Aggregation operators

Agenda:

- **Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)**

- **Introduction:**

- Expression Operators
- Arithmetic Expression Operators
- Array Expression Operators
- Bitwise Operators
- Boolean Expression Operators
- Comparison Expression Operators
- Conditional Expression Operators
- Custom Aggregation Expression Operators
- Data Size Operators
- Date Expression Operators
- Literal Expression Operator
- Miscellaneous Operators
- Object Expression Operators
- Set Expression Operators
- String Expression Operators
- Text Expression Operator
- Timestamp Expression Operators
- Trigonometry Expression Operators
- Type Expression Operators
- Accumulators ($group, $bucket, $bucketAuto, $setWindowFields)
- Accumulators (in Other Stages)
- Variable Expression Operators
- Window Operators

The aggregation pipeline operators are compatible with MongoDB Atlas and on-premise environments.

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

Types:

Expression Type	**Description**	**Syntax**
Accumulators	Perform calculations on entire groups of documents	
* $sum	Calculates the sum of all values in a numeric field within a group.	"$fieldName": { $sum: "$fieldName" }
* $avg	Calculates the average of all values in a numeric field within a group.	"$fieldName": { $avg: "$fieldName" }
* $min	Finds the minimum value in a field within a group.	"$fieldName": { $min: "$fieldName" }
* $max	Finds the maximum value in a field within a group.	"$fieldName": { $max: "$fieldName" }
* $push	Creates an array containing all unique or duplicate values from a field	"$arrayName": { $push: "$fieldName" }
* $addToSet	Creates an array containing only unique values from a field within a group.	"$arrayName": { $addToSet: "$fieldName" }
* $first	Returns the first value in a field within a group (or entire collection).	"$fieldName": { $first: "$fieldName" }
* $last	Returns the last value in a field within a group (or entire collection).	"$fieldName": { $last: "$fieldName" }

Average GPA of All students:

The MongoDB $avg returns the average value of numeric values. The syntax of the $avg is as follows:

```
{ $avg: <expression> }
```
Code language: HTML, XML (xml)

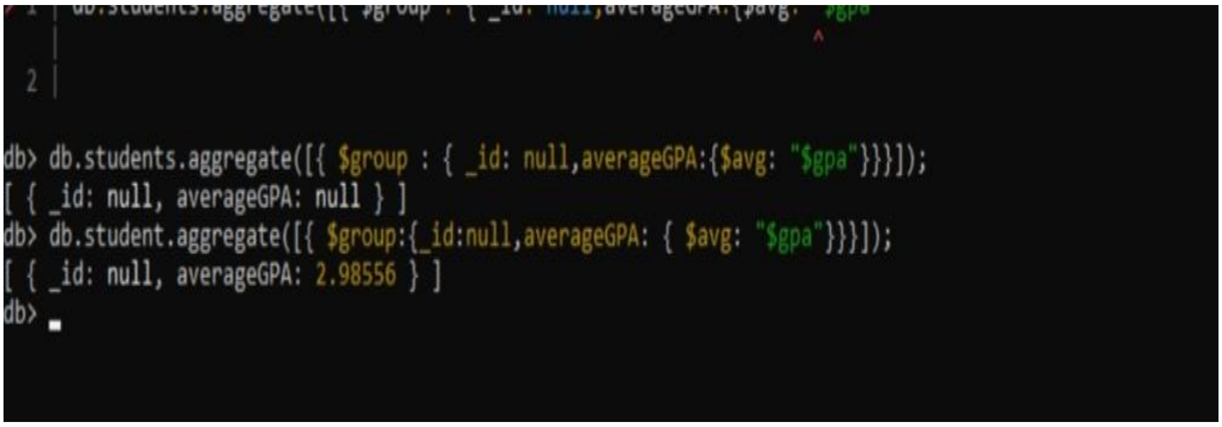
The $avg ignores the non-numeric and missing values. If all values are non-numeric, the $avg returns null

Input:

```
JavaScript

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

Output:



```
db> db.students.aggregate([{ $group : { _id: null,averageGPA:{$avg: "$gpa"}}}]);
[ { _id: null, averageGPA: null } ]
db> db.student.aggregate([{ $group:{_id:null,averageGPA: { $avg: "$gpa"}}}]);
[ { _id: null, averageGPA: 2.98556 } ]
db>
```

Output explaination:

The first line connects to the students collection in the database db

db.students.aggregate(...) performs an aggregation on the students collection. Aggregation pipelines consist of multiple stages that transform data into a desired format.

The $group stage is used to group documents together based on specified criteria and perform operations on each group.

In this case, the pipeline groups all documents in the students collection and calculates the average value of the gpa field.

Within the $group stage:

_id: null specifies that we are not grouping the documents based on any particular field. Instead, all documents will be placed in a single group

averageGPA: {$avg: "$gpa"} calculates the average value of the gpa field for the grouped documents and assigns the result to the field named averageGPA. The $avg operator is used to calculate the average.

The entire aggregation pipeline is wrapped in double square brackets [].

The last line db> indicates that the MongoDB shell is ready to accept new commands.

Running this code will first try to calculate the average GPA but will result in null because there might be no documents in the students collection. The second line corrects the collection name from students to student and successfully calculates the average GPA which is 2.98556.

- `$group`: Groups all documents together.
  - `_id: null`: Sets the group identifier to null (optional, as there's only one group in this case).
  - `averageGPA`: Calculates the average value of the "gpa" field using the `$avg`operator.

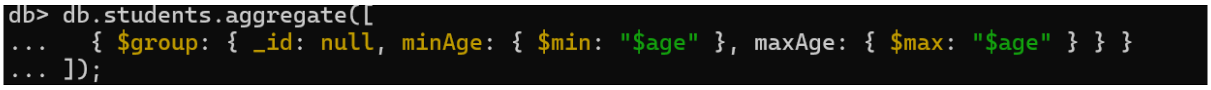
Minimum and maximum age:

Managing data efficiently within a database system is critical for the performance of applications. MongoDB, a popular NoSQL database system, offers various features to handle data effectively, including special compare values known as Min key and Max key. This tutorial will provide an in-depth understanding of what Min key and Max key are in code.

MongoDB and how they can be applied in real-world scenarios withBefore diving into the intricacies of Min key and Max key, it’s vital to grasp their fundamental purpose. Min key represents the lowest possible value in the BSON type ordering, and conversely, Max key denotes the highest possible value in the BSON type ordering. As unlikely as it might seem to have practical applications for the highest and lowest value types, MongoDB utilizes these special types as sentinels in various operations like queries, sharding, and indexing.

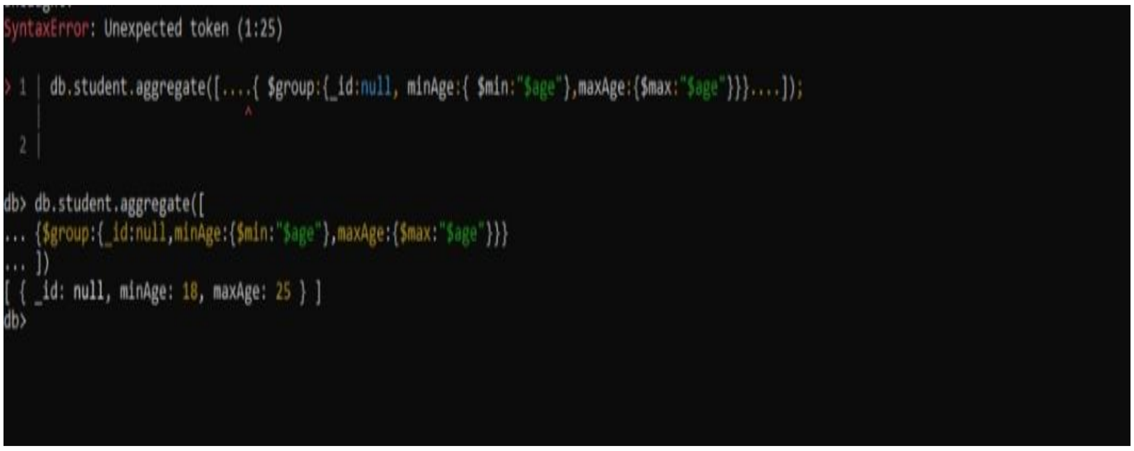
Mongodb-6

Input:



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

Output:



```
SyntaxError: Unexpected token (1:25)

> 1 | db.student.aggregate([....{ $group:{_id:null, minAge:{ $min:"$age"},maxAge:{$max:"$age"}}}....]);
    |                          ^
  2 |

db> db.student.aggregate([
... {$group:{_id:null,minAge:{$min:"$age"},maxAge:{$max:"$age"}}}
... ])
[ { _id: null, minAge: 18, maxAge: 25 } ]
db>
```

Output explaination:

The first line connects to the students collection in the database db.

db.students.aggregate(...) performs an aggregation on the students collection. Aggregation pipelines consist of multiple stages that transform data into a desired format.

The $group stage is used to group documents together based on specified criteria and perform operations on each group.

In this case, the pipeline groups all documents in the students collection and calculates the average value of the gpa field.

Within the $group stage:

_id: null specifies that we are not grouping the documents based on any particular field. Instead, all documents will be placed in a single group.

averageGPA: {$avg: "$gpa"} calculates the average value of the gpa field for the grouped documents and assigns the result to the field named averageGPA. The $avg operator is used to calculate the average.

The entire aggregation pipeline is wrapped in double square brackets [].

The last line db> indicates that the MongoDB shell is ready to accept new commands.

Running this code will first try to calculate the average GPA but will result in null because there might be no documents in the students collection. The second line corrects the collection name from students to student and successfully calculates the average GPA which is 2.98556.

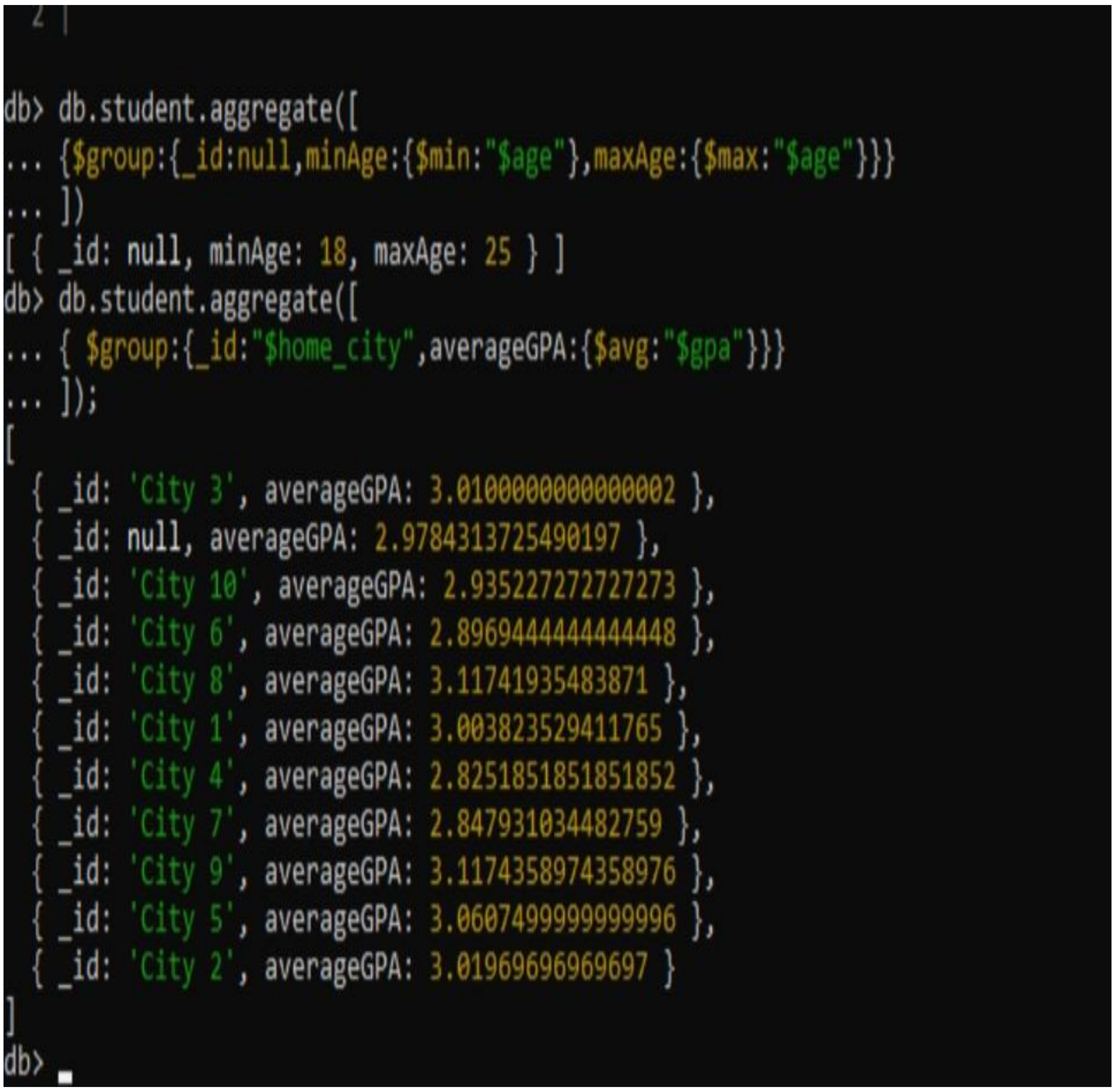
- Similar to the previous example, it uses `$group` to group all documents.
- `minAge`: Uses the `$min` operator to find the minimum value in the "age" field.
- `maxAge`: Uses the `$max` operator to find the maximum value in the "age" field

How to get averageGPA to all home cities.:

Input:

Db.student.aggregate([{$group:{_id:””$home_city”,averageGPA:{$avg:”$gpa”}}}]);

Output:



```
db> db.student.aggregate([
... {$group:{_id:null,minAge:{$min:"$age"},maxAge:{$max:"$age"}}}
... ])
[ { _id: null, minAge: 18, maxAge: 25 } ]
db> db.student.aggregate([
... { $group:{_id:"$home_city",averageGPA:{$avg:"$gpa"}}}
... ]);
[
  { _id: 'City 3', averageGPA: 3.0100000000000002 },
  { _id: null, averageGPA: 2.9784313725490197 },
  { _id: 'City 10', averageGPA: 2.935227272727273 },
  { _id: 'City 6', averageGPA: 2.8969444444444448 },
  { _id: 'City 8', averageGPA: 3.11741935483871 },
  { _id: 'City 1', averageGPA: 3.003823529411765 },
  { _id: 'City 4', averageGPA: 2.8251851851851852 },
  { _id: 'City 7', averageGPA: 2.847931034482759 },
  { _id: 'City 9', averageGPA: 3.1174358974358976 },
  { _id: 'City 5', averageGPA: 3.0607499999999996 },
  { _id: 'City 2', averageGPA: 3.01969696969697 }
]
db>
```

Output explaination:

The code consists of two aggregation pipelines querying a collection named student. Let's break down each pipeline:

Pipeline 1: Finding Minimum and Maximum Age

db.student.aggregate([...]) defines the aggregation pipeline for the student collection.

The first stage uses the $group operator:

_id: null specifies that we are not grouping the documents based on any particular field. Instead, all documents will be placed in a single group.

minAge: {$min: "$age"} finds the minimum value of the age field for the grouped documents and assigns the result to the field named minAge. The $min operator returns the minimum value.

maxAge: {$max: "$age"} finds the maximum value of the age field for the grouped documents and assigns the result to the field named maxAge. The $max operator returns the maximum value.

The entire aggregation pipeline is wrapped in double square brackets [].

The result is displayed at the bottom, showing a single document with:

_id: null (because we didn't group by any field)

minAge: 18 (minimum age in the collection)

maxAge: 25 (maximum age in the collection)

Pipeline 2: Calculating Average GPA by City

Similar to the first pipeline, this one uses db.student.aggregate([...]) to define an aggregation pipeline.

The $group stage is used again:

_id: "$home_city" groups the documents by the home_city field. This means documents with the same city will be grouped together.

averageGPA: {$avg: "$gpa"} calculates the average value of the gpa field for each group and assigns the result to the field named averageGPA. The $avg operator is used to calculate the average.

The entire aggregation pipeline is wrapped in double square brackets [].

The result contains documents with:

_id: City name (since we grouped by city)

averageGPA: Average GPA for students in that city

Pushing all course into a single array:

Explanation:

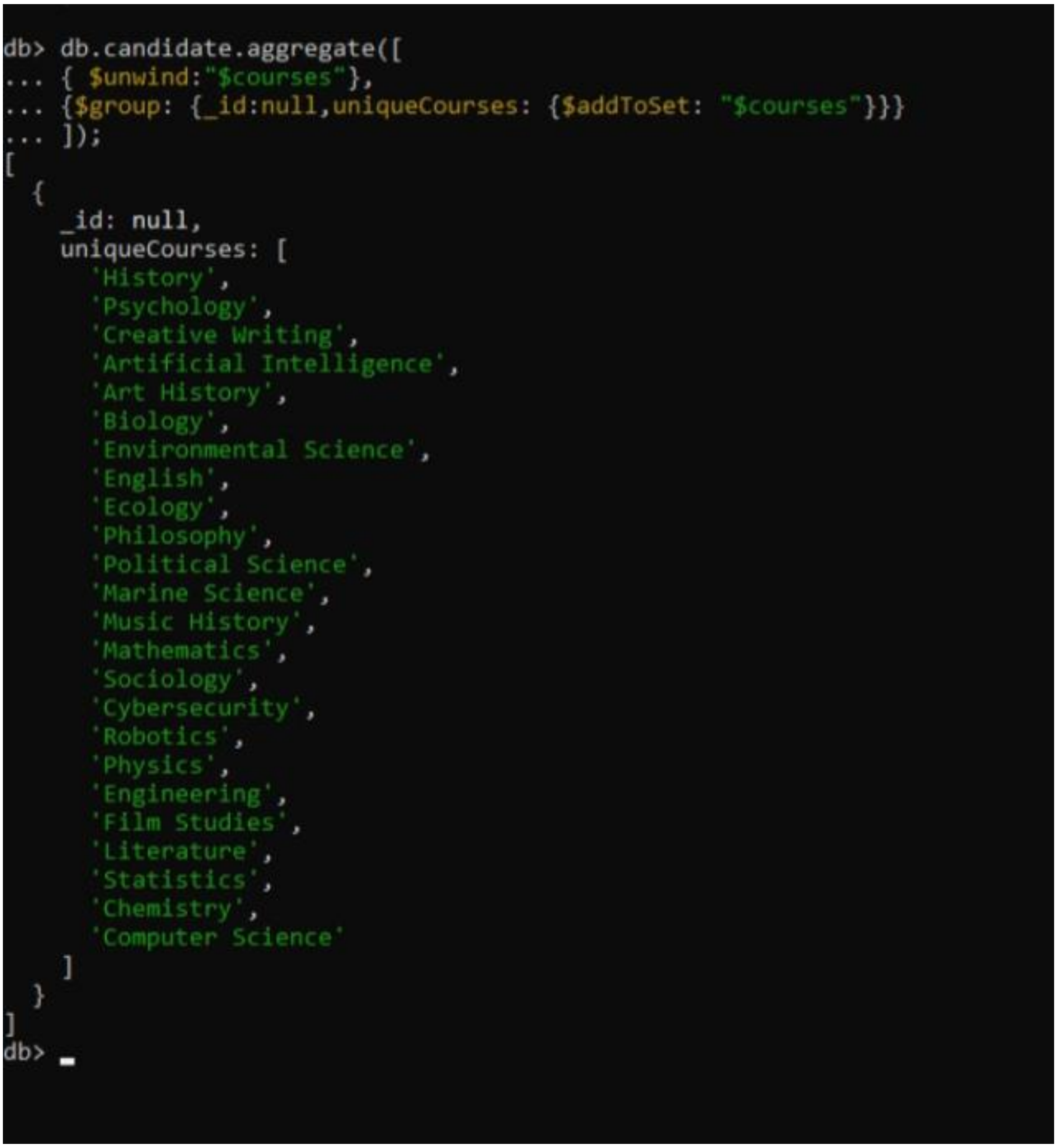
- `$project`: Transforms the input documents.
    - `_id: 0`: Excludes the `_id` field from the output documents.
    - `allCourses`: Uses the `$push` operator to create an array. It pushes all elements from the "courses" field of each student document into the `allCourses` .

Collect unique courses offerd (using $add To Set):

Input:

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

Output:



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

Output explaination:

The first line db.candidate.aggregate([...]) defines the aggregation pipeline for the candidate collection in the database db.

The first stage uses the $unwind operator:

$unwind: "$courses" unwinds the courses array for each document in the pipeline. This means if a candidate has multiple courses in the courses array, the pipeline will create a separate document for each course.

The second stage (which is missing in the image but implied based on the output) likely uses the $group operator.

_id: null specifies that we are not grouping the documents based on any particular field. Instead, all documents will be placed in a single group.

uniqueCourses: {$addToSet: "$courses"} uses the $addToSet operator to add unique values from the courses field to the uniqueCourses array. The $addToSet operator ensures that only distinct courses are included in the final result.

The entire aggregation pipeline is wrapped in double square brackets [].

The last line db> indicates that the MongoDB shell is ready to accept new commands.

The result at the bottom shows a single document with:

_id: null (because we din't group by any field)

uniqueCourses: An array containing all the unique courses offered by students (without duplicates)

In essence, this code snippet helps you find all the distinct courses offered by students in the candidate collection.