Class 7

Aggregation pipeline:

- 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)

1.$match: Filters the documents to pass only the documents that match thespecified condition(s).

{ $match: { status: "A" } }

2.$group: Groups input documents by the specified identifier expression andapplies the accumulator expressions.

{

$group: {

_id: "$status",

total: { $sum: "$amount" }

}

}

3.$project: Passes along the documents with the requested fields.

{

$project:

{name: 1,

status: 1

}

}

4.$sort: Sorts all input documents and returns them to the pipeline in sortedorder.

{ $sort: { age: -1 } }

5.$limit: Limits the number of documents passed to the next stage.

{ $limit: 5 }

6.$skip: Skips the first n documents where n is the specified skip number andpasses the remaining documents to the next stage.

{ $skip: 10 }

7.**$unwind:** Deconstructs an array field from the input documents to output adocument for each element.

{ $unwind: "$sizes" }

Lets Build new Dataset

```
_id: 4
name : "David"
age : 20
major : "Computer Science"
▾ scores : Array (3)
    0: 98
    1: 95
    2: 87
```

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
])
```

Output:

```
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

The pipeline starts by finding all students in the students6 collection who are older than 23 years old.Then, it sorts these students in descending order of their age, so the oldest students appear first.Finally, the output only shows thestudent's name and age, discarding the automatically generated _id field.

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 } } }
... ])
```

Output:

```
[
  { _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 }
]
```

This pipeline starts by grouping all students in the students6 collection basedon their declared major. Within each major group, it calculates two statistics:

- The average age of students in that major (averageAge).
- The total number of students in that major (totalStudents).

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

```
db.students6.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      averageScore: { $avg: "$scores" }
    }
  },
  { $match: { averageScore: { $gt: 85 } } },
  { $skip: 1 } // Skip the first document
])
```

Output:

```
db> db.students6.aggregate([
...   {
...     $project: {
...       _id: 0,
...       name: 1,
...       averageScore: { $avg: "$scores" }
...     }
...   },
...   { $match: { averageScore: { $gt: 85 } } }, // Filter by average sc
...   { $skip: 1 } // Skip the first document
... ])
[ { name: 'David', averageScore: 93.33333333333333 } ]
db>
```

The pipeline starts by iterating through each student document in the students6 collection. For each student, it calculates the average score from the"scores" array (assuming it exists and contains numerical values).It then creates a new document for each student that includes only the name and the calculated averageScore. The _id field is discarded.Finally, it filters the documents created in step 3, keeping only those students whose averageScoreis greater than 85.