MongoDB

Aggregation pipeline

What is an Aggregation Pipeline?

An aggregation pipeline in MongoDB is a sequence of stages that process data, transforming it into meaningful results. Each stage takes the output of the previous stage as input, performing operations like filtering, grouping, sorting, and calculating values.

Common Aggregation Stages:

- $match: Filters documents based on specified conditions.
- $project: Includes or excludes fields from the documents, and can also add new calculated fields.
- $group: Groups documents by a specified key and calculates accumulated values.
- $sort: Sorts documents based on specified fields.
- $limit: Limits the number of documents returned.
- $skip: Skips a specified number of documents.

➢ $match ,$sort, $project

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

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

MongoDB aggregation pipeline that operates on the students6 collection. It consists of three stages:

1. **$match**:Filters documents where the age field is greater than 23.

This stage selects only students who are older than 23 years old.

2. **$sort**:Sorts the filtered documents in descending order based on the age field.

This stage arranges the students in descending order of their age.

3. **$project**:Includes the name and age fields in the output documents, excluding the _id field.

This stage selects only the relevant information (name and age) for each student and removes the unnecessary _id field.

Overall, the pipeline:

Filters students older than 23.

Sorts them by age in descending order.

Selects only the name and age fields for each student.

This pipeline effectively retrieves and formats information about students who meet the specified age criteria.

Output:

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

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

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
db> db.Students6.aggregate([ { $match: { age: { $lt: 23 } } }, { $sort: { age: -1 } }] )
[
```

Breakdown of the Pipeline:

1. **$match:** This stage filters documents based on a specified condition.
    - age: { $lt: 23 }: This filter selects documents where the age field is less than 23.
2. **$sort:** This stage sorts the documents based on a specified field.
    - age: -1: Sorts the documents by the age field in descending order (oldest first).

Overall:

The pipeline will:

1. Select all students with an age less than 23 from the Students6 collection.
2. Sort these students by age in descending order.

The result will be a list of students younger than 23, ordered from oldest to youngest.

Output:

```
[
  {
    _id: 2,
    name: 'Bob',
    age: 22,
    major: 'Mathematics',
    scores: [ 90, 88, 95 ]
  },
  {
    _id: 4,
    name: 'David',
    age: 20,
    major: 'Computer Science',
    scores: [ 98, 95, 87 ]
  }
```

Other example:

```
db> db.Students6.aggregate([ { $match: { scores: { $eq: 98 } } }, { $sort: { age: 1 } }, { $project: { _id: 0, name: 1, age: 1,scores:1 } }] )
[ { name: 'David', age: 20, scores: [ 98, 95, 87 ] } ]
db> db.Students6.aggregate([ { $match: { scores: { $lt: 98 } } }, { $sort: { age: 1 } }, { $project: { _id: 0, name: 1, age: 1,scores:1 } }] )
[
  { name: 'David', age: 20, scores: [ 98, 95, 87 ] },
  { name: 'Bob', age: 22, scores: [ 90, 88, 95 ] },
  { name: 'Eve', age: 23, scores: [ 80, 77, 93 ] },
  { name: 'Alice', age: 25, scores: [ 85, 92, 78 ] },
  { name: 'Charlie', age: 28, scores: [ 75, 82, 89 ] }
]
db>
```

➢ $group

1.

```
db> db.Students6.aggregate([ { $group:{_id:"$major",averageAge:{$avg:"$age"},totalStudents:{$sum:1}}}])
[
  { _id: 'Biology', averageAge: 23, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 },
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 }
]
db>
```

$group stage: This stage groups documents by a specified key and calculates accumulated values.

_id: "$major": This groups the documents by the major field, creating groups for each unique major value.

averageAge: {$avg: "$age"}: Calculates the average age for each group.

totalStudents: {$sum: 1}: Counts the total number of students in each group.

Output: The output shows the results of the aggregation, with each document representing a group:

_id: The major name.

averageAge: The average age of students in the major.

totalStudents: The total number of students in the major.

2.

```
db> db.Students6.aggregate([ { $group:{_id:"$major"}}])
[
  { _id: 'Biology' },
  { _id: 'Computer Science' },
  { _id: 'Mathematics' },
  { _id: 'English' }
]
db>
```

$group stage: This stage groups documents by a specified key.

_id: "$major": This groups the documents by the major field, creating groups for each unique major value.

Output: The output shows the results of the aggregation, with each document representing a group:

_id: The major name.

In essence:

This aggregation pipeline lists all the unique major values present in the Students6 collection.

Example:

Biology

Computer Science

Mathematics

English

3.

```
]
db> db.Students6.aggregate([ { $group:{_id:"$major",maximunAge:{$max:"$age"},totalStudents:{$sum:1}}}])
[
  { _id: 'English', maximunAge: 28, totalStudents: 1 },
  { _id: 'Mathematics', maximunAge: 22, totalStudents: 1 },
  { _id: 'Computer Science', maximunAge: 25, totalStudents: 2 },
  { _id: 'Biology', maximunAge: 23, totalStudents: 1 }
]
db>
```

$group stage: This stage groups documents by a specified key and calculates accumulated values.

- **_id: "$major":** This groups the documents by the major field, creating groups for each unique major value.
- **maximunAge: {$max:"$age"}:** Calculates the maximum age for each group.
- **totalStudents: {$sum: 1}:** Counts the total number of students in each group.

Output: The output shows the results of the aggregation, with each document representing a group:

- **_id:** The major name.
- **maximunAge:** The maximum age of students in the major.
- **totalStudents:** The total number of students in the major.

➢ $Skip:

The **$skip** stage in MongoDB is used to skip a specified number of documents from the beginning of the result set. It's often used in conjunction with the $limit stage for pagination.

```
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 provided image shows a MongoDB aggregation pipeline applied to the students6 collection. Let's break down the stages:

1. $project:

- Creates a new document shape.
- Excludes the _id field by setting it to 0.
- Includes the name field.
- Calculates the averageScore by averaging the values in the scores array for each document.

2. $match:

- Filters the documents based on the calculated averageScore.
- Only keeps documents where the averageScore is greater than 85.

3. $skip:

- Skips the first document in the resulting set.

Output:

- The final result is a single document containing the name and averageScore of the second student with an average score greater than 85. In this case, it's 'David' with an average score of 93.33333333333333.