

Class 6: Aggregation Operators

Agenda:-

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

Introduction:-

Aggregation operators in MongoDB are powerful tools that allow you to process and analyze data stored in your collections. These operators work together within an **aggregation pipeline**, which is a series of stages that transform documents into summarized results.

Here's a breakdown of key concepts:-

- **Aggregation Pipeline:** This pipeline consists of multiple stages, where each stage performs a specific operation on the documents. Stages can filter documents, group them together, and perform calculations on the grouped data.
- **Aggregation Operators:** These operators are the building blocks within each stage of the pipeline. They carry out various tasks like filtering, grouping, sorting, performing mathematical operations, and more. MongoDB offers a rich set of aggregation operators to handle different data manipulation needs.

Common examples of aggregation operators include:-

- **\$match:** Filters documents based on a set of criteria.
- **\$group:** Groups documents together based on shared field values.
- **\$sum:** Calculates the sum of a numerical field across all documents in a group.
- **\$avg:** Calculates the average of a numerical field within a group.
- **\$push:** Adds elements to an array field.

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" }

Explanation:-

While you can't directly use a .csv file with MongoDB, let's imagine you have the data from students.csv imported into a MongoDB collection named "students".

Here's how aggregation operators can help you analyze student data:

Example 1: Find the average test score

```
db.students.aggregate([  
  
  { $group: { _id: null, averageScore: { $avg: "$testScore" } } }  
  
])
```

This pipeline uses two operators:

- **\$group:** Groups all documents together (since `_id: null`).
- **\$avg:** Calculates the average value of the "testScore" field and assigns the result to "averageScore".

The output will be a document with a single field, "averageScore", showing the average test score across all students.

Example 2: Find students with scores above 90 and their corresponding majors

```
db.students.aggregate([  
  
  { $match: { testScore: { $gt: 90 } } },  
  
  { $group: { _id: "$major", students: { $push: { name: "$name", score: "$testScore" } } } }  
  
])
```

This pipeline uses two operators:

- **\$match:** Filters documents where the "testScore" is greater than 90.
- **\$group:** Groups the remaining documents by "major".
- **\$push:** Within each group, creates an array named "students" containing documents with the student's name and their score.

The output will show documents with each major (where students scored above 90) as the `_id` and an array named "students" containing details of those students in that major.

Average GPA of All Students:-

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

JavaScript

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

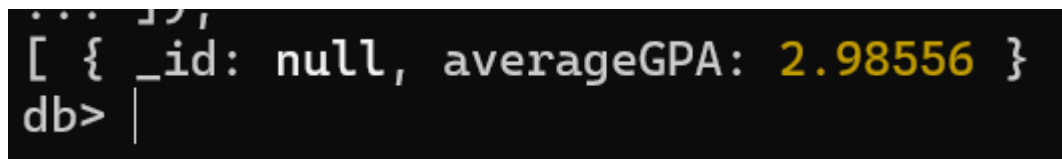
1.db.students.aggregate: This line initiates the aggregation pipeline on the "students" collection.

2.[...]: The square brackets define an array that holds the pipeline stages.

3.{ \$group: ... }: This is the first stage that uses the \$group operator.

- **_id: null:** This specifies that we don't need documents grouped by any particular field. We want all documents grouped together as a single entity.
- **averageGPA: { \$avg: "\$gpa" }:** This calculates the average value of the "gpa" field and assigns the result to a new field named "averageGPA". The \$avg operator computes the average.

OUTPUT:-

A terminal window with a dark background showing the output of a MongoDB aggregation command. The output is a JSON array containing one object: [{"_id": null, "averageGPA": 2.98556}]. The prompt "db>" is visible on the line below the output.

```
[ { "_id": null, averageGPA: 2.98556 } ]
db> |
```

```
[
  { "_id": 1, "name": "Alice", "gpa": 3.8 },
  { "_id": 2, "name": "Bob", "gpa": 3.5 },
  { "_id": 3, "name": "Charlie", "gpa": 3.9 }
]
```

Running the aggregation pipeline would produce an output like:

```
{
  "_id" : null,
  "averageGPA" : 3.7333333333333333
}
```

This indicates that the average GPA among all three students is approximately 3.73.

Maximum And Minimum Age:-

Syntax:

When used in other supported stages, `$max` has one of two syntaxes:

- `$max` has one specified expression as its operand:
`{ $max: <expression> }`
- `$max` has a list of specified expressions as its operand:
`{ $max: [<expression1>, <expression2> ...] }`

CODE:-

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

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

OUTPUT:-

```
[ { _id: null, minAge: 18, maxAge: 25 } ]
```

Explanation:

- 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.

Pushing All Courses into a Single Array:-

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

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

Explanation:

1. **db.students.aggregate**: This line initiates the aggregation pipeline on the "students" collection.
2. **[...]**: The square brackets define an array that holds the pipeline stages.
3. **{ \$project: ... }**: This is the first (and potentially only) stage that uses the \$project operator.
 - **id: 0**: This instruction excludes the "_id" field from the output documents. By default, MongoDB assigns a unique "_id" field to every document. Here, we're indicating that we don't want this field in the results.
 - **allCourses: { \$push: "\$courses" }**: This creates a new field named "allCourses" in the output documents. The \$push operator adds an element to an array. In this case, it pushes the entire "courses" array (presumably containing a list of courses a student is enrolled in) into the new "allCourses" field.

Result:-

- This will return a list of documents, each with an **allCourses** array containing all unique courses offered (assuming courses might be duplicated across students).

BUT:-

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

This is because our Array is incorrect :)

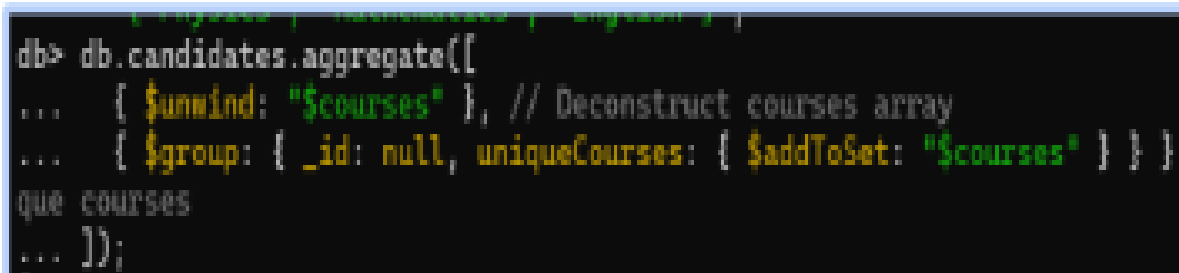
Collect Unique Courses Offered (Using \$addToSet):-

Here's how to collect unique courses offered using the [\\$addToSet](#) operator in MongoDB:-

```

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

```



```

db> db.candidates.aggregate([
...   { $unwind: "$courses" }, // Deconstruct courses array
...   { $group: { _id: null, uniqueCourses: { $addToSet: "$courses" } } }
que courses
... ]);

```

Explanation:-

1. **db.courses.aggregate:** This line initiates the aggregation pipeline on the "courses" collection (assuming this is the name of your collection that stores course information).
2. **[...]:** The square brackets define an array that holds the pipeline stages. In this case, there's only one stage.
3. **{ \$group: ... }:** This is the first (and only) stage that uses the \$group operator.
 - **_id: "\$department":** This specifies how documents should be grouped. Here, documents are grouped based on the value of the "department" field. Each department will have its own group.
 - **uniqueCourses: { \$addToSet: "\$courseCode" }:** This part achieves the core functionality of finding unique courses offered within each department:
 - **uniqueCourses:** This defines the name of the field in the output document that will store the unique course codes.
 - **\$addToSet:** This operator ensures that only distinct values from the "courseCode" field are added to the "uniqueCourses" array. So, if a department offers the same course multiple times with the same code, it will only be included once in the "uniqueCourses" array.

Output:-

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

```

There will be one document for each department in your "courses" collection. Each document's "uniqueCourses" array will contain the distinct course codes offered within that particular department.