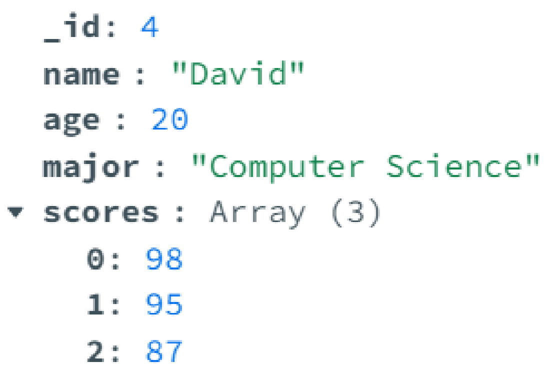
AGGREGATION PIPELINE

Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more.

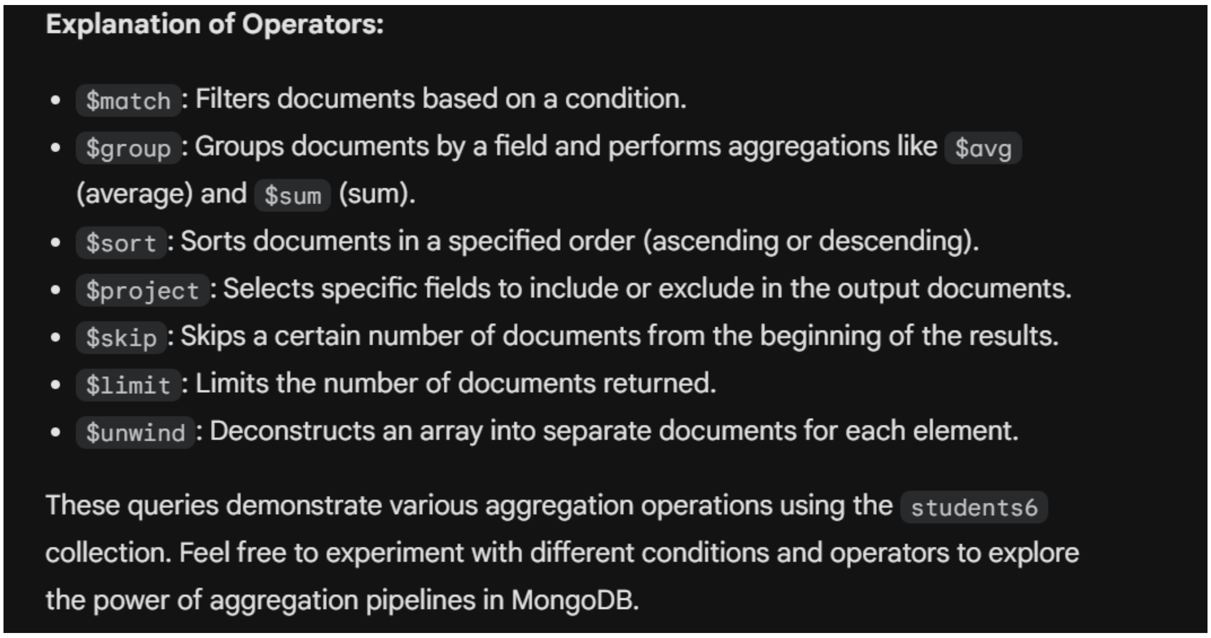
Aggregation pipelines can have one or more "stages". The order of these stages are important. Each stage acts upon the results of the previous stage.

Upload the new collection with name “students6”.



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

Explanation:

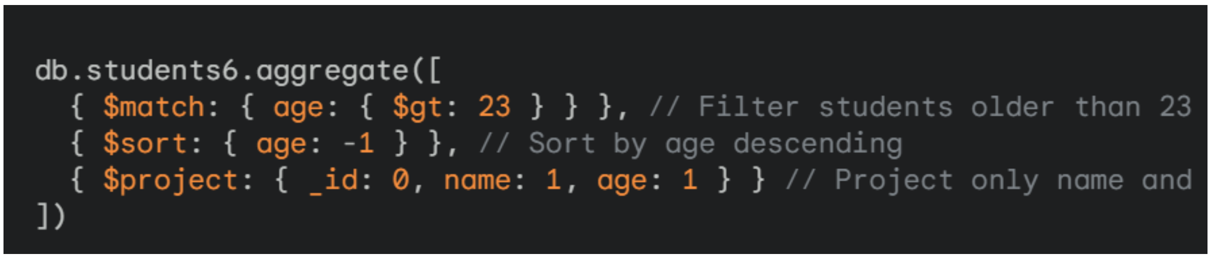


Explanation of Operators:

- `$match`: Filters documents based on a condition.
- `$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.

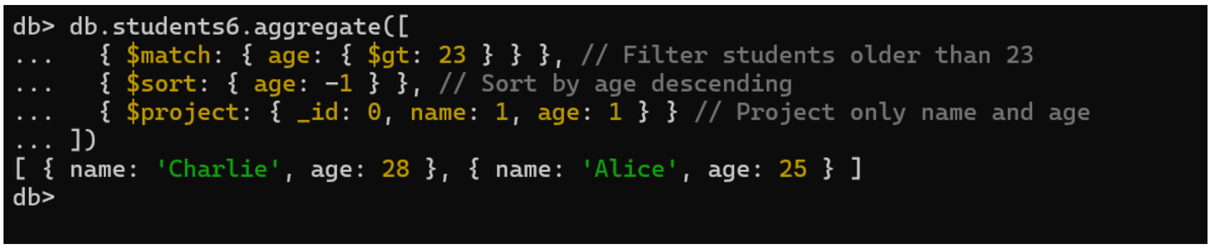
These queries demonstrate various aggregation operations using the `students6` collection. Feel free to experiment with different conditions and operators to explore the power of aggregation pipelines in MongoDB.

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



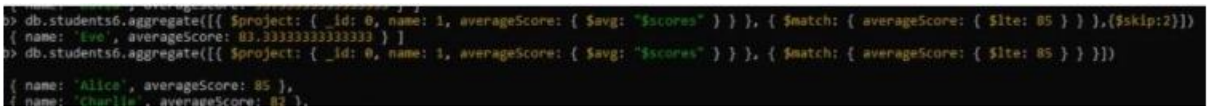
```
db.students6.aggregate([
  { $match: { age: { $gt: 23 } } }, // Filter students older than 23
  { $sort: { age: -1 } }, // Sort by age descending
  { $project: { _id: 0, name: 1, age: 1 } } // Project only name and
])
```

Output:



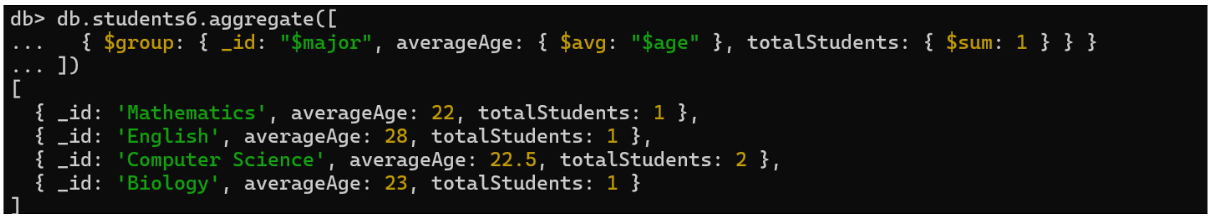
```
db> db.students6.aggregate([
...   { $match: { age: { $gt: 23 } } }, // Filter students older than 23
...   { $sort: { age: -1 } }, // Sort by age descending
...   { $project: { _id: 0, name: 1, age: 1 } } // Project only name and age
... ])
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

1.B.Find students with age less than 23, sorted by age in ascending order, and only return name and score with output:



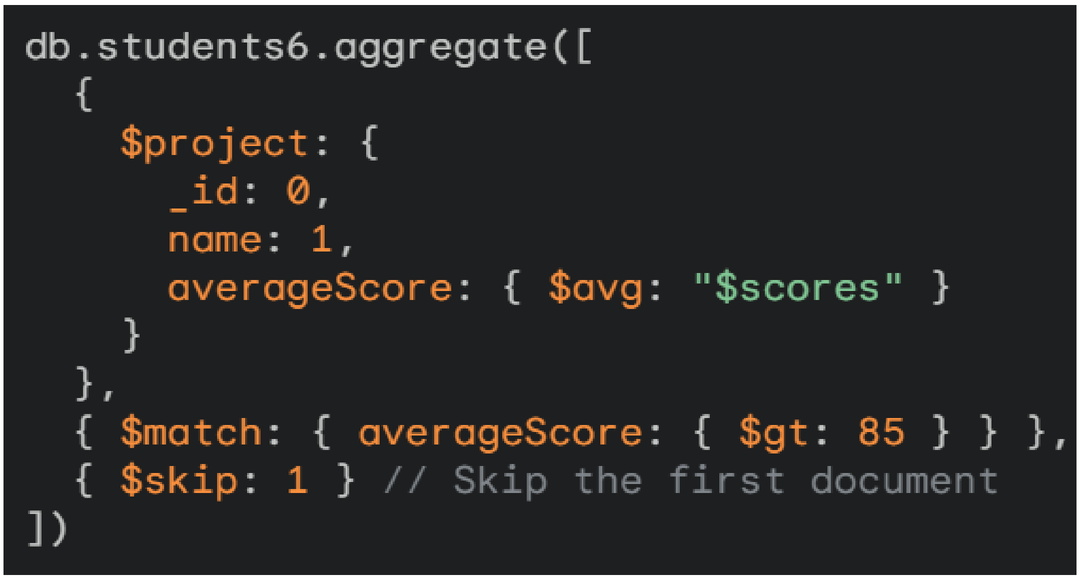
```
b> db.students6.aggregate([{ $project: { _id: 0, name: 1, averageScore: { $avg: "$scores" } } }, { $match: { averageScore: { $lte: 85 } } },{$skip:2}])
{ name: 'Eve', averageScore: 83.33333333333333 } ]
b> db.students6.aggregate([{ $project: { _id: 0, name: 1, averageScore: { $avg: "$scores" } } }, { $match: { averageScore: { $lte: 85 } } }])

{ name: 'Alice', averageScore: 85 },
{ name: 'Charlie', averageScore: 82 },
```

2.Group students by major, calculate average age and total number of students in each major with output:



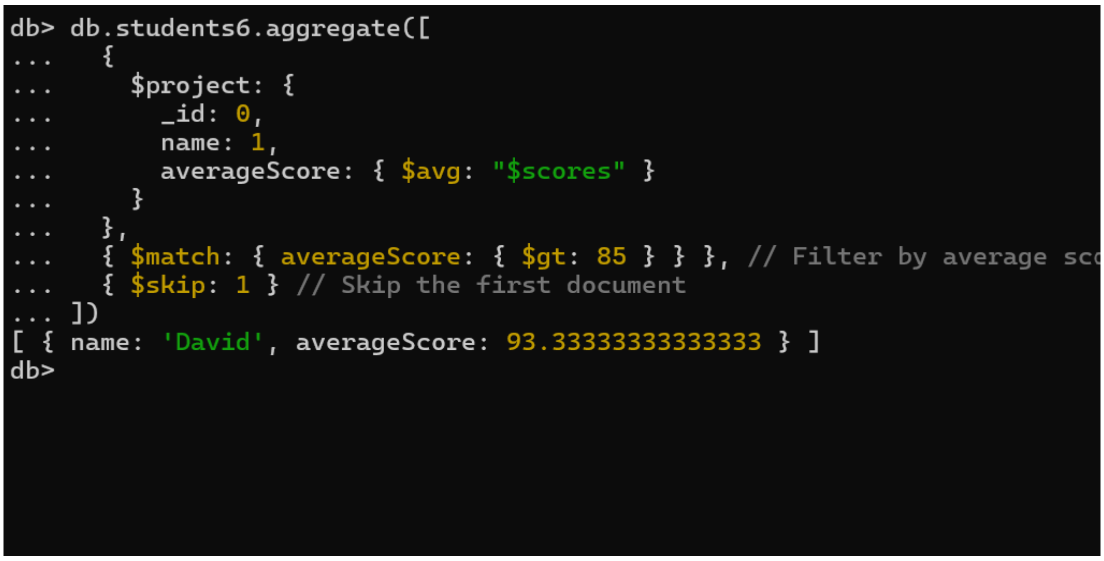
```
db> db.students6.aggregate([
...   { $group: { _id: "$major", averageAge: { $avg: "$age" }, totalStudents: { $sum: 1 } } }
... ])
[
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 },
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 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
])
```

Output:



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