CLASS 5 : Projection Operators

Projection operators specify the fields returned by an operation.

Projection operators:

- $
- $elemMatch
- $slice

$

The positional **$** operator limits the contents of an <array>to return the first element that matches the query condition on the array.

$elemMatch

The **$elemMatch** projection operator takes an explicit condition argument. This allows you to project based on a condition not in the query, or if you need to project based on multiple fields in the array's embedded documents

$slice

The **$slice** projection operator specifies the number of elements in an array to return in the query result.

1. Retrieve Name, Age, and GPA

```
db> db.candidates.find({},{name:1,age:1,gpa:1});
[
  {
    _id: ObjectId('669923074a062f6e859a18cf'),
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('669923074a062f6e859a18d0'),
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8
  },
  {
    _id: ObjectId('669923074a062f6e859a18d1'),
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2
  },
  {
    _id: ObjectId('669923074a062f6e859a18d2'),
    name: 'Emily Jones',
    age: 21,
    gpa: 3.6
  },
  {
    _id: ObjectId('669923074a062f6e859a18d3'),
    name: 'David Williams',
    age: 23,
    gpa: 3
  },
  {
    _id: ObjectId('669923074a062f6e859a18d4'),
    name: 'Fatima Brown',
    age: 18,
    gpa: 3.5
  },
  {
    _id: ObjectId('669923074a062f6e859a18d5'),
    name: 'Gabriel Miller',
    age: 24,
    gpa: 3.9
  },
  {
    _id: ObjectId('669923074a062f6e859a18d6'),
    name: 'Hannah Garcia',
    age: 20,
    gpa: 3.3
  },
  {
    _id: ObjectId('669923074a062f6e859a18d7'),
    name: 'Isaac Clark',
    age: 22,
    gpa: 3.7
  },
  {
    _id: ObjectId('669923074a062f6e859a18d8'),
    name: 'Jessica Moore',
    age: 19,
    gpa: 3.1
  },
  {
    _id: ObjectId('669923074a062f6e859a18d9'),
    name: 'Kevin Lewis',
    age: 21,
    gpa: 4
  },
```

Here only name, age and gpa are displayed.

Variation: Exclude fields

```
]
db> db.candidates.find({},{_id:0,course:0});
[
  {
    name: 'Alice Smith',
    age: 20,
    courses: [ 'English', 'Biology', 'Chemistry' ],
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    name: 'Bob Johnson',
    age: 22,
    courses: [ 'Computer Science', 'Mathematics', 'Physics' ],
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Charlie Lee',
    age: 19,
    courses: [ 'History', 'English', 'Psychology' ],
    gpa: 3.2,
    home_city: 'Chicago',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Emily Jones',
    age: 21,
    courses: [ 'Mathematics', 'Physics', 'Statistics' ],
    gpa: 3.6,
    home_city: 'Houston',
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    name: 'David Williams',
    age: 23,
    courses: [ 'English', 'Literature', 'Philosophy' ],
    gpa: 3,
    home_city: 'Phoenix',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Fatima Brown',
    age: 18,
    courses: [ 'Biology', 'Chemistry', 'Environmental Science' ],
    gpa: 3.5,
    home_city: 'San Antonio',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    name: 'Gabriel Miller',
    age: 24,
    courses: [ 'Computer Science', 'Engineering', 'Robotics' ],
    gpa: 3.9,
    home_city: 'San Diego',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
```

Here _id, courses are exluded .

$slice

2. Retrieve all candidates with first two courses

```
db> db.candidates.find({},{name:1,courses:{$slice:2}});
[
  {
    _id: ObjectId('669923074a062f6e859a18cf'),
    name: 'Alice Smith',
    courses: [ 'English', 'Biology' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d0'),
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d1'),
    name: 'Charlie Lee',
    courses: [ 'History', 'English' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d2'),
    name: 'Emily Jones',
    courses: [ 'Mathematics', 'Physics' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d3'),
    name: 'David Williams',
    courses: [ 'English', 'Literature' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d4'),
    name: 'Fatima Brown',
    courses: [ 'Biology', 'Chemistry' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d5'),
    name: 'Gabriel Miller',
    courses: [ 'Computer Science', 'Engineering' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d6'),
    name: 'Hannah Garcia',
    courses: [ 'History', 'Political Science' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d7'),
    name: 'Isaac Clark',
    courses: [ 'English', 'Creative Writing' ]
  },
  {
    _id: ObjectId('669923074a062f6e859a18d8'),
    name: 'Jessica Moore',
    courses: [ 'Biology', 'Ecology' ]
  },
  {
```

Return only the name field and the first 2 elements of the courses array for each document.

CLASS 6 : Aggregation Operators

- Aggregation means grouping together
- For ex : sum, avg, min, 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" }

1. **Average GPA of all students**

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

_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.

2.Minimum and Maximum Age:

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

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.

This shows only min age and max age.

3. **Average GPA of all home cities:**

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

Group the documents in the students collection by the home_city field.

For each group, calculate the average GPA (gpa) and store it in a field called averageGPA.

The result will be a set of documents where each document represents a unique home_city and the corresponding average GPA of students from that city.

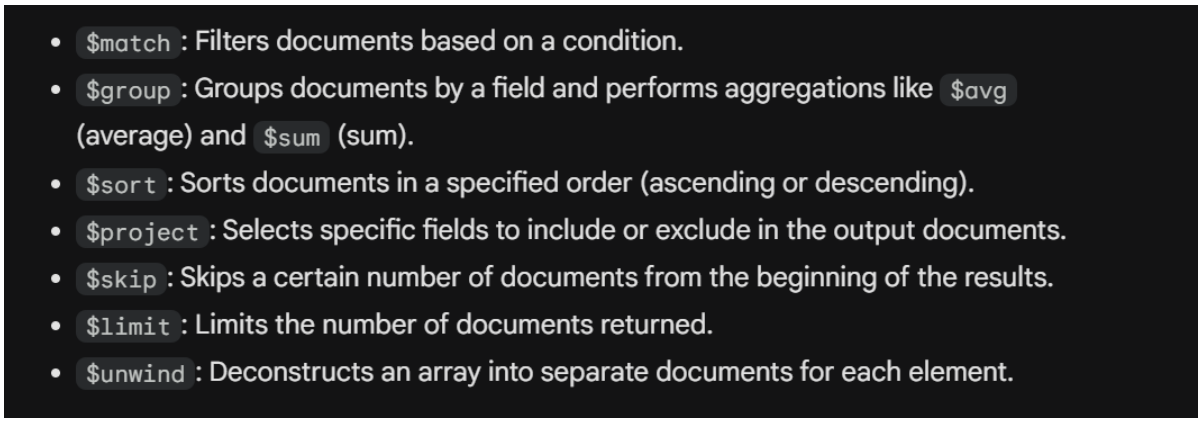
4. Pushing all courses into a single array:

```
db> db.students.aggregate([
... {$unwind:"$courses"},
... {$group:{_id:null,allCourses:{$push:"$courses"}}},
... {$project:{_id:0,allCourses:1}}
... ]);
[
  {
    allCourses: [
      "['English', 'Computer Science', 'Physics', 'Mathematics']",
      "['Physics', 'English']",
      "['Physics', 'Computer Science', 'Mathematics', 'History']",
      "['Mathematics', 'History', 'English']",
      "['English', 'Computer Science', 'Mathematics', 'History']",
      "['History', 'Physics', 'Computer Science', 'Mathematics']",
      "['Mathematics', 'History', 'Physics']",
      "['Mathematics', 'English']",
      "['History', 'Physics', 'Computer Science']",
      "['History', 'Physics', 'English', 'Mathematics']",
      "['Computer Science', 'Mathematics', 'History', 'English']",
      "['Mathematics', 'Computer Science', 'Physics']",
      "['Mathematics', 'Computer Science']",
      "['History', 'Physics', 'Computer Science']",
      "['English', 'History']",
      "['History', 'Physics', 'Mathematics']",
      "['English', 'History', 'Physics', 'Computer Science']",
      "['English', 'History', 'Physics', 'Mathematics']",
      "['English', 'Physics']",
      "['Computer Science', 'Physics', 'History', 'Mathematics']",
      "['Physics', 'Computer Science', 'English']",
      "['Computer Science', 'English', 'History']",
      "['Mathematics', 'English', 'Computer Science', 'Physics']",
      "['Computer Science', 'Physics', 'Mathematics', 'History']",
      "['Physics', 'Computer Science', 'English', 'Mathematics']",
      "['History', 'Computer Science']",
      "['Computer Science', 'History', 'Physics', 'English']",
      "['History', 'English', 'Computer Science', 'Mathematics']",
      "['English', 'Mathematics', 'Computer Science', 'Physics']",
      "['History', 'Computer Science', 'Mathematics', 'English']",
      "['History', 'Physics', 'Computer Science']",
      "['Mathematics', 'Computer Science']",
      "['Mathematics', 'Computer Science']",
      "['History', 'Computer Science', 'Physics', 'Mathematics']",
      "['Computer Science', 'English', 'Physics', 'History']",
      "['History', 'Mathematics', 'Physics', 'English']",
      "['Computer Science', 'Mathematics', 'English', 'Physics']",
      "['English', 'Mathematics', 'Computer Science']",
      "['English', 'History']",
      "['Mathematics', 'History', 'Physics']",
      "['English', 'Physics', 'Computer Science', 'History']",
      "['History', 'Computer Science']",
      "['English', 'Physics', 'Mathematics', 'Computer Science']",
      "['English', 'Physics']",
      "['English', 'Computer Science', 'Physics']",
      "['Mathematics', 'Physics', 'Computer Science', 'English']",
      "['Physics', 'Computer Science']",
      "['History', 'Mathematics', 'Physics']",
      "['Computer Science', 'English', 'History']",
      "['English', 'History', 'Computer Science']",
      "['Physics', 'History', 'Mathematics']",
      "['Computer Science', 'History']",
      "['English', 'History', 'Mathematics', 'Computer Science']",
      "['History', 'Computer Science']",
      "['Physics', 'English', 'Mathematics', 'History']",
      "['History', 'English', 'Physics', 'Computer Science']",
```

The final output is a single document with an allCourses array containing all the courses from all students.

CLASS 7 : Aggregation pipleline

Allow you to transform and analyze your data in a pipeline format. An aggregation pipeline consists of multiple stages that process documents, each stage performing an operation on the documents and passing the results to the next stage.



Benefits of Aggregation Pipeline:

- **Flexible and powerful**:

  Allows complex transformations and analyses.

- **Scalable**:

  Can handle large datasets efficiently.

- **Modular**:

  Each stage in the pipeline can be developed and tested independently.

1.To find students with age greater than 23, sorted by age in descending order and only return name and age.

```
db> db.students6.aggregate([
... {$match:{age:{$gt:23}}},
... {$sort:{age:-1}},
... {$project:{_id:0,name:1,age:1}}
... ]);
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

Here students older than 23 sorted by age in descending order and displays only name and age.

2.To find students with age greater than 23, sorted by age in ascending order and only return name and age.

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

Here students younger than 23 sorted by age in ascending order and displays only name and age.

3.Group students by major, calculate average age and total number of students in each major.

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

Here average age and total number of students in each major is calculated.

4.Find students with an average score (from scores array) above 85 and skip the first document.

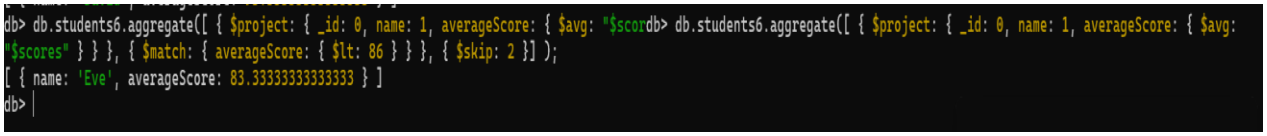
```
db> db.students6.aggregate([ { $project: { _id: 0, name: 1, averageScore: { $avg: "$scores" } } }, { $match: { averageScore: { $gt: 85 } } }, { $skip: 1 }] );
[ { name: 'David', averageScore: 93.33333333333333 } ]
db>
```

The $project stage calculates the average score for each student and includes only the name and averageScore fields.

The $match stage filters out students whose averageScore is not greater than 85.

The $skip stage skips the first document in the filtered result set, so only the subsequent documents are included in the final output.

5.Find students with an average score (from scores array) below 86 and skip the first 2 documents.



The $project stage calculates the average score for each student and includes only the name and averageScore fields.

The $match stage filters out students whose averageScore is not less than 86.

The $skip stage skips the first two documents in the filtered result set, resulting in no documents in the final output.