Aggregation Operators

Aggregation in MongoDB is a powerful feature that allows for complex data transformations and computations on collections of documents. It enables users to group, filter, and manipulate data to produce summarized results.

It is typically performed using the MongoDB Aggregation Pipeline which is a framework for data aggregation modeled on the concept of data processing pipelines. In this article, We will learn about Aggregation in MongoDB in detail by covering various aspects related to MongoDB Aggregation.

What is aggregation?

MongoDB Aggregation is a database process that allows us to perform complex data transformations and computations on collections of documents or rows.

It enables us to group, filter, and manipulate data to produce summarized results. MongoDB Aggregation is typically carried out using the aggregation pipeline, which is a framework for data aggregation modeled on the concept of data processing pipelines.

Each stage of the pipeline transforms the documents as they pass through it and allowing for operations like filtering, grouping, sorting, reshaping and performing calculations on the data.

Syntax:

```
db.collection_name.aggregate([
  // Stage 1 definition,
  // Stage 2 definition,
  // ... more stages
]);
```

Here's a breakdown of the syntax:

- **db.collection_name**: This specifies the collection on which you want to perform the aggregation.
- **.aggregate :** This method initiates the aggregation process.
- **[]:** This defines the aggregation pipeline, which is an array of stages that your data goes through.
- **// Stage definition:** Each element within the square brackets represents a stage in the pipeline. Each stage definition specifies an aggregation operator and its arguments.

Common Aggregation Stages:

- **$match:** Filters documents based on a condition.
- **$group:** Groups documents together based on a specified field and allows for calculations on the grouped data using operators like **$sum, $avg, $max**, and **$min.**
- **$sort:** Sorts the documents within a group or the entire result set.

- **$project:** Selects or transforms fields in the output documents.
- There are many other stages available for various functionalities

Examples for Expression type:

```
db> db.SStudents.aggregate([ { $group: { _id:null, totalAGE: { $sum: "$age"},totalGPA:{$sum:"$gpa"} } }] );
[ { _id: null, totalAGE: 252, totalGPA: 42 } ]
db> db.SStudents.aggregate([ { $group: { _id:null, averageAGE: { $avg: "$age"},averageGPA:{$avg:"$gpa"} } }] );
[ { _id: null, averageAGE: 21, averageGPA: 3.5 } ]
db> db.SStudents.aggregate([ { $group: { _id:null, minAGE: { $min: "$age"},minGPA:{$min:"$gpa"} } }] );
[ { _id: null, minAGE: 18, minGPA: 3 } ]
db> db.SStudents.aggregate([ { $group: { _id:null, maxAGE: { $max: "$age"},maxGPA:{$max:"$gpa"} } }] );
[ { _id: null, maxAGE: 24, maxGPA: 4 } ]
db> db.SStudents.aggregate([ { $group: { _id:null, collectionAGE: { $push: "$age"},collectionGPA:{$push:"$gpa"}
[
  {
    _id: null,
    collectionAGE: [
      20, 22, 19, 21, 23,
      18, 24, 20, 22, 19,
      21, 23
    ],
    collectionGPA: [
      3.4, 3.8, 3.2, 3.6,   3,
      3.5, 3.9, 3.3, 3.7, 3.1,
        4, 3.5
    ]
  }
]
```

In above examples

Line 1:

- This line performs an aggregation on the "students" collection in the database **(db.students.aggregate).**
- The aggregation pipeline contains an array with one element, which defines the grouping operation **([{ $group: {...}}]).**
- The $group operator is used to group documents together based on certain criteria and perform calculations on the grouped data.
  - In this case, **id: null** sets the **_id** field of the grouped documents to null.

- totalAGE: **{ $sum: "$age" }** calculates the sum of the **"age"** field for each group and stores the result in a new field named "totalAGE".
- totalGPA: **{$sum:"$gpa"}** calculates the sum of the **"gpa"** field for each group and stores the result in a new field named "totalGPA".

Line 2:

- This line performs another aggregation on the **"students"** collection.
- Similar to the first line, it uses the **$group** operator to group documents.

- Here, it again sets the _id field to null using id: null.
- averageAGE**: { $avg: "$age" }** calculates the average of the **"age"** field for each group and stores the result in a new field named "averageAGE".
- averageGPA**: {$avg: "$gpa"}** calculates the average of the **"gpa"** field for each group and stores the result in a new field named "averageGPA".

Line 3:

This line groups again using the $group operator.

- It sets _id to null with id: null.
- minAGE: **{ $min: "$age" }** finds the minimum value of the "age" field for each group and stores it in a new field named "minAGE".
- minGPA: **{$min: "$gpa"}** finds the minimum value of the "gpa" field for each group and stores it in a new field named "minGPA"

Line 4:

This line is similar to line 3, but it uses the $max operator to find the maximum values.

- Similar to previous lines, it sets _id to null with id: null.
- maxAGE: **{ $max: "$age" }** finds the maximum value of the "age" field for each group and stores it in a new field named "maxAGE".
- maxGPA: **{$max:"$gpa"}** finds the maximum value of the "gpa" field for each group and stores it in a new field named "maxGPA".

Line 5:

This line groups again using the $group operator. Here, it collects all the values into arrays instead of performing calculations.

- It sets _id to null with id: null.
- collectionAGE: **{ $push: "$age" }** creates a new field named "collectionAGE" and pushes all the "age" values from the documents into this array.
- collectionGPA: **{$push:"$gpa"}** creates a new field named "collectionGPA" and pushes all the "gpa".

Line 6:

- This line performs an aggregation on the "SStudents" collection in the database (db.SStudents.aggregate).
- The aggregation pipeline contains an array with one element, which defines the grouping operation ([{ $group: {...}}]).
- The $group operator is used to group documents together based on certain criteria and perform calculations on the grouped data.

- In this case, id: null sets the _id field of the grouped documents to null.
- setAGE: **{ $addToSet: "$age" }** uses the $addToSet operator to add unique entries of the "age" field for each group and stores the results in a new field named "setAGE".
- setGPA: **{$addToSet:"$gpa"}** uses the $addToSet operator to add unique entries of the "gpa" field for each group and stores the results in a new field named "setGPA".

Line 7:

- This line performs another aggregation on the "SStudents" collection.
- It uses the $group operator to group documents, similar to line 1.

- Here, it again sets the _id field to null using id: null.
- firstAGE**: { $first: "$age" }** finds the first value of the "age" field for each group and stores the result in a new field named "firstAGE".
- firstGPA: **{$first: "$gpa"}** finds the first value of the "gpa" field for each group and stores the result in a new field named "firstGPA".

Line 8:

This line groups again using the $group operator.

- It sets _id to null with id: null.
- lastAGE: **{ $last: "$age" }** finds the last value of the "age" field for each group and stores it in a new field named "lastAGE".
- lastGPA: **{$last:"$gpa"}** finds the last value of the "gpa" field for each group and stores it in a new field named "lastGPA".

MongoDB

To get Average GPA for all home cities

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
db> db.SStudents.aggregate([ { $group: { _id:"$home_city", averageGPA: { $avg: "$gpa" } } }] );
[
  { _id: 'Phoenix', averageGPA: 3 },
  { _id: 'New York City', averageGPA: 3.4 },
  { _id: 'Los Angeles', averageGPA: 3.8 },
  { _id: 'San Jose', averageGPA: 3.7 },
  { _id: null, averageGPA: 4 },
  { _id: 'Dallas', averageGPA: 3.3 },
  { _id: 'Jacksonville', averageGPA: 3.5 },
  { _id: 'Houston', averageGPA: 3.6 },
  { _id: 'San Antonio', averageGPA: 3.5 },
  { _id: 'Chicago', averageGPA: 3.2 },
  { _id: 'San Diego', averageGPA: 3.9 },
  { _id: 'Austin', averageGPA: 3.1 }
]
```

- This line performs an aggregation on the "students" collection in the database (db.students.aggregate).
- The aggregation pipeline contains an array with one element, which defines the grouping operation ([{ $group: {...}}]).
- The $group operator is used to group documents together based on certain criteria and perform calculations on the grouped data.

- In this case, id: "$home_city" groups the documents by their "home_city" field. The _id field of the grouped documents will be set to the value of the "home_city" field for each group.
- averageGPA: { $avg: "$gpa" } calculates the average of the "gpa" field for each group and stores the result in a new field named "averageGPA".

```
]
db> db.Students.aggregate([ { $group: { _id:"$home_city", averageGPA: { $avg: "$gpa" } } }] );
[
  { _id: 'City 7', averageGPA: 2.847931034482759 },
  { _id: 'City 4', averageGPA: 2.8251851851851852 },
  { _id: 'City 6', averageGPA: 2.8969444444444448 },
  { _id: 'City 5', averageGPA: 3.0607499999999996 },
  { _id: 'City 10', averageGPA: 2.935227272727273 },
  { _id: 'City 9', averageGPA: 3.1174358974358976 },
  { _id: 'City 3', averageGPA: 3.0497297297297297 },
  { _id: 'City 1', averageGPA: 3.003823529411765 },
  { _id: 'City 8', averageGPA: 3.11741935483871 },
  { _id: null, averageGPA: 2.9784313725490197 },
  { _id: 'City 2', averageGPA: 3.01969696969697 }
```

$Push:

In MongoDB, the $push operator is used within the aggregation framework to create or modify an array within the output documents of an aggregation pipeline. It allows you to add elements to an existing array or create a new array if it doesn't exist.

```
db> db.sstudents.aggregate([
... {$group:{
... _id:null,collectAGE:{$push:"$age"},
... collectGPA:{$push:"$gpa"}
... }}
... ]);
[
  {
    _id: null,
    collectAGE: [
      20, 22, 19, 21, 23,
      18, 24, 20, 22, 19,
      21, 23
    ],
    collectGPA: [
      3.4, 3.8, 3.2, 3.6,   3,
      3.5, 3.9, 3.3, 3.7, 3.1,
        4, 3.5
    ]
  }
]
db>
```

This line groups again using the $group operator. Here, it collects all the values into arrays instead of performing calculations.

- It sets _id to null with id: null.
- collectAGE: { $push: "$age" } creates a new field named "collectAGE" and pushes all the "age" values from the documents into this array.
- collectGPA: {$push:"$gpa"} creates a new field named "collectGPA" and pushes all the "gpa".

$addToSet:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
db> db.Students.aggregate([ { $group: { _id:"$name", Couses: { $addToSet: "$courses" } } }] );
[
  {
```

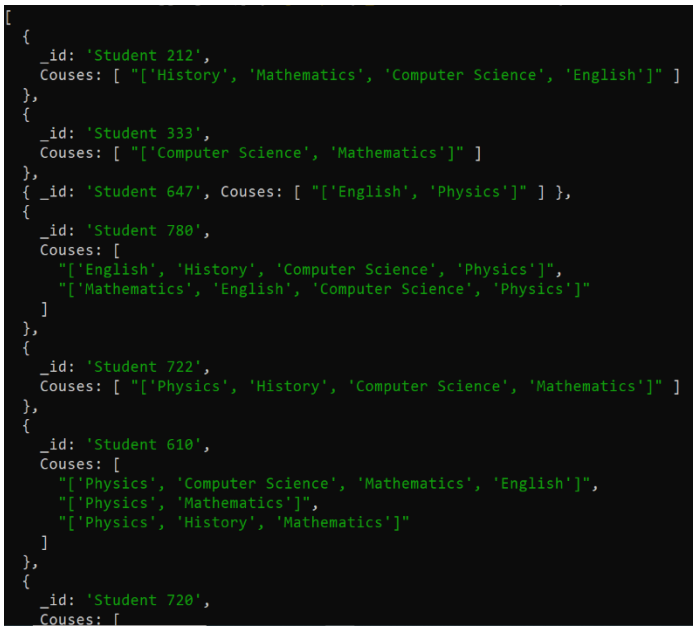
the **$addToSet** operator is used to add elements to an array field in a document. Here's a breakdown of its functionality:

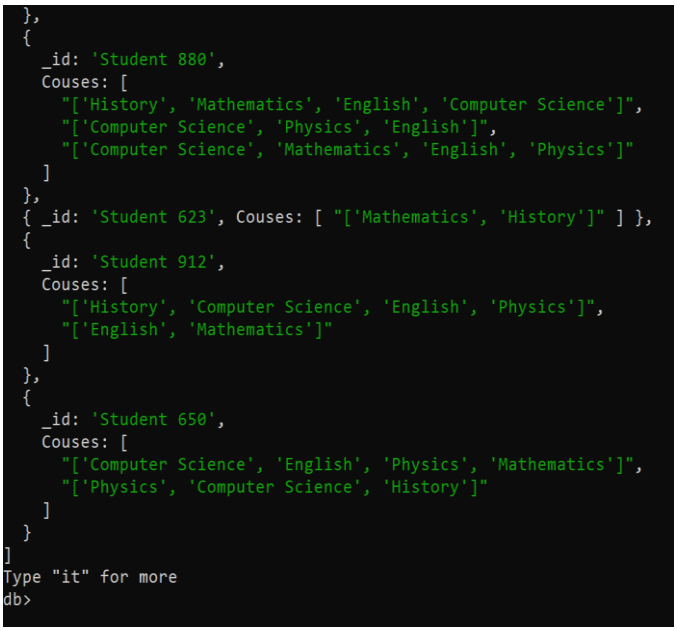
- **Uniqueness:** The key feature of $addToSet is ensuring that only **unique** elements are added to the array. It checks if the element you're trying to add already exists in the array based on a comparison of the entire element value.

- **Single Element vs. Array:** If you provide an array as the element to add**, $addToSet** will treat the entire array as a single element and add it only if the exact array doesn't exist already.
- **Existing Duplicates:** It's important to note that **$addToSet** only prevents adding new duplicates. If there are already duplicate elements within the array before using **$addToSet**, it won't remove them.

Here are some use cases for **$addToSet**:

- **Adding unique items to a shopping cart:** Imagine a shopping cart document with an "items" array. You can use **$addToSet** to add new products to the cart, ensuring no duplicates are accidentally added.
- **Tracking unique skills on a user profile:** In a user profile document, you could have a "skills" array. **$addToSet** would be useful to add new skills the user acquires without creating duplicates.
- **Creating a list of unique followers:** You could use **$addToSet** to update a user document, adding new followers to a "followers" array while preventing duplicate entries.





Collect Unique Courses Offered (Using $addToSet):

In MongoDB, the $addToSet operator is used within the aggregation framework to create an array containing **unique** entries from a specified field. It ensures that each element appears only once in the resulting array, even if it exists multiple times in the original documents.

Example 1:

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

"Science", "English"], unwinding will create separate documents for each course, with the student ID (_id) and the corresponding course name.

- **$group: {_id: "$student_id", uniqueCourses: {$addToSet: "$courses"}}:** This stage groups the unwound documents by their student ID using _id: "$student_id". Then, it calculates the unique courses for each student group using the $addToSet operator.

- $addToSet ensures that each course appears only once in the "uniqueCourses" array for each student. Even if a student has the same course listed multiple times in the original "courses" array, it will only be included once in the "uniqueCourses" array after unwinding and grouping.

The **$unwind** operator is used in a different scenario. It's specifically designed to work with arrays within documents. Here's how it functions:

- **Deconstructing Arrays:** $unwind takes an array field from a document and replicates the document for each element in the array. In essence, it "unwinds" the array into separate documents.
- **Filtering Out Missing Arrays:** By default, **$unwind** skips processing documents where the array field is missing, null, or empty.

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
[
  {
    _id: null,
    uniqueCourses: [
      'Music History',
      'Chemistry',
      'Film Studies',
      'Political Science',
      'Creative Writing',
      'Physics',
      'English',
      'Computer Science',
      'History',
      'Statistics',
      'Philosophy',
      'Environmental Science',
      'Marine Science',
      'Art History',
      'Sociology',
      'Psychology',
      'Literature',
      'Artificial Intelligence',
      'Biology',
      'Robotics',
      'Mathematics',
      'Engineering',
      'Ecology',
      'Cybersecurity'
    ]
  }
]
```

The output of this pipeline will be a new set of documents, each containing:

- **_id:** The student ID from the original document.
- **uniqueCourses:** An array containing the unique courses for that student.