Experiment 06

Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

Let's execute a detailed MongoDB aggregation pipeline using various operations such as $match, $group, $sort, $project, and $skip. Here is a step-by-step example

Sample Data

Assume we have a students collection with documents like this:

```
{
    "name": "John Doe",
    "age": 21,
    "major": "Computer Science",
    "gpa": 3.5
  },
  {
    "name": "Jane Smith",
    "age": 22,
    "major": "Mathematics",
    "gpa": 3.8
  },
  {
    "name": "Alice Johnson",
    "age": 23,
```

```
"major": "Computer Science",
 "gpa": 3.9
},
{
 "name": "Bob Brown",
 "age": 21,
 "major": "Physics",
 "gpa": 3.2
}
```

Aggregation Pipeline

The following pipeline filters students by major, groups them by major to calculate the average GPA and count, sorts the results by average GPA, projects the necessary fields, and skips the first document.

```
  },
{
  $project: {
    _id: 0,                          // Exclude the _id field
    major: "$_id",                   // Include major field
    avgGPA: 1,                       // Include average GPA
    count: 1                         // Include student count
  }
 },
 {
  $skip: 1                           // Skip the first document
 } db.students.aggregate([
```

```
  {
    $match: { major: "Computer Science" }  // Filter students by major
  },
  {
    $group: {
      _id: "$major",                        // Group by major
      avgGPA: { $avg: "$gpa" },             // Calculate average GPA
      count: { $sum: 1 }                    // Count number of students
    }
  },
  {
    $sort: { avgGPA: -1 }                   // Sort by average GPA in descending order

])
```

Explanation of the Pipeline:

$match: Filters documents to include only those with the major "Computer Science".

$group: Groups the documents by the major field and calculates the average GPA and count of students.

$sort: Sorts the grouped results by average GPA in descending order.

$project: Projects the required fields (major, avgGPA, and count) and excludes the _id field.

$skip: Skips the first document in the sorted results.

Aggregation Pipeline Example 2

Now, let's create a more complex pipeline that includes additional operators like $lookup and $unwind. Assume we have another collection courses:

```
  {
    "student_id": "student_id",
    "course_name": "Database Systems",
    "grade": "A"
  }
```

We will join this with the students collection to include course details.

```
db.students.aggregate([
  {
    $match: { gpa: { $gte: 3.0 } }        // Filter students with GPA >= 3.0
  },
  {
    $lookup: {
      from: "courses",
      localField: "_id",
      foreignField: "student_id",
      as: "courses"
    }
  },
  {
    $unwind: "$courses"                   // Deconstruct the courses array
  },
  {
    $group: {
      _id: "$_id",
      name: { $first: "$name" },
      gpa: { $first: "$gpa" },
```

```
    courses: { $push: "$courses.course_name" }
      }
    },
    {
      $project: {
        _id: 0,
        name: 1,
        gpa: 1,
        courses: 1
      }
    },
    {
      $sort: { gpa: -1 }                    // Sort by GPA in descending order
    },
    {
      $skip: 2                              // Skip the first two documents
    }
  ])
```

Explanation of the Pipeline:

$match: Filters documents to include only those with GPA >= 3.0.

$lookup: Joins the students collection with the courses collection based on the student ID.

$unwind: Deconstructs the courses array field from the input documents to output a document for each element.

$group: Groups the documents by student _id and aggregates the courses into an array.

$project: Projects the desired fields and excludes the _id field.

$sort: Sorts the output documents by GPA in descending order.

$skip: Skips the first two documents

Aggregation Pipeline Example3

Additional queries to demonstrate aggregation operators

a. Using $limit and $addFields

```
db.students.aggregate([
  {
   $match: { major: "Mathematics" }
  },
  {
   $group: {
     _id: "$major",
     totalGPA: { $sum: "$gpa" },
     numStudents: { $sum: 1 }
   }
  },
  {
   $addFields: {
     avgGPA: { $divide: ["$totalGPA", "$numStudents"] }}
  },
  {
   $sort: { avgGPA: -1 }
  },
  {
   $limit: 5                          // Limit the number of results to 5
  }
])
```

b. Using $out to Save Results to a New Collection

```
db.students.aggregate([
  {
   $match: { age: { $gte: 20 } }
  },
  {
   $group: {
     _id: "$age",
     avgGPA: { $avg: "$gpa" },
     studentNames: { $push: "$name" }
   }
  },
  {
   $sort: { avgGPA: 1 }
  },
  {
   $out: "students_by_age"              // Output results to a new collection
  }
])
```

These examples demonstrate various stages of the aggregation pipeline in MongoDB, showing how to filter, group, sort, project, skip, limit, and output data.

Lets Build new Dataset:

- Download document here

Example 04

Find students with age greater than 23, sorted by age in descending order, and only return name and age

```
db.students6.aggregate([
  { $match: { age: { $gt: 23 } } }, // Filter students older than 23
  { $sort: { age: -1 } }, // Sort by age descending
  { $project: { _id: 0, name: 1, age: 1 } } // Project only name and
])
```

Solution

```
db> db.students6.aggregate([
...   { $match: { age: { $gt: 23 } } }, // Filter students older than 23
...   { $sort: { age: -1 } }, // Sort by age descending
...   { $project: { _id: 0, name: 1, age: 1 } } // Project only name and age
... ])
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

Example 05

Find students with age less than 23, sorted by name in ascending order, and only return name and score

```
db> db.std6.aggregate([{$match :{age:{$gt:23}}},{$sort:{age:1}},{$project:{_id:0,name:1,age:1}}])
[ { name: 'Alice', age: 25 }, { name: 'Charlie', age: 28 } ]
```

Example 06

Group students by major, calculate average age and total number of students in each major:

```
db> db.students6.aggregate([
...   { $group: { _id: "$major", averageAge: { $avg: "$age" }, totalStudents: { $sum: 1 } } }
... ])
[
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 },
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 },
  { _id: 'Biology', averageAge: 23, totalStudents: 1 }
]
```

Example 07

Find students with an average score (from scores array) above 85 and skip the first document

```
db> db.std6.aggregate([{$project:{_id:0,name:1,averageScore:{$avg:"$scores"}}},{$match:{averageScore:{$gt:85}}},{$skip:1}])
[ { name: 'David', averageScore: 93.33333333333333 } ]
db>
```