Class 6: Aggregation Operators
Experiment 5

Introduction

● Aggregation is a way of processing a large number of documents In a collection by means of passing them through different ways.

● For example min , max , avg etc

Syntax

db.collection.aggregate(<AGGREGATE OPERATION>

Average GPA of All Students

To find average gpa of all the students

Example:

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

Output:

```
db> db.student.aggregate([ { $group: { _id: null, averageGPA: { $avg: "$gpa" } } }] );
[ { _id: null, averageGPA: 3.7813626373626374 } ]
db>
```

Explanation:

- $group: Groups all documents together.
  - _id: null: Sets the group identifier to null .
  - averageGPA: Calculates the average value of the "gpa" using the $avg operator.

Minimum and Maximum Age:

Example:

```
db.student.aggregate([
  { $group: {_id:1 , minAge :{$min : “$age”} , maxAge :
  {$max: “$age”}}}]);
```

Output :

```
db> db.student.aggregate([{$group:{_id:1 ,minAge:{$min:"$age"},maxAge:{$max:"$age"}}}]);
[ { _id: 1, 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 calculate Average GPA for all home cities

Example:

```
db.student.aggregate([
   {$group:{_id: “$home_city” , averageGPA :{$avg : $gpa   } } } ] );
```

Output:

```
db> db.student.aggregate([{$group:{_id:"$home_city",averageGPA:{$avg:"$gpa"}}}]);
[
  { _id: 'City 1', averageGPA: 3.8203225806451617 },
  { _id: 'City 7', averageGPA: 3.6064 },
  { _id: 'City 5', averageGPA: 3.8850000000000002 },
  { _id: 'City 9', averageGPA: 3.9200000000000004 },
  { _id: 'City 3', averageGPA: 3.7868965517241375 },
  { _id: 'City 8', averageGPA: 3.8958620689655175 },
  { _id: 'City 2', averageGPA: 3.8329032258064517 },
  { _id: 'City 4', averageGPA: 3.5957692307692306 },
  { _id: 'City 6', averageGPA: 3.7025806451612904 },
  { _id: null, averageGPA: 3.7747857142857146 },
  { _id: 'City 10', averageGPA: 3.7352272727272724 }
```

Collect Unique Courses Offered (Using $addToSet):

Example:

```
db.candidates.aggregate([
{ $unwind: "$courses" },
{ $group: { _id: 1 , uniqueCourses: { $addToSet: "$courses" } }
}]);
```

Output:

```
db> db.candidates.aggregate([{$unwind: "$courses"},{$group:{_id:1 , uniqueCourses:{$addToSet:"$courses"
... }}}]);
[
  {
    _id: 1,
    uniqueCourses: [
      'English',
      'Robotics',
      'Sociology',
      'Psychology',
      'Political Science',
      'History',
      'Ecology',
      'Artificial Intelligence',
      'Cybersecurity',
      'Music History',
      'Literature',
      'Film Studies',
      'Creative Writing',
      'Computer Science',
      'Mathematics',
      'Philosophy',
      'Physics',
      'Art History',
      'Statistics',
      'Environmental Science',
      'Marine Science',
      'Engineering',
      'Chemistry',
      'Biology'
    ]
  }
]
```

Class 7: Aggregation pipeline

Experiment 6

Introduction

Aggregation Pipeline and its operators run with the

db.collection.aggregate() method do not modify

documents in a collection , the pipeline contains a $group, $sort, $project , $merge etc stages.

A. Finding students with age greater than 25 , sorted by age in descending order, and only return name and age.

```
db.students6.aggregate([
{$match:{age:{$gt:25}}},
{$sort:{age:-1}},
{$project :{_id:1 , name:1 , age:1}}])
```

Output:

```
db> db.students6.aggregate([ {$match:{age:{$gt:25}}}, {$sort:{age:-1}}, {$project:{_id:1 ,name:1 , age:1}}]);
[ { _id: 3, name: 'Charlie', age: 28 } ]
db>
```

B. Find students with age less than 20, sorted by name in ascending order, and only return name and score

```
db.students6.aggregate([
{$match:{age:{$lt:23}}},
{$sort:{age:1}},
{$project :{_id:0, name:1 , age:1}}])
```

Output:

```
db> db.students6.aggregate([ {$match:{age:{$lt:25}}}, {$sort:{age:1}}, {$project:{_id:0 ,name:1 , age:1}}]);
[
  { name: 'David', age: 20 },
  { name: 'Bob', age: 22 },
  { name: 'Eve', age: 23 }
]
db>
```

Grouping students by major, calculating average age and total number of students in each major:

```
db.students6.aggregate([
{$group:{_id: “$major” , averageAge :{$avg: “age”},
totalStudents:{$sum :1}}}
]);
```

Output:

```
db> db.students6.aggregate([ {$group:{_id:"$major" , averageAge:{$avg:"$age"},totalStudents:{$sum:1}}}])
[
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 },
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Biology', averageAge: 23, totalStudents: 1 },
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 }
]
db>
```

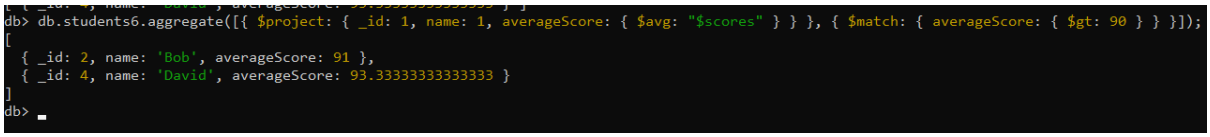
Finding students with an average above 90 .

db.students6.aggregate([

{$project:{_id:1 , name:1, averageScore :{$avg:”$scores}

}},{$match:{averageScore:{$gt:90}}}]);

Output:



Finding students with an average score below 80 and skip the first document .

db.students6.aggregate([

{$project:{_id:1 , name:1, averageScore :{$avg:”$scores}

}},{$match:{averageScore:{$lt:85}}},{skip:1}]);

Output:

```
db> db.students6.aggregate([ { $project: { _id: 0, name: 1, averageScore: { $avg: "$scores" } } }, { $match: { averageScore: { $lt: 85 } } }, { $skip:1 }]);
[ { name: 'Eve', averageScore: 83.33333333333333 } ]
db>
fwd-i-search: _
```