EXPRIMENT 05

AGGREGATION OPERATORS

Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.).

Aggregation operations process data records and return computed results. In this tutorial, we will explore how to use the aggregation framework to calculate the minimum (MIN), maximum (MAX), sum (SUM), average (AVG), and count (COUNT) .

The aggregation framework in MongoDB is like a pipeline, where documents enter and are transformed as they pass through multiple stages. The stages include filtering, projecting, grouping, and sorting documents. Through these stages, you can shape the data to summarize or calculate metrics easily.

Operator	Meaning
$count	Calculates the quantity of documents in the given group.
$max	Displays the maximum value of a document's field in the collection.
$min	Displays the minimum value of a document's field in the collection.
$avg	Displays the average value of a document's field in the collection.
$sum	Sums up the specified values of all documents in the collection.
$push	Adds extra values into the array of the resulting document.

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" }

Counting Documents:

To count documents in a collection, we use **$count**:

Input:

```
db> db.std.aggregate([ {$count:"name"}]);
```

This code is querying a collection named "std" in a database named "db". The aggregation framework in MongoDB allows you to process data and return computed results. In this case, the

`$count` aggregation operator is being used to count the number of documents in the "std" collection.

Output:

```
db> db.std.aggregate([ {$count:"name"}]);
[ { name: 500 } ]
db>
```

The output shows that there are 500 documents in the "std" collection. The name field in the output document reflects the field that was used in the **`$count`** aggregation operator.

Calculating MINIMUM and MAXIMUM values:

For finding the minimum and maximum sale prices you can use the **$group** stage with **$min** and **$max** accumulators:

Example 1:

Input:

```
db> db.std.aggregate([
... {
... $group:{
... _id:null,
... minGPA:{$min:"$gpa"},
... maxGPA:{$max:"$gpa"},
... }}
... ]);
```

To find the minimum and maximum grade point averages (GPA) in the collection.

Output:

- **db.std.aggregate -** This line initiates the aggregation pipeline. It tells MongoDB to perform an aggregation operation on the std collection in the current database.
- **[{ $group: {...}}] -** This defines the stages of the aggregation pipeline. In this case, there's just a single stage that uses the $group operator.

- **$group -** The $group operator is used to group documents together based on specified criteria and perform calculations on those groups. Here's what it's doing in this specific case:
- **_id: null -** This sets the _id field for the grouped output document to null. Since we're not grouping by any particular field, we set it to null to indicate a single group encompassing all documents.
- **minGPA: { $min: "$gpa" } -** This finds the minimum GPA using the **`$min`** operator. It looks at the field named `"gpa"` in each document and returns the lowest value.
- **maxGPA: { $max: "$gpa" } -** This finds the maximum GPA using the `$max` operator. It looks at the field named `"gpa"` in each document and returns the highest value.

```
db> db.std.aggregate([
... {
... $group:{
... _id:null,
... minGPA:{$min:"$gpa"},
... maxGPA:{$max:"$gpa"},
... }}
... ]);
[ { _id: null, minGPA: 2.01, maxGPA: 3.99 } ]
db>
```

This shows that the minimum GPA in the `std` collection is 2.01 and the maximum GPA is 3.99.

Example 2:

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

- **db.std.aggregate -** This line initiates the aggregation pipeline. It tells MongoDB to perform an aggregation operation on the `std` collection in the current database.
- **[{ $group: {...}}] -** This defines the stages of the aggregation pipeline. In this case, there's just a single stage that uses the `$group` operator.
- **$group -** The `$group` operator is used to group documents together based on specified criteria and perform calculations on those groups. Here's what it's doing in this specific case:

- **_id: null** - This sets the `_id` field for the grouped output document to `null`. Since we're not grouping by any particular field, we set it to null to indicate a single group encompassing all documents.
- **minAge: { $min: "$age" }** - This finds the minimum age using the `$min` operator. It looks at the field named `"age"` in each document and returns the lowest value.
- **maxAge: { $max: "$age" }** - This finds the maximum age using the `$max` operator. It looks at the field named `"age"` in each document and returns the highest value.

The output of the query shows that the minimum age in the `std` collection is 18 and the maximum age is 25.

Calculating the sum and average:

Similar to MIN and MAX, we can calculate the total revenue (using **$sum**) and the average price (using **$avg**) .

$sum:

Calculates and returns the collective sum of numeric values.

Input:

```
db> db.sstudents.aggregate([
...   { $group:
...   { _id: null, totalAGE: { $sum: "$age" },
...   totalGPA: { $sum: "$gpa" } }}]);
```

Output:

```
db> db.sstudents.aggregate([
...   { $group:
...   { _id: null, totalAGE: { $sum: "$age" },
...   totalGPA: { $sum: "$gpa" } }}]);
[ { _id: null, totalAGE: 252, totalGPA: 42 } ]
db>
```

- The code calculates total age and GPA for all students in a collection named **`sstudents.`**
- It uses the aggregation pipeline to process student data.
- It groups all students together (using **`_id: null`**) and then sums their ages and GPAs.
- The final output is a single document with the total age and total GPA for all students.

$avg:

Returns the average value of the numeric values.

Example 1 : Calculating the average **gpa** of student database

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

- **db.std.aggregate:** This line initiates the aggregation pipeline. It tells MongoDB to perform an aggregation operation on the std collection in the current database.
- **[{ $group: {...}}]:** This defines the stages of the aggregation pipeline. In this case, there is just a single stage that uses the $group operator.
- **$group:** The $group operator is used to group documents together based on specified criteria and perform calculations on those groups. Here, it is being used to calculate the average GPA.
- **_id: null:** This sets the _id field for the grouped output document to null. Since we're not grouping by any particular field, we set it to null to indicate a single group encompassing all documents.
- **averageGPA: { $avg: "$gpa" }:** This calculates the average GPA using the $avg operator. It looks at the field named "gpa" in each document in the std collection and returns the average value.

The average GPA in the std collection is 2.98556.

Example 2: Calculating the average gpa of candidates database

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

This shows that the average GPA in the candidates collection is 3.5.

How to get Average GPA for all home cities?

Input:

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

Output:

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

- **db.students.aggregate:** This line tells MongoDB to perform an aggregation operation on the students collection in the current database.
- **[{ $group: {...}}]:** This defines the stages of the aggregation pipeline. In this case, there's just a single stage that uses the $group operator.

- $group: The $group operator is used to group documents together based on specified criteria and perform calculations on those groups. Here, it's being used to calculate the average GPA for each student's home city.
- **id: "$home_city"**: This sets the _id field for the grouped output document to the student's home city. This groups the documents by home city.
- **averageGPA: {$avg: "$gpa"}:** This calculates the average GPA using the $avg operator. It looks at the field named "gpa" in each document in the students collection and returns the average value for each group (home city).

The output of the query shows average GPA for each city listed in the collection. For instance, the first entry shows that the average GPA for students from "City 9" is 3.117.

$push :

$push returns an array of *all* values that result from applying an expression to documents.

Input:

```
db> db.sstudents.aggregate([
... {$group:{
... _id:null,collectAGE:{$push:"$age"},
... collectGPA:{$push:"$gpa"}
... }}
... ]);
```

The code defines an array `students` containing example student data. This is the initial input for the aggregation pipeline.

Output:

```
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
    ]
  }
]
```

- **$match:** This stage filters the input data, keeping only students with the major "Computer Science".
- **$group:** This stage groups the remaining students by their age. It creates a new document for each age group with two fields:
- **_id:** This holds the value of the grouping field (age in this case). By default, it's included in the output.
- **count:** This field uses the $sum: `1` operator to count the number of students in each age group.
- **$project:** This stage defines the format of the final output documents.
- **_id: 0:** This removes the default grouping key `_id` from the output.
- **age: "$_id":** This renames the grouping key `_id` to a clearer name "age" in the output.
- **count: 1:** This keeps the count of students in each age group.

$addToSet:

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

Input:

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

The input to this code snippet is a collection named "candidates" in a MongoDB database. Each document within the "candidates" collection likely has a field named "courses" which is an array. This array contains strings representing the courses a particular candidate has taken.

Output:

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

The output of the code snippet is a new collection containing documents with information about all unique courses across all candidates.

Each document in the output collection will have:

- **_id:** This field is set to null because the aggregation doesn't group by any specific criteria.
- **uniqueCourses:** This field is an array that contains all the unique course names found in the "courses" arrays of every document in the original "candidates" collection.

The output of the query would be a new collection that shows all unique courses a candidate has ever taken. There would be one document for each unique course. Each document would likely have a field named _id set to null and a field named uniqueCourses that is an array containing all of the unique course names found in the candidates collection.