AGGREGATE PIPELINE

❖ DEFNATION:

An aggregation pipeline consists of one or more stages that process documents:

- Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.
- The documents that are output from a stage are passed to the next stage.
- An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.
- $skip etc. students encourage to execute several queries to demonstrate various aggregation opertaors.

❖ Let's Build New Dataset:

- Download collection here
- Upload the new collection with name "students6"

❖ Explanations:

Explanation of Operators:

- `$match` : Filters documents based on a condition.
- `$group` : Groups documents by a field and performs aggregations like `$avg` (average) and `$sum` (sum).
- `$sort` : Sorts documents in a specified order (ascending or descending).
- `$project` : Selects specific fields to include or exclude in the output documents.
- `$skip` : Skips a certain number of documents from the beginning of the results.
- `$limit` : Limits the number of documents returned.
- `$unwind` : Deconstructs an array into separate documents for each element.

These queries demonstrate various aggregation operations using the `students6` collection. Feel free to experiment with different conditions and operators to explore the power of aggregation pipelines in MongoDB.

JSON FILE:

1.Find the 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:

```
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>
```

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

```
db> db.students6.aggregate([ { $match: { age: { $lt: 23 } } }, {
 $sort: { name: 1 } }, { $project: { _id: 0, name: 1, scores: 1
} }] )
```

Output:

```
db> db.students6.aggregate([ { $match: { age: { $lt: 23 } } }, {
 $sort: { name: 1 } }, { $project: { _id: 0, name: 1, scores: 1
} }] )
[
  { name: 'Bob', scores: [ 90, 88, 95 ] },
  { name: 'David', scores: [ 98, 95, 87 ] }
```

3.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 }
]
```

4.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>
```

3.Find students with an average score (from scores array) below 86 and skip the first 2 documents

```
db> db.students6.aggregate([
... {$project:{
... _id:0,
... name:1,
... averageScore:{$avg:"$scores"}
... }
... },
... {$match:{averageScore:{$lt:86}}},
... {$skip:1}
... ])
```

Output:

```
[
  { name: 'Charlie', averageScore: 82 },
  { name: 'Eve', averageScore: 83.33333333333333 }
]
```