ARREGATION PIPELINE

The **aggregation pipeline** in MongoDB is a powerful framework for data processing. It consists of one or more stages that process documents. Each stage performs an operation on input documents, such as filtering, grouping, or calculating values. The output from one stage becomes the input for the next, allowing you to build complex data transformations.

The operations of aggregation pipeline are

$match- Filters the documents based on a specified condition, similar to a query. This is often used as an initial stage to reduce the amount of data passing through the pipeline.

$group-Groups documents by a specified identifier expression and applies accumulator expressions (like sum, average, max, min) to each group. This is useful for operations like counting, summing, or averaging values in the documents.

$sort-Sorts the documents based on a specified field or fields. Sorting can be in ascending or descending order.

$project-Reshapes each document by including, excluding, or adding new fields. This is used to transform the structure of the documents.

$limit-Restricts the number of documents passing through the pipeline to a specified number.

$skip-Skips over a specified number of documents and passes the remaining documents to the next stage.

$unwind- Deconstructs an array field from the input documents to output a document for each element of the array. This is useful for normalizing data.

$addFields-Adds new fields to documents with specified values.

$out-Writes the resulting documents to a specified collection. This stage is used for creating new collections or replacing existing ones with the results of the aggregation.

$sample- Randomly selects the specified number of documents from its input.

Now upload the new collection called “students6” by using mongodb compass

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

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
admin     40.00 KiB
config    72.00 KiB
db       288.00 KiB
dbs      640.00 KiB
local     72.00 KiB
db> show collections
candidates
db
locations
players
student
student_permission
db> show collections
candidates
db
```

Now we have to find the students with age greater than 23 it could be sorted by descending order to obtain only name and age

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

$gt-greater than
Age: -1 represents sorting in desending order.

The output of the above code is

```
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
```

Now we have to find the students with age less than 23 it could be sorted by descending order to obtain only name and age

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

$lt-less than
Age:-1 represents sorting in descending order.

The output of the above code is

```
[ { name: 'Bob', age: 22 }, { name: 'David', age: 20 } ]
db>
```

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> db.students6.aggregate([
... {$group:{_id:"$major",sumAge:{$sum:"$age"},totalstudents:{$sum:1}}}
... ])
[
  { _id: 'Mathematics', sumAge: 22, totalstudents: 1 },
  { _id: 'Biology', sumAge: 23, totalstudents: 1 },
  { _id: 'Computer Science', sumAge: 45, totalstudents: 2 },
  { _id: 'English', sumAge: 28, totalstudents: 1 }
]
```

Now to group students by minor to calculate average age and total number of students in each major 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 a command

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

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

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([{$project:{_id:0,name:1,averageScore:{$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([{$project:{_id:0,name:1,averageScore:{$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:{$lt:99}}},{$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 }
```