MONGO DB

EXPERIMENT-02

2. a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.

b. Develop a MongoDB query to display the first 5 documents from the results obtained in a.[use of limit and find]

MongoDB Query to Select Specific Fields and Exclude Others

Documents in our MongoDB collection, each brimming with information. But sometimes, we only need a specific set of jewels, not the entire . This is where MongoDB's projection operator comes to the rescue, allowing you to curate the data you retrieve and discard the rest, optimizing performance and bandwidth.

Art of Projection:

MongoDB's find operation, the workhorse for data retrieval, can be enhanced with a projection document. This document acts as a map, guiding MongoDB on which fields to include and exclude from the returned results.

There are two primary ways to wield the projection document:

1.Including Specific Fields:

Here, we explicitly list the fields we desire. Set the corresponding field names in the projection document to 1.

For example:

db.collectionName.find({}, {fieldName1: 1, fieldName2: 1})

This query retrieves all documents (empty query {}) and includes only fieldName1 and fieldName2 in the results.

2.Excluding Unwanted Fields:

Here we set the fields you want to exclude to 0.

Syntax:

db.collectionName.find({}, {fieldName1: 0, fieldName2: 0})

This query retrieves all documents but omits fieldName1 and fieldName2 from the response.

Why Projection Matters:

- **Reduced Network Traffic:**
  By fetching only the necessary fields, we minimize the amount of data transferred between our application and the MongoDB server. This is especially crucial for large collections or applications with limited bandwidth.
- **Enhanced Performance:**
  Since MongoDB doesn't need to process and transmit unwanted fields, query execution becomes faster.
- **Data Security:**
  Sensitive information can be excluded from the results, adding an extra layer of security.

MongoDB's projection operator empowers you to curate our data retrieval, optimizing performance, clarity, and security. By selectively choosing which fields to include or exclude, we can navigate your document treasure trove with precision, unearthing only the information you seek!

The Theory Behind the Magic:

MongoDB employs a document-oriented data model, where documents are self-contained units containing key-value pairs (fields and their values). By selectively choosing which fields to return, you dictate the structure of the returned documents, tailoring them to your specific needs. This reduces the amount of data transferred between the database server and your application, leading to significant performance gains, especially when dealing with large collections.

A Unique Twist: Performance Optimization

The beauty of field selection lies not just in data reduction but also in optimizing queries. By excluding unnecessary fields from the projection, MongoDB can perform more efficient scans and filtering operations on the remaining data. This translates to faster queries and improved overall application responsiveness.

- Including the _id field is optional but often recommended for easier document identification.
- Excluding fields you don't need is generally good practice, but ensure you have the necessary information for further processing.

Example 01:

1.Find Method:

The find method is used to retrieve documents from a collection. It takes two arguments:

- Query Document (Optional):Filters the documents based on certain criteria.
- Projection Document: Specifies which fields to include or exclude from the results.

Find all students, but only return name

```
db> db.student.find({},{name:1});
[
  { _id: ObjectId('665899c18fb275953df12837'), name: 'Alice Smith' },
  { _id: ObjectId('665899c18fb275953df12838'), name: 'Bob Johnson' },
  { _id: ObjectId('665899c18fb275953df12839'), name: 'Charlie Lee' },
  { _id: ObjectId('665899c18fb275953df1283a'), name: 'Emily Jones' },
  { _id: ObjectId('665899c18fb275953df1283b'), name: 'David Williams' },
  { _id: ObjectId('665899c18fb275953df1283c'), name: 'Fatima Brown' },
  { _id: ObjectId('665899c18fb275953df1283d'), name: 'Gabriel Miller' },
  { _id: ObjectId('665899c18fb275953df1283e'), name: 'Hannah Garcia' },
  { _id: ObjectId('665899c18fb275953df1283f'), name: 'Isaac Clark' },
  { _id: ObjectId('665899c18fb275953df12840'), name: 'Jessica Moore' },
  { _id: ObjectId('665899c18fb275953df12841'), name: 'Kevin Lewis' },
  { _id: ObjectId('665899c18fb275953df12842'), name: 'Lily Robinson' }
]
```

Example 02:

- The first argument to find is an empty document (`{}`), meaning all documents will be returned.
- The second argument is the projection document:
- name: 1, age: 1, and blood group: A+` indicate that these fields should be included.
- _id: 0, description: 0 specify that these fields should be excluded.
  - To include a field: Set its value to `1`.
  - To exclude a field: Set its value to `0`.

```
db> db.student.find({},{name:1,age:1,blood_group:"A+",_id:0});
[
  { name: 'Alice Smith', age: 20, blood_group: 'A+' },
  { name: 'Bob Johnson', age: 22, blood_group: 'A+' },
  { name: 'Charlie Lee', age: 19, blood_group: 'A+' },
  { name: 'Emily Jones', age: 21, blood_group: 'A+' },
  { name: 'David Williams', age: 23, blood_group: 'A+' },
  { name: 'Fatima Brown', age: 18, blood_group: 'A+' },
  { name: 'Gabriel Miller', age: 24, blood_group: 'A+' },
  { name: 'Hannah Garcia', age: 20, blood_group: 'A+' },
  { name: 'Isaac Clark', age: 22, blood_group: 'A+' },
  { name: 'Jessica Moore', age: 19, blood_group: 'A+' },
  { name: 'Kevin Lewis', age: 21, blood_group: 'A+' },
  { name: 'Lily Robinson', age: 23, blood_group: 'A+' }
]
```

Example 03:

To include a field: Set its value to `1`.

```
db> db.student.find({},{name:1,age:1});
[
  {
    _id: ObjectId('665899c18fb275953df12837'),
    name: 'Alice Smith',
    age: 20
  },
  {
    _id: ObjectId('665899c18fb275953df12838'),
    name: 'Bob Johnson',
    age: 22
  },
  {
    _id: ObjectId('665899c18fb275953df12839'),
    name: 'Charlie Lee',
    age: 19
  },
  {
    _id: ObjectId('665899c18fb275953df1283a'),
    name: 'Emily Jones',
    age: 21
  },
  {
    _id: ObjectId('665899c18fb275953df1283b'),
    name: 'David Williams',
    age: 23
  },
  {
    _id: ObjectId('665899c18fb275953df1283c'),
    name: 'Fatima Brown',
    age: 18
  },
  {
    _id: ObjectId('665899c18fb275953df1283d'),
    name: 'Gabriel Miller',
    age: 24
```

MongoDB query to display the first 5 documents from the results

Syntax:

db.<collection_name>.find({<query>}).limit(5)

db.<collection_name>:

Replace `<collection_name>` with the actual name of your MongoDB collection.

find({<query>}):

This clause defines the filtering criteria for selecting documents from your previous query. It's optional if you want to retrieve the first 5 documents from all documents in the collection.

.limit(5):

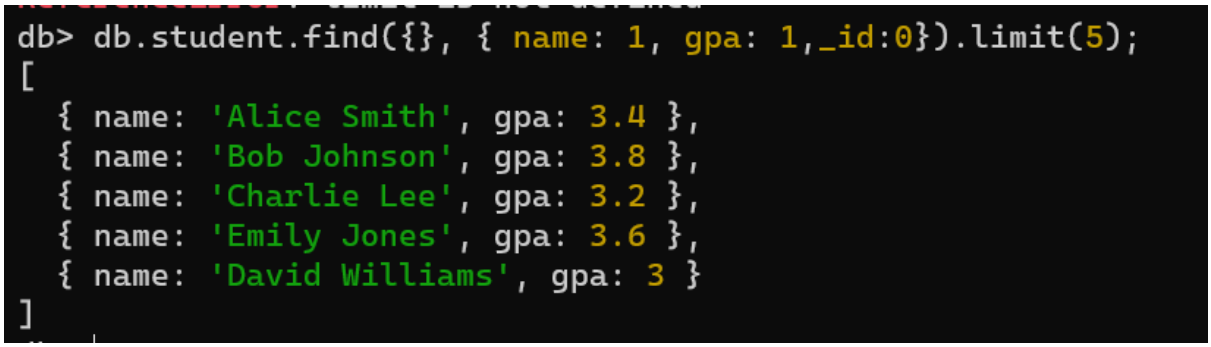
This operator limits the number of documents returned by the query to 5. It's applied after the filtering stage (`find({<query>})`).

Example 01:

Assuming you have a collection named `student` and you want to retrieve the first 5 student include name and gpa.

Syntax:

db.collection name.find({},{field1:value}).limit(5);



Example 02:

Return the data who all have gpa greater than or equal to 3.5 with limit 5.

```
db> db.student.find({ gpa:{$gte: 3.5}}, { name: 1,gpa:1,_id:0}).limit(6);
[
  { name: 'Bob Johnson', gpa: 3.8 },
  { name: 'Emily Jones', gpa: 3.6 },
  { name: 'Fatima Brown', gpa: 3.5 },
  { name: 'Gabriel Miller', gpa: 3.9 },
  { name: 'Isaac Clark', gpa: 3.7 },
  { name: 'Kevin Lewis', gpa: 4 }
]
db>
```

Example 03:

Filter to only the first 8 documents.

```
db> db.student.find({}, { name: 1, gpa: 1, courses: { $slice: 2 } ,_id:0}).limit(8);
[
  { name: 'Alice Smith', courses: [ 'English', 'Biology' ], gpa: 3.4 },
  {
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics' ],
    gpa: 3.8
  },
  { name: 'Charlie Lee', courses: [ 'History', 'English' ], gpa: 3.2 },
  {
    name: 'Emily Jones',
    courses: [ 'Mathematics', 'Physics' ],
    gpa: 3.6
  },
  {
    name: 'David Williams',
    courses: [ 'English', 'Literature' ],
    gpa: 3
  },
  {
    name: 'Fatima Brown',
    courses: [ 'Biology', 'Chemistry' ],
    gpa: 3.5
  },
  {
    name: 'Gabriel Miller',
    courses: [ 'Computer Science', 'Engineering' ],
    gpa: 3.9
  },
  {
    name: 'Hannah Garcia',
    courses: [ 'History', 'Political Science' ],
    gpa: 3.3
  }
db>
```

1.Filtering:

The **find** method with a query document allows you to specify conditions for selecting documents from the collection. The document passed to **`find` defines field-value comparisons or logical expressions for filtering.**

2.Chaining Operations:

MongoDB operations like `find` and `limit` can be chained together. The `limit` operation is applied after the filtering logic in `find`.

3.Result Set Limitation:

The `limit` operator restricts the number of documents returned by the query. It's a performance optimization technique, especially when dealing with large collections, as it reduces the amount of data transferred between the database and your application

In essence, this query first retrieves the documents matching the filtering criteria and then limits the output to only the first 5 matching documents.