AGGREGATION OPERATORS

Aggregation operations process multiple documents and return computed results. You can use aggregation operations to:

- Group values from multiple documents together.
- Perform operations on the grouped data to return a single result.
- Analyze data changes over time.

Syntax:

db.collection.aggregate (<AGGREGATE OPERATION>)

Types

Accumulators :

Accumulators are operators that maintain their state (e.g. totals, maximums, minimums, and related data) as documents progress through the pipeline.

$sum :

Calculates and returns the sum of numeric values. $sum ignores non-numeric values.

$avg :

Returns the average value of the numeric values. $avg ignores non-numeric values.

$min :

The $min updates the value of the field to a specified value *if* the specified value is **less than** the current value of the field.

$max :

The $max operator updates the value of the field to a specified value if the specified value is greater than the current value of the field.

$push :

If the value is an array, $push appends the whole array as a single element.

$addToSet :

$addToSet returns an array of all unique values that results from applying an expression to each document in a group.

$first :

This means $first returns the first order type for the documents between the beginning of the partition and the current document.

$last :

$last returns the entire array from the last document. It does not traverse array elements.

Average GPA of all students:

```
db.candidates.aggregate([{ $group: { _id:null, averageGPA: { $avg: “$gpa”}}]);
```

OUTPUT:

```
db> db.candidates.aggregate([{ $group: { _id:null, averageGPA: { $avg: "$gpa"}}}]);
[ { _id: null, averageGPA: 3.5 } ]
```

Minimum and Maximum age :

```
db.candidates.aggregate([{ $group: { _id: null, minAge: { $min:”$age”}, maxAge: {$max: “$age”}}}]);
```

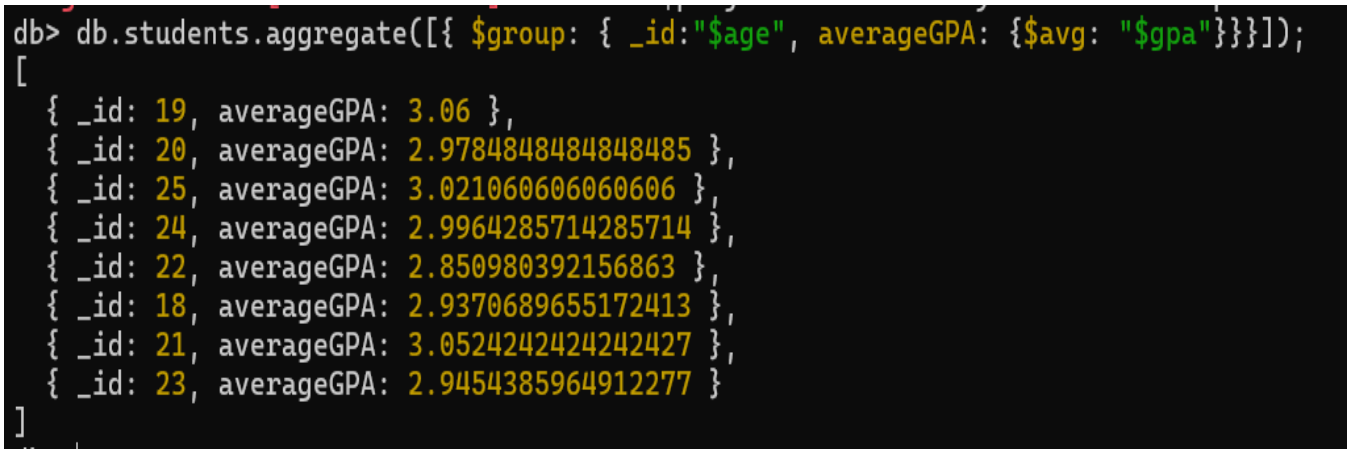
OUTPUT:

```
db> db.candidates.aggregate([{ $group: { _id: null, minAge: { $min:"$age"}, maxAge: {$max: "$age"}}}]);
[ { _id: null, minAge: 18, maxAge: 24 } ]
```

Average GPA for all home cities:

```
db.students.aggregate([{ $group: { _id:"$age", averageGPA: {$avg: "$gpa"}}}]);
```

OUTPUT:



Pushing all Courses into a Single Array

```
db.students.aggregate([{ $project: { _id:0, allCourses: { $push: “$courses”}}}]);
```

OUTPUT:

```
db> db.students.aggregate([
... {$project: { _id:0, allCourses: { $push: "$courses"}}}
... ]);
MongoServerError[Location31325]: Invalid $project :: caused by :: Unknown expression $push
```

Collect Unique Courses Offered (Using $addToSet)

```
db.candidates.aggregate([{$unwind: “$courses”},{ $group: { _id:null, uniqueCourses: { $addToSet: “$courses”}}}]);
```

OUTPUT:

```
db> db.candidates.aggregate([{$unwind: "$courses"},{ $group: { _id:null, uniqueCourses: { $addToSet: "$courses"}}}]);
[
  {
    _id: null,
    uniqueCourses: [
      'Literature',
      'English',
      'Political Science',
      'Creative Writing',
      'Biology',
      'Robotics',
      'Sociology',
      'Chemistry',
      'Music History',
      'Marine Science',
      'Physics',
      'Mathematics',
      'Statistics',
      'Philosophy',
      'Film Studies',
      'Psychology',
      'History',
      'Environmental Science',
      'Artificial Intelligence',
      'Art History',
      'Cybersecurity',
      'Computer Science',
      'Ecology',
      'Engineering'
    ]
```

```
db.students.aggregate([{$unwind: “$home_city”},{ $group: { _id:null, uniqueCourses: { $addToSet: “$home_city”}}}]);
```

OUTPUT:

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.

- **$match :**

  This aggregation stage behaves like a find. It will filter documents that match the query provided.

- **$group :**

  This aggregation stage groups documents by the unique _id expression provided.

- **$sort :**

  This aggregation stage groups sorts all documents in the specified sort order.

- **$project :**

  This aggregation stage passes only the specified fields along to the next aggregation stage.

- **$limit :**

  This aggregation stage limits the number of documents passed to the next stage.

- **$unwind :**

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

- **$skip :**

  $skip takes a positive integer that specifies the maximum number of documents to skip.

1.A) Find students with age greater than 20, sorted by age in descending order, and only return name and age.

```
db.students6.aggregate([{ $match: { age: { $gt: 20}}},
{ $sort: { age: -1}},{ $project: { _id: 0, name:1, age:
1}}])
```

OUTPUT:

```
db> db.students6.aggregate([{ $match: { age: { $gt: 20}}}, { $sort: { age: -1}},{ $project: { _id: 0, name:1, age: 1}}])

[
  { name: 'Charlie', age: 28 },
  { name: 'Alice', age: 25 },
  { name: 'Eve', age: 23 },
  { name: 'Bob', age: 22 }
]
```

B) Find students with age less than 26,sorted by name in ascending order , and only return name and score.

```
db.students6.aggregate([{ $match: { age: { $lt: 26}}},
{ $sort: { age: -1}},{ $project: { _id: 0, name:1, age:
1}}])
```

OUTPUT:

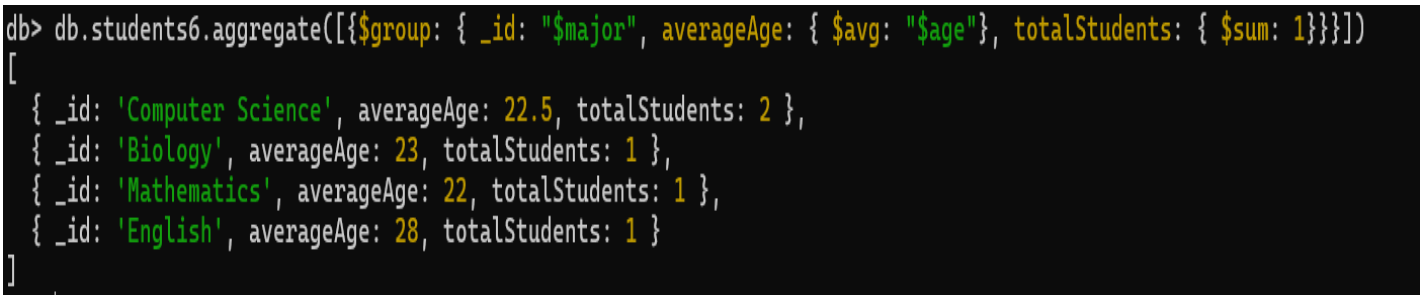
```
db> db.students6.aggregate([{ $match: { age: { $lt: 26}}}, { $sort: { age: -1}},{ $project: { _id: 0, name:1, age: 1}}])

[
  { name: 'Alice', age: 25 },
  { name: 'Eve', age: 23 },
  { name: 'Bob', age: 22 },
  { name: 'David', age: 20 }
]
```

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

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

OUTPUT:



3.A) Find students with an average score (from scores array) above 75 and skip the first document.

```
db.students6.aggregate([{ $project: { _id:0, name:1, averageScore: { $avg: “$scores”}}},{ $match: { averageScore: { $gt:75 }}}, { $skip: 1}])
```

OUTPUT:

```
db> db.students6.aggregate([{ $project: { _id:0, name:1, averageScore: { $avg: "$scores"}}},{ $match: { averageScore: {
$gt:75 }}}, { $skip: 1}])
[
  { name: 'Bob', averageScore: 91 },
  { name: 'Charlie', averageScore: 82 },
  { name: 'David', averageScore: 93.33333333333333 },
  { name: 'Eve', averageScore: 83.33333333333333 }
]
```

3.B)Find students with an average score (from scores array) below 86 and skip the first 2 documents.

```
db.students6.aggregate([{ $project: { _id:0, name:1,
averageScore: { $avg: “$scores”}}},{ $match: { averageScore:
{ $lt: 96 }}}, { $skip: 2}])
```

OUTPUT:

```
db> db.students6.aggregate([{ $project: { _id:0, name:1, averageScore: { $avg: "$scores"}}},{ $match: { averageScore: {
$lt: 96 }}}, { $skip: 2}])
[
  { name: 'Charlie', averageScore: 82 },
  { name: 'David', averageScore: 93.33333333333333 },
  { name: 'Eve', averageScore: 83.33333333333333 }
]
```