Aggregation Pipeline

Introduction

MongoDB's aggregation pipeline is a powerful framework for data aggregation and transformation, allowing complex operations to be performed on data stored within MongoDB collections. The pipeline consists of multiple stages, each performing a specific operation on the documents passing through it. This report provides an overview of the aggregation pipeline

1. $match

- **Description**: Filters documents to pass only those that meet the specified conditions.

2. $group

- **Description**: Groups documents by a specified identifier and can perform aggregation operations on grouped data.

3.$project

- **Description**: Reshapes documents by including, excluding, or adding new fields.

4. $sort

- **Description**: Sorts documents in the specified order

5. $limit

- **Description**: Limits the number of documents passed to the next stage.

6. $skip

- **Description**: Skips a specified number of documents.

7. $unwind

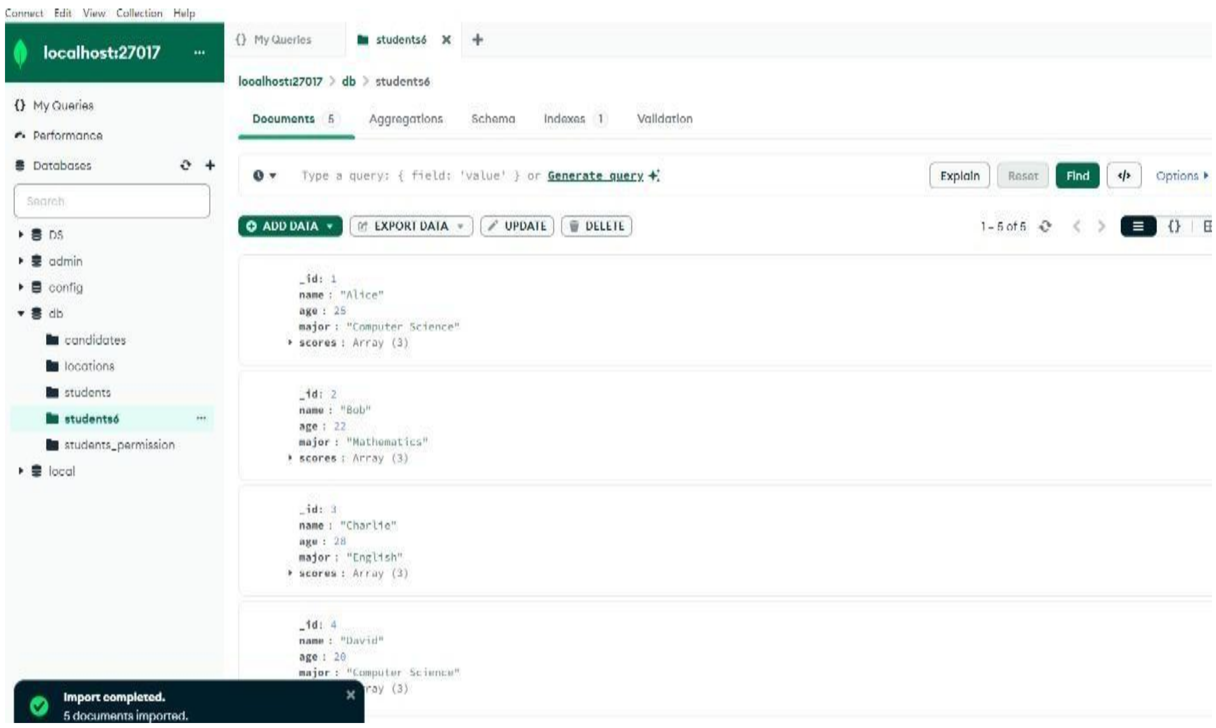
- **Description**: Deconstructs an array field from the input documents to output a document for each element of the array.

8. $lookup

- **Description**: Performs a left outer join to another collection.

9. $out

- **Description**: Writes the resulting documents of the aggregation pipeline to a collection.



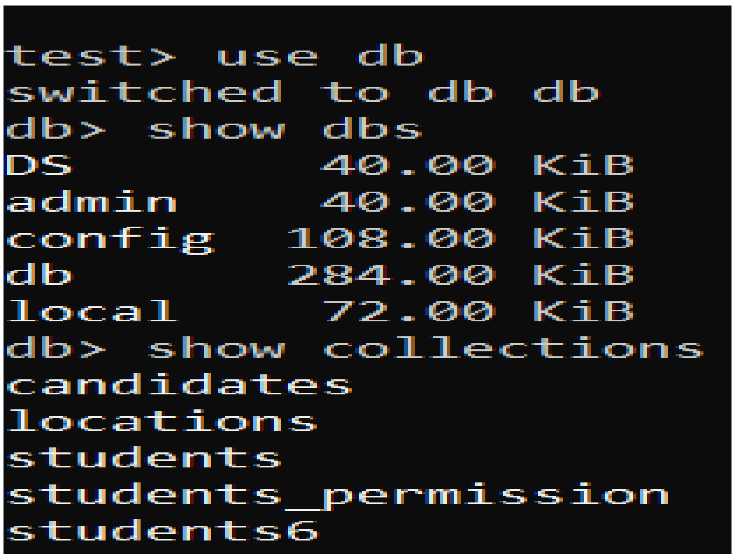
```
_id: 1
name : "Alice"
age : 25
major : "Computer Science"
scores : Array (3)

_id: 2
name : "Bob"
age : 22
major : "Mathematics"
scores : Array (3)

_id: 3
name : "Charlie"
age : 28
major : "English"
scores : Array (3)

_id: 4
name : "David"
age : 20
major : "Computer Science"
```

Import completed.
5 documents imported.

To switch this collection have to use some commands they are

use db show dbs

show collections



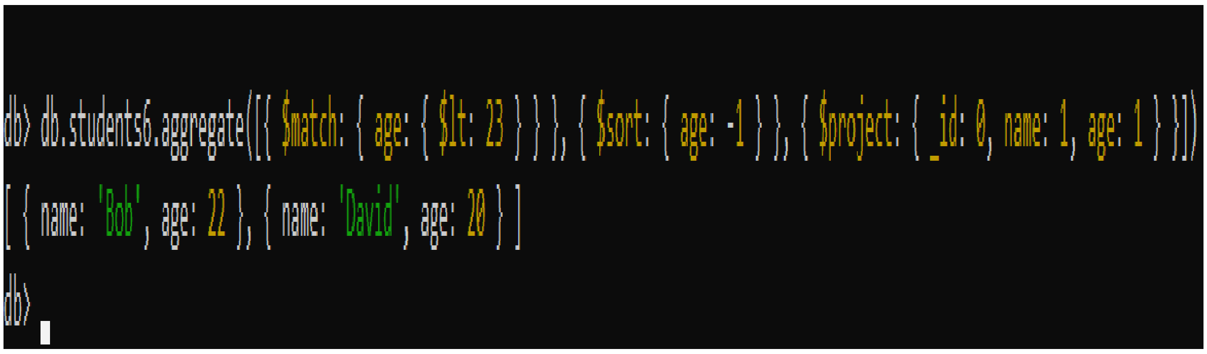
```
test> use db
switched to db db
db> show dbs
DS          40.00 KiB
admin       40.00 KiB
config     108.00 KiB
db         284.00 KiB
local       72.00 KiB
db> show collections
candidates
locations
students
students_permission
students6
```

❖ $match,$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}}])



```
db> db.students6.aggregate([{ $match: { age: { $lt: 23 } } }, { $sort: { age: -1 } }, { $project: { _id: 0, name: 1, age: 1 } }])
[ { name: 'Bob', age: 22 }, { name: 'David', age: 20 } ]
db>
```

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

Here,

$lt:represents less than.

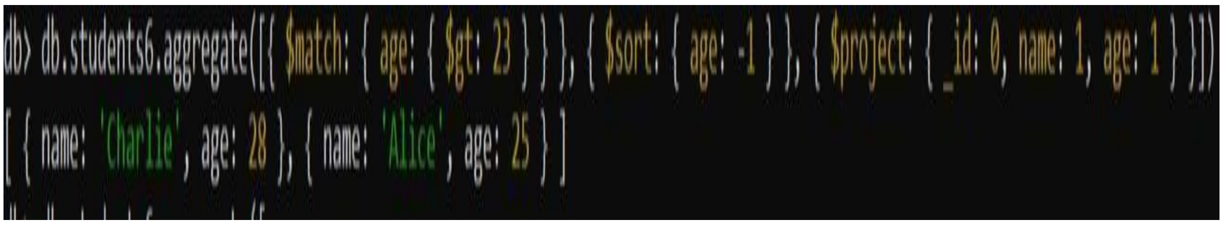
$gt:reprents greater than.

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}}])



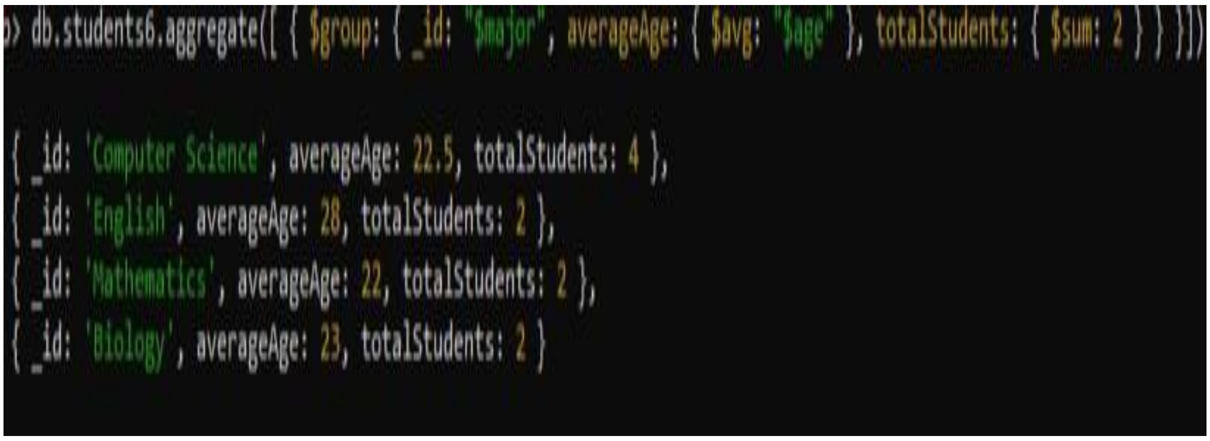
```
db> db.students6.aggregate([{ $match: { age: { $gt: 23 } } }, { $sort: { age: -1 } }, { $project: { _id: 0, name: 1, age: 1 } }])
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
```

❖ **$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

.

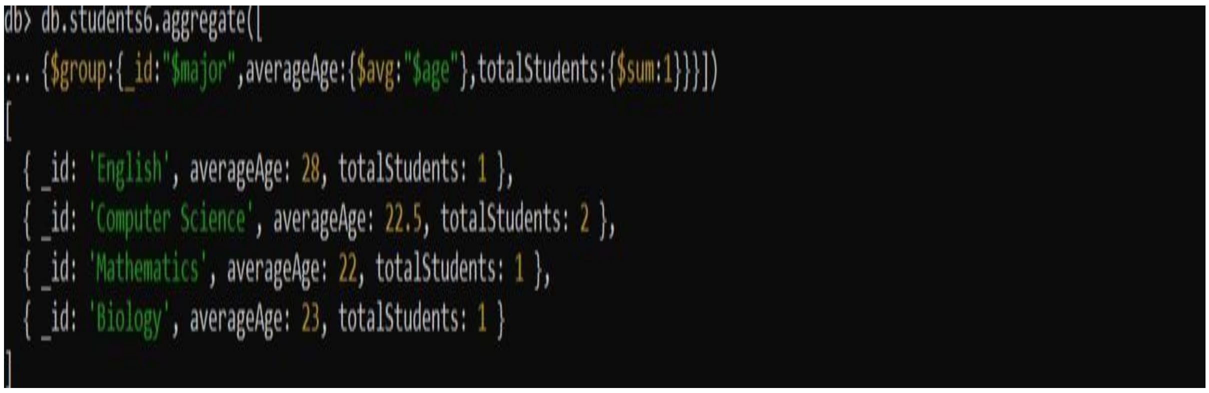
db.students6.aggregate([{$group:{_id:"$major",averageAge:{$avg:"$ age"},totalStudents:{$sum:2}}}])



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

{ _id: 'Computer Science', averageAge: 22.5, totalStudents: 4 },
{ _id: 'English', averageAge: 28, totalStudents: 2 },
{ _id: 'Mathematics', averageAge: 22, totalStudents: 2 },
{ _id: 'Biology', averageAge: 23, totalStudents: 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

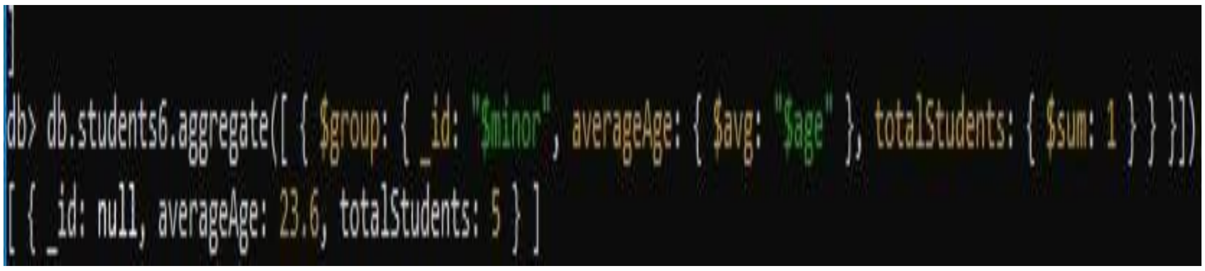
db.students6.aggregate([{$group:{_id:"$major",averageAge:{$avg:"$ age"},totalStudents:{$sum:1}}}])



```
db> db.students6.aggregate([
... {$group:{_id:"$major",averageAge:{$avg:"$age"},totalStudents:{$sum:1}}}])
[
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 },
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 },
  { _id: 'Biology', averageAge: 23, totalStudents: 1 }
]
```

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

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



```
]
db> db.students6.aggregate([ { $group: { _id: "$minor", averageAge: { $avg: "$age" }, totalStudents: { $sum: 1 } } }])
[ { _id: null, averageAge: 23.6, totalStudents: 5 } ]
```

❖ **<u>$project,$skip:</u>**

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

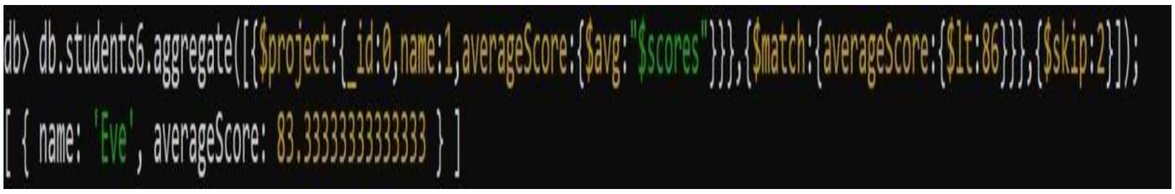
db.students6.aggregate([{$project:{_id:0,name:1averageSc ore:{$avg:"

$scores"}}},{$match:{averageScore:{$gt:85}}},{$skip:1}])

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

db.students6.aggregate([{$project:{_id:0,name:1averageSc ore:{$avg:"

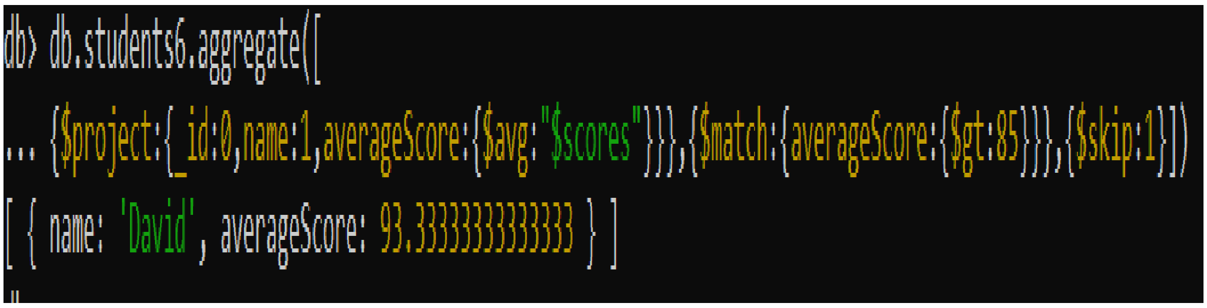
$scores"}}},{$match:{averageScore:{$lt:86}}},{$skip:2}])



```
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<u> skip the first one document</u> to do this so have to use a command is

db.students6.aggregate([{$project:{name:1,averageScore:{ $avg:"$sco
res"}}},{$match:{averageScore:{$lt:95}}},{$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 } ]
```