CLASS:6

AGGREGATION OPERATORS

INTRODUCTION:

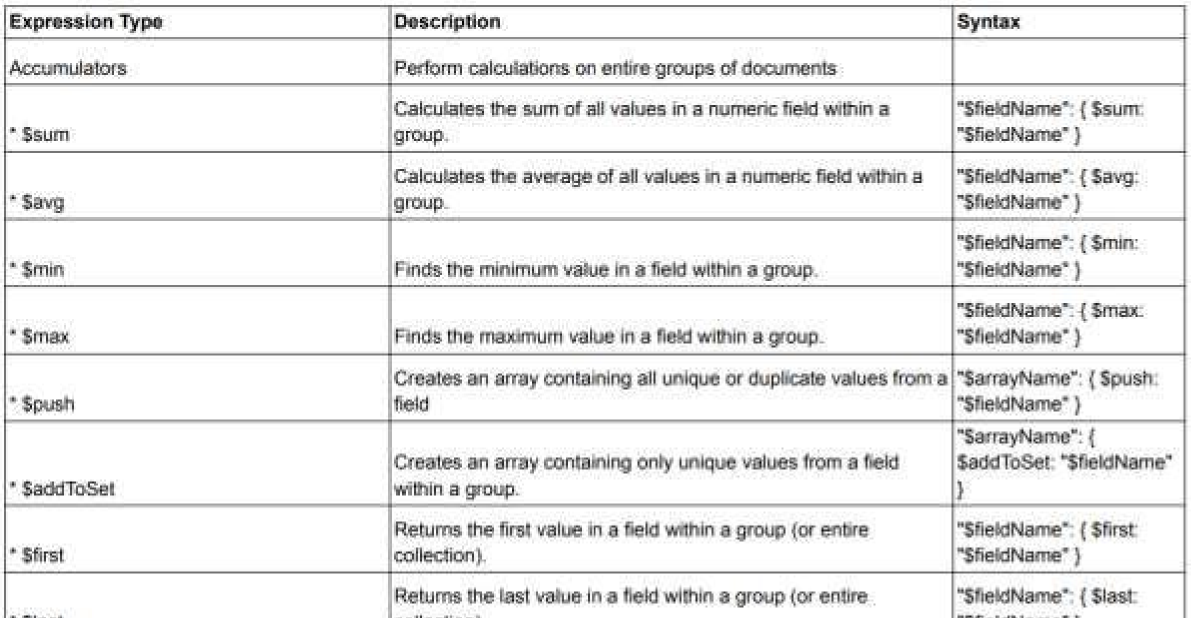
Aggregation means the gathering of things together. In Computer Science, data aggregation means the grouping of data to prepare combined data sets helpful in generating better information. Aggregation in MongoDB groups the data from various collections and then performs various operations to generate one combined result.

Aggregation operations summarize data by performing calculations on a group of values. They take multiple rows and return a single result. Common examples include COUNT, SUM, AVG, MIN, and MAX.

Aggregation method syntax and use

```
db.collection.aggregate(<AGGREGATE OPERATION>)
```

We can use the MongoDB aggregate method to perform the aggregate operation on the documents in MongoDB. We will understand this method better with the help of some practical examples. First, let us take a look at various expressions used in the aggregate operation. Following are the expression types used in the MongoDB aggregated operation.

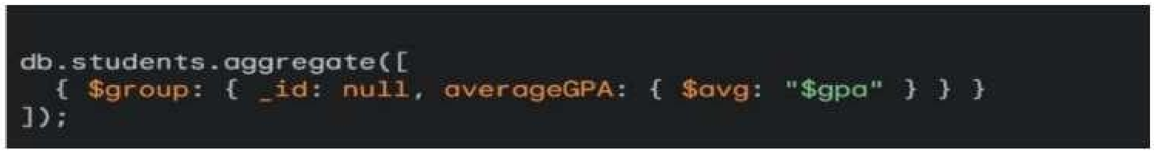


Expression Type	Description	Syntax
Accumulators	Perform calculations on entire groups of documents	
* $sum	Calculates the sum of all values in a numeric field within a group.	"$fieldName": { $sum: "$fieldName" }
* $avg	Calculates the average of all values in a numeric field within a group.	"$fieldName": { $avg: "$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" }

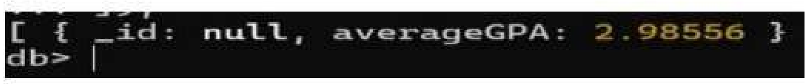
AVERAGE GPA OF ALL STUDENTS:

In MongoDB, you can calculate the average GPA of all students using the aggregation framework. Here's how:

1. Use the aggregate function to initiate the aggregation pipeline.
2. Include a stage with the $group operator.
3. Within the $group stage, specify an accumulator expression with avg to calculate the average.
4. Define what field (e.g., "gpa") to calculate the average on.
5. Optionally, include a filter stage before $group to filter students based on specific criteria. This pipeline calculates the average GPA for all students in the collection.



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



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

EXPLANATION:

In reference with the above code

*$group: Groups all documents together.

_id: null: Sets the group identifier to null (optional, as there's only one group I In this case).

averageGPA: Calculates the average value of the "gpa" field using the $avg operator.

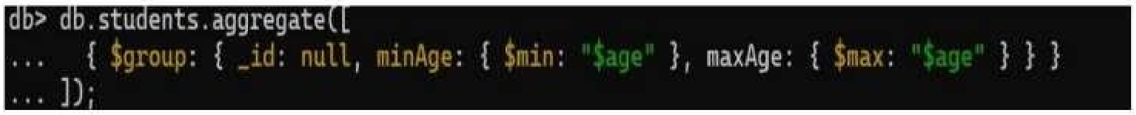
MINIMUM AND MAXIMUM AGE:

While aggregation typically involves summary functions like average, you can achieve minimum and maximum age using different approaches:

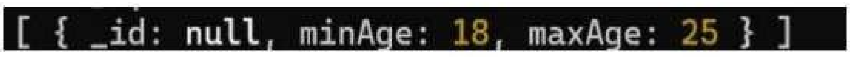
*Seperate aggregation.

*custom class.

Both methods achieve finding minimum and maximum age, but the choice depends on your specific needs and data structure.



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

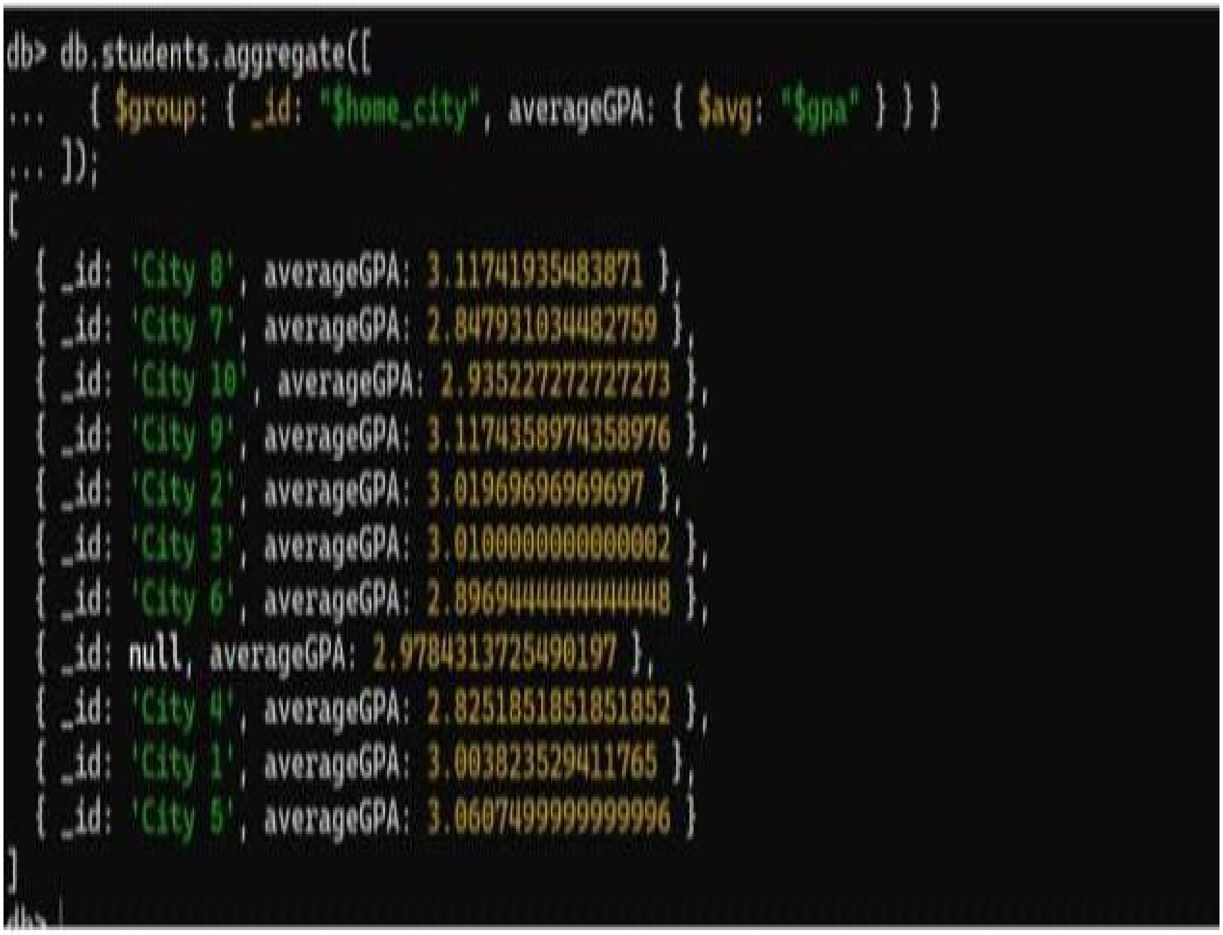


```
[ { _id: null, minAge: 18, maxAge: 25 } ]
```

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

To get average GPA for all home cities:

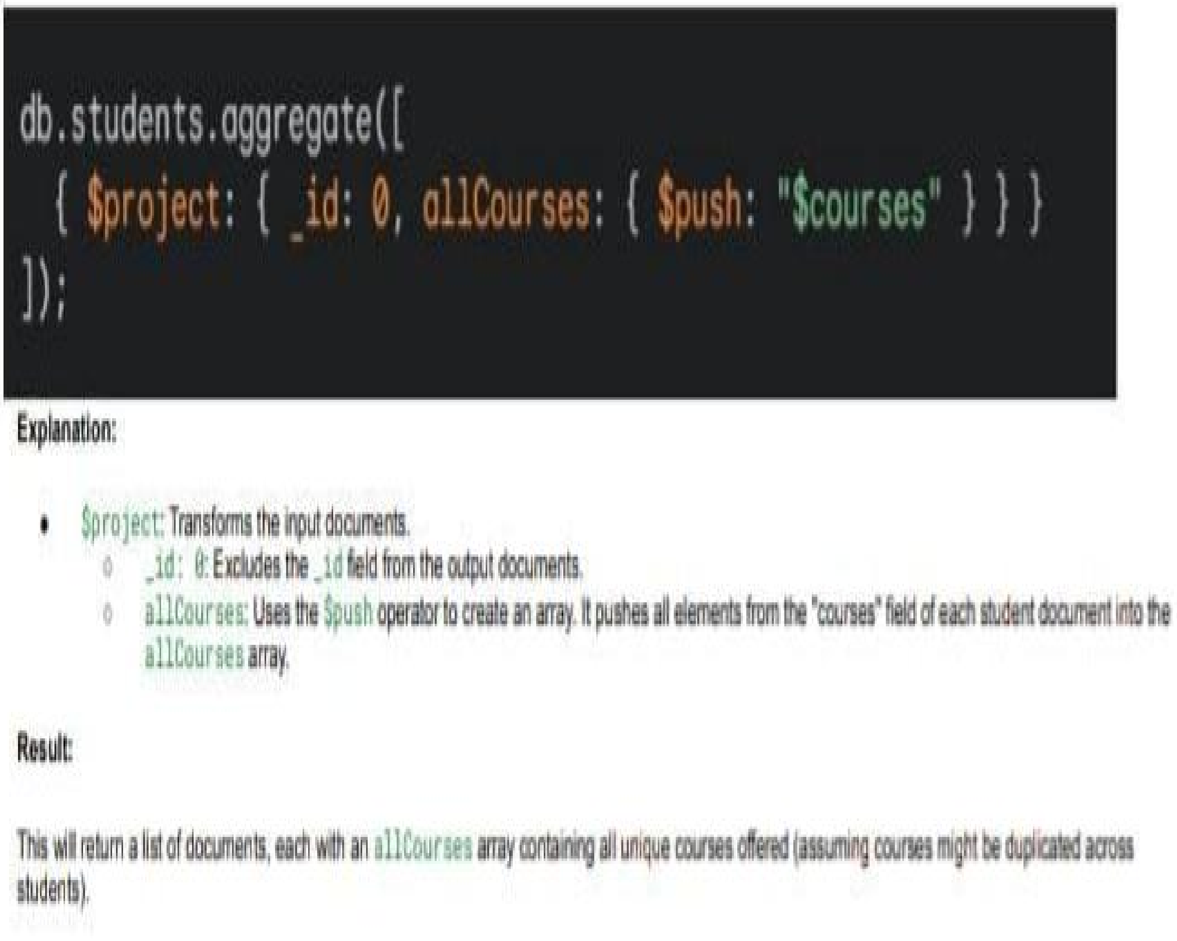


```
db> db.students.aggregate([
...    { $group: { _id: "$home_city", averageGPA: { $avg: "$gpa" } } }
... ]);
[
  { _id: 'City 8', averageGPA: 3.11741935483871 },
  { _id: 'City 7', averageGPA: 2.847931034482759 },
  { _id: 'City 10', averageGPA: 2.935227272727273 },
  { _id: 'City 9', averageGPA: 3.1174358974358976 },
  { _id: 'City 2', averageGPA: 3.0196969696969697 },
  { _id: 'City 3', averageGPA: 3.0100000000000002 },
  { _id: 'City 6', averageGPA: 2.8969444444444448 },
  { _id: null, averageGPA: 2.9784313725490197 },
  { _id: 'City 4', averageGPA: 2.8251851851851852 },
  { _id: 'City 1', averageGPA: 3.003823529411765 },
  { _id: 'City 5', averageGPA: 3.0607499999999996 }
]
db>
```

PUSHING ALL COURSES INTO SINGLE ARRAY:

EXPLANATION:

- Aggregation Pipeline: Utilize the aggregate function to initiate the aggregation pipeline.
- Unwind Courses: Consider using the $unwind operator if your documents have an array field containing courses.



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

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` array.

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).

• Empty Array Initialization: Introduce a stage with the $project operator. Within $project, define a new field (e.g., allCourses) to hold the combined courses and initialize it as an empty array ([]).

• Push Courses: Still within the $project stage, use the $push accumulator to append each course document (unwound or original) to the allCourses array.

Example (assuming courses are stored as separate documents):

1. db.students.aggregate([{ $project: { allCourses: { $push: "$$ROOT" } } } // Push entire document])

2. db.students.aggregate([{ $unwind: "$courses" }, // Deconstruct courses array.

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:

1. Filtering Unwanted Fields:

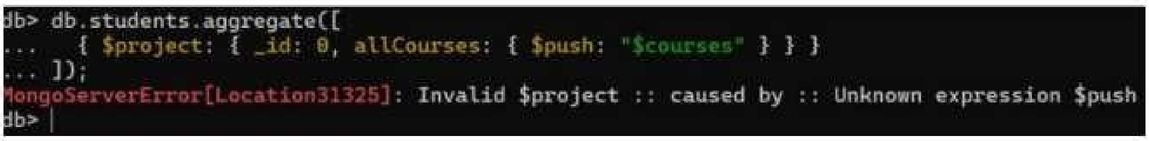
If you want to exclude specific fields while pushing elements, you can leverage the $project stage with exclusion. Here's an example assuming you want to push all courses but exclude the "_id" field:

This approach uses $objectToArray to convert the document to an array of key-value pairs, excludes _id using projection, and then unwinds and extracts the course data.

2. Conditional Push Based on Field Values:

If you want to conditionally push courses based on a specific field value, you can utilize the $cond operator within $project:

The $setDifference stage (optional) removes any null values that might be pushed due to the conditional logic.



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

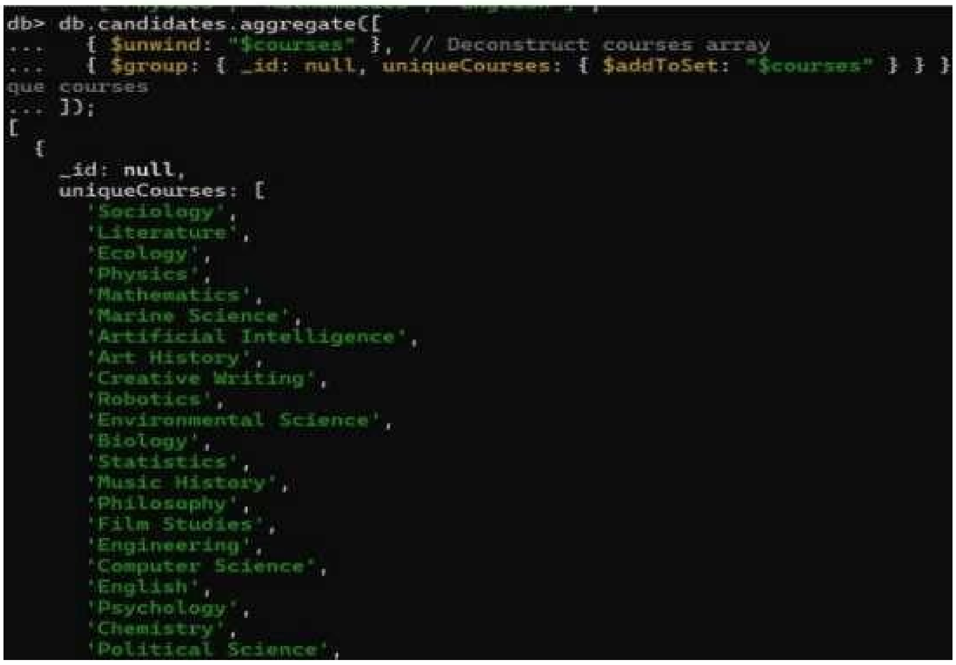
This is because our Array is incorrect :)

Collect Unique Courses Offered (Using $addToSet):

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
... ]);
[
  {
    _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',
```