AGGREGATION PIPELINE

The Aggregation Pipeline is a series of data processing stages that can be combined to perform complex data transformations. Each stage in the pipeline processes the output of the previous stage, allowing you to chain multiple operations together to achieve the desired result.

Key Features of the Aggregation Pipeline:

1. **Flexibility**: The Aggregation Pipeline allows you to perform a wide range of data transformations, from simple filtering and sorting to complex data aggregations and analysis.
2. **Efficiency:** The pipeline is optimized for performance, allowing you to process large datasets quickly and efficiently.
3. **Scalability**: The Aggregation Pipeline can handle large datasets and scale horizontally, making it suitable for big data applications.
4. **Expressiveness**: The pipeline provides a rich set of operators and expressions, allowing you to perform complex data transformations and analysis.

Aggregation Pipeline Stages:

The Aggregation Pipeline consists of several stages, each with its own specific function. Here are some of the most common stages:

1. **$match:** Filters documents based on a condition.
2. **$project:** Transforms documents by adding, removing, or modifying fields.
3. **$filter:** Filters documents based on a condition.
4. **$sort**: Sorts documents based on one or more fields.
5. **$group:** Groups documents based on one or more fields and performs aggregation operations.
6. **$unwind**: Deconstructs arrays and creates separate documents for each element.

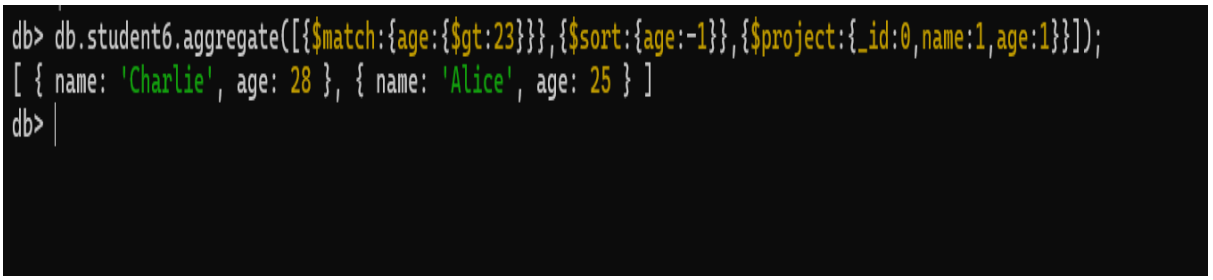
7. **$lookup**: Performs a left outer join with another collection.
8. **$out**: Writes the output of the pipeline to a new collection.

Aggregation Pipeline Operators:

The Aggregation Pipeline provides a rich set of operators that can be used to perform various data transformations and analysis. Here are some examples:

1. **$sum:**Calculates the sum of a field.
2. **$avg:** Calculates the average of a field.
3. **$max:** Returns the maximum value of a field.
4. **$min:** Returns the minimum value of a field.
5. **$push:** Adds an element to an array.
6. **$addToSe**t**:** Adds an element to an array, ensuring that only unique values are included.
7. **$multiply:** Multiplies two fields.

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



Explanation:

1. db.student6.aggregate([...]);

- This starts an aggregation pipeline on the student6 collection within your MongoDB database.

2. {$match:{age:{$gt:23}}};

- $match: This stage filters the documents.
- {age:{$gt:23}}: This filters documents where the "age" field is greater than 23.

3. {$sort:{age:-1}};

- $sort: This stage sorts the documents.
- {age:-1}: This sorts documents in descending order based on the "age" field.

4. {$project: {_id:0,name:1,age:1}};

- $project: This stage reshapes the documents, specifying which fields to keep and how to format them.
- {_id:0,name:1,age:1}:
- _id:0 removes the default _id field.
- name:1 keeps the "name" field as is.
- age:1 keeps the "age" field as is.

The pipeline will produce an array of documents, each containing only the "name" and "age" fields. The documents will be sorted in descending order by age, ensuring that the oldest student appears first.

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 }
]
```

Explanation:

1. db.students6.aggregate([...]): This starts an aggregation pipeline on the "students6" collection.
2. { $group: { ... }}: This stage uses the $group operator to group documents based on a field and perform calculations within each group.
    - _id: "$major": This specifies that the grouping should be done based on the "major" field in each document.
    - averageAge: { $avg: "$age" }: This calculates the average age of all students within the same major using the $avg operator on the "age" field.
    - totalStudents: { $sum: 1 }: This counts the total number of students within the same major using the $sum operator with a value of 1 for each document in the group.
3. The output: The output is a list of documents, each representing a unique major. Each document includes the following information:
    - _id: The major name.
    - averageAge: The average age of students in that major.
    - totalStudents: The total number of students in that major.

In the provided example, you have these majors and their corresponding data:

- Mathematics: 1 student with an average age of 22.
- English: 1 student with an average age of 28.
- Computer Science: 2 students with an average age of 22.5.
- Biology: 1 student with an average age of 23.

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
])
```

Explanation:

The code snippet is a MongoDB aggregation pipeline that performs the following actions:

1. **$project**: It selects only the name and averageScore fields from each document, and calculates the average of the scores field as averageScore.
2. **$match**: It filters the results to include only documents where averageScore is greater than 85.
3. **$skip:** It skips the first document in the filtered results.

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 output of this pipeline is a single document with the following fields:

- **name: David**
- **averageScore: 93.33333333333333**

This indicates that David has the highest average score (greater than 85) among all the students, excluding the first student in the collection.