PROJECTION LIMIT AND SELECTORS

In my MongoDB, I have a student collection with 10 records having fields name and roll. One record of this collection is:

```
{
   "_id" : ObjectId("53d9feff55d6b4dd1171dd9e"),
   "name" : "Swati",
   "roll" : "80",
}
```

I want to retrieve the field roll only for all 10 records in the collection as we would do in traditional database by using:

```
SELECT roll FROM student
```

I went through many blogs but all are resulting in a query which must have WHERE clause in it, for example:

```
db.students.find({ "roll": { $gt: 70 })
```

The query is equivalent to:

SELECT * FROM student WHERE roll > In my MongoDB, I have a student collection with 10 records having fields name and roll. One record of this collection is:

```
{
   "_id" : ObjectId("53d9feff55d6b4dd1171dd9e"),
   "name" : "Swati",
   "roll" : "80",
}
```

I want to retrieve the field roll only for all 10 records in the collection as we would do in traditional database by using:

```
SELECT roll FROM student
```

I went through many blogs but all are resulting in a query which must have WHERE clause in it, for example:

```
db.students.find({ "roll": { $gt: 70 })
```

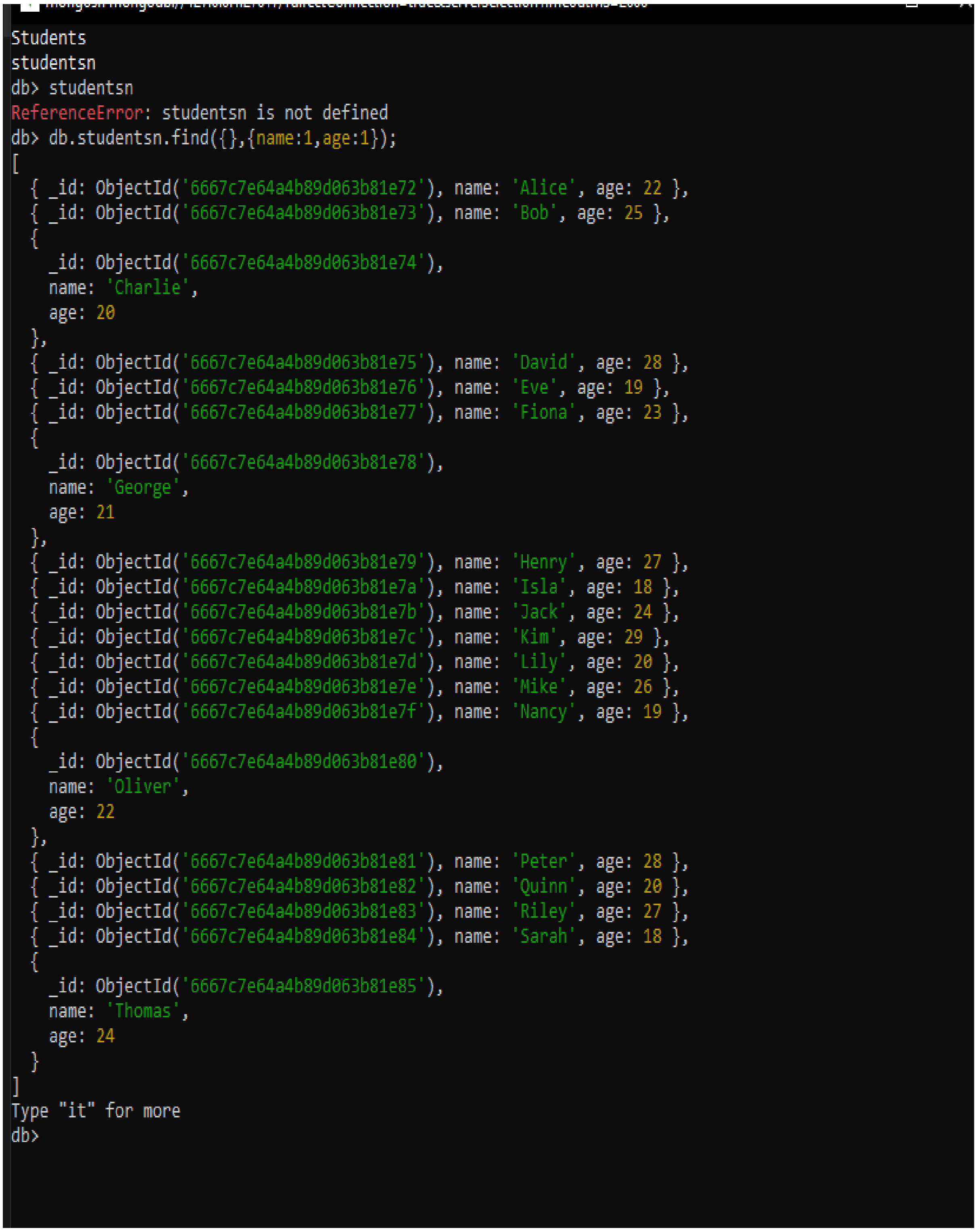
The query is equivalent to:

```
SELECT * FROM student WHERE roll > 70
```

Get selected attributes:

The **Attribute** Pattern provides for easier indexing the documents, targeting many similar fields per document. By moving this subset of data into a key-value subdocument, we can use non-deterministic field names, add additional qualifiers to the information, and more clearly state the relationship of the original field.

•



```
Students
studentsn
db> studentsn
ReferenceError: studentsn is not defined
db> db.studentsn.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: 20
  },
  { _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>
```

```
// Get only the name and age for all students
db.students.find({}, { name: 1, age: 1 });
```

Explaination:

Selecting the Database:

db = studentsn This line attempts to select a database named "studentsn". However, in MongoDB, databases are created implicitly when you first insert data into a collection. So, if a database named "studentsn" doesn't exist yet, it will be created automatically upon running this line.

Error Handling (Incorrect Database Selection):

ReferenceError: studentsn is not defined This error message indicates that the variable "studentsn" is not defined as a database. As mentioned earlier, MongoDB databases are created implicitly, so selecting a non-existent database using db = database_name will result in this error.

Correcting the Database Selection:

db = db.studentsn This line correctly selects the database using the db object. In MongoDB, the db object refers to the current database. Since there is no database named "studentsn" explicitly created, this line essentially sets the current database to the one being used by the collection "studentsn" (which might have been created implicitly earlier).

Finding Documents in the Collection:

db.studentsn.find((), (name: 1, age: 1)); This line is the actual query that retrieves data from the collection "studentsn". Let's break down what each part does:

db.studentsn specifies the collection "studentsn" within the current database.

.find() is a method that finds documents in the collection.

((), (name: 1, age: 1)) is an optional argument to the .find() method. The first empty set of parentheses () specifies the query filter. In this case, it's an empty filter, meaning all documents in the collection will be returned. The second part (name: 1, age: 1) is a projection document that specifies which fields to include in the results. Here, only the "name" and "age" fields will be returned for each document.

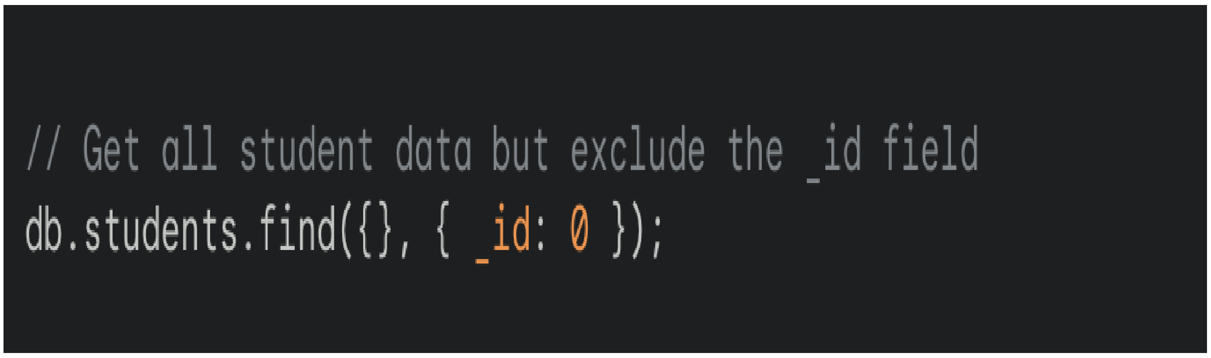
Output: The query successfully retrieves documents from the collection "studentsn" and displays them. Each document shows its _id field (automatically generated by MongoDB for each document) along with the projected fields "name" and "age".

Overall, the code connects to MongoDB, selects the database being used by the collection "studentsn", finds all documents in that collection, and projects only the "name" and "age" fields for each document

Get selected attributes:

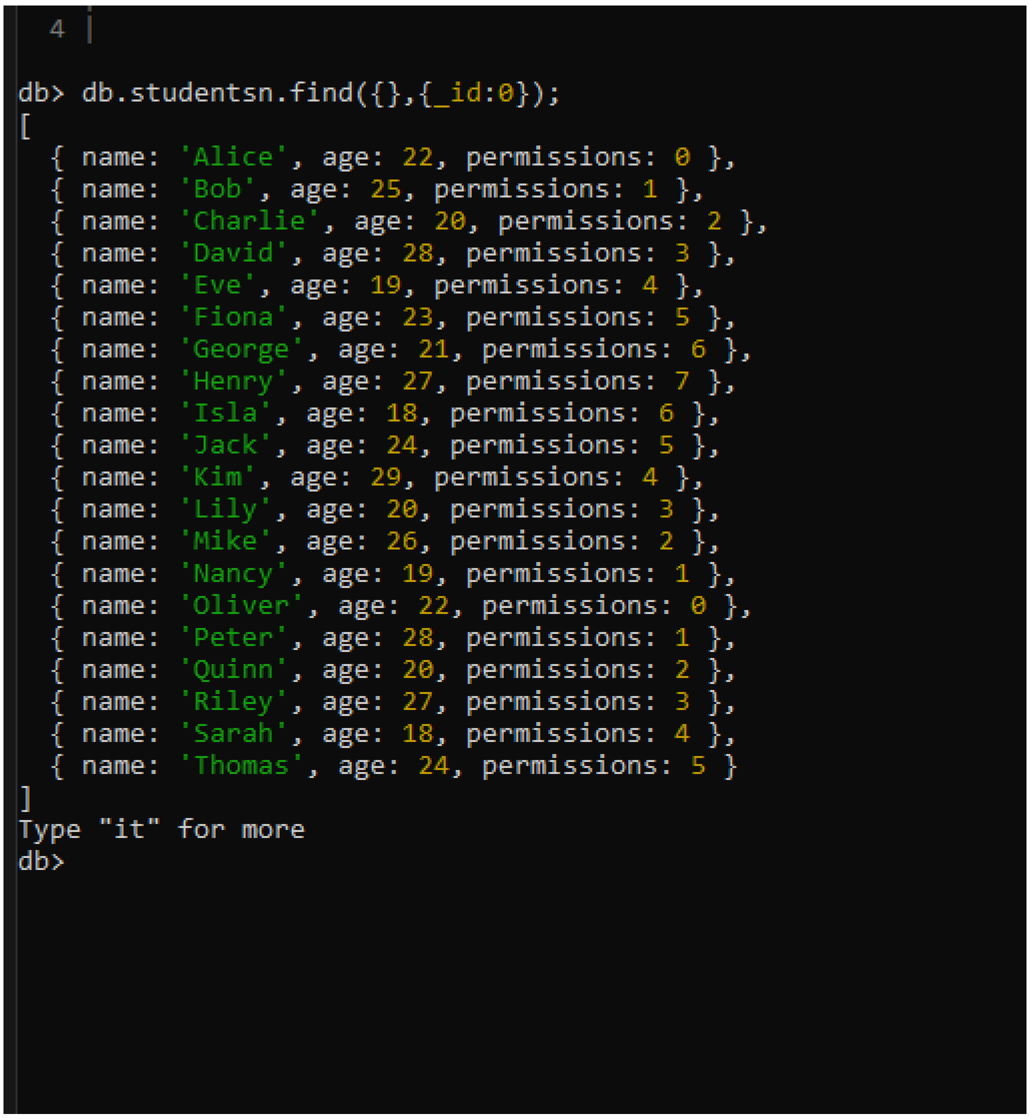
Finds the distinct values for a specified field across a single collection or view and returns the results in an array.
It give a collection you want to filter a subset of attributes .That is the place projection is used

Input:



```
// Get all student data but exclude the _id field
db.students.find({}, { _id: 0 });
```

Output:



```
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>
```

Output Explaination:

Selecting the Database:

db = studentsn This line attempts to select a database named "studentsn". However, in MongoDB, databases are created implicitly when you first insert data into a collection. So, if a database named "studentsn" doesn't exist yet, it will be created automatically upon running this line

Error Handling (Incorrect Database Selection):

ReferenceError: studentsn is not defined This error message indicates that the variable "studentsn" is not defined as a database. As mentioned earlier, MongoDB databases are created implicitly, so selecting a non-existent database using db = database_name will result in this error.

Correcting the Database Selection:

db = db.studentsn This line correctly selects the database using the db object. In MongoDB, the db object refers to the current database. Since there is no database named "studentsn" explicitly created, this line essentially sets the current database to the one being used by the collection "studentsn" (which might have been created implicitly earlier).

Finding Documents in the Collection:

db.studentsn.find((), (name: 1, age: 1)); This line is the actual query that retrieves data from the collection "studentsn". Let's break down what each part does:

db.studentsn specifies the collection "studentsn" within the current database.

.find() is a method that finds documents in the collection.

((), (name: 1, age: 1)) is an optional argument to the .find() method. The first empty set of parentheses () specifies the query filter. In this case, it's an empty filter, meaning all documents in the collection will be returned. The second part (name: 1, age: 1) is a projection document that specifies which fields to include in the results. Here, only the "name" and "age" fields will be returned for each document.

Output: The query successfully retrieves documents from the collection "studentsn" and displays them. Each document shows its _id field (automatically generated by MongoDB for each document) along with the projected fields "name" and "age".

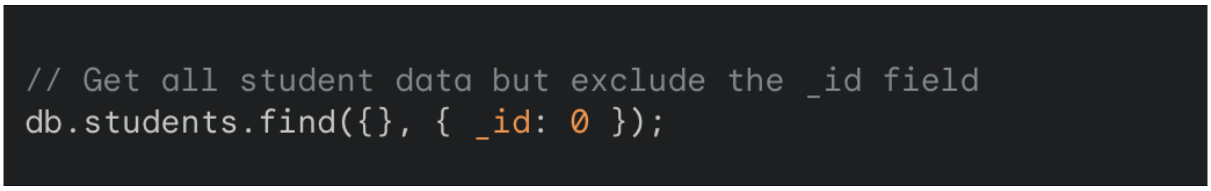
Overall, the code connects to MongoDB, selects the database being used by the collection "studentsn", finds all documents in that collection, and projects only the "name" and "age" fields for each document.

Ignore attributes:

In MongoDB, the **Ignore Attribute** is used to:

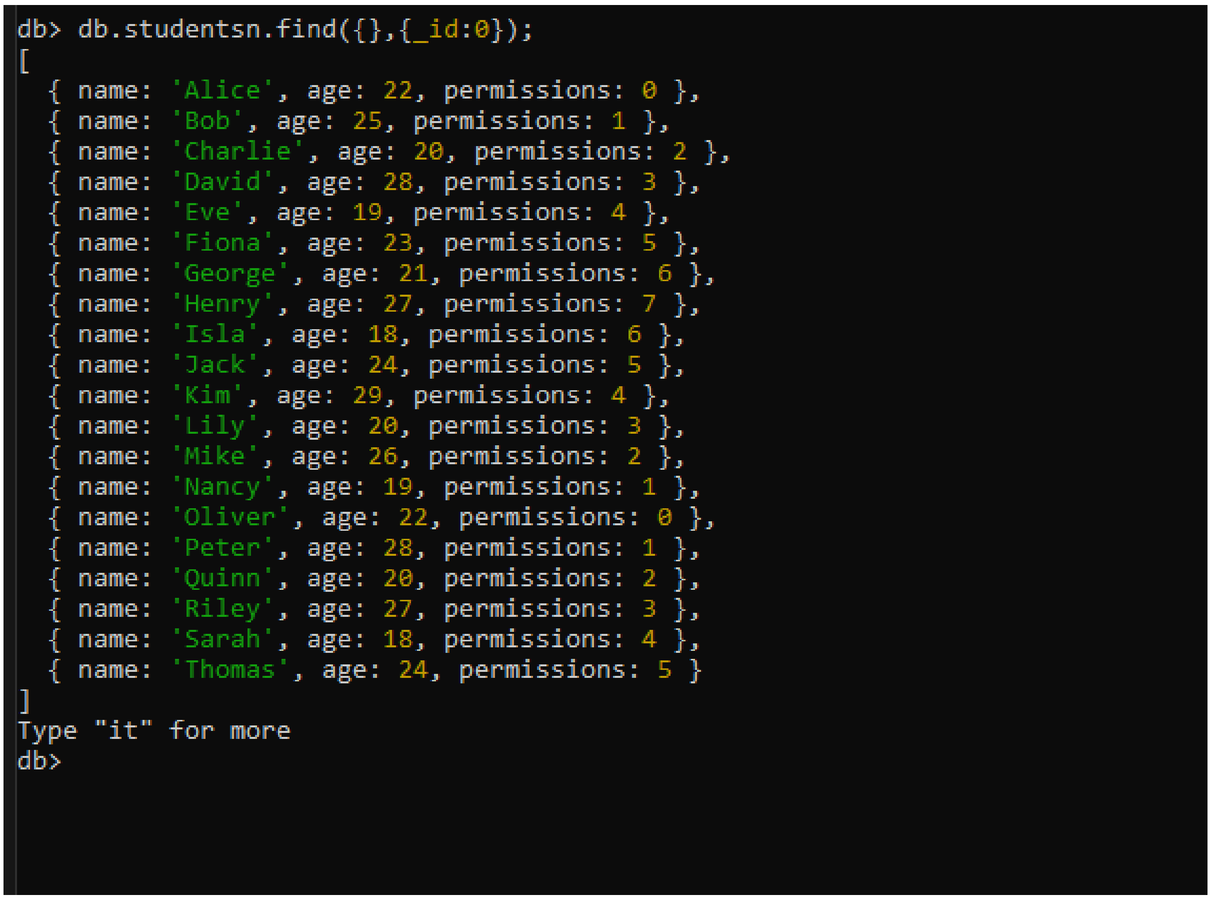
1. **Ignore a property when persisting an entity to the database**[1].
2. If you specify an inclusion of a field that does not exist in the document, **$project ignores that field inclusion and does not add the field to the document**[2]

Input:



```
// Get all student data but exclude the _id field
db.students.find({}, { _id: 0 });
```

Output:



```
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>
```

Output explaination:

Selecting the Database:

db = studentsn This line attempts to select a database named "studentsn". However, in MongoDB, databases are created implicitly when you first insert data into a collection. So, if a database named "studentsn" doesn't exist yet, it will be created automatically upon running this line.

Error Handling (Incorrect Database Selection):

ReferenceError: studentsn is not defined This error message indicates that the variable "studentsn" is not defined as a database. As mentioned earlier, MongoDB databases are created implicitly, so selecting a non-existent database using db = database_name will result in this error.

Correcting the Database Selection:
db = db.studentsn This line correctly selects the database using the db object. In MongoDB, the db object refers to the current database. Since there is no database named "studentsn" explicitly created, this line essentially sets the current database to the one being used by the collection "studentsn" (which might have been created implicitly earlier).

Finding Documents in the Collection:
db.studentsn.find((), (name: 1, age: 1)); This line is the actual query that retrieves data from the collection "studentsn". Let's break down what each part does:
db.studentsn specifies the collection "studentsn" within the current database.
.find() is a method that finds documents in the collection.
((), (name: 1, age: 1)) is an optional argument to the .find() method. The first empty set of parentheses () specifies the query filter. In this case, it's an empty filter, meaning all documents in the collection will be returned. The second part (name: 1, age: 1) is a projection document that specifies which fields to include in the results. Here, only the "name" and "age" fields will be returned for each document.
Output: The query successfully retrieves documents from the collection "studentsn" and displays them. Each document shows its _id field (automatically generated by MongoDB for each document) along with the projected fields "name" and "age".
Overall, the code connects to MongoDB, selects the database being used by the collection "studentsn", finds all documents in that collection, and projects only the "name" and "age" fields for each document.

Retriveing specific fields from nested object:

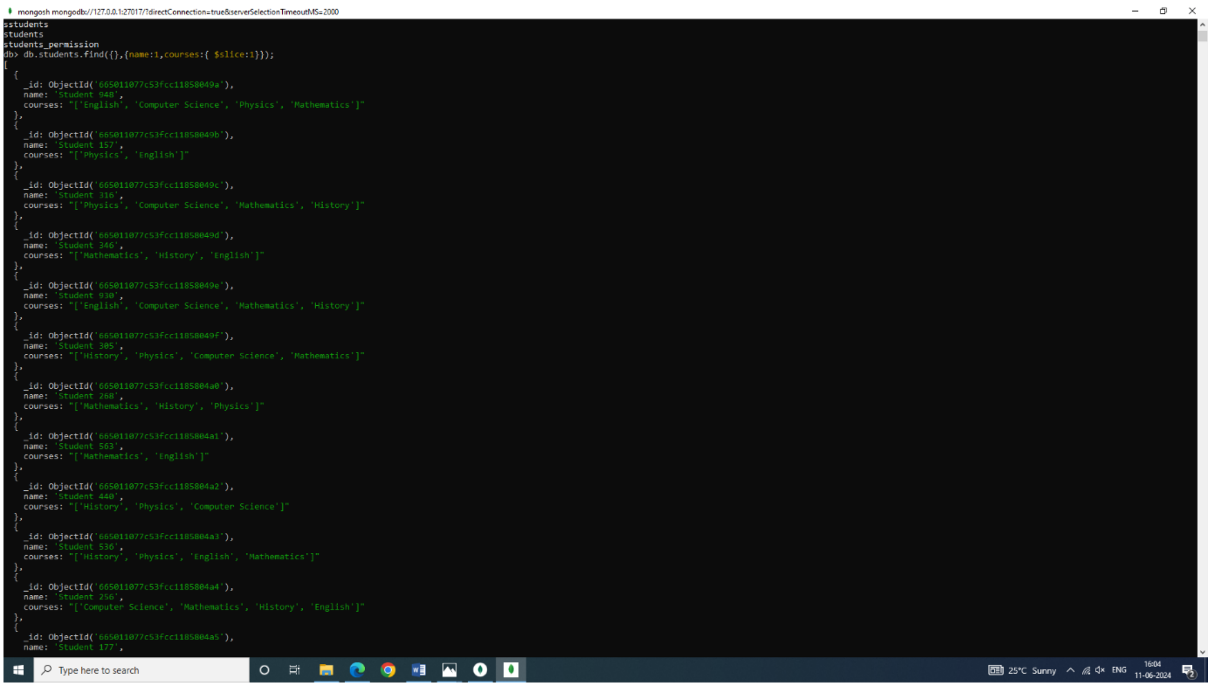
Sometimes you might want to **retrieve** only a few fields from a deeply nested structure. The projection parameter in the find method allows to specify the fields you want to include or exclude from the results: db.users.find({}, { 'address.city': 1, 'address.state': 1 })

Try $slice

Input:

```
// Get student name and only the first course from the courses array
db.students.find({}, {
  name: 1,
  courses: { $slice: 1 }
});
```

Output:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
sstudents
students
students_permission
db> db.students.find({},{name:1,courses:{ $slice:1}});
[
  {
    _id: ObjectId('665011077c53fcc11858049a'),
    name: 'Student 948',
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']"
  },
  {
    _id: ObjectId('665011077c53fcc11858049b'),
    name: 'Student 157',
    courses: "['Physics', 'English']"
  },
  {
    _id: ObjectId('665011077c53fcc11858049c'),
    name: 'Student 316',
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']"
  },
  {
    _id: ObjectId('665011077c53fcc11858049d'),
    name: 'Student 346',
    courses: "['Mathematics', 'History', 'English']"
  },
  {
    _id: ObjectId('665011077c53fcc11858049e'),
    name: 'Student 930',
    courses: "['English', 'Computer Science', 'Mathematics', 'History']"
  },
  {
    _id: ObjectId('665011077c53fcc11858049f'),
    name: 'Student 305',
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']"
  },
  {
    _id: ObjectId('665011077c53fcc1185804a0'),
    name: 'Student 268',
    courses: "['Mathematics', 'History', 'Physics']"
  },
  {
    _id: ObjectId('665011077c53fcc1185804a1'),
    name: 'Student 563',
    courses: "['Mathematics', 'English']"
  },
  {
    _id: ObjectId('665011077c53fcc1185804a2'),
    name: 'Student 440',
    courses: "['History', 'Physics', 'Computer Science']"
  },
  {
    _id: ObjectId('665011077c53fcc1185804a3'),
    name: 'Student 536',
    courses: "['History', 'Physics', 'English', 'Mathematics']"
  },
  {
    _id: ObjectId('665011077c53fcc1185804a4'),
    name: 'Student 256',
    courses: "['Computer Science', 'Mathematics', 'History', 'English']"
  },
  {
    _id: ObjectId('665011077c53fcc1185804a5'),
    name: 'Student 177',
```

Output explaination:

db.students.find({}): This line uses the find method to retrieve documents from the "students" collection. The empty curly braces {} specify that all documents in the collection should be returned.

{name: 1, courses: 1}: This part of the code defines which fields to return from the documents. The 1 indicates that only the "name" and "courses" fields should be included in the output, and any other fields will be excluded.

The output shows multiple student documents with their names and enrolled courses.

Here are some additional points to note:

The ObjectId values displayed are unique identifiers assigned to each document in the collection.

The find method can be used with more complex queries to filter and sort data based on specific criteria.

I can't provide information about the specific program or application this code snippet originates from as it only shows a portion of the code.

Benefits of projection:

- Reduces data transferred between the database and your application.
- Improves query performance by retrieving only necessary data.
- Simplifies your code by focusing on the specific information you need.

Limit:

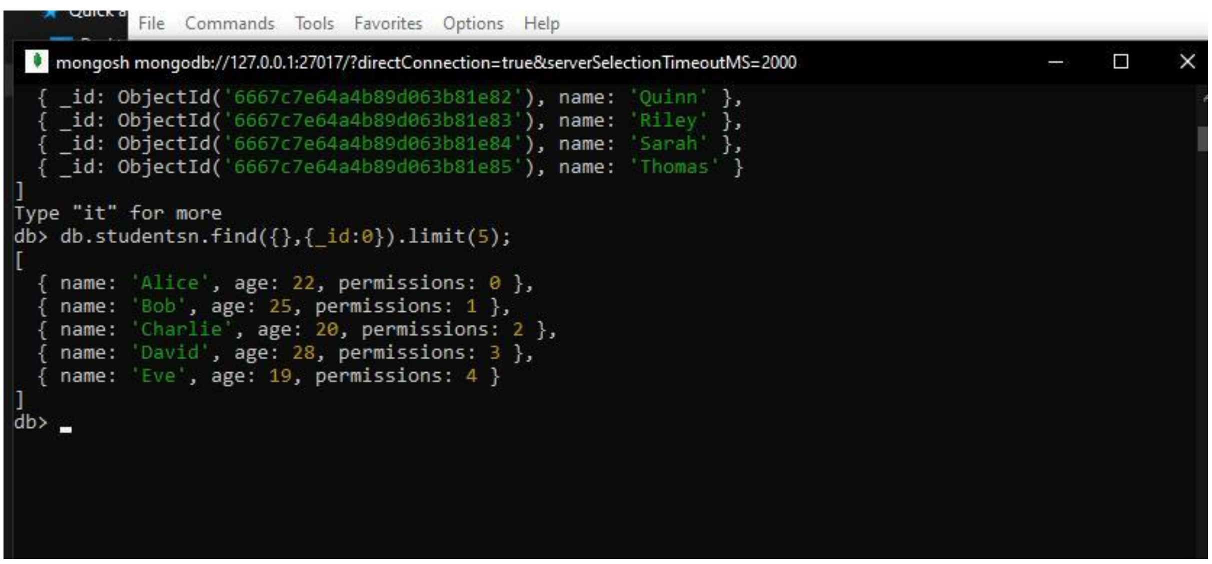
- The `limit` operator is used with the `find` method.
- It's chained after the filter criteria or any sorting operations.
- Syntax: `db.collection.find({filter}, {projection}).limit(number)`

Get first five documents:

Input:

```
// Assuming you have already executed a query on the student collection
// Limit the results to the first 5 documents
db.students.find({}, { _id: 0 }).limit(5);
```

Output:


File Commands Tools Favorites Options Help
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

```
  { _id: ObjectId('6667c7e64a4b89d063b81e82'), name: 'Quinn' },
  { _id: ObjectId('6667c7e64a4b89d063b81e83'), name: 'Riley' },
  { _id: ObjectId('6667c7e64a4b89d063b81e84'), name: 'Sarah' },
  { _id: ObjectId('6667c7e64a4b89d063b81e85'), name: 'Thomas' }
]
Type "it" for more
db> db.studentsn.find({},{_id:0}).limit(5);
[
  { 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 }
]
db>
```

Output explaination:

db.students.find({},{_id: 0}): This line uses the find method to retrieve documents from the "students" colection.

The first curly braces {} specify the query criteria. In this case, it selects all documents because it's empty.

The second curly braces {_id: 0} specify which fields to exclude from the output. The 0 here indicates that the "_id" field should not be included.

.limit(5): This limits the number of documents returned to 5.

The output shows five student documents, each with an "_id" field (which is hidden by the query) and other fields like "name", "age", and "permissions".

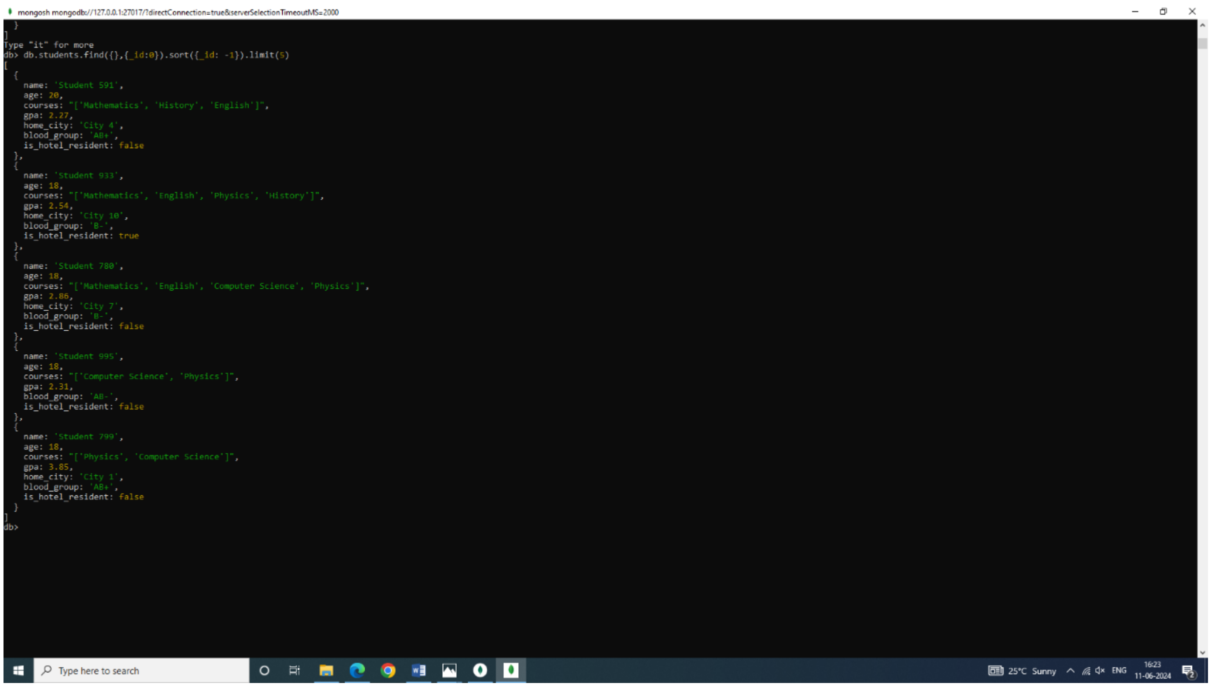
Here are some additional points to note:

MongoDB uses documents to store data. Documents are similar to JSON objects with key-value pairs.

The "_id" field is a special field in MongoDB that uniquely identifies each document in a collection. It's automatically generated by MongoDB when a document is inserted.

The find method is a versatile method for retrieving data from MongoDB collections. You can use it with various query filters and options to target specific documents.

Limiting results:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
  }
]
Type "it" for more
db> db.students.find({},{_id:0}).sort({_id: -1}).limit(5)
[
  {
    name: 'Student 591',
    age: 20,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    name: 'Student 933',
    age: 18,
    courses: "['Mathematics', 'English', 'Physics', 'History']",
    gpa: 2.54,
    home_city: 'City 10',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    name: 'Student 780',
    age: 18,
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']",
    gpa: 2.86,
    home_city: 'City 7',
    blood_group: 'B-',
    is_hotel_resident: false
  },
  {
    name: 'Student 995',
    age: 18,
    courses: "['Computer Science', 'Physics']",
    gpa: 2.31,
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    name: 'Student 799',
    age: 18,
    courses: "['Physics', 'Computer Science']",
    gpa: 3.85,
    home_city: 'City 1',
    blood_group: 'AB+',
    is_hotel_resident: false
  }
]
db>
```

Output explaination:

db.students.find({}): This line uses the find method to retrieve documents from the "students" collection. The empty curly braces {} specify that all documents in the collection should be returned.

{name: 1, courses: 1}: This part of the code defines which fields to return from the documents. The 1 indicates that only the "name" and "courses" fields should be included in the output, and any other fields will be excluded.

The output shows multiple student documents with their names and enrolled courses.

Here are some additional points to note:

The ObjectId values displayed are unique identifiers assigned to each document in the collection.

The find method can be used with more complex queries to filter and sort data based on specific criteria.

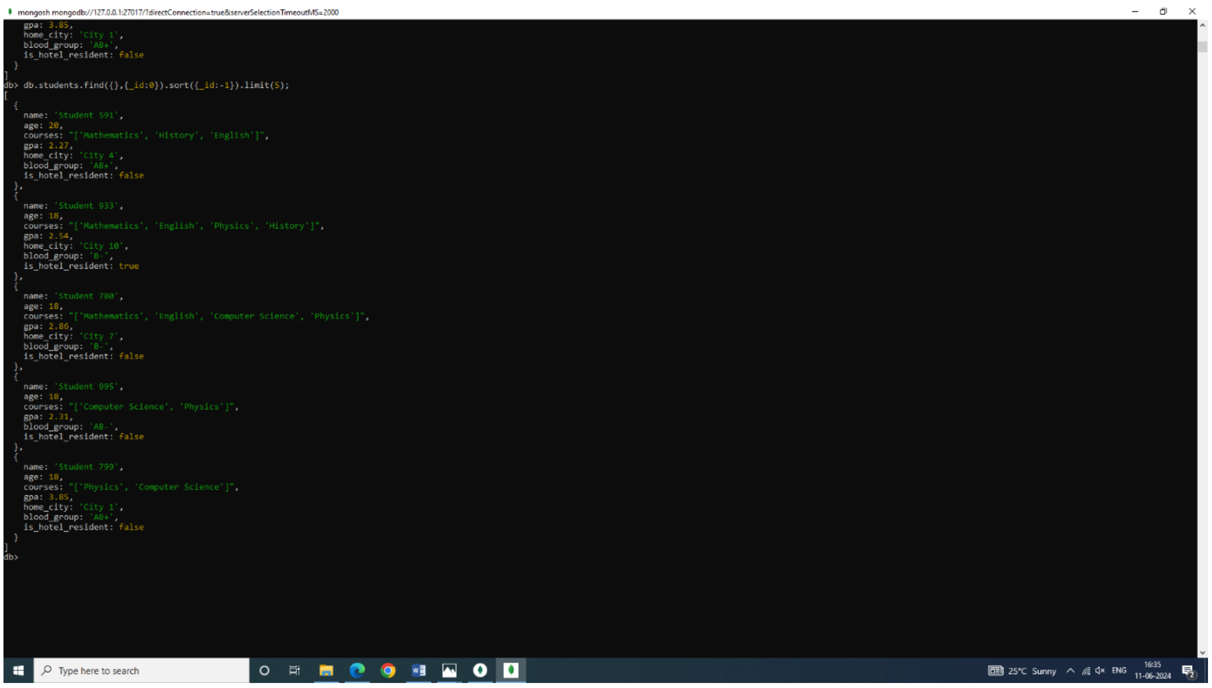
I can't provide information about the specific program or application this code snippet originates from as it only shows a portion of the code.

I want top 10 RESULTS:

Input:

```
// Sort documents in descending order by _id and limit to 5
db.students.find({}, { _id: 0 }).sort({ _id: -1 }).limit(5);
```

Output:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
    gpa: 3.85,
    home_city: 'City 1',
    blood_group: 'AB+',
    is_hotel_resident: false
  }
]
db> db.students.find({},{_id:0}).sort({_id:-1}).limit(5);
[
  {
    name: 'Student 591',
    age: 20,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    name: 'Student 933',
    age: 18,
    courses: "['Mathematics', 'English', 'Physics', 'History']",
    gpa: 2.54,
    home_city: 'City 10',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    name: 'Student 780',
    age: 18,
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']",
    gpa: 2.86,
    home_city: 'City 7',
    blood_group: 'B-',
    is_hotel_resident: false
  },
  {
    name: 'Student 995',
    age: 18,
    courses: "['Computer Science', 'Physics']",
    gpa: 2.31,
    blood_group: 'AB-',
    is_hotel_resident: false
  },
  {
    name: 'Student 799',
    age: 18,
    courses: "['Physics', 'Computer Science']",
    gpa: 3.85,
    home_city: 'City 1',
    blood_group: 'AB+',
    is_hotel_resident: false
  }
]
db>
```

Output explaination:

db.students.find({}): This line uses the find method to retrieve documents from the "students" collection. The empty curly braces {} specify that all documents in the collection should be returned.

{name: 1, courses: 1}: This part of the code defines which fields to return from the documents. The 1 indicates that only the "name" and "courses" fields should be included in the output, and any other fields will be excluded.

The output shows multiple student documents with their names and enrolled courses.

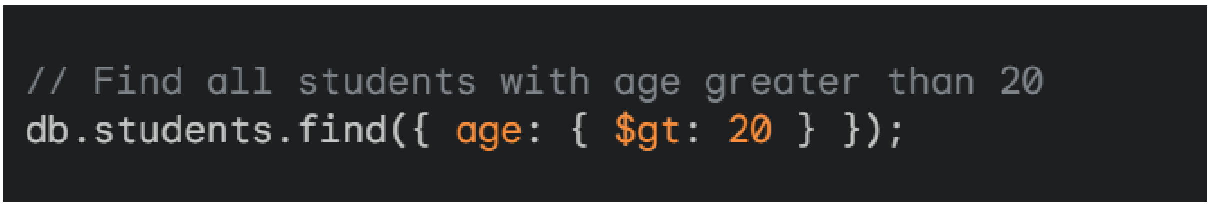
Here are some additional points to note:

Experiment:3

Comparison gt lt:

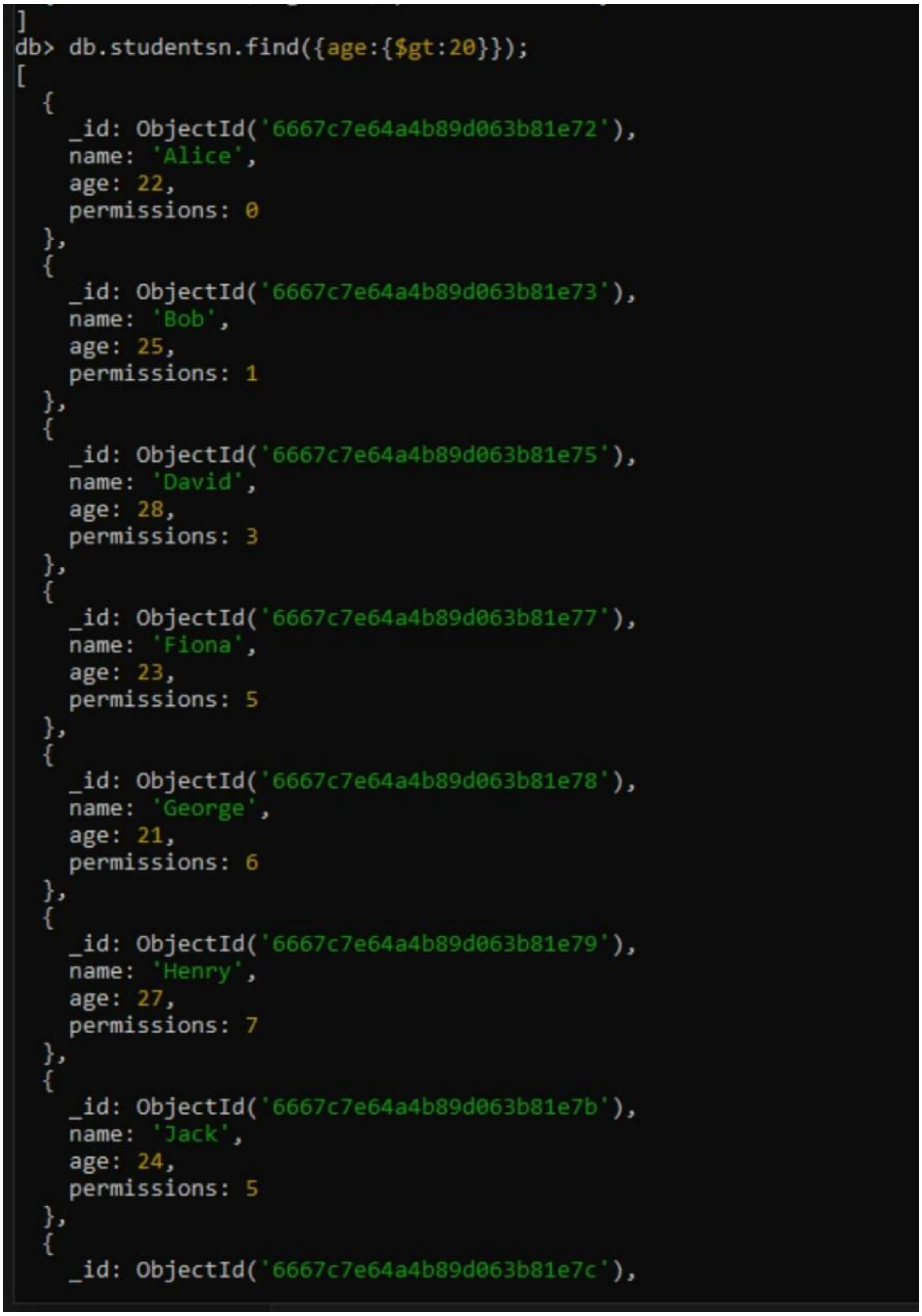
MongoDB comparison operators are a set of special operators used to perform comparison operations, including equality, inequality, greater than, less than, greater than or equal to, and less than or equal to. They can be used for conditional filtering in both aggregate and normal queries. This page summarizes the comparison operators in MongoDB

Input:



```
// Find all students with age greater than 20
db.students.find({ age: { $gt: 20 } });
```

Output:



```
]
db> db.studentsn.find({age:{$gt:20}});
[
  {
    _id: ObjectId('6667c7e64a4b89d063b81e72'),
    name: 'Alice',
    age: 22,
    permissions: 0
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e73'),
    name: 'Bob',
    age: 25,
    permissions: 1
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e75'),
    name: 'David',
    age: 28,
    permissions: 3
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e77'),
    name: 'Fiona',
    age: 23,
    permissions: 5
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e78'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e79'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e7b'),
    name: 'Jack',
    age: 24,
    permissions: 5
  },
  {
    _id: ObjectId('6667c7e64a4b89d063b81e7c'),
```

Output explanation:

db.students.find({}): This part uses the find method to retrieve documents from the "students" collection.

{age:{$gt:20}}: This part filters the documents based on the value of the "age" field. The $gt operator is a comparison operator that checks if a field is greater than a specified value. In this case, the query selects documents where the "age" field is greater than 20.

The output shows multiple student documents where the "age" field is greater than 20. Each document contains all fields stored for that student.

Here are some additional points to note:

MongoDB collections store data in JSON-like documents.

The find method is a versatile method for retrieving data from MongoDB collections. You can use it with various query filters and options to target specific documents.

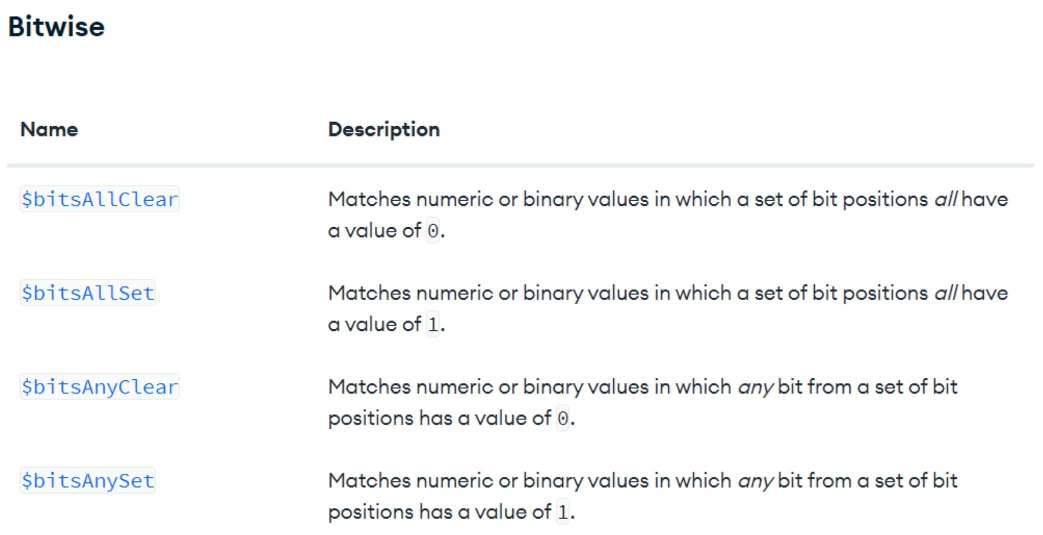
The $gt operator is one of many comparison operators available in MongoDB. You can use it to filter documents based on equality, greater than, less than, and other comparisons.

Lets take new data set:

- New Students Permission dataset link

Explanation: Collection name: students_permission

- **name:** Student's name (string)
- **age:** Student's age (number)
- **permissions:** Bitmask representing user permissions (number
- **Bitwise value:**
- Bitwise operators return data based on bit position conditions.
- Bitwise operators return data based on bit
- In our example its a 32 bit each bit representing different things
- Bitwise value 7 means all access 7 -> 111
- Bitwise types:



Bitwise

Name	Description
`$bitsAllClear`	Matches numeric or binary values in which a set of bit positions *all* have a value of `0`.
`$bitsAllSet`	Matches numeric or binary values in which a set of bit positions *all* have a value of `1`.
`$bitsAnyClear`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `0`.
`$bitsAnySet`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `1`.

-
- Query:

MongoDB Query is a fundamental aspect of MongoDB that allows users to fetch data from the database. Similar to **SQL** queries in traditional databases, MongoDB queries provide simplicity and flexibility in retrieving specific data based on certain criteria or conditions. In this article, We will learn about the MongoDB Query in detail by performing various **Queries on document** of MongoDB collection and so on.

MongoDB Query

- MongoDB **Query** allows retrieving data from the MongoDB database. MongoDB Query provides simplicity in the process of fetching data from the database, it's similar to SQL queries in SQL Database language.
- While performing a query operation, one can also use criteria or conditions that can be used to retrieve specific data from the database.
- MongoDB provides the function names as **db.collection_name.find()** to operate query operations on the database.

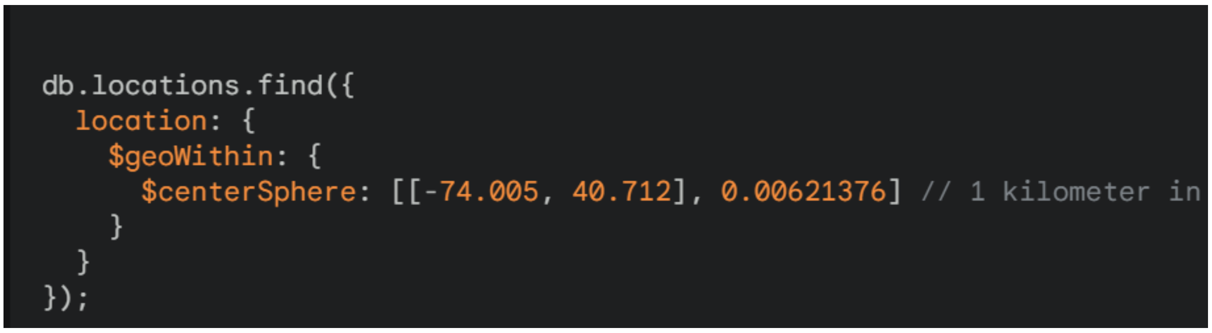
GEOSPATIAL:

Geospatial indexes **support queries on data stored as GeoJSON objects or legacy coordinate pairs**. You can use geospatial indexes to improve performance for queries on geospatial data or to run certain geospatial queries. MongoDB provides two types of geospatial indexes: 2dsphere Indexes, which support queries that interpret geometry on a sphere.

- Official Documentation link
- Create collection called "locations"
- Upload the dataset using json link

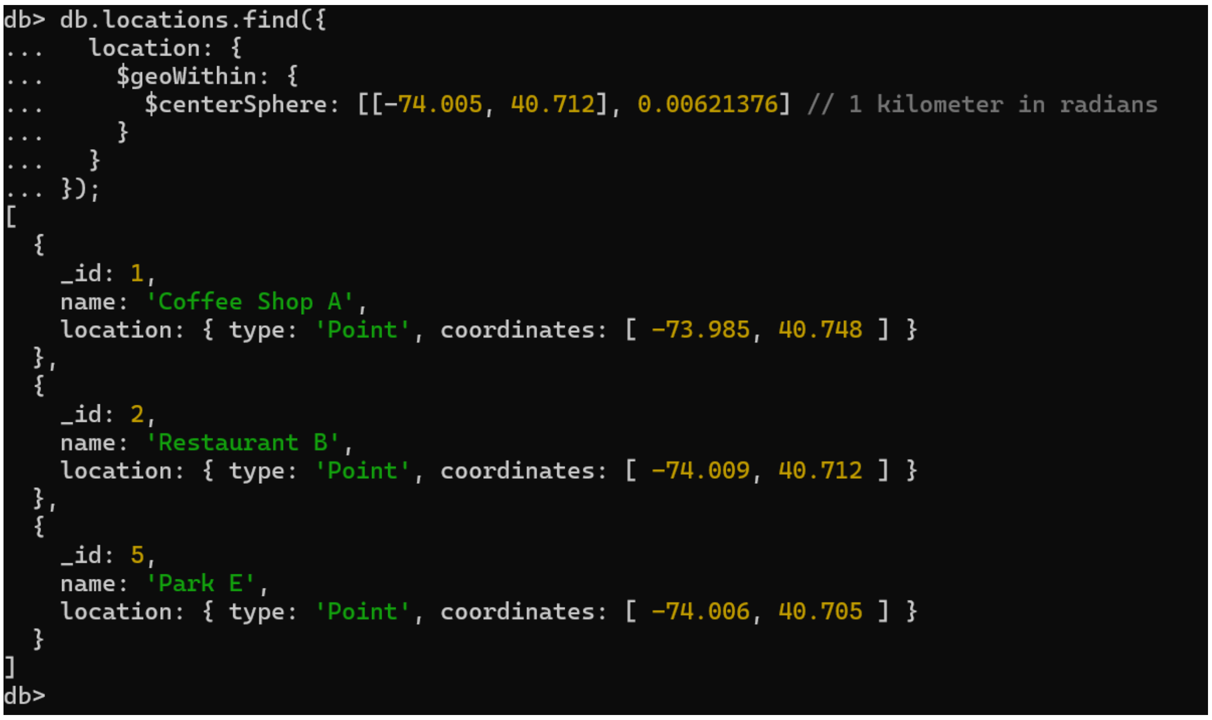
Geospatial query:

Input:



```
db.locations.find({
  location: {
    $geoWithin: {
      $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in
    }
  }
});
```

Output:



```
db> db.locations.find({
...   location: {
...     $geoWithin: {
...       $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in radians
...     }
...   }
... });
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
db>
```

Code explanation:

db.students.find(): This part uses the find method to retrieve documents from the "students" collection. The empty parentheses () specify that all documents in the collection should be returned

(_id: 0): This part excludes the "_id" field from the output documents. The 0 here indicates that the "_id" field should not be included.

.sort({"_id": -1}) .limit(5): This sorts he results by the "_id" field in descending order (indicated by -1) and limits the output to the top 5 documents using the .limit(5) method.

The output shows the five most recently added student documents, excluding the "_id" field, which contains a unique identifier for each document. Each document includes fields like "name", "age", "courses", "gpa", and "is_hotel_resident".

Here are some additional points to note:

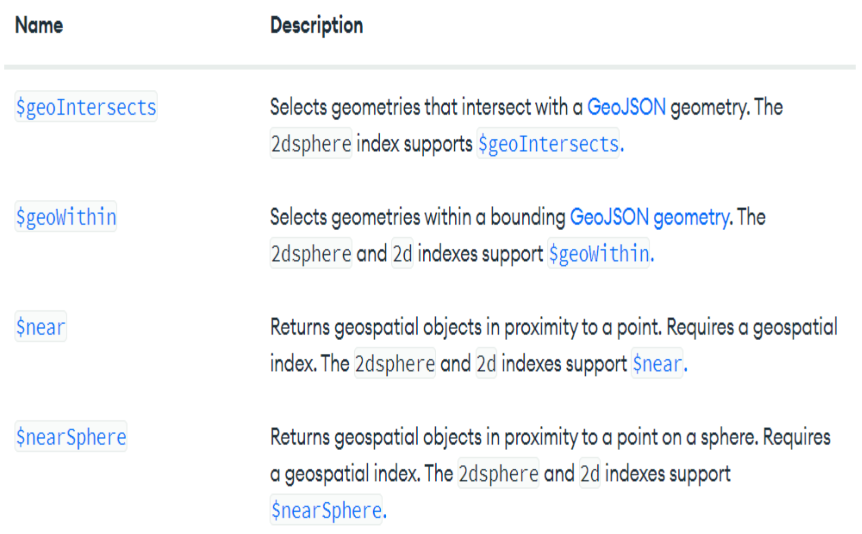
MongoDB collections store data in JSON-like documents.

The _id field is a special field that uniquely identifies each document in a collection and it's automatically generated by MongoDB when a document is inserted.

The find method is a versatile method for retrieving data from MongoDB collections. You can use it with various query filters and options to target specific documents.

Data types and operations:

- **DataType**
  - Point
  - Line String
  - Polygon



Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The `2dsphere` and `2d` indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.

In MongoDB, the documents are stores in BSON, which is the binary encoded format of JSON and using BSON we can make remote procedure calls in MongoDB. BSON data format supports various data-types. Below are the enlisted MongoDB data types:
1. String: This is the most commonly used data type in MongoDB to store data, BSON strings are of UTF-8. So, the drivers for each programming language convert from the string format of the language to UTF-8 while serializing and de-serializing BSON. The string must be a valid UTF-8.