demonstrates following:

Mobile_Specs(Mobile_Name,RAM,Price,Camera)

```
// Create the Mobile_Specs collection

db.createCollection("Mobile_Specs");


// Insert multiple mobile specifications

db.Mobile_Specs.insertMany([

   {Mobile_Name: "Realme", RAM: 16, Price: 15000, Camera: 17},

   {Mobile_Name: "Oppo", RAM: 32, Price: 20000, Camera: 45},

   {Mobile_Name: "Redmi", RAM: 64, Price: 18000, Camera: 12},

   {Mobile_Name: "Poco", RAM: 32, Price: 27000, Camera: 68},

   {Mobile_Name: "Iphone", RAM: 512, Price: 70000, Camera: 90},

   {Mobile_Name: "Realme", RAM: 32, Price: 25000, Camera: 40},

   {Mobile_Name: "Itel", RAM: 16, Price: 12000, Camera: 12}

]);


db.Mobile_Specs.find();

//Find Mobiles with 16GB RAM Using $where

db.Mobile_Specs.find({$where: function() { return this.RAM == 16; }});

//Limit the Display Records to 5

db.Mobile_Specs.find().limit(5);

//Sort Mobiles in Ascending Order by Price

db.Mobile_Specs.find().sort({ Price: 1 });
```

```
//Sort Mobiles in Descending Order by RAM
db.Mobile_Specs.find().sort({ RAM: -1 });
//Skip the First 5 Records
db.Mobile_Specs.find().skip(5).pretty();
```

9.

a. Implement Map reduces operation using MongoDB.

Problem: College student data (FE,SE,TE,BE)

```
// Step 1: Create the "Students" collection
db.createCollection("Students");

// Step 2: Insert documents into the "Students" collection
db.Students.insertMany([
  {Stud_Id: "Pic01", Stud_Name: "Christine", Stud_Year: "SE", Pending_Fees: 25000},
  {Stud_Id: "Pic02", Stud_Name: "Sydney", Stud_Year: "TE", Pending_Fees: 40000},
  {Stud_Id: "Pic03", Stud_Name: "Chandler", Stud_Year: "FE", Pending_Fees: 7000},
  {Stud_Id: "Pic04", Stud_Name: "Joshua", Stud_Year: "TE", Pending_Fees: 30000},
  {Stud_Id: "Pic05", Stud_Name: "Jeremy", Stud_Year: "SE", Pending_Fees: 20000},
  {Stud_Id: "Pic06", Stud_Name: "Joey", Stud_Year: "FE", Pending_Fees: 37000},
```

```
   {Stud_Id: "Pic07", Stud_Name: "Mary", Stud_Year: "SE", Pending_Fees: 44000},
   {Stud_Id: "Pic08", Stud_Name: "Martha", Stud_Year: "BE", Pending_Fees: 50000},
   {Stud_Id: "Pic09", Stud_Name: "Monica", Stud_Year: "BE", Pending_Fees: 70000}
]);

// Step 3: Define the Map function
var mapFunction1 = function() {
   emit(this.Stud_Year, this.Pending_Fees);
};

// Step 4: Define the Reduce function
var reduceFunction1 = function(keyStud_Year, Pending_Fees) {
   return Array.sum(Pending_Fees);
};

// Step 5: Execute the MapReduce operation
db.Students.mapReduce(
   mapFunction1,
   reduceFunction1,
   {
      out: "Pending_Fees_List"
   }
```

```
);

// Step 6: Query the result of MapReduce
db.Pending_Fees_List.find();
```

10. & 13.

Consider the following database:
Employee (emp_no, name, skill, pay rate)

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

// 1. Insert one document
db.Employee.insertOne({emp_no: 1011, name: "Srinidhi", skill: "Developer", pay_rate: 20000});

// 2. Insert multiple documents
db.Employee.insertMany([
    {emp_no: 1012, name: "Ovi", skill: "Tester", pay_rate: 25000},
    {emp_no: 1013, name: "Sanchi", skill: "Analyst Trainee", pay_rate: 35000},
    {emp_no: 1014, name: "Suresh", skill: "Assistant Trainee", pay_rate: 22000},
    {emp_no: 1015, name: "Girish", skill: "Programmer", pay_rate: 29000}
]);

// 3. Find all documents
db.Employee.find();

// 4. Remove a document using $where
db.Employee.deleteOne({$where: function() { return this.emp_no == 1013; }});
```

```
// 5. Update a document using $where (example for emp_no 1012)
db.Employee.updateOne(
    {$where: function() { return this.emp_no == 1012; }},
    {$set: {pay_rate: 26000}}
);

// 6. Upsert a document
db.Employee.updateOne(
    {emp_no: 1016},
    {$set: {name: "Nilisha", skill: "Senior Developer", pay_rate: 40000}},
    {upsert: true}
);

db.Employee.save({
    emp_No: 1016, // This will insert a new document since Emp_No is unique and does not exist
    name: "Nilisha",
    skill: "Senior Developer",
    pay_Rate: 40000
});
```

12.

Consider the following database:
Duty_allocation (posting_no, emp_no, day, shift(day/night))

```
// Create the Duty_allocation collection
db.createCollection("Duty_allocation");

// Insert one document into the collection
db.Duty_allocation.insertOne({
   Posting_no: 101,
   Emp_no: "E001",
   Day: "Monday",
   Shift: "day"
});

// Insert multiple documents into the collection
db.Duty_allocation.insertMany([
   { Posting_no: 102, Emp_no: "E002", Day: "Tuesday", Shift: "night" },
   { Posting_no: 103, Emp_no: "E003", Day: "Wednesday", Shift: "day" },
   { Posting_no: 104, Emp_no: "E004", Day: "Thursday", Shift: "night" }
]);

// Remove a document using $where
db.Duty_allocation.remove({
```

```
    $where: function() {
        return this.Emp_no === "E001";
    }
});

// Update a document using $where
db.Duty_allocation.update(
    { $where: function() { return this.Posting_no === 102; } },
    { $set: { Shift: "day" } }
);

// Upsert a document using save() (this will insert or update)
db.Duty_allocation.save({
    Posting_no: 105, // This will create a new document since it does not exist
    Emp_no: "E005",
    Day: "Friday",
    Shift: "night"
});

db.Duty_allocation.insertOne({
    Posting_no: 105,
    Emp_no: "E005",
 Day: "Friday",
    Shift: "night"
});
```