### Problem Statement 4 (AGGREGATION & INDEXING USING MONGODB)

This task involves creating a MongoDB collection called `Student\_Data`, performing aggregation operations, and working with indexing. Let's go through the steps in detail:

---

### Step 1: Create the Collection `Student\_Data`

We’ll begin by creating the `Student\_Data` collection and inserting some sample documents.

```javascript

use school // Switch to the database

db.Student\_Data.insertMany([

{ Student\_ID: 1, Student\_Name: "Alice", Department: "Computer Science", Marks: 85 },

{ Student\_ID: 2, Student\_Name: "Bob", Department: "Electronics", Marks: 75 },

{ Student\_ID: 3, Student\_Name: "Charlie", Department: "Mechanical", Marks: 90 },

{ Student\_ID: 4, Student\_Name: "David", Department: "Computer Science", Marks: 80 },

{ Student\_ID: 5, Student\_Name: "Eve", Department: "Electronics", Marks: 88 },

{ Student\_ID: 6, Student\_Name: "Frank", Department: "Mechanical", Marks: 65 },

{ Student\_ID: 7, Student\_Name: "Grace", Department: "Computer Science", Marks: 95 },

{ Student\_ID: 8, Student\_Name: "Hannah", Department: "Mechanical", Marks: 92 },

{ Student\_ID: 9, Student\_Name: "Ivy", Department: "Electronics", Marks: 78 },

{ Student\_ID: 10, Student\_Name: "Jack", Department: "Mechanical", Marks: 70 }

]);

```

---

### Step 2: Queries for Aggregation Operations

#### 1. Display All Students Based on Their Departments Along with the Average Marks of a Particular Department

We will use the `$group` stage of the aggregation pipeline to calculate the average marks for each department.

```javascript

db.Student\_Data.aggregate([

{

$group: {

\_id: "$Department", // Group by Department

averageMarks: { $avg: "$Marks" } // Calculate average marks

}

},

{

$lookup: {

from: "Student\_Data", // Lookup to get students in the department

localField: "\_id", // Department

foreignField: "Department", // Department in student records

as: "students"

}

}

]).pretty();

```

- \*\*Explanation\*\*:

- The `$group` stage groups the students by their `Department` and calculates the average marks for each department.

- The `$lookup` stage joins the department data with the list of students in that department.

#### 2. Display the Number of Students Associated with Each Department

We use the `$group` stage to count the number of students per department.

```javascript

db.Student\_Data.aggregate([

{

$group: {

\_id: "$Department", // Group by department

studentCount: { $sum: 1 } // Count number of students

}

}

]).pretty();

```

- \*\*Explanation\*\*: The `$sum: 1` operation adds 1 for each student in a department, effectively counting them.

#### 3. Display List of Students with the Highest Marks in Each Department in Descending Order of Marks

We can use the `$sort` and `$group` stages to find the student with the highest marks per department.

```javascript

db.Student\_Data.aggregate([

{

$sort: { Marks: -1 } // Sort students by Marks in descending order

},

{

$group: {

\_id: "$Department", // Group by Department

highestMarksStudent: { $first: "$Student\_Name" }, // Get the student with the highest marks

highestMarks: { $first: "$Marks" } // Get the marks of the student

}

}

]).pretty();

```

- \*\*Explanation\*\*:

- The `$sort` stage orders the documents by `Marks` in descending order.

- The `$group` stage uses `$first` to select the student with the highest marks in each department (because of the prior sort).

---

### Step 3: Create Indexes

#### 4. Create an Index on the Field `Student\_ID`

Creating an index on `Student\_ID` helps improve query performance when searching by `Student\_ID`.

```javascript

db.Student\_Data.createIndex({ Student\_ID: 1 });

```

- \*\*Explanation\*\*: The `1` indicates an ascending index on the `Student\_ID` field.

#### 5. Create an Index on Fields `Student\_Name` and `Department`

To optimize queries that filter by both `Student\_Name` and `Department`, we create a composite index.

```javascript

db.Student\_Data.createIndex({ Student\_Name: 1, Department: 1 });

```

- \*\*Explanation\*\*: This creates an ascending index on both `Student\_Name` and `Department` fields.

---

### Step 4: Drop Indexes

#### 6. Drop an Index on Field `Student\_ID`

If you no longer need the index on `Student\_ID`, you can drop it.

```javascript

db.Student\_Data.dropIndex({ Student\_ID: 1 });

```

- \*\*Explanation\*\*: This drops the index on `Student\_ID`.

#### 7. Drop an Index on Fields `Student\_Name` and `Department`

Similarly, you can drop the composite index created on `Student\_Name` and `Department`.

```javascript

db.Student\_Data.dropIndex({ Student\_Name: 1, Department: 1 });

```

- \*\*Explanation\*\*: This drops the composite index on `Student\_Name` and `Department`.

---

### Step 5: Additional Index Management

#### Show All Indexes

To view all the indexes created on the `Student\_Data` collection, run:

```javascript

db.Student\_Data.getIndexes();

```

- \*\*Explanation\*\*: This will list all indexes in the collection, including the default `\_id` index and any custom indexes you've created.

---

### Summary of Queries and Operations

1. \*\*Display students by department with average marks\*\*: Uses `$group` and `$lookup` aggregation stages.

2. \*\*Display the number of students per department\*\*: Uses `$group` with `$sum`.

3. \*\*Display the highest marks student per department\*\*: Uses `$sort` and `$group` with `$first`.

4. \*\*Create an index on `Student\_ID`\*\*: Uses `createIndex()` on `Student\_ID`.

5. \*\*Create a composite index on `Student\_Name` and `Department`\*\*: Uses `createIndex()` on both fields.

6. \*\*Drop an index on `Student\_ID`\*\*: Uses `dropIndex()` on `Student\_ID`.

7. \*\*Drop a composite index on `Student\_Name` and `Department`\*\*: Uses `dropIndex()` on both fields.

8. \*\*View all indexes in a collection\*\*: Uses `getIndexes()`.

---

### Running the Queries

1. \*\*Open MongoDB Shell\*\* (`mongosh`).

2. \*\*Switch to the database\*\*:

```javascript

use school

```

3. \*\*Execute the queries\*\*: Run each of the aggregation queries and indexing commands as outlined above.

These operations will help you manage data in the `Student\_Data` collection using aggregation and indexing to optimize queries and retrieve useful information.