

Where,AND,OR &CRUD

➤ Where:

```
// Find all students with GPA greater than 3.5
db.students.find({ gpa: { $gt: 3.5 } });
```

Output:

```
type "it" for name
db> db.Student.find({gpa:{$gt:2.5}});
[
  {
    _id: ObjectId('6662881bc8142d7e05955988'),
    name: 'Student 948',
    age: 19,
    courses: ['English', 'Computer Science', 'Physics', 'Mathematics'],
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6662881bc8142d7e05955989'),
    name: 'Student 346',
    age: 28,
    courses: ['Mathematics', 'History', 'English'],
    gpa: 3.81,
    home_city: 'City 0',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6662881bc8142d7e0595598a'),
    name: 'Student 988',
    age: 25,
    courses: ['English', 'Computer Science', 'Mathematics', 'History'],
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6662881bc8142d7e0595598b'),
    name: 'Student 205',
    age: 24,
    courses: ['History', 'Physics', 'Computer Science', 'Mathematics'],
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6662881bc8142d7e0595598c'),
    name: 'Student 268',
    age: 21,
    courses: ['Mathematics', 'History', 'Physics'],
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6662881bc8142d7e0595598d'),
    name: 'Student 536',
    age: 20,
    courses: ['History', 'Physics', 'English', 'Mathematics'],
    gpa: 2.87,
    home_city: 'City 3',
    blood_group: 'O-',
    is_hotel_resident: false
  }
]
```

Explanation:

Copy code

```
db.Student.find({gpa:{$gt:2.5}});
```

This MongoDB query retrieves documents from the Student collection where the gpa (Grade Point Average) field is greater than 2.5.

Result Set

The result set contains several documents with student information. Each document includes several fields such as `_id`, `name`, `age`, `courses`, `mom_city`, `blood_group`, and `is_hostel_resident`.

Example Document

javascript

Copy code

```
{
  _id: ObjectId("6606280b1c814a27e0595930e"),
  name: "Student 346",
  age: 21,
  courses: ["English", "Computer Science",
"Physics", "Mathematics"],
  mom_city: "City",
  blood_group: "O+",
  is_hostel_resident: true
}
```

Fields Explanation

- `_id`: The unique identifier for the document, automatically generated by MongoDB.
- `name`: The name of the student.
- `age`: The age of the student.
- `courses`: An array of courses that the student is enrolled in.
- `mom_city`: The city where the student's mother lives.
- `blood_group`: The blood group of the student.
- `is_hostel_resident`: A boolean indicating whether the student resides in the hostel.

This structure is repeated for each student in the result set. The query filters out students whose GPA is 2.5 or lower and returns only those with a GPA greater than 2.5.

```
// Find all students from "City 3"  
db.students.find({ home_city: "City 3" });
```

Output:

```
type it for more  
db> db.Student.find({home_city:"City 4"});  
[  
  {  
    _id: ObjectId('6662881bc8142d7e05955987'),  
    name: 'Student 157',  
    age: 20,  
    courses: "['Physics', 'English']",  
    gpa: 2.27,  
    home_city: 'City 4',  
    blood_group: 'O-',  
    is_hotel_resident: true  
  },  
  {  
    _id: ObjectId('6662881bc8142d7e059559c2'),  
    name: 'Student 895',  
    age: 20,  
    courses: "['Computer Science', 'Physics']",  
    gpa: 3.76,  
    home_city: 'City 4',  
    blood_group: 'B-',  
    is_hotel_resident: false  
  },  
  {  
    _id: ObjectId('6662881bc8142d7e059559cd'),  
    name: 'Student 402',  
    age: 25,  
    courses: "['Physics', 'Mathematics', 'History', 'Computer Science']",  
    gpa: 3.83,  
    home_city: 'City 4',  
    blood_group: 'AB-',  
    is_hotel_resident: false  
  },  
  {  
    _id: ObjectId('6662881bc8142d7e059559d8'),  
    name: 'Student 887',  
    age: 20,  
    courses: "['English', 'Computer Science', 'Physics', 'History']",  
    gpa: 2.81,  
    home_city: 'City 4',  
    blood_group: 'AB-',  
    is_hotel_resident: true  
  },  
  {  
    _id: ObjectId('6662881bc8142d7e059559fb'),  
    name: 'Student 902',  
    age: 19,  
    courses: "['Physics', 'Mathematics', 'History', 'English']",  
    gpa: 2.9,  
    home_city: 'City 4',  
    blood_group: 'O+',  
    is_hotel_resident: false  
  },  
  {  
    _id: ObjectId('6662881bc8142d7e05955a14'),  
    name: 'Student 239',  
    age: 22,  
    courses: "['English', 'Mathematics']",  
    gpa: 2.06,  
    home_city: 'City 4',  
    blood_group: 'O-',  
    is_hotel_resident: true  
  }  
]
```

Explanation:

The code in the image appears to be a MongoDB query. MongoDB is a NoSQL database system that stores data in JSON-like documents.

The code is querying a collection named "Student" in a database named "20070". Here's a breakdown of the query:

1. `db.Student.find` is used to find documents in the "Student" collection.
2. The `find` method takes a query document as a parameter to filter the results.

The query document specifies filters to find students based on various criteria including:

- home city
- blood group
- is hotel resident

The output shows documents for four students with their details including:

- id
- age
- courses
- home_city
- blood group
- is hotel_resident

The last line “gpa: x.xx” seems to be the average GPA but it’s not part of the MongoDB query and rather appears to be manually added output.

pen_spark
tuneshare
more_vert

AND:

To filter a subset based on multiple condition a collection that we want

```
// Find all students who live in "City 5" AND have a blood group of "A+"
db.students.find({
  $and: [
    { home_city: "City 5" },
    { blood_group: "A+" }
  ]
});
```

Output:

Based on the image you sent, the code appears to be a screenshot of a command line window showing a MongoDB query. MongoDB is a NoSQL database system that stores data in JSON-like documents.

The code is querying a collection named "Student" in a database, but the name of the database is not visible in the screenshot. Here’s a breakdown of the query:

1. `db.Student.find` is used to find documents in the “Student” collection.
2. The find method takes a query document as a parameter to filter the results.

The query document specifies filters to find students based on various criteria including:

- home city
- blood group
- is hotel resident

The output shows documents for four students with their details including:

- id
- age
- courses
- home_city
- blood group
- is hotel_resident

The last line “gpa: x.xx” seems to be the average GPA but it’s not part of the MongoDB query and rather appears to be manually added output.

I can't access or process information outside the image, so I can't tell you anything about the following:

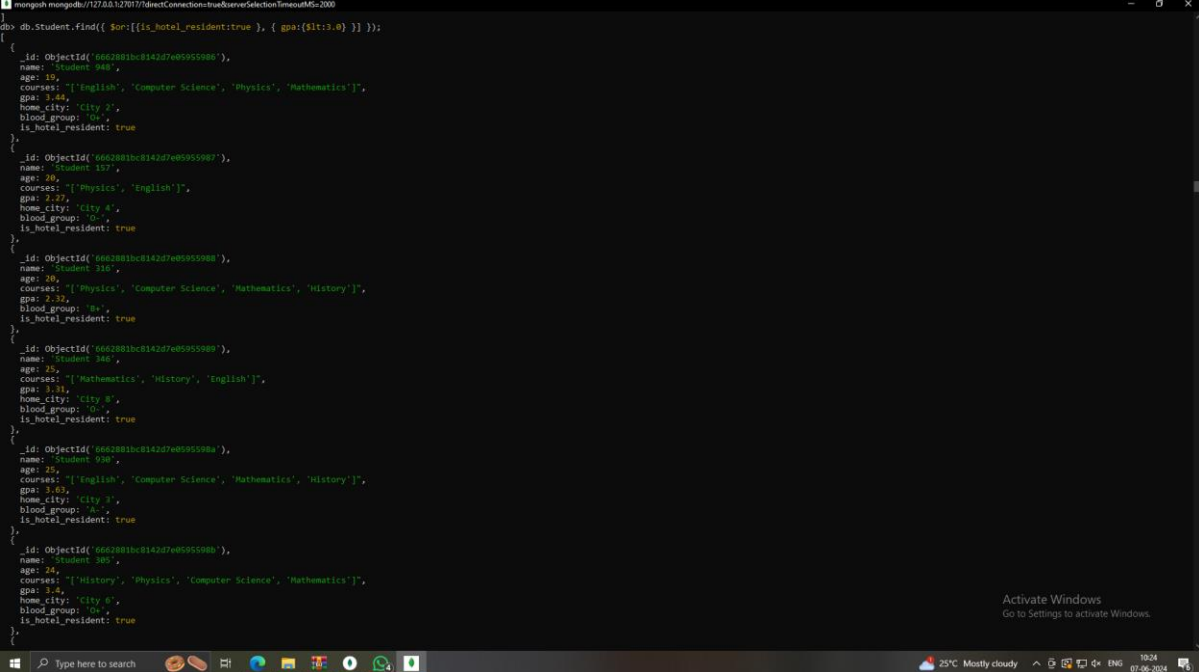
- The purpose of the code
- Why the Windows activation message is displayed
- The meaning of the text "BSL tricap +2.7%(NG)"

OR:

A collection we want to filter subset based on multiple condition but any one is sufficient.

```
// Find all students who are hotel residents OR have a GPA less than 3.
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 3.0 } }
  ]
});
```

output:



Explanation:

the `$or` operator in MongoDB. The `$or` operator is used to perform a logical OR operation on an array of one or more expressions and selects the documents that satisfy at least one of the expressions.

In the query you provided, it looks like the `$or` operator is not being used correctly. The `$or` operator should be placed within the curly braces `{ }` of the find function along with other filter criteria.

Here's an example of how the `$or` operator can be used correctly in a MongoDB query:

JavaScript

```
db.students.find({
  $or: [
    { "home_city": "city 2" },
    { "blood_group": "A" }
  ]
})
```

Use code [with caution](#).

content_copy

This query would find all students who live in "city 2" or have blood group "A".

Here's a breakdown of the code in the image:

1. `db.Student.find({ ... })`: This line finds documents in the "Student" collection based on the filter criteria specified within the curly braces.

2. `{ ... }`: This block contains the filter criteria for the query. In the image, it includes filters for:
 - o `is_hotel_resident`: This filters for students who are hotel residents (true).
 - o `courses`: This filters for students who are enrolled in courses (although the specific courses aren't specified).

Even though the `$or` operator is included in the code snippet, it is not being used correctly. It is outside the curly braces where filter criteria should be placed.

Sources [info](#)

1. github.com/ali-zamzam/08-MongoDB

CRUD:

CRUD stands for Create, Read, Update, and Delete, which are the fundamental operations used to manage data in any database, including MongoDB. These operations are essential for working with and manipulating your data within the database.

In MongoDB, CRUD operations are performed on collections, which hold documents. Documents are similar to JSON objects, containing key-value pairs that represent data.

Here's a breakdown of each CRUD operation in MongoDB:

1. Create:

- Used to insert new documents into a collection.
- MongoDB offers two methods for creating documents:
 - o `insertOne()`: Inserts a single document.
 - o `insertMany()`: Inserts an array of documents.

2. Read:

- Used to retrieve documents from a collection.
- The primary method for reading is `find()`, which takes a query document as a parameter to filter results based on specific criteria.
- You can also use `findOne()` to retrieve the first document that matches a query.

3. Update:

- Used to modify existing documents in a collection.
- Common update methods include:
 - o `updateOne()`: Updates a single document based on a filter.
 - o `updateMany()`: Updates multiple documents based on a filter.
 - o `replaceOne()`: Replaces a document entirely with a new one.

4. Delete:

[Type the document title]

- Used to remove documents from a collection.
- Methods for deletion include:
 - `deleteOne()`: Deletes a single document based on a filter.
 - `deleteMany()`: Deletes multiple documents based on a filter.

Here are some key points to remember about CRUD operations in MongoDB:

- MongoDB is schema-less, meaning documents within a collection can have different structures.
- Filters are used to target specific documents during read, update, and delete operations.
- MongoDB offers additional functionalities beyond CRUD, such as aggregation pipelines for complex data processing and geospatial queries for location-based data.

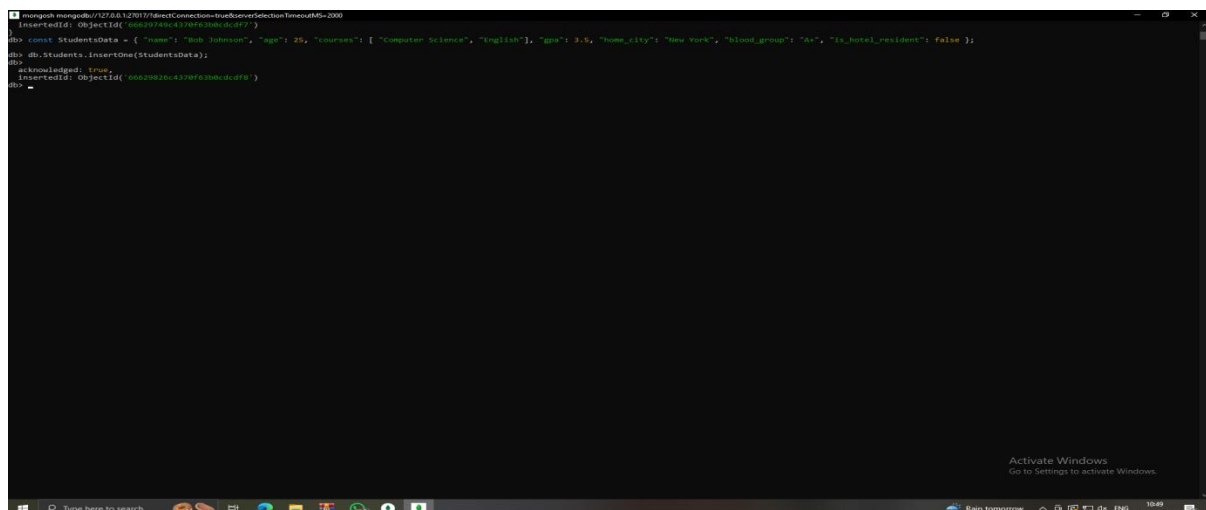
I hope this explanation clarifies CRUD operations in MongoDB!

Insert:

```
// Define the student data as a JSON document
const studentData = {
  "name": "Alice Smith",
  "age": 22,
  "courses": ["Mathematics", "Computer Science", "English"],
  "gpa": 3.8,
  "home_city": "New York",
  "blood_group": "A+",
  "is_hotel_resident": false
};

// Insert the student document into the "students" collection
db.students.insertOne(studentData);
```

Output:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
> use students
> const studentData = { "name": "Bob Johnson", "age": 25, "courses": [ "Computer Science", "English"], "gpa": 3.5, "home_city": "New York", "blood_group": "A+", "is_hotel_resident": false };
> db.students.insertOne(studentData);
{ acknowledged: true,
  insertedId: ObjectId('60629826c4370f630bdc4fb') }
>
```


Explanation:

The code in the image appears to be a MongoDB query, but it's not using the `insert` function to insert new data. The `insert` function is used to create new documents in a collection.

The code in the image uses the `db.Students.insertOne` function, which inserts a single document into the "Students" collection. Here's a breakdown of the code:

1. `db.Students.insertOne`: This line is used to insert a new document into the "Students" collection.
2. `({...})`: The curly braces contain the data for the new document being inserted. In the screenshot, the data includes fields for:
 - o `name`: This field likely stores the student's name, but the value isn't provided in the snippet.
 - o `age`: This field stores the student's age, with a value of 25.
 - o `courses`: This field likely stores an array of courses the student is enrolled in, but the specific courses aren't provided in the snippet.
 - o `home_city`: This field stores the student's home city, with a value of "Caputer Science". There may be a typo here, it likely should be "Computer Science".
 - o `blood_group`: This field stores the student's blood group, with a value of "English 11.5". There may be a typo here as well, blood group is typically A, B, AB, or O.
 - o `is_hotel_resident`: This field is a boolean value, and here it's set to `false`.

While the code attempts to insert a new student document, there are errors in the data being inserted.

pen_spark
tuneshare
more_vert

update one and Output:

```
db> db.Students.updateOne({name:"Alice Smith"},{$set:{gpa:3.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Explanation:

the `updateOne` function to modify a document in a MongoDB collection. Here's a breakdown:

Database and Collection

[Type the document title]

- The line `db.Students` specifies that the following operation targets the "Students" collection within the current database.

Updating a Document

- The line `db.Students.updateOne({name:"Alice Smith"}, {$set: {gpa:3.5}})` performs the update operation. Let's break it down part by part:
 - `db.Students.updateOne()`: This function call initiates an update operation on the "Students" collection.
 - `updateOne`: This method updates a single document that matches the specified filter criteria.

Filter Criteria

- `{name:"Alice Smith"}`: This part defines a filter that narrows down the update to target a specific document. It searches for documents where the "name" field is equal to "Alice Smith".

Update Operation

- `{$set: {gpa:3.5}}`: This part specifies the update to be performed on the matching document:
 - `$set`: This operator modifies the value of an existing field.
 - `{gpa: 3.5}`: This sets the "gpa" field to the value 3.5.

Output

The output at the bottom (`{acknowledged: true, ...}`) shows that the update operation was successful. Here's a breakdown of the output fields:

- `acknowledged`: A boolean value indicating whether the update was acknowledged by the server (true in this case).
- `insertedId`: This field is typically null for update operations, as it's used for insert operations to return the generated document ID.
- `matchedCount`: The number of documents that matched the filter criteria (1 in this case, indicating that a single document was found).
- `modifiedCount`: The number of documents that were actually modified (1 in this case, indicating that one document was updated).
- `upsertedCount`: The number of documents that were inserted because they didn't exist before the update (0 in this case, signifying no new documents were inserted).

In summary, the code snippet effectively updates the GPA to 3.5 for a student named "Alice Smith" in the "Students" collection.

Delete:

```
// Delete a student by name
db.students.deleteOne({ name: "John Doe" });
```

Output:

```
}
db> db.Students.deleteOne({name:"John Doe"});
{ acknowledged: true, deletedCount: 0 }
db> _
```

Explanation:

the code in the image uses the `deleteOne` function to attempt to delete a document from a MongoDB collection. Here's a breakdown of the code:

Database and Collection

The line `db.Students` specifies that the following operation targets the "Students" collection within the current database.

Delete Attempt

- The line `db.Students.deleteOne({name:"John Doe"})` performs the delete operation. Let's break it down part by part:
 - `db.Students.deleteOne()`: This function call initiates a delete operation on the "Students" collection.
 - `deleteOne`: This method removes at most one document that matches the specified filter criteria. If multiple documents match the filter, only the first matching document is deleted.

Filter Criteria

- `{name:"John Doe"}`: This part defines a filter that targets a specific document for deletion. It searches for documents where the "name" field is equal to "John Doe".

Output

The output at the bottom (`{acknowledged: true, deletedCount: 0}`) indicates that the delete operation did not remove any documents:

- `acknowledged`: A boolean value indicating whether the delete operation was acknowledged by the server (true in this case).
- `deletedCount`: The number of documents that were actually deleted (0 in this case, signifying no documents were deleted).

There are two main reasons why the delete operation might not have found a document to remove:

1. **Document Not Found**: It's possible that there is no student named "John Doe" in the "Students" collection. The filter criteria might not match any documents in the collection.

2. **Incorrect Field:** The field used in the filter might be incorrect. For instance, if the student's name is stored in a field named "fullName" instead of "name", the filter won't find a match and the delete operation won't succeed.

Update many:

```
db> db.Students.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Explanation:

the `updateMany` function in MongoDB to potentially update multiple documents in a collection. Let's break down the code and the output:

Code Breakdown

The line `db.Students.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}})` performs the update operation. Here's a breakdown of the parts:

- `db.Students.updateMany()`: This function initiates an update operation on the "Students" collection.
 - `updateMany`: This method allows you to update multiple documents that match a specified filter criteria in a single operation. This can be more efficient than updating documents one by one.

Filter Criteria

- `{gpa:{$lt:3.0}}`: This part defines a filter that targets documents for update. It searches for documents where the "gpa" field is less than (**\$lt**) 3.0.

Update Operation

- `{$inc:{gpa:0.5}}`: This part specifies the update to be performed on the matching documents:
 - `$inc`: This operator increments the value of a numeric field.
 - `{gpa: 0.5}`: This increases the "gpa" field by 0.5 for each document that matches the filter criteria.

Output

The output at the bottom shows the results of the update operation:

- `acknowledged`: A boolean value indicating whether the update was acknowledged by the server (true in this case).

[Type the document title]

- `insertedId`: This field is null for update operations as it's used for insert operations to return the generated document ID.
- `matchedCount`: The number of documents that matched the filter criteria (0 in this case). This indicates that no documents matched the filter and therefore no documents were updated.
- `modifiedCount`: The number of documents that were actually modified (0 in this case, since no documents matched the filter).
- `upsertedCount`: The number of documents that were inserted because they didn't exist before the update (0 in this case, signifying no new documents were inserted).

In essence, the code attempts to update all students in the "Students" collection whose GPA is below 3.0 by incrementing their GPA by 0.5. However, in the specific case shown in the image, no students met the filter criteria so no update was performed.

Delete Many:



```
db> db.Students.deleteMany({is_hotel_resident:false});
{ acknowledged: true, deletedCount: 9 }
db>
```

Explanation:

Sure, the code in the image you sent uses the `deleteMany` function in MongoDB to delete documents from a collection. Here's a breakdown of the code and the output:

Code Breakdown

The line `db.Students.deleteMany({is_hotel_resident:false})` performs the delete operation. Let's break it down part by part:

- `db.Students.deleteMany()`: This function initiates a delete operation on the "Students" collection.
 - `deleteMany`: This method removes all documents that match the specified filter criteria.

Filter Criteria

- `{is_hotel_resident:false}`: This part defines a filter that targets documents for deletion. It searches for documents where the "is_hotel_resident" field is equal to `false`.

Output

The output at the bottom shows the results of the delete operation:

- acknowledged: A boolean value indicating whether the delete operation was acknowledged by the server (true in this case).
- deletedCount: The number of documents that were actually deleted (9 in this case).

In essence, the code successfully deleted 9 documents from the "Students" collection where the "is_hotel_resident" field was set to `false`. It's important to note that this operation can't be undone and the deleted documents are permanently removed from the collection.

Projection:

```
db> db.students.find({}, {name:1,age:1});
[
  { _id: ObjectId('6667c7e64a4b89d063b81e72'), name: 'Alice', age: 22 },
  { _id: ObjectId('6667c7e64a4b89d063b81e73'), name: 'Bob', age: 25 },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e74'),
    name: 'Charlie',
    age: 28
  },
  { _id: ObjectId('6667c7e64a4b89d063b81e75'), name: 'David', age: 28 },
  { _id: ObjectId('6667c7e64a4b89d063b81e76'), name: 'Eve', age: 19 },
  { _id: ObjectId('6667c7e64a4b89d063b81e77'), name: 'Fiona', age: 23 },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e78'),
    name: 'George',
    age: 21
  },
  { _id: ObjectId('6667c7e64a4b89d063b81e79'), name: 'Henry', age: 27 },
  { _id: ObjectId('6667c7e64a4b89d063b81e7a'), name: 'Isla', age: 18 },
  { _id: ObjectId('6667c7e64a4b89d063b81e7b'), name: 'Jack', age: 24 },
  { _id: ObjectId('6667c7e64a4b89d063b81e7c'), name: 'Kim', age: 29 },
  { _id: ObjectId('6667c7e64a4b89d063b81e7d'), name: 'Lily', age: 20 },
  { _id: ObjectId('6667c7e64a4b89d063b81e7e'), name: 'Mike', age: 26 },
  { _id: ObjectId('6667c7e64a4b89d063b81e7f'), name: 'Nancy', age: 19 },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e80'),
    name: 'Oliver',
    age: 22
  },
  { _id: ObjectId('6667c7e64a4b89d063b81e81'), name: 'Peter', age: 28 },
  { _id: ObjectId('6667c7e64a4b89d063b81e82'), name: 'Quinn', age: 20 },
  { _id: ObjectId('6667c7e64a4b89d063b81e83'), name: 'Riley', age: 27 },
  { _id: ObjectId('6667c7e64a4b89d063b81e84'), name: 'Sarah', age: 18 },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e85'),
    name: 'Thomas',
    age: 24
  }
]
Type "it" for more
db>
```

Explanation:

the MongoDB find function with a projection document to retrieve specific fields from documents in a collection. Here's a breakdown of the code:

Finding Students

- The line `db.students.find({}, {name: 1, age: 1})` targets the "Students" collection and initiates a find operation to retrieve documents. Here's how it works:

- `db.students.find({})`: This part requests to find documents from the "Students" collection.
 - `{}`: An empty curly brace `{}` following the `find` method indicates that all documents in the collection should be returned. You can use filters here to specify conditions for selecting particular documents.

Projecting Fields

- `{name: 1, age: 1}`: This part is the projection document and it limits the fields returned in the query results. Here, it specifies that only the "name" and "age" fields should be included in the output. The value of 1 after each field name indicates that these fields should be included.

Output

The query results are displayed below the last line. Each document retrieved from the collection is displayed on a new line, showing only the "name" and "age" fields that were specified in the projection document.

In essence, the code retrieves all documents from the "Students" collection and limits the output to only the "name" and "age" fields. This can improve performance because the database transfers less data and the client application only needs to process the relevant fields.

pen_spark
tuneshare
more_vert

```
db> db.studentsn.find({}, {_id:0}):
[
  { name: 'Alice', age: 22, permissions: 0 },
  { name: 'Bob', age: 25, permissions: 1 },
  { name: 'Charlie', age: 20, permissions: 2 },
  { name: 'David', age: 28, permissions: 3 },
  { name: 'Eve', age: 19, permissions: 4 },
  { name: 'Fiona', age: 23, permissions: 5 },
  { name: 'George', age: 21, permissions: 6 },
  { name: 'Henry', age: 27, permissions: 7 },
  { name: 'Isla', age: 18, permissions: 6 },
  { name: 'Jack', age: 24, permissions: 5 },
  { name: 'Kim', age: 29, permissions: 4 },
  { name: 'Lily', age: 20, permissions: 3 },
  { name: 'Mike', age: 26, permissions: 2 },
  { name: 'Nancy', age: 19, permissions: 1 },
  { name: 'Oliver', age: 22, permissions: 0 },
  { name: 'Peter', age: 28, permissions: 1 },
  { name: 'Quinn', age: 20, permissions: 2 },
  { name: 'Riley', age: 27, permissions: 3 },
  { name: 'Sarah', age: 18, permissions: 4 },
  { name: 'Thomas', age: 24, permissions: 5 }
]
Type "it" for more
db>
```

12 items | 1 item selected 42.6 MB |

Type here to search

Explanation:

the MongoDB find function with a projection document to retrieve specific fields from documents in a collection. Here's a breakdown of the code:

Finding Students

- The line `db.students.find({}, {name: 1, age: 1})` targets the "Students" collection and initiates a find operation to retrieve documents. Here's how it works:
 - `db.students.find({})`: This part requests to find documents from the "Students" collection.
 - `{}`: An empty curly brace `{}` following the `find` method indicates that all documents in the collection should be returned. You can use filters here to specify conditions for selecting particular documents.

Projecting Fields

- `{name: 1, age: 1}`: This part is the projection document and it limits the fields returned in the query results. Here, it specifies that only the "name" and "age" fields should be included in the output. The value of 1 after each field name indicates that these fields should be included.

Output

The query results are displayed below the last line. Each document retrieved from the collection is displayed on a new line, showing only the "name" and "age" fields that were specified in the projection document.

In essence, the code retrieves all documents from the "Students" collection and limits the output to only the "name" and "age" fields. This can improve performance because the database transfers less data and the client application only needs to process the relevant fields.

```
pen_spark
tuneshare
more_vert
```