AGGREGATION OPERATORS

$avg, $min,$max, $push, $addToSet

Aggregation operation where students encourage to execute several queries to demonstrate various aggregation operators.

INTRODUCTION:

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.

SYNTAX:

db.collection.aggregate(<AGGREGATE OPERATION>

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" }

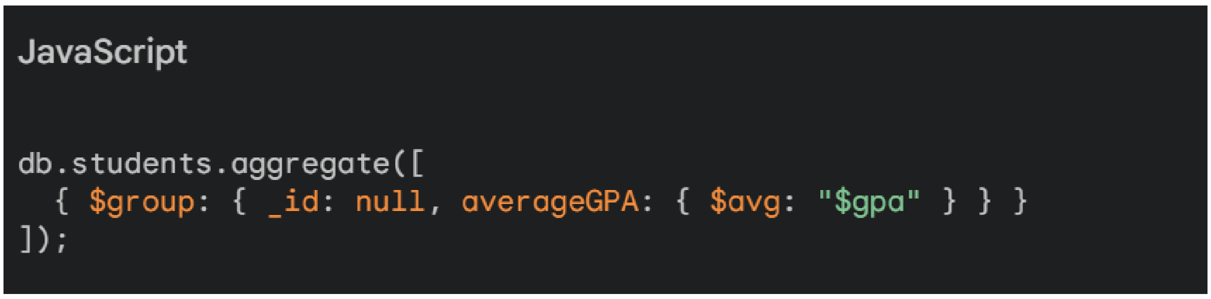
* $last	Returns the last value in a field within a group (or entire collection).	"$fieldName": { $last: "$fieldName" }

AVERAGE GPA OF ALL STUDENTS:

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.

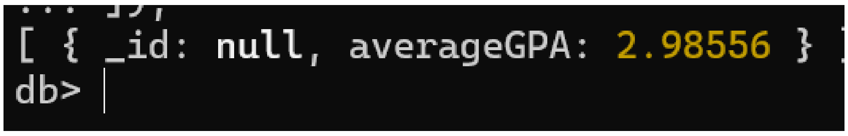
This pipeline calculates the average GPA for all students in the collection.



JavaScript

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

OUTPUT:



```
... ]);
[ { _id: null, averageGPA: 2.98556 }
db>
```

EXPLANATION:

In reference with the above code

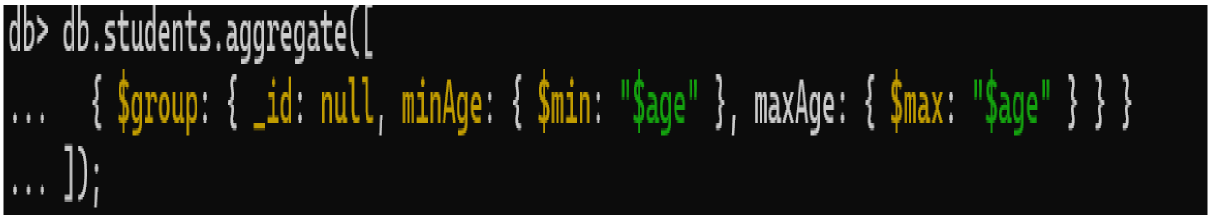
*`$group`: Groups all documents together.
        `_id: null`: Sets the group identifier to null (optional, as there's only one group in this case).
        `averageGPA`: Calculates the average value of the "gpa" field using the `$avg` operator.

MINIMUM AND MAXIMUM AGE:

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

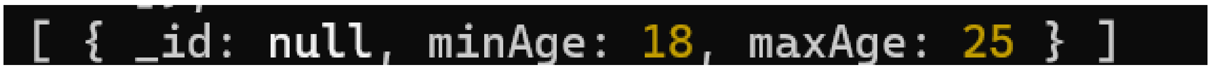
1. **Custom Class:** If age is a property of the part class (e.g., `Student`), you could include getter methods for minimum and maximum age within the whole class. These methods would likely iterate through the part objects (e.g., students in an `Order` class) to find the minimum and maximum age.
2. **Separate Aggregation:** You could perform a separate aggregation using a library like Java Stream API. This approach wouldn't directly rely on the "has-a" relationship but process the part objects (students) independently to find min and max age.

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" } } }
... ]);
```

OUTPUT:



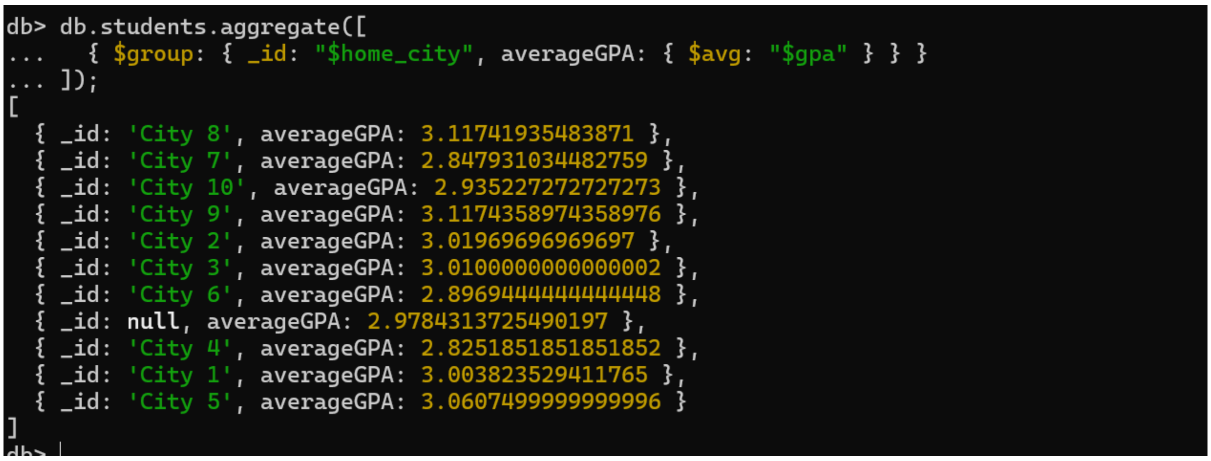
```
[ { _id: null, minAge: 18, maxAge: 25 } ]
```

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.

- **Student Stream:** Obtain a Stream of all student objects (likely from the `Order` class's student list).
- **Map to Age:** Use the `map` function to extract the age property from each student object and create a new Stream of just ages.
- **Min and Max:** Utilize the `min` and `max` functions (available in Stream API) on the age Stream to find the minimum and maximum values.

HOW 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: 3.01969696969697 },
  { _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 `$unwind` operator if your documents have an array field containing courses. This deconstructs the array into separate documents, one for each course. Skip this step if courses are already stored as separate documents.
- **Empty Array Initialization:** Introduce a stage with the `$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

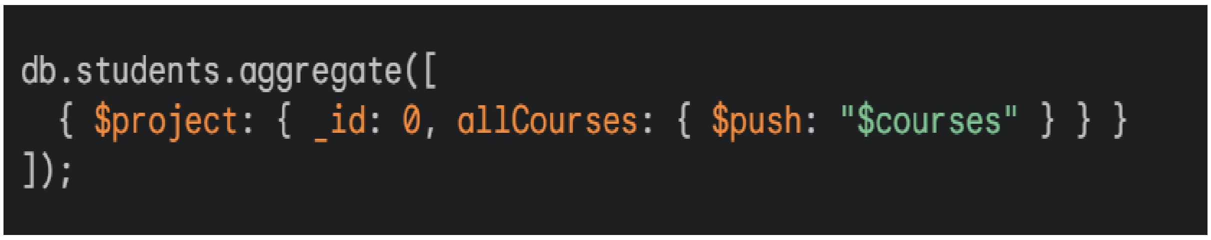
])

2.db.students.aggregate([

{ $unwind: "$courses" }, // Deconstruct courses array

{ $project: { allCourses: { $push: "$courses" } } } // Push each course

])



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

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:**

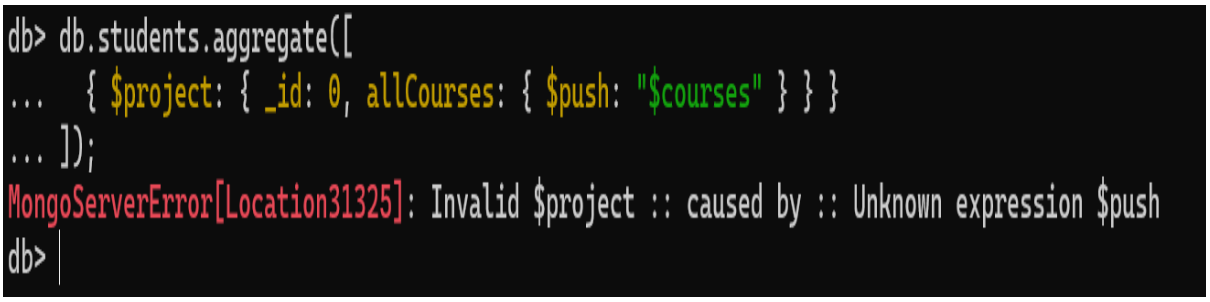
If you want to exclude specific fields while pushing elements, you can leverage the `$project` stage with exclusion. Here's an example assuming you want to push all courses but exclude the "_id" field:

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 UINQUE COURSES OFFERED USING $ADD TO SET:

Here's how to collect unique forces offered using the `$addToSet` operator in MongoDB aggregation:

1. Aggregation Pipeline:

Utilize the `aggregate` function to initiate the aggregation pipeline.

2. $group Stage:

Include a stage with the `$group` operator to group documents potentially containing an array of forces offered.

3. Specify _id:

Within the `$group` stage, define `_id: null` (or another field for specific grouping if needed).

4. $addToSet for Forces:

Also within `$group`, use the `$addToSet` accumulator with the field name containing the forces offered (e.g., "forces"). This ensures only unique elements are added to the resulting array.

JavaScript
```
db.militaryUnits.aggregate([
  {
    $group: {
      _id: null, // Group all documents
      uniqueForces: { $addToSet: "$forces" } // Collect unique forces
offered
    }
  }
])
```

EXPLANATION:

- This pipeline processes all documents in the `militaryUnits` collection.
- The `$group` stage groups all documents into a single document.
- Inside `$group`, `_id: null` instructs it to consider all documents as a single group for force collection.
- The `$addToSet` operator, applied to the "forces" field, ensures only unique entries from the "forces" field across all documents are added to the newly created `uniqueForces` array in the resulting document.

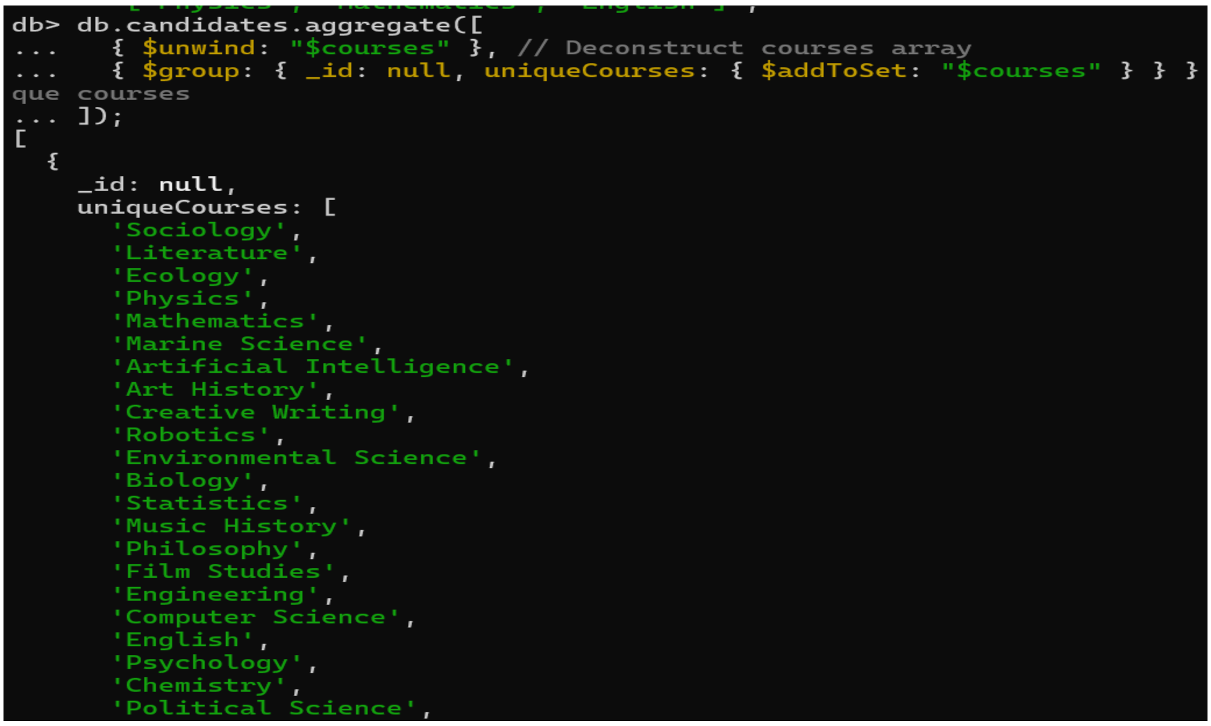
RESULT:

The resulting document will have:

- `_id`: The value you specified (here, `null`).
- `uniqueForces`: An array containing all unique forces offered across all military units in the collection.

share
more_vert

WHAT DOES IT DO:?



```
       [ 'Physics', 'Mathematics', 'English' ],
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',
```