CLASS:7

AGGREGATION PIPELINE

AGENDA:

Execute Aggregation Pipeline and its operations (pipeline must

contain $match, $group, $sort, $project.

- $skip etc. students encourage to
  execute several queries to

demonstrate various aggregation operators.

Aggregation operation is a process where data is processed and returned in a structured format. It allows you to perform operations like grouping, filtering, sorting, and transforming data in a collection.

TO BUILD NEW DATABASE

- Download collection here
- Upload the new collection with name “students6”.

It includes:

[{ "_id": 1, "name": "Alice", "age": 25, "major": "Computer Science", "scores": [85, 92, 78] },

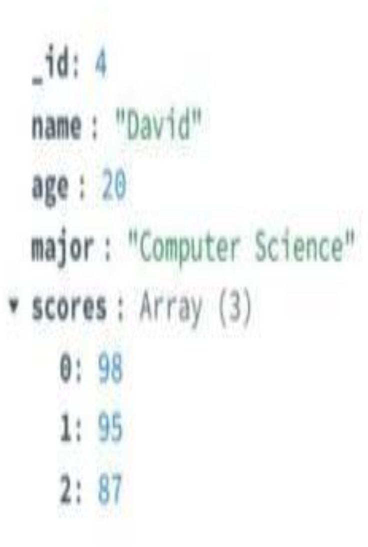
{"_id": 2, "name": "Bob", "age": 22, "major": "Mathematics", "scores": [90, 88, 95] },

{ "_id": 3, "name": "Charlie", "age": 28, "major": "English", "scores": [75, 82, 89] },

{ "_id": 4, "name": "David", "age": 20, "major": "Computer Science", "scores": [98, 95, 87] },

{ "_id": 5, "name": "Eve", "age": 23, "major": "Biology", "scores": [80, 77, 93] }]

Like:



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

EXPLANATION:

* Input: Aggregation pipelines operate on collections of documents in your

database.

* Operators: These are the building blocks that perform specific actions on the

data at each stage. Here are some common types:

o Filtering: $match keeps only documents matching certain criteria.

o Projection: $project selects specific fields to include or exclude from

the output.

o Grouping: $group organizes documents into groups based on

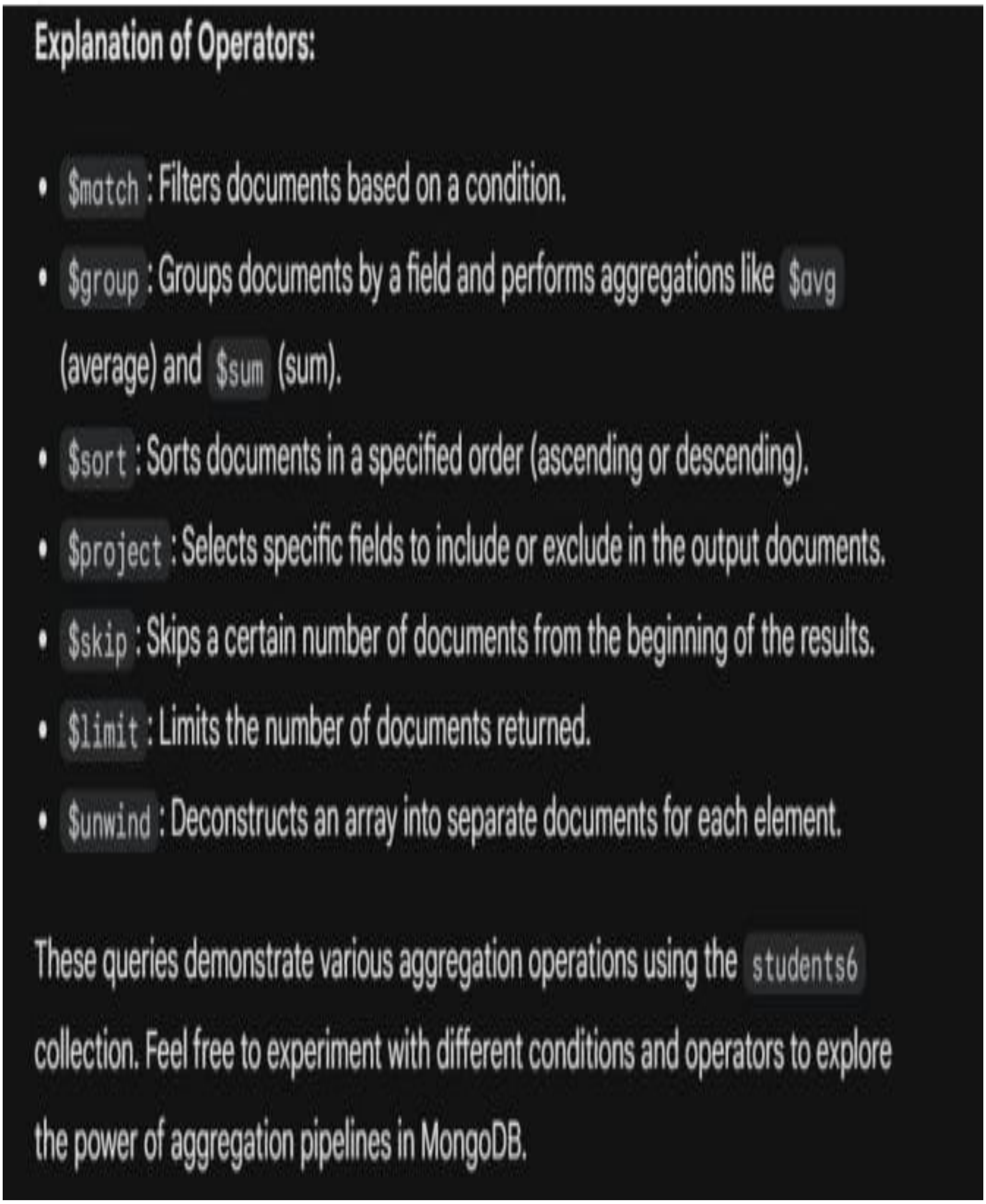
shared field values.

* Multi-Stage: You can chain multiple operators together. Each stage processes

the output of the previous one.

* Output: The final stage defines the results. You can return the entire

transformed data set or just the aggregated values.

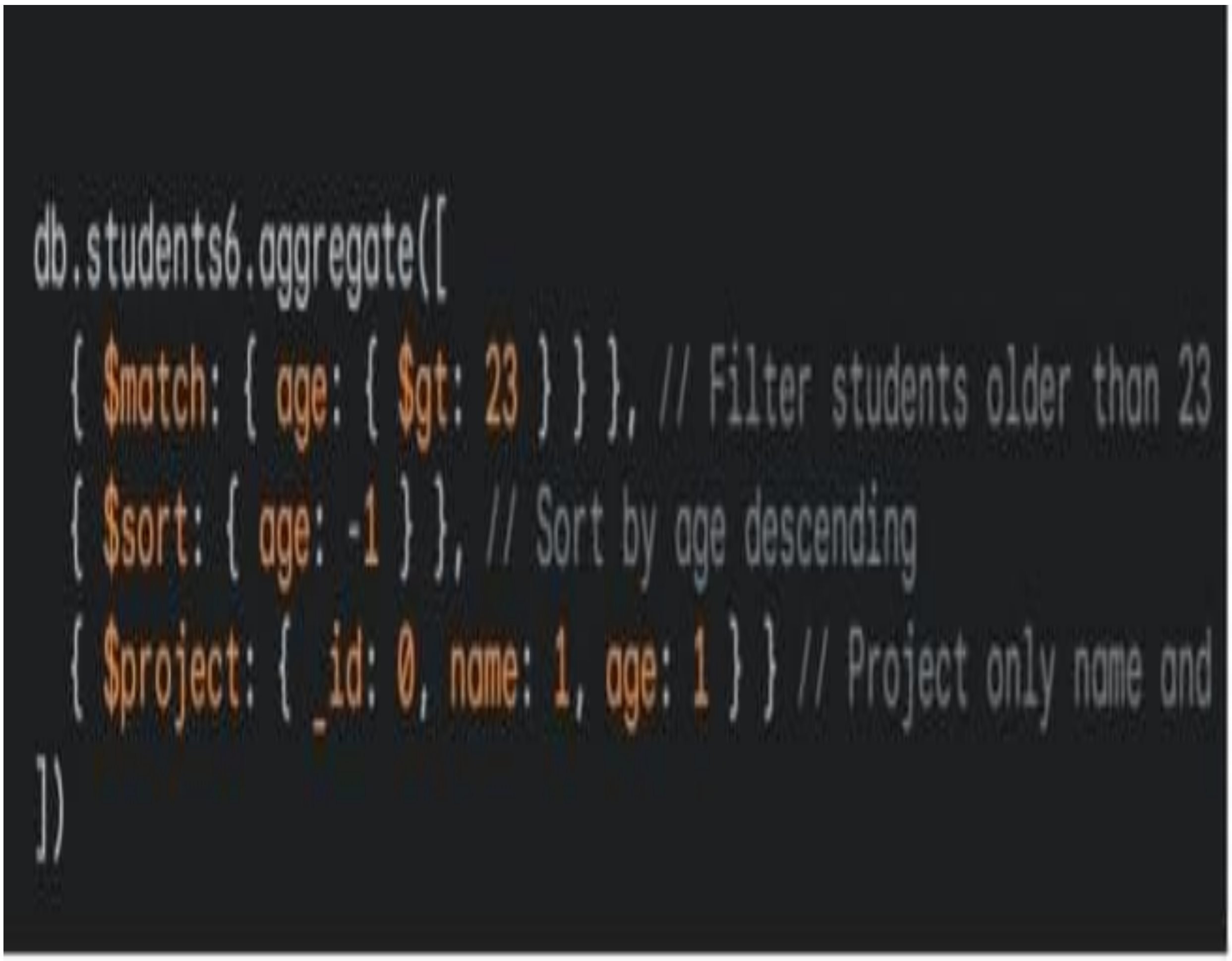


Explanation of Operators:

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

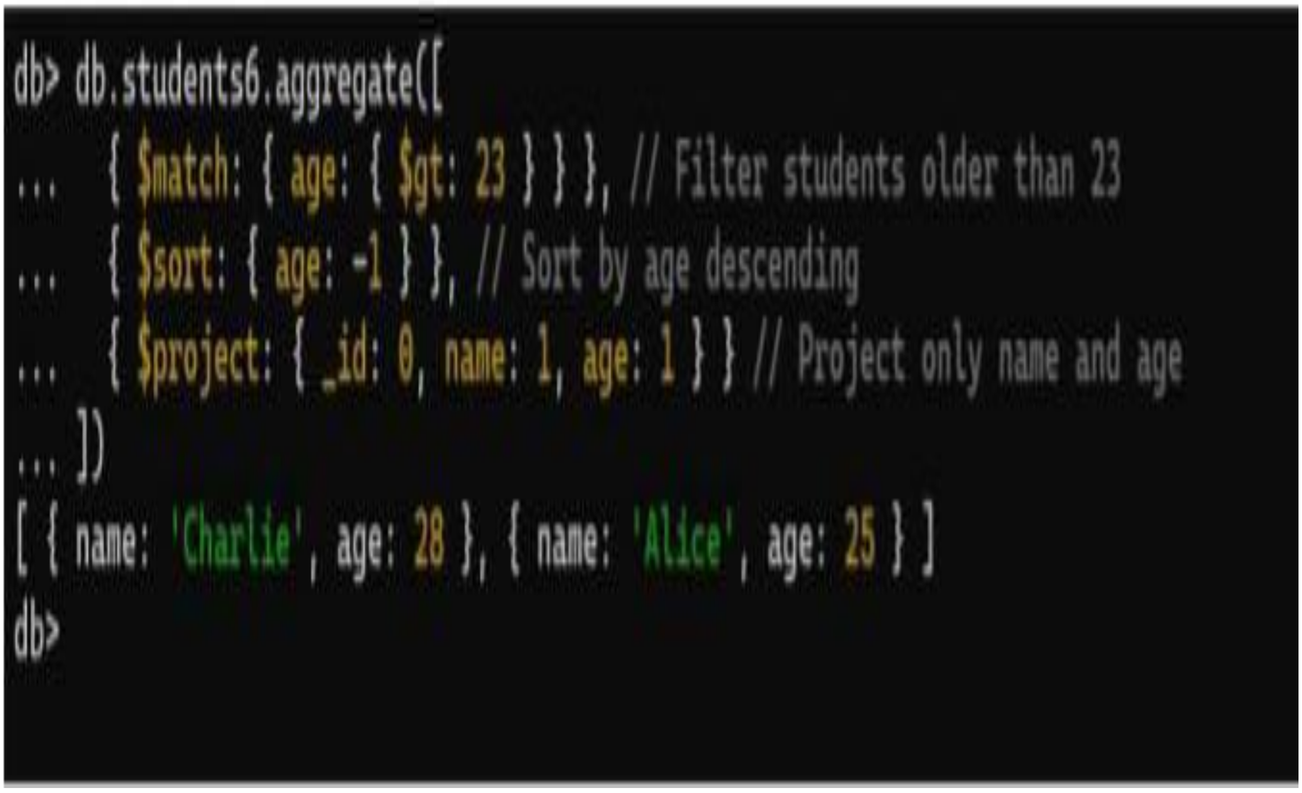
1) TO FIND STUDENTS WITH AGE GREATER THAN 23 ,SORTED BY AGE IN DESCENDING ORDER,AND ONLY RETURN NAME AND AGE



```
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
])
```

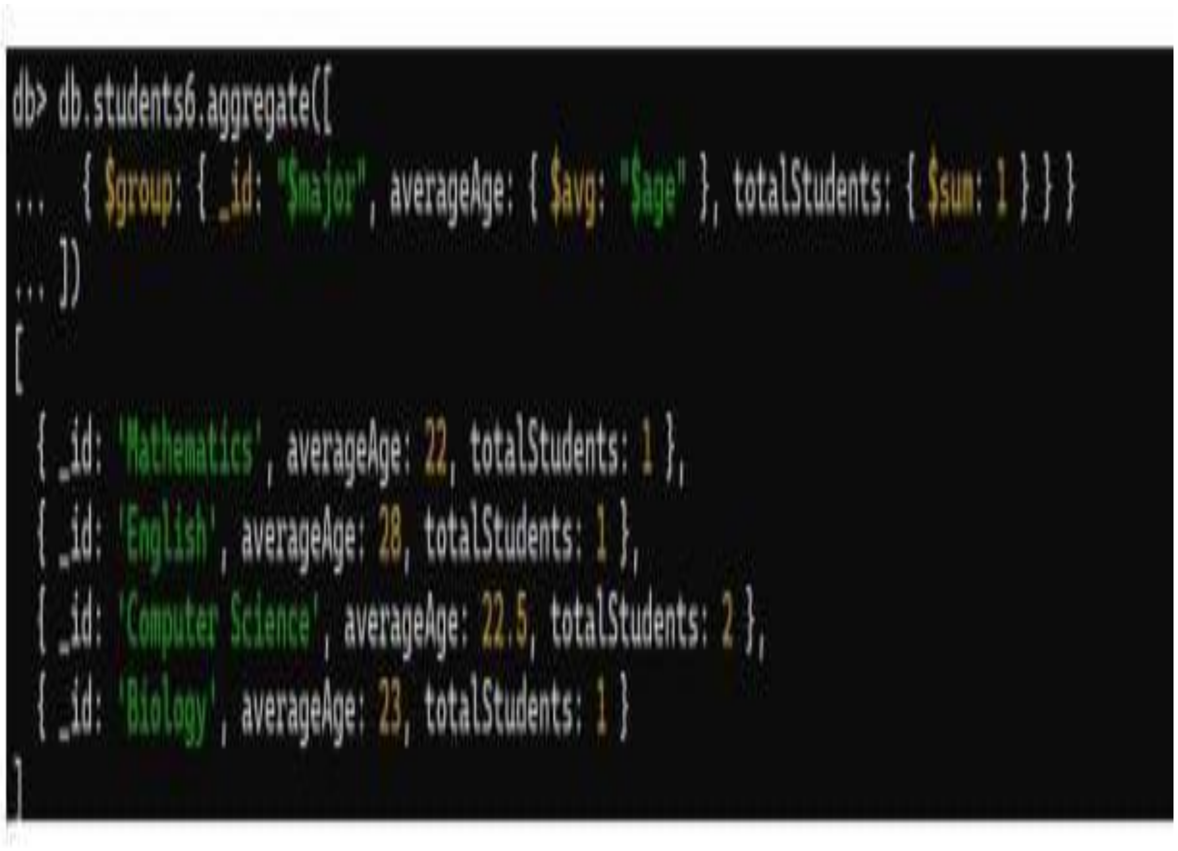
HERE,we use three opertions $match,$sort,$project.

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) Group studentd 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) TO FIND STUDENTS WITH AN AVERAGE SCRE (FROM SCORES ARRAY) ABOVE 85 AND SKIP THE FIRST DOCUMENT.

It has 5 stages:

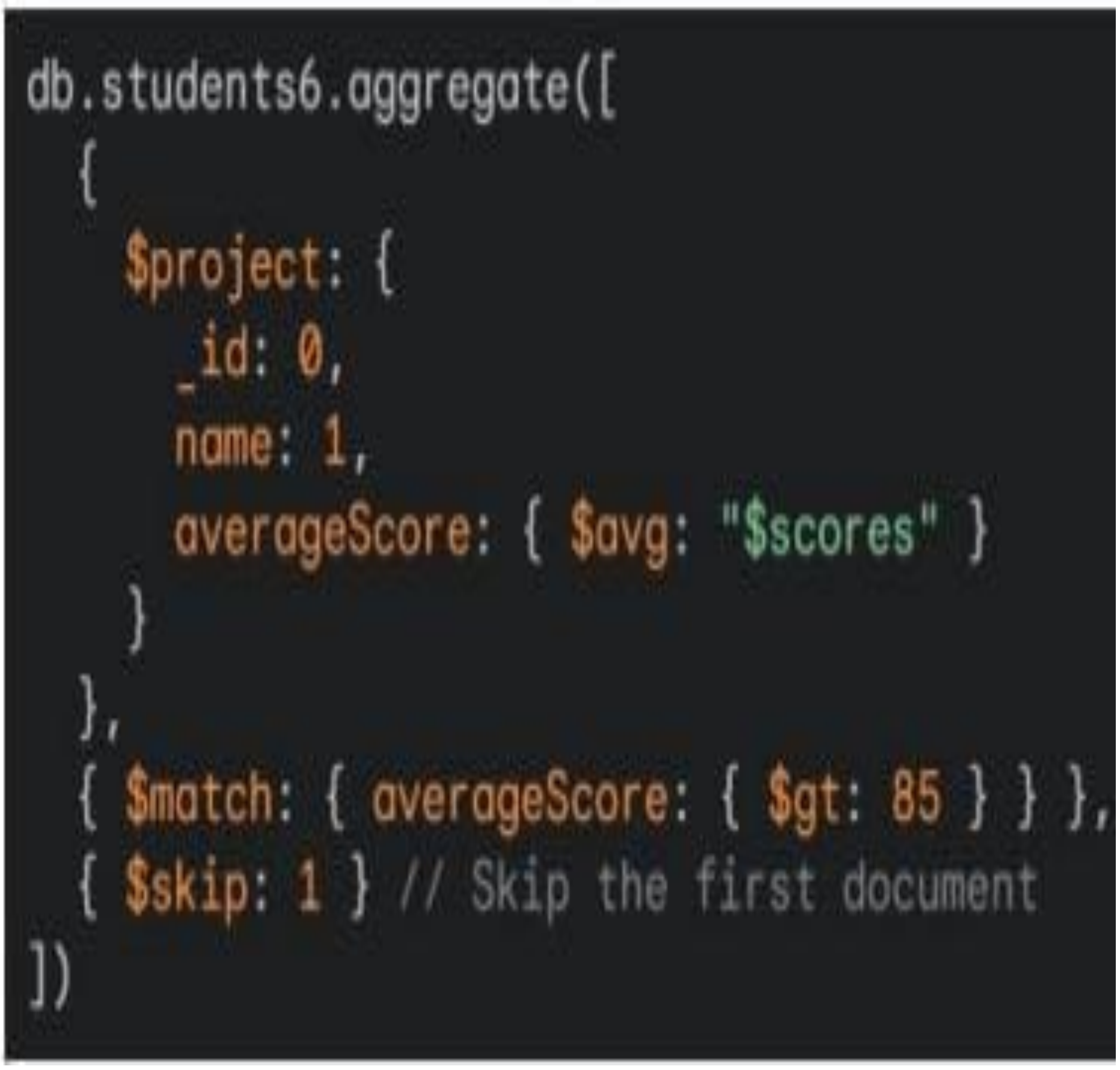
$unwind stage

$group stage

$match stage

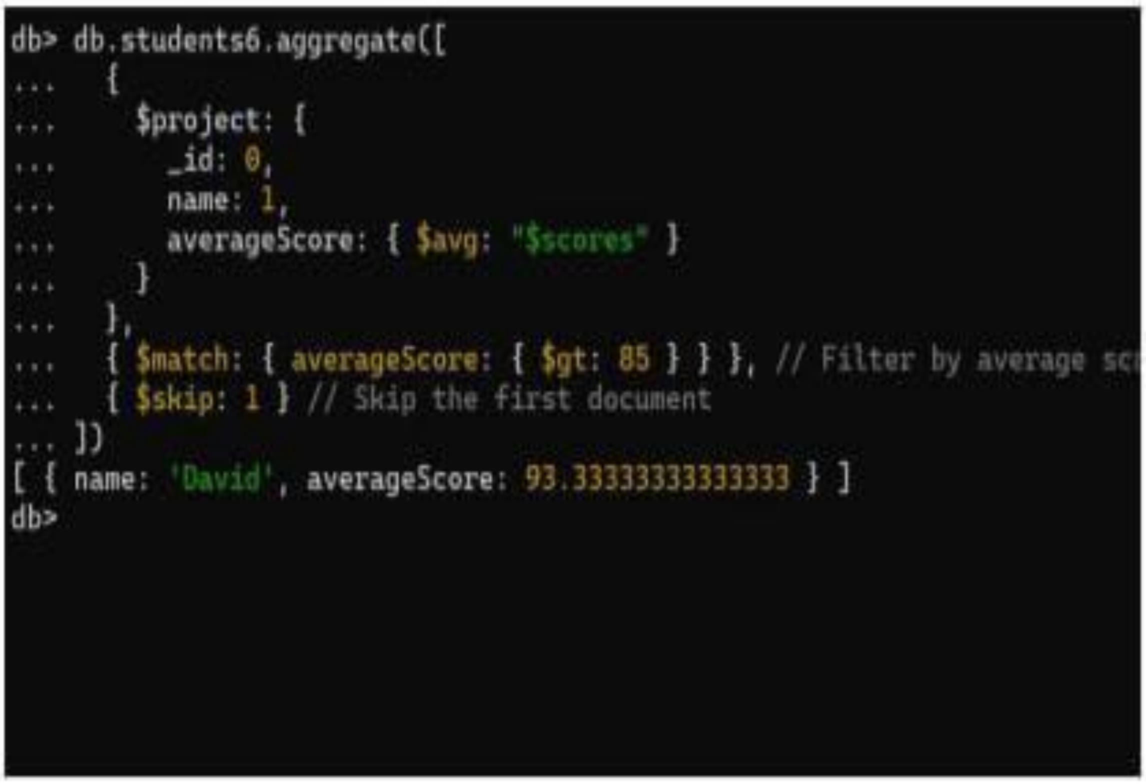
$sort stage

$skip stage



```
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>
```