MongoDB

AGGREGATION PIPELINES

Aggregation in MongoDB involves processing data records and returning computed results. The aggregation framework is powerful and can be used to perform a wide variety of operations. Here are the key operators used in MongoDB aggregation pipelines:

Aggregation Pipeline Stages

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

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

2. **$group**: Groups input documents by a specified identifier expression and applies the accumulator expressions to each group.

```
{ $group: { _id: "$status", total: { $sum: 1 } } }
```

3. **$project**: Reshapes each document in the stream by adding, removing, or renaming fields.

```
{ $project: { item: 1, total: 1, status: 1 } }
```

4. **$sort**: Sorts all input documents and returns them in sorted order.

```
{ $sort: { total: -1 } }
```

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

```
{ $limit: 5 }
```

6. **$skip**: Skips the first n documents and passes the remaining documents to the next stage.

```
{ $skip: 10 }
```

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

```
{ $unwind: "$items" }
```

8. **$lookup**: Performs a left outer join to another collection in the same database to filter in documents from the "joined" collection for processing.

```
{
  $lookup: {
    from: "otherCollection",
    localField: "fieldName",
    foreignField: "foreignFieldName",
    as: "alias"
  }
}
```

9. **$addFields**: Adds new fields to documents.

```
{ $addFields: { newField: "$existingField" } }
```

10. **$replaceRoot**: Replaces the input document with the specified document.

```
{ $replaceRoot: { newRoot: "$newDoc" } }
```

11. **$count**: Returns a count of the number of documents input to the stage.

```
{ $count: "total" }
```

12. **$facet**: Processes multiple aggregation pipelines within a single stage on the same set of input documents.

```
{
  $facet: {
    "groupedByStatus": [{ $group: { _id: "$status", count: { $sum: 1 } } }],
    "groupedByCategory": [{ $group: { _id: "$category", count: { $sum: 1 } } }]
  }
}
```

Aggregation Pipeline Expressions

1. **$sum**: Calculates and returns the sum of numeric values.

```
{ $sum: "$quantity" }
```

2. **$avg**: Calculates the average value of the numeric values.

```
{ $avg: "$quantity" }
```

3. **$min**: Returns the minimum value.

```
{ $min: "$quantity" }
```

4. **$max**: Returns the maximum value.

```
{ $max: "$quantity" }
```

5. **$push**: Returns an array of all values for the group.

```
{ $push: "$item" }
```

6. **$addToSet**: Returns an array of unique values.

```
{ $addToSet: "$item" }
```

7. **$first**: Returns the first value in the group.

```
{ $first: "$item" }
```

8. **$last**: Returns the last value in the group.

```
{ $last: "$item" }
```

9. **$concat**: Concatenates strings.

```
{ $concat: ["$firstName", " ", "$lastName"] }
```

10. **$toLower**: Converts a string to lowercase.

```
{ $toLower: "$name" }
```

11. **$toUpper**: Converts a string to uppercase.

```
{ $toUpper: "$name" }
```

12. **$substr**: Returns a substring of a string.

```
{ $substr: ["$name", 0, 3] }
```

13. **$ifNull**: Returns the value if it is not null; otherwise, it returns the replacement value.

```
{ $ifNull: ["$field", "replacementValue"] }
```

Example Aggregation Pipeline

Here's an example of how an aggregation pipeline might look:

```
db.orders.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },
  { $sort: { total: -1 } },
  { $limit: 5 },
  { $project: { _id: 0, customer: "$_id", total: 1 } }
]);
```

In this example:

1. $match filters documents with status "A".
2. $group groups documents by cust_id and calculates the sum of the amount for each group.
3. $sort sorts the grouped results by total in descending order.
4. $limit restricts the result to the top 5 documents.
5. $project reshapes the documents to include only the customer and total fields, renaming _id to customer.

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
])
```

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

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

1. **B. 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 } ]
db>
```

2. **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 }
]
```

3. 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
])
```

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

```
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>
```
