AGGREGATION OPERATIONS

Aggregation Operations:

Aggregation operations process multiple documents and return computed results. You can use aggregation operations to:

- Group values from multiple documents together.
- Perform operations on the grouped data to return a single result.
- Analyze data changes over time.

To perform aggregation operations, you can use:

- Aggregation pipelines, which are the preferred method for performing aggregations.
- Single purpose aggregation methods, which are simple but lack the capabilities of an aggregation pipeline.

Operations can be performed are:

In MongoDB, the aggregation framework allows for complex data processing and transformation tasks, similar to SQL's "**GROUP BY**" and "**HAVING**" clauses. It operates through a pipeline consisting of multiple stages, each transforming the documents as they pass through.

Accumulator Operators

Accumulator operators are used within the **$group** stage to perform calculations on the grouped data.

1. $min

Finds the minimum value.

Syntax:

```
{ $min: "<expression>" }
```

Example:

```
db.collection.aggregate([
{ $group: { _id: "$category", minPrice: { $min: "$price" } } } ])
```

2.$max:

Finds the maximum value.

Syntax:

```
{ $max: "<expression>" }
```

Example:

```
db.collection.aggregate([
{$group:{_id:"$category",maxPrice:{
$max:"$price"}}}])
```

3.$sum:

Calculates the sum of numeric values. Use $sum: 1 to count the number of documents.

Syntax:

```
   {$sum:"<expression>"}
```

Example:

```
   db.collection.aggregate([
{$group:{_id:"$category",totalPrice:{$sum:"$price"}}}])
```

4.$avg:

Calculates the average of numeric values.

Syntax:

```
   { $avg: "<expression>" }
```

Example:

```
 db.collection.aggregate([
 { $group: { _id: "$category", avgPrice: { $avg: "$price" } } }])
```

5.$push:

Adds values to an array.

Syntax:

```
   { $push: "<expression>" }
```

Example:

```
db.collection.aggregate([

{ $group: { _id: "$category", prices: { $push: "$price" } } }])
```

6.$addToSet:

Adds unique values to an array (no duplicates).

Syntax:

```
{ $addToSet: "<expression>" }
```

Example:

```
db.collection.aggregate([
{ $group: { _id: "$category", uniquePrices: { $addToSet: "$price"
 }}}])
```

7.$First:

Returns the first value in a group of documents.

Syntax:

```
{ $first: "<expression>" }
```

Example:

```
db.collection.aggregate([
{ $group: { _id: "$category", firstPrice: { $first: "$price" } } }])
```

8.$last:

Returns the last value in a group of documents.

Syntax:

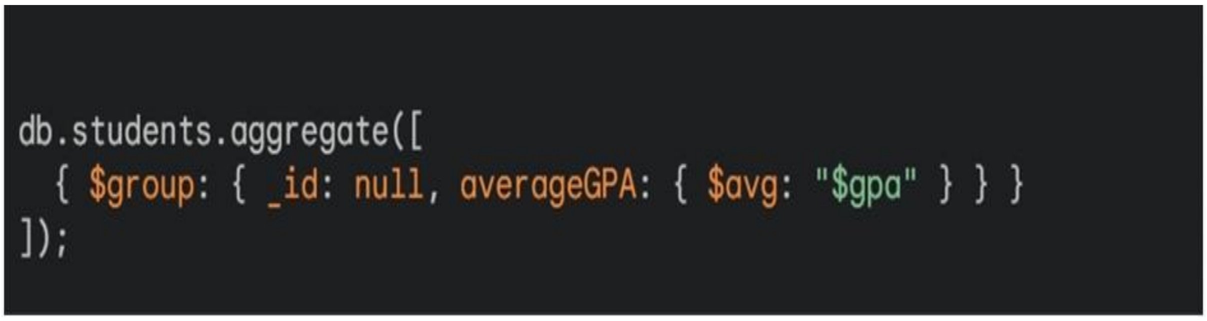
```
{ $last: "<expression>" }
```

Example:

db.collection.aggregate([

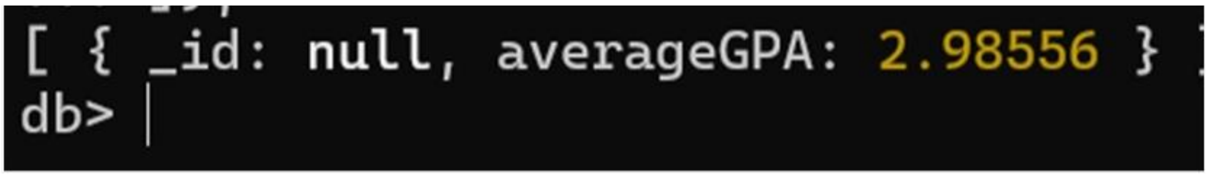
{ $group: { _id: "$category", lastPrice: { $last: "$price" } } }])

Average GPA of all Students:



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

Output:



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

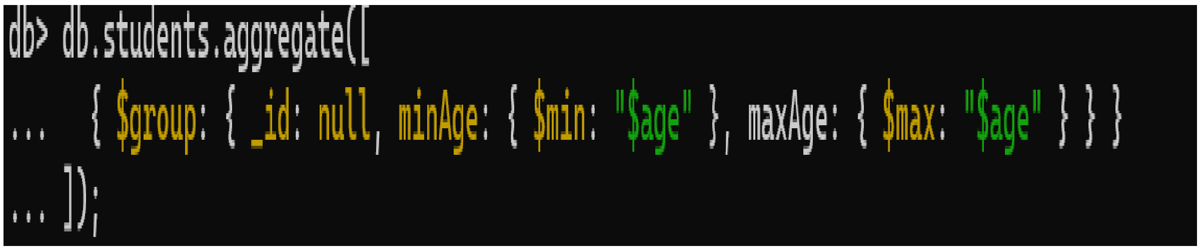
Explanation:

- `db.students.aggregate(...)`: This initiates an aggregation operation on the `students` collection.
- `[ ... ]`: The aggregation pipeline is defined within these square brackets. In this case, it consists of a single stage.
- `{ $group: { ... } }`: This is the `$group` stage in the aggregation pipeline. It groups documents by a specified identifier and can perform various operations, such as calculating averages, sums, etc.

- `_id: null`: This specifies that all documents should be grouped into a single group. Setting `_id` to `null` means that there is no grouping by a particular field, so all documents are treated as a single group.
- `averageGPA: { $avg: "$gpa" }`: This calculates the average of the `gpa` field across all documents in the collection. The result will be stored in the `averageGPA` field.

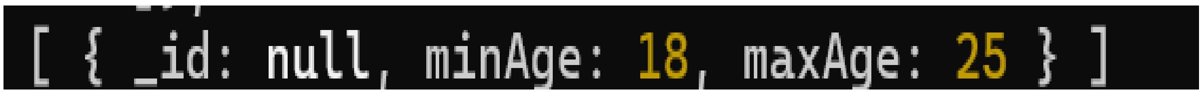
In summary, this code calculates the average GPA of all the documents in the `students` collection and returns it in a field named `averageGPA`.

Minimum and Maximum Age:



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

Output:



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

Explanation:

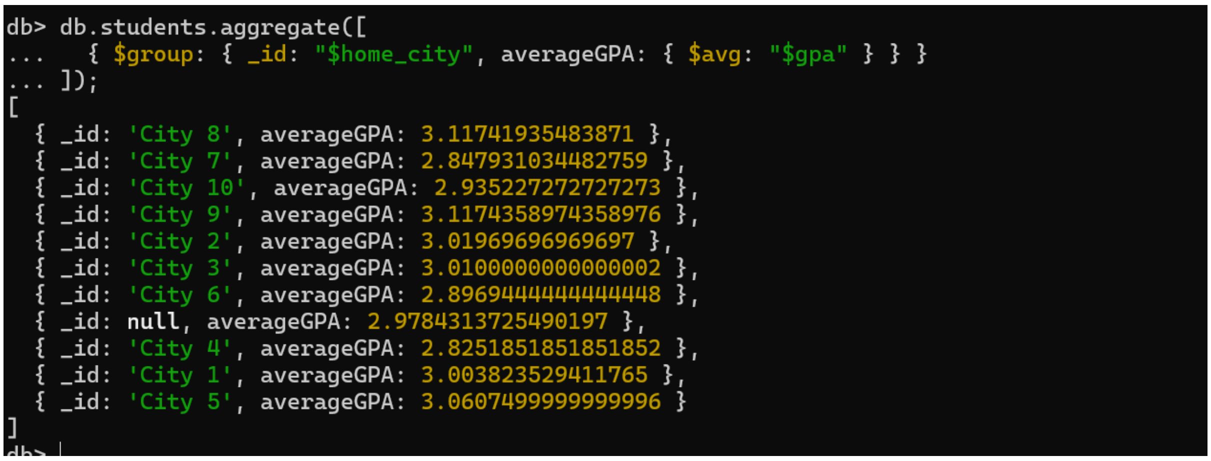
This MongoDB aggregation query calculates the minimum and maximum ages of students in the `students` collection. Here's a breakdown of the code:

- `db.students.aggregate(...)`: This initiates an aggregation operation on the `students` collection.
- `[ ... ]`: The aggregation pipeline is defined within these square brackets. In this case, it consists of a single stage.
- `{ $group: { ... } }`: This is the `$group` stage in the aggregation pipeline. It groups documents by a specified identifier and can perform various operations, such as calculating minimums, maximums, sums, etc.
    - `_id: null`: This specifies that all documents should be grouped into a single group. Setting `_id` to `null` means that there is no grouping by a particular field, so all documents are treated as a single group.
    - `minAge: { $min: "$age" }`: This calculates the minimum value of the `age` field across all documents in the collection. The result will be stored in the `minAge` field.
    - `maxAge: { $max: "$age" }`: This calculates the maximum value of the `age` field across all documents in the collection. The result will be stored in the `maxAge` field.

In summary, this code calculates the minimum and maximum ages of all the documents in the `students` collection and returns these values in the fields `minAge` and `maxAge`.

How to get Average GPA for all home cities:

Code and output:



```
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 }
]
```

Explanation:

This MongoDB aggregation query calculates the average GPA of students grouped by their home city in the `students` collection. Here's a breakdown of the code and its output:

- `db.students.aggregate(...)`: This initiates an aggregation operation on the `students` collection.
- `[ ... ]`: The aggregation pipeline is defined within these square brackets. In this case, it consists of a single stage.
- `{ $group: { ... } }`: This is the `$group` stage in the aggregation pipeline. It groups documents by a specified identifier and can perform various operations, such as calculating averages, sums, etc.

  - `_id: "$home_city"`: This specifies that documents should be grouped by the `home_city` field. Each unique value of `home_city` will form a group.

- `averageGPA: { $avg: "$gpa" }`: This calculates the average of the `gpa` field for each group (each unique `home_city`). The result will be stored in the `averageGPA` field.

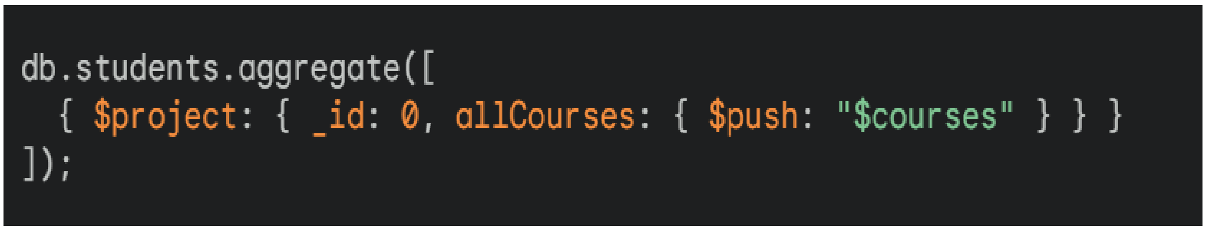
Output Explanation:

The output is a list of documents, each representing a group of students from the same city, along with the calculated average GPA for that group:

- Each document in the output represents a city (`_id` field) and the corresponding average GPA (`averageGPA` field) of students from that city.
- For example, for 'City 8', the average GPA is approximately 3.171.

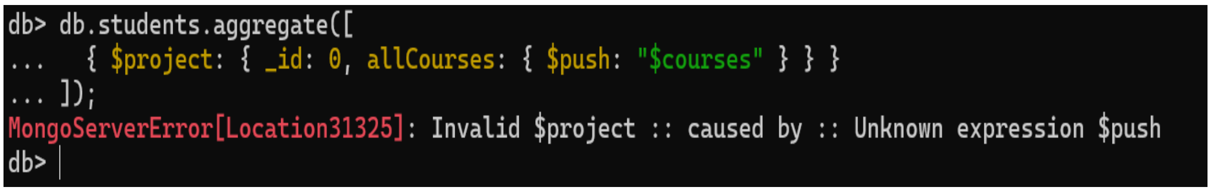
- The entry with `_id: null` represents documents where the `home_city` field is missing or `null`, with an average GPA calculated for such entries.

Pushing All Courses into a Single Array:



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

Output:



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

Explanation:

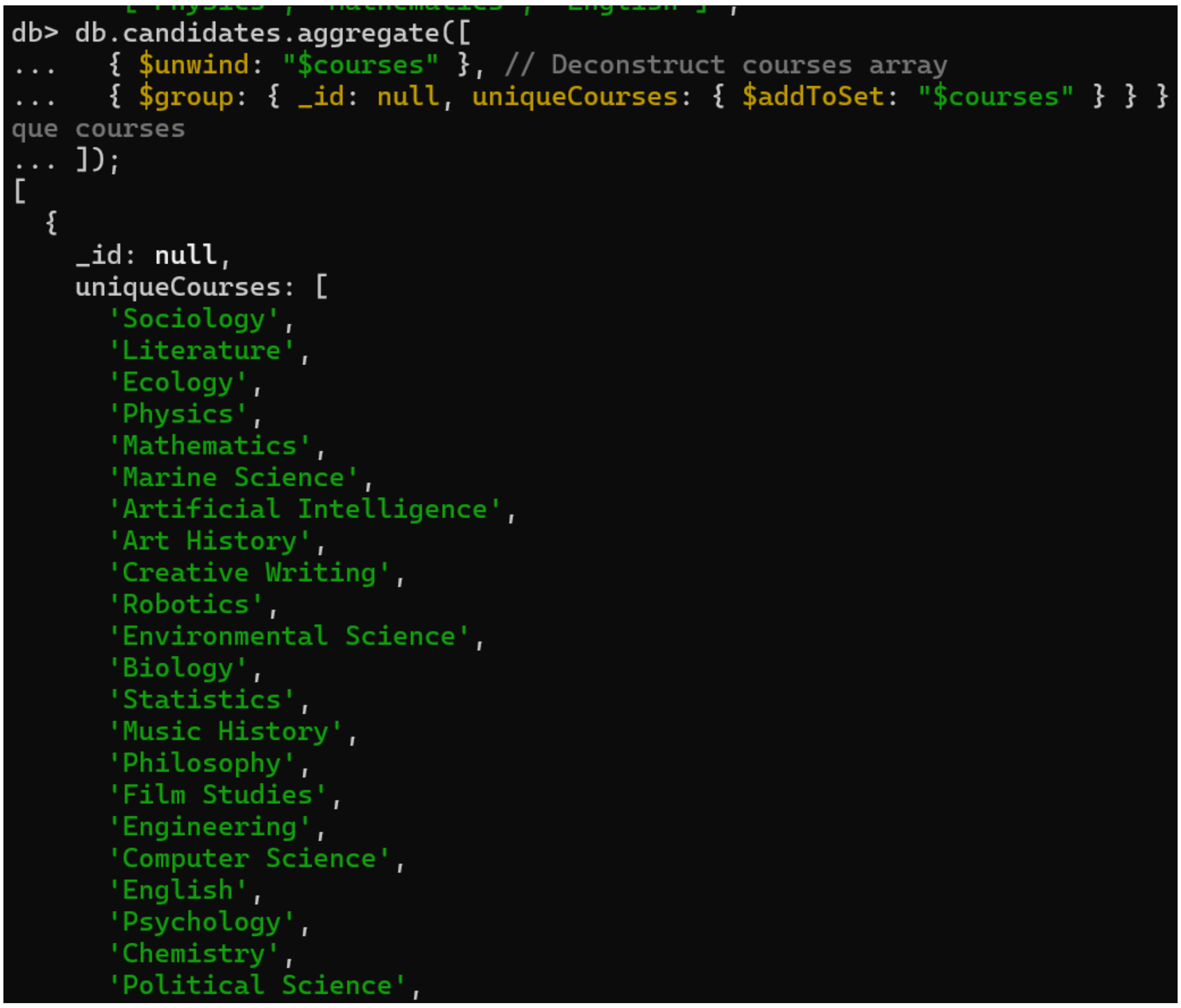
This MongoDB aggregation query projects all the courses from each document in the `students` collection into an array called `allCourses`. Here's a breakdown of the code:

- `db.students.aggregate(...)`: This initiates an aggregation operation on the `students` collection.
- `[ ... ]`: The aggregation pipeline is defined within these square brackets. In this case, it consists of a single stage.
- `{ $project: { ... } }`: This is the `$project` stage in the aggregation pipeline. It reshapes each document in the stream, usually by adding, removing, or renaming fields.
  - `_id: 0`: This excludes the `_id` field from the output documents. By setting it to `0`, the `_id` field will not be included in the resulting documents.
  - `allCourses: { $push: "$courses" }`: This creates a new field `allCourses` and populates it with the value of the `courses` field from each document. However, `$push` is typically used in the `$group` stage to append values to an array. In this context, it will simply place each `courses` field into the `allCourses` field for each document.

In summary, this code creates a new field `allCourses` in each document, containing the value of the `courses` field from the same document, and excludes the `_id` field from the output. If the intention is to combine all `courses` fields from all documents into a single array, a `$group` stage would be needed instead.

Collect Unique Courses Offered (Using $addToSet):

Code with 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',
```

Explanation:

The code you sent appears to be a MongoDB aggregation pipeline that is used to deconstruct an array of courses and create a new document with a set of unique courses. Let's break down the code step by step:

1. **`db.candidates.aggregate([ ... ])`**: This line initiates the aggregation pipeline. It starts by specifying the collection to operate on, which in this case is `candidates`. The `aggregate` method then takes an array of aggregation stages that will be applied sequentially to transform the data.
2. **First Stage: `{$group: { ... }}`**: This stage groups the documents in the `candidates` collection. It uses an empty group identifier (`id: null`) since we are not interested in grouping them based on a specific field. The important part here is the `uniqueCourses` field. It uses the `$addToSet` operator to create an array of unique courses from the `courses` field within each document.
3. **Second Stage: `{$unwind: "$courses"}`**: This stage deconstructs the `courses` array from each document in the results of the previous stage. After this stage, each document will have a single course instead of an array.
4. **Final Output**: The aggregation pipeline returns a new set of documents, each containing an `_id` field (set to null in this case) and a `uniqueCourses` field that contains a distinct set of courses extracted from the original `courses` array.

In essence, this aggregation pipeline helps flatten the `courses` array and remove duplicates to create a collection of documents with unique courses.