# AGGREGATION PIPELINE

The MongoDB **Aggregation pipeline** is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a **multi-stage pipeline** that transforms the documents into aggregated results.

Each stage performs an operation on the input documents and passes the results to the next stage. The stages can filter, group, and modify the documents in various ways.

It encourage to execute several queries to demonstrate various **Aggregation operators.**

## Here are some common operators in aggregation pipeline:

1. **$match**: Filters the documents to pass only those that match the specified condition to the next pipeline stage.

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

3. **$project:** Reshapes each document in the stream, such as by adding, removing, or renaming fields that is to include and exclude fields.

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

5. **$limit:** limits the number of documents returned.

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

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

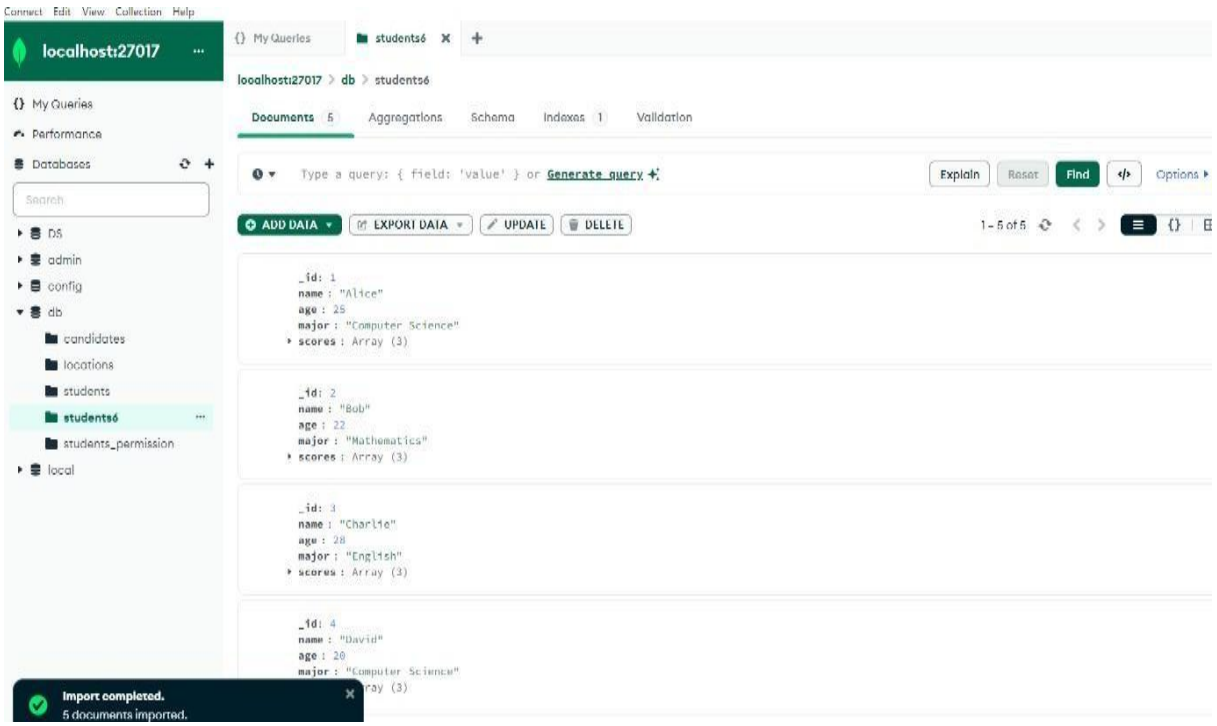
8. **$lookup:** Performs a left outer join to a collection in the same database to filter in documents from the "joined" collection for processing.

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

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

The order of the stages is crucial because the output of one stage becomes the input of the next.

Now lets import a new collection called **“students6”** through mongo compass.

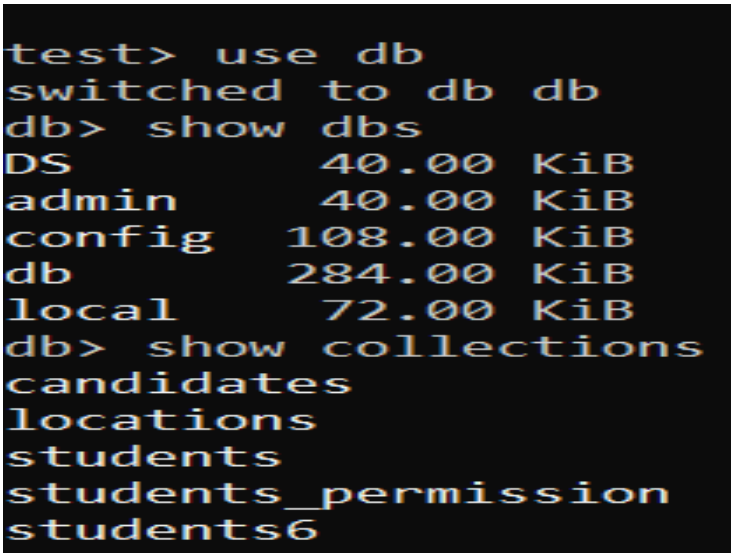



To switch this collection have to use some commands they are

**use db**

**show dbs**

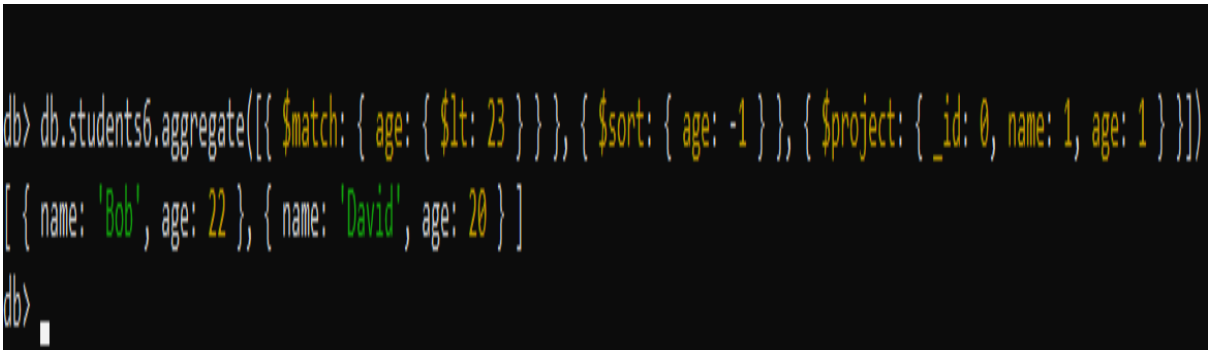
**show collections**



## ❖ $match,$sort:

Now to find students with age **less than** 23 it could be sorted by descending order to obtain only name and age we use a command

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




According to the output Bob and David are 22 and 20 year students respectively.

Here,

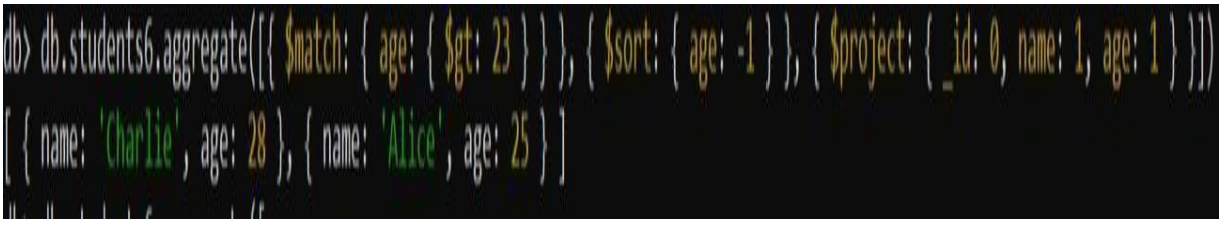
**$lt**:represents less than.

**$gt**:reprents greater than.

**Age:(-1)**:-represents sorting in descending order.

Again Now to find students with age **greater than** 23 it could be sorted by descending order to obtain only name and age we use a command

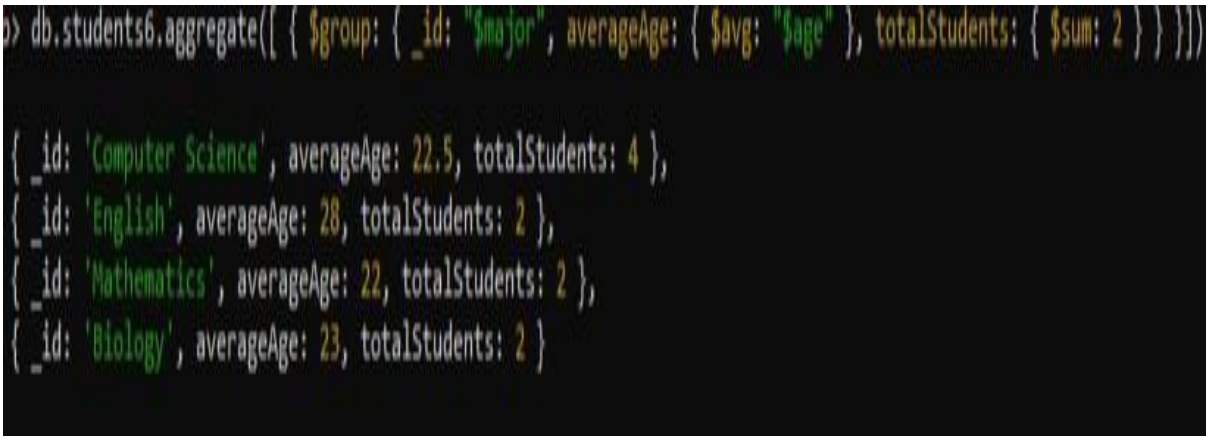
**db.students6.aggregate([{$match:{age:{$gt:23}}},{$sort:{age:-1}},{$project:{_id:0,name:1,age:1}}])**




## ❖ $group:

Now to group students by major to calculate average age and total number of students in each major using **sum:2** we use a command

**db.students6.aggregate([{$group:{_id:"$major",averageAge:{$avg:"$age"},totalStudents:{$sum:2}}}])**

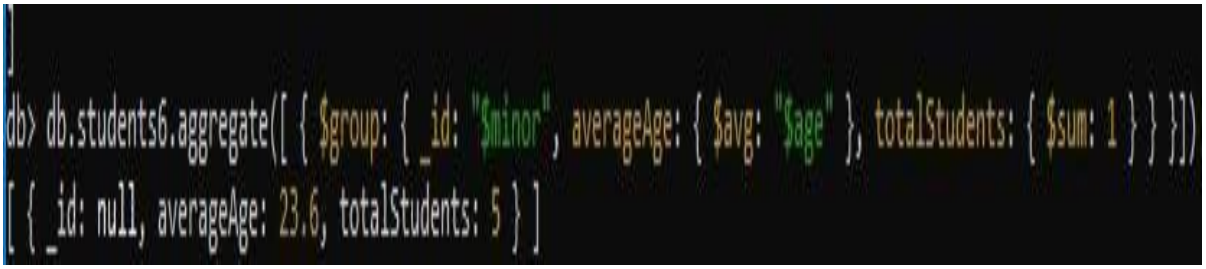



Now to group students by **major** to calculate average age and total number of students in each maajor using **sum:1** we use a command

**db.students6.aggregate([{$group:{_id:"$major",averageAge:{$avg:"$age"},totalStudents:{$sum:1}}}])**


Now to group students by **minor** to calculate average age and total number of students in each maajor using **sum:1** we use a command

**db.students6.aggregate([{$group:{_id:"$minor",averageAge:{$avg:"$age"},totalStudents:{$sum:1}}}])**



## ❖ $project,$skip:

Here to find students with an average score (from scores array) **above** 85 and skip the first document to do this so have to use a command is

**db.students6.aggregate([{$project:{_id:0,name:1averageScore:{$avg:"$scores"}}},{$match:{averageScore:{$gt:85}}},{$skip:1}])**

```
db> db.students6.aggregate([
... {$project:{_id:0,name:1,averageScore:{$avg:"$scores"}}},{$match:{averageScore:{$gt:85}}},{$skip:1}])
[ { name: 'David', averageScore: 93.33333333333333 } ]
```

Again now to find students with an average score (from scores array) **below** 86 and skip the first two document to do this so have to use a command is

**db.students6.aggregate([{$project:{_id:0,name:1averageScore:{$avg:"$scores"}}},{$match:{averageScore:{$lt:86}}},{$skip:2}])**

```
db> db.students6.aggregate([{$project:{_id:0,name:1,averageScore:{$avg:"$scores"}}},{$match:{averageScore:{$lt:86}}},{$skip:2}]);
[ { name: 'Eve', averageScore: 83.33333333333333 } ]
```

Here to find students name with an average score (from scores array) above 95 and skip the first one document to do this so have to use a command is

**db.students6.aggregate([{$project:{name:1,averageScore:{$avg:"$scores"}}},{$match:{averageScore:{$lt:95}}},{$skip:1}])**

```
db> db.students6.aggregate([ { $project: { name: 1, averageScore: { $avg: "$scores" } } }, { $match: { averageScore: { $lt:95
ip: 1 }])
[
  { _id: 2, name: 'Bob', averageScore: 91 },
  { _id: 3, name: 'Charlie', averageScore: 82 },
  { _id: 4, name: 'David', averageScore: 93.33333333333333 },
  { _id: 5, name: 'Eve', averageScore: 83.33333333333333 }
]
```