Class 5:-Projection operation

Projection

Projections in MongoDB are a powerful tool for controlling the data retrieved from queries. They allow you to specify exactly which fields you want to return from documents in a collection, offering several advantages:

• **Reduced Data Transfer**: By fetching only the necessary fields, you minimize the amount of data transferred between the client and server, leading to faster queries and improved network efficiency.

• **Enhanced Performance:** Less data to process translates to faster query execution times on both the client application and MongoDB server.

• **Focused Results:** Projections help you retrieve only the relevant data for your specific needs. This is especially beneficial when working with large documents containing numerous fields, many of which might not be essential for your current task.

• **Data Confidentiality:** Projections can be used to safeguard sensitive information. By excluding specific fields containing confidential data from the projection document, you can ensure they are not unintentionally returned in query results.

Introduction to projection operators

1.$:-

This operator projects the first element in the array type field that matches the query condition.

Syntax:- { <query>},{<array type field>.$:1}

2. $elemMatch:-

This operator projects the first element in the array type field that matches the condition given in the $elemMatch

Syntax:- { <array type field>:{$elemMatch:{<query>}}}

3.$slice:-

This operator limits the number of elements projected from an array. We can also use skip and limit in this operator.

Syntax:- {<array type field>:{$slice:<number of elements>}}

Example 1:-Retrieve Name ,Age ,and GPA

db.collections.find({},{ name: 1, age: 1, gpa: 1});

```
db> db.collections.find({},{name:1, age:1,gpa:1}).count()
12
db> db.collections.find({},{name:1, age:1,gpa:1})
[
  {
    _id: ObjectId('667c21d4fef4401250753918'),
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('667c21d4fef4401250753919'),
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8
  },
  {
    _id: ObjectId('667c21d4fef440125075391a'),
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2
  },
  {
    _id: ObjectId('667c21d4fef440125075391b'),
    name: 'Emily Jones',
    age: 21,
    gpa: 3.6
  },
  {
    _id: ObjectId('667c21d4fef440125075391c'),
    name: 'David Williams',
    age: 23,
    gpa: 3
  },
  {
    _id: ObjectId('667c21d4fef440125075391d'),
    name: 'Fatima Brown',
    age: 18,
    gpa: 3.5
  },
  {
    _id: ObjectId('667c21d4fef440125075391e'),
    name: 'Gabriel Miller',
    age: 24,
    gpa: 3.9
  },
  {
    _id: ObjectId('667c21d4fef440125075391f'),
    name: 'Hannah Garcia',
    age: 20,
    gpa: 3.3
  },
  {
    _id: ObjectId('667c21d4fef4401250753920'),
    name: 'Isaac Clark',
    age: 22,
    gpa: 3.7
  },
  {
    _id: ObjectId('667c21d4fef4401250753921'),
    name: 'Jessica Moore',
    age: 19,
    gpa: 3.1
  },
  {
    _id: ObjectId('667c21d4fef4401250753922'),
    name: 'Kevin Lewis',
    age: 21,
    gpa: 4
  },
  {
    _id: ObjectId('667c21d4fef4401250753923'),
    name: 'Lily Robinson',
```

- `db.candidates.find()` is the basic command to find documents in the `candidates` collection.
- `{}` is the query filter, and since it's empty, it means you want to find all documents.
- `{name: 1, age: 1, gpa: 1}` is the projection that specifies which fields to include in the result. A value of `1` means that the field will be included in the output.

Example 2:- Variation: Exclude Fields

db.collections.find({}, { _id: 0, courses: 0})

```
db> db.collections.find({},{_id:0,courses:0})
[
  {
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2,
    home_city: 'Chicago',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Emily Jones',
    age: 21,
    gpa: 3.6,
    home_city: 'Houston',
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    name: 'David Williams',
    age: 23,
    gpa: 3,
    home_city: 'Phoenix',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Fatima Brown',
    age: 18,
    gpa: 3.5,
    home_city: 'San Antonio',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    name: 'Gabriel Miller',
    age: 24,
    gpa: 3.9,
    home_city: 'San Diego',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    name: 'Hannah Garcia',
    age: 20,
```

Example 3:- Projection Operator($elemMatch)

Find Candidates enrolled in “Computer Science” with Specific Projection

```
db> db.collections.find({ courses:{$elemMatch:{$eq:"Computer Science"}}},{name: 1,"courses.$":1})
[
  {
    _id: ObjectId('667c21d4fef4401250753919'),
    name: 'Bob Johnson',
    courses: [ 'Computer Science' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391e'),
    name: 'Gabriel Miller',
    courses: [ 'Computer Science' ]
  },
  {
    _id: ObjectId('667c21d4fef4401250753922'),
    name: 'Kevin Lewis',
    courses: [ 'Computer Science' ]
  }
]
```

Example 4:- Projection Operator($slice)

Retrieve all candidates with first two courses

```
db> db.collections.find({},{name: 1,courses:{$slice:2}})
[
  {
    _id: ObjectId('667c21d4fef4401250753918'),
    name: 'Alice Smith',
    courses: [ 'English', 'Biology' ]
  },
  {
    _id: ObjectId('667c21d4fef4401250753919'),
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391a'),
    name: 'Charlie Lee',
    courses: [ 'History', 'English' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391b'),
    name: 'Emily Jones',
    courses: [ 'Mathematics', 'Physics' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391c'),
    name: 'David Williams',
    courses: [ 'English', 'Literature' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391d'),
    name: 'Fatima Brown',
    courses: [ 'Biology', 'Chemistry' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391e'),
    name: 'Gabriel Miller',
    courses: [ 'Computer Science', 'Engineering' ]
  },
  {
    _id: ObjectId('667c21d4fef440125075391f'),
    name: 'Hannah Garcia',
    courses: [ 'History', 'Political Science' ]
  },
  {
    _id: ObjectId('667c21d4fef4401250753920'),
    name: 'Isaac Clark',
    courses: [ 'English', 'Creative Writing' ]
  },
  {
    _id: ObjectId('667c21d4fef4401250753921'),
    name: 'Jessica Moore',
    courses: [ 'Biology', 'Ecology' ]
  },
  {
    _id: ObjectId('667c21d4fef4401250753922'),
    name: 'Kevin Lewis',
    courses: [ 'Computer Science', 'Artificial Intelligence' ]
  },
  {
    _id: ObjectId('667c21d4fef4401250753923'),
    name: 'Lily Robinson',
    courses: [ 'History', 'Art History' ]
  }
]
db>
```

Class 6:-Aggregation operation

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.

Syntax:- 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 aggregrate operation.

EXPRESSION:-

1.$sum:-Sums up the defined value from all the documents in a collection.

Syntax:- db.collection. aggregate ([{$group: {_id: $<Field Name>,<Field Label>: {$sum: <Field Name,Number or Operation>}}}])

2.$avg:- Calculates the average values from all the documents in a collection.

Syntax:- db.collection. aggregate ([{$group: {_id : $<Field Name>,<Field Label> ($Savg: <Field Name,Number or Operation }}}])

3.$min:- Gives the minimum of all the values of documents in a collection.

Syntax:- db.collection. aggregate([{$group: {_id: $<Field Name>,<Field Label> ($min: <Field Name, Number or Operation>}}}])

4.$max:- Gives the maximum of all the values of documents in a collection.

Syntax:- db.collection. aggregate ([{$group: {_id: $<Field Name>,<Field Lable> {$max <Field Name, Number or Operation> }}}])

5.$push:- Inserts values to an array of the resulting document

Syntax:- db.collection. aggregate([{$group: (id: $<Field Name>,<Field Label> : {$push:<Field Name >}}}])

6.$addToSet:- Inserts values to an array of the resulting document, but does not create duplicates in the resulting document.

Syntax:- db.collection. aggregate([{$group: {_id $<Field Name>,<Field Label> {$addToSet:<Field Name> }}}])

7.$first:- Gives the first document from the source document.

Syntax:- db.collection. aggregate([{$group: {_id : $<Field Name>,<Field Label> : {$first:<Field Name>}}}])

8.$list:- Gives the last document from the source document.

Syntax:- db.collection. aggregate([{$group: {_id : $<Field Name>,<Field Label> : ($last: <Field Name>}}}]

Average GPA of all Students

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

Answer:-

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

Explanation:

- $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 operat

Minimum and Maximum Age:

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

Answer:-

```
[ { _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

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: 2.8479310344827 59 },
  { _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 }
]
```

Pushing All Courses into a Single Array

```
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)

BUT

This is because our Array is incorrect

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

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',
```

Class 7:-Aggregation pipeline

The basic overview of aggregation pipeline:

1.$match: Filters the documents to pass only the documents that match the specified condition(s).

{ $match: { status: "A" } }

2.$group: Groups input documents by the specified identifier expression and applies the accumulator expressions.

```
{
 $group: {
_id: "$status",
 total: { $sum: "$amount" }
 }
 }
```

3.$project: Passes along the documents with the requested fields.

```
{
 $project: {
 name: 1,
 status: 1
}
 }
```

4.$sort: Sorts all input documents and returns them to the pipeline in sorted order.

{ $sort: { age: -1 } }

5.$limit: Limits the number of documents passed to the next stage.

{ $limit: 5 }

6.$skip: Skips the first n documents where n is the specified skip number and passes the remaining documents to the next stage.

{ $skip: 10 }

7.$unwind: Deconstructs an array field from the input documents to output a document for each element.

{ $unwind: "$sizes" }

BUILD A NEW DATASET:

```
_id: 4
name : "David"
age : 20
major : "Computer Science"
▾ scores : Array (3)
    0: 98
    1: 95
    2: 87
```

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

```
db.students6.aggregate([
  { $match: { age: { $gt: 23 } } }, // Filter students older than 23
  { $sort: { age: -1 } }, // Sort by age descending
  { $project: { _id: 0, name: 1, age: 1 } } // Project only name and
])
```

The pipeline starts by finding all students in the students6 collection who are older than 23 years old.Then, it sorts these students in descending order of their age, so the oldest students appear first.Finally, the output only shows the student's name and age, discarding the automatically generated _id field.

Output:-

```
db> db.students6.aggregate([
...   { $match: { age: { $gt: 23 } } }, // Filter students older than 23
...   { $sort: { age: -1 } }, // Sort by age descending
...   { $project: { _id: 0, name: 1, age: 1 } } // Project only name and age
... ])
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

2. 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 } } }
... ])
```

This pipeline starts by grouping all students in the students6 collection based on their declared major. Within each major group, it calculates two statistics:

- The average age of students in that major (averageAge).
- The total number of students in that major (totalStudents).

Output:-

```
[
  { _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 }
]
```

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

```
db.students6.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      averageScore: { $avg: "$scores" }
    }
  },
  { $match: { averageScore: { $gt: 85 } } },
  { $skip: 1 } // Skip the first document
])
```

The pipeline starts by iterating through each student document in the students6 collection. For each student, it calculates the average score from the "scores" array (assuming it exists and contains numerical values).It then creates a new document for each student that includes only the name and the calculated averageScore. The _id field is discarded.Finally, it filters the documents created in step 3, keeping only those students whose averageScore is greater than 85.

Output:-

```
[ { name: 'David', averageScore: 93.33333333333333 } ]
```