MONGODB

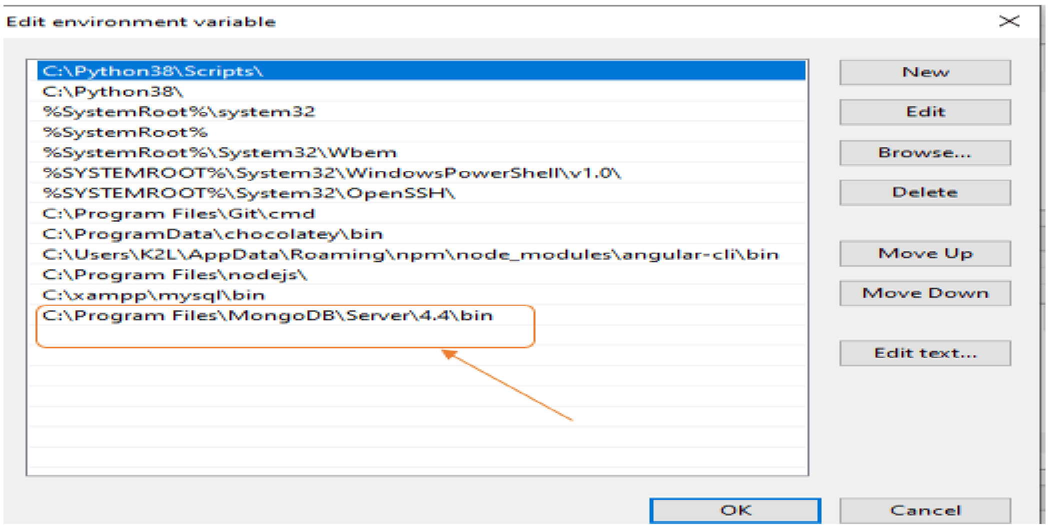
MONGO DB INSTALLATION:

1.Search for MongoDB website & Install: Visit the official mongoDB website for downloading the mongoDb compass and the shell.
Here, You can select any version, Windows, and package according to your requirement. For Windows, we need to choose:

- Version: 7.0.4
- OS: Windows x64
- Package: msi

2.After installation: When the download is complete open the msi file and click the *next button*.accept the End-User License Agreement Now click the ***Finish button*** to complete the MongoDB installation process:

3.Navigating the location:Check where the download file is located and then edit the path in the Now, to create an environment variable open system properties >> Environment Variable >> System variable >> path >> Edit Environment variable and paste the copied link to your environment system and click OK.



Edit environment variable

```
C:\Python38\Scripts\
C:\Python38\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\ProgramData\chocolatey\bin
C:\Users\K2L\AppData\Roaming\npm\node_modules\angular-cli\bin
C:\Program Files\nodejs\
C:\xampp\mysql\bin
C:\Program Files\MongoDB\Server\4.4\bin
```

4.After setting the environment variable, we will run the MongoDB server, i.e. mongod. So, open the command prompt and run the following command.

5. Now we are going to connect our server (mongod) with the mongo shell. So, keep that mongod window and open a new command prompt window and write **mongo.** Now, our mongo shell will successfully connect to the mongod.Now you can make a new database, collections, and documents in your shell.

INTRODUCTION

MongoDB, the most popular NoSQL database, is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON (similar to JSON format).
SQL databases store data in tabular format. This data is stored in a predefined data model which is not very much flexible for today's real-world highly growing applications. **Modern applications are more networked, social and interactive than ever**. Applications are storing more and more data and are accessing it at higher rates.

Where do we use MongoDB?

- Big Data: If you have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharding your database.
- Unstable Schema: Adding a new column in RDBMS is hard whereas MongoDB is schema-less. Adding a new field does not effect old documents and will be very easy.

- Distributed data Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

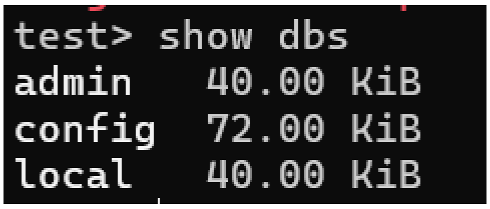
Features of MongoDB:

- Document Oriented: MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like CPU, RAM, Hard disk, etc.
- Indexing: Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.
- Scalability: MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. Also, new machines can be added to a running database.

Database & Collection

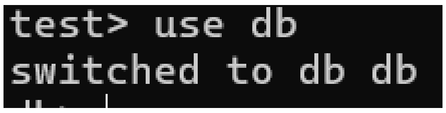
Databases, collections, documents are important parts of MongoDB without them you are not able to store data on the MongoDB server. A Database contains a collection, and a collection contains documents and the documents contain data, they are related to each other.

To view a database we use the command:
show dbs



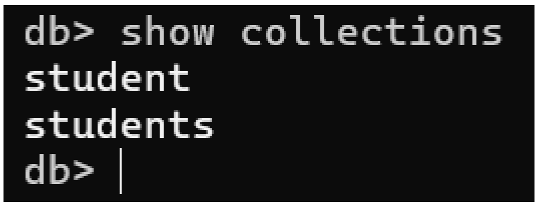
```
test> show dbs
admin   40.00 KiB
config  72.00 KiB
local   40.00 KiB
```

To use the database we use the command:
use db



```
test> use db
switched to db db
```

Collections:
To show the collections stored in mongodb compass we use the command
show collections



```
db> show collections
student
students
db>
```

DataTypes in MongoDB:

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.

2. Integer: In MongoDB, the integer data type is used to store an integer value. We can store integer data type in two forms 32 -bit signed integer and 64 – bit signed integer.

3.Double: The double data type is used to store the floating-point values.

4. Boolean: The boolean data type is used to store either true or false.

5. Null: The null data type is used to store the null value.

6. Array: The Array is the set of values. It can store the same or different data types values in it. In MongoDB, the array is created using square brackets([]).

MongoDB Methods:

1.Find() Method:

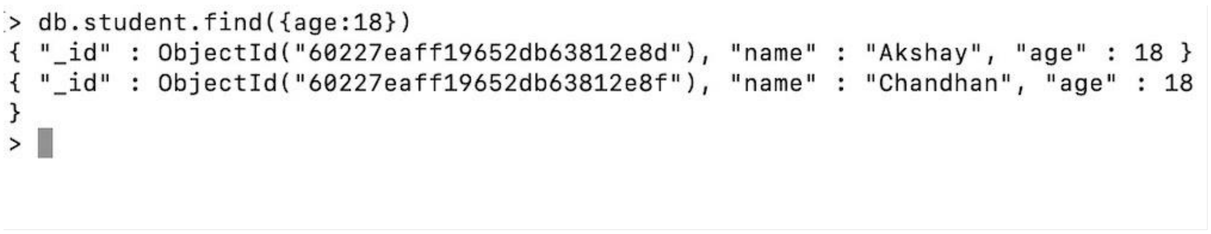
The **find ()** method in MongoDB **selects documents in a collection** that matches the specified conditions and returns a cursor to the selected documents

Syntax

```
db.Collection_name.find(selection_criteria,projection,options)
```

Example:

```
db.student.find({age:18})
```



```
> db.student.find({age:18})
{ "_id" : ObjectId("60227eaff19652db63812e8d"), "name" : "Akshay", "age" : 18 }
{ "_id" : ObjectId("60227eaff19652db63812e8f"), "name" : "Chandhan", "age" : 18
}
>
```

2.sort() Method:

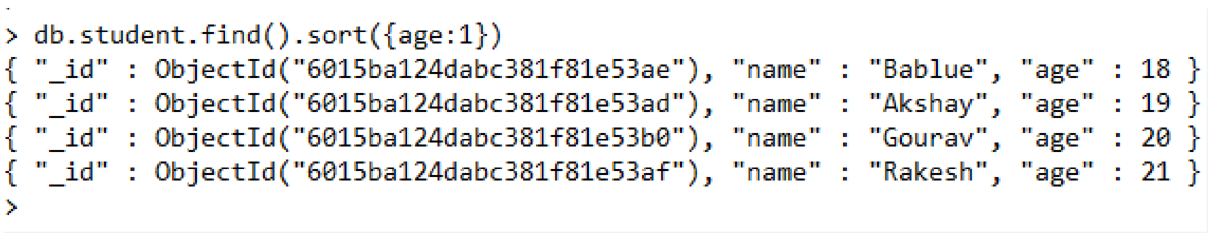
The **sort()** method specifies the order in which the query returns the matching documents from the given collection. You must apply this method to the cursor before retrieving any documents from the database. It takes a document as a parameter that contains a field: value pair that defines the sort order of the result set. The value is 1 or -1 specifying an ascending or descending sort respectively.

Syntax:

```
db.Collection_Name.sort({field_name:1 or -1})
```

Example:

```
db.student.find().sort({age:1})
```



```
> db.student.find().sort({age:1})
{ "_id" : ObjectId("6015ba124dabc381f81e53ae"), "name" : "Bablue", "age" : 18 }
{ "_id" : ObjectId("6015ba124dabc381f81e53ad"), "name" : "Akshay", "age" : 19 }
{ "_id" : ObjectId("6015ba124dabc381f81e53b0"), "name" : "Gourav", "age" : 20 }
{ "_id" : ObjectId("6015ba124dabc381f81e53af"), "name" : "Rakesh", "age" : 21 }
>
```

3.count() Method:

The count() method counts the number of documents that match the selection criteria. It returns the number of documents that match the selection criteria. It takes two arguments first one is the selection criteria and the other is optional.

Syntax:

```
db.Collection_name.count()
```

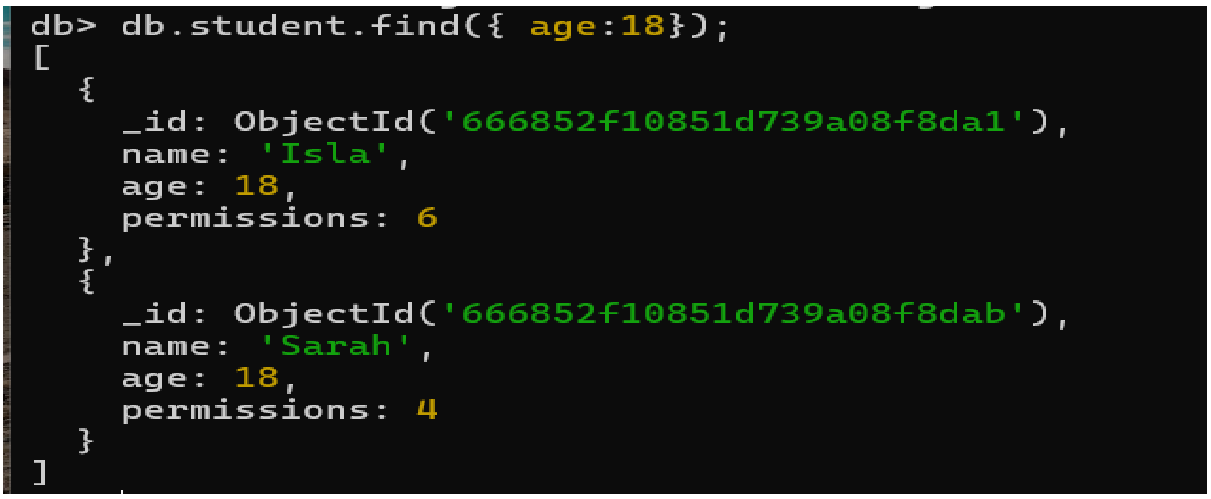
Example:

`db.student.count()`

<u>WHERE,AND,OR & CRUD OPERATIONS:</u>

WHERE:

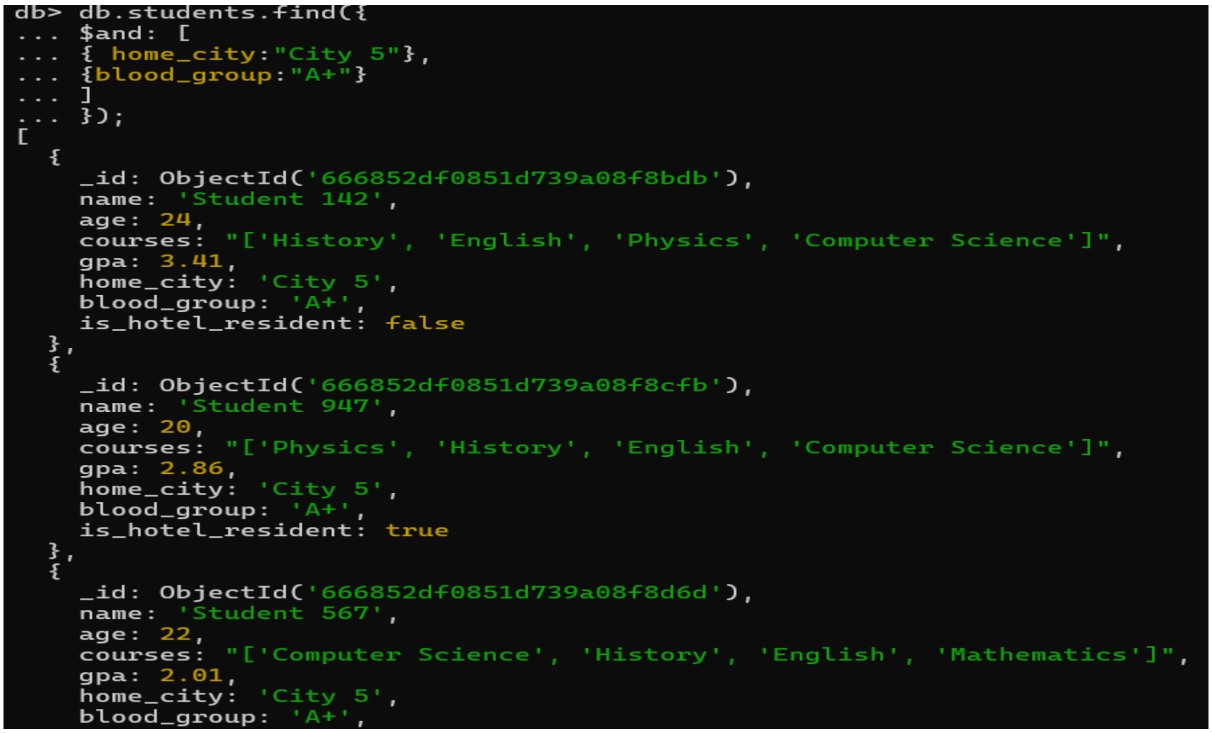
- The collection you want to FILTER a subset based on a condition,this is WHERE operation is used.



```
db> db.student.find({ age:18});
[
  {
    _id: ObjectId('666852f10851d739a08f8da1'),
    name: 'Isla',
    age: 18,
    permissions: 6
  },
  {
    _id: ObjectId('666852f10851d739a08f8dab'),
    name: 'Sarah',
    age: 18,
    permissions: 4
  }
]
```

AND:

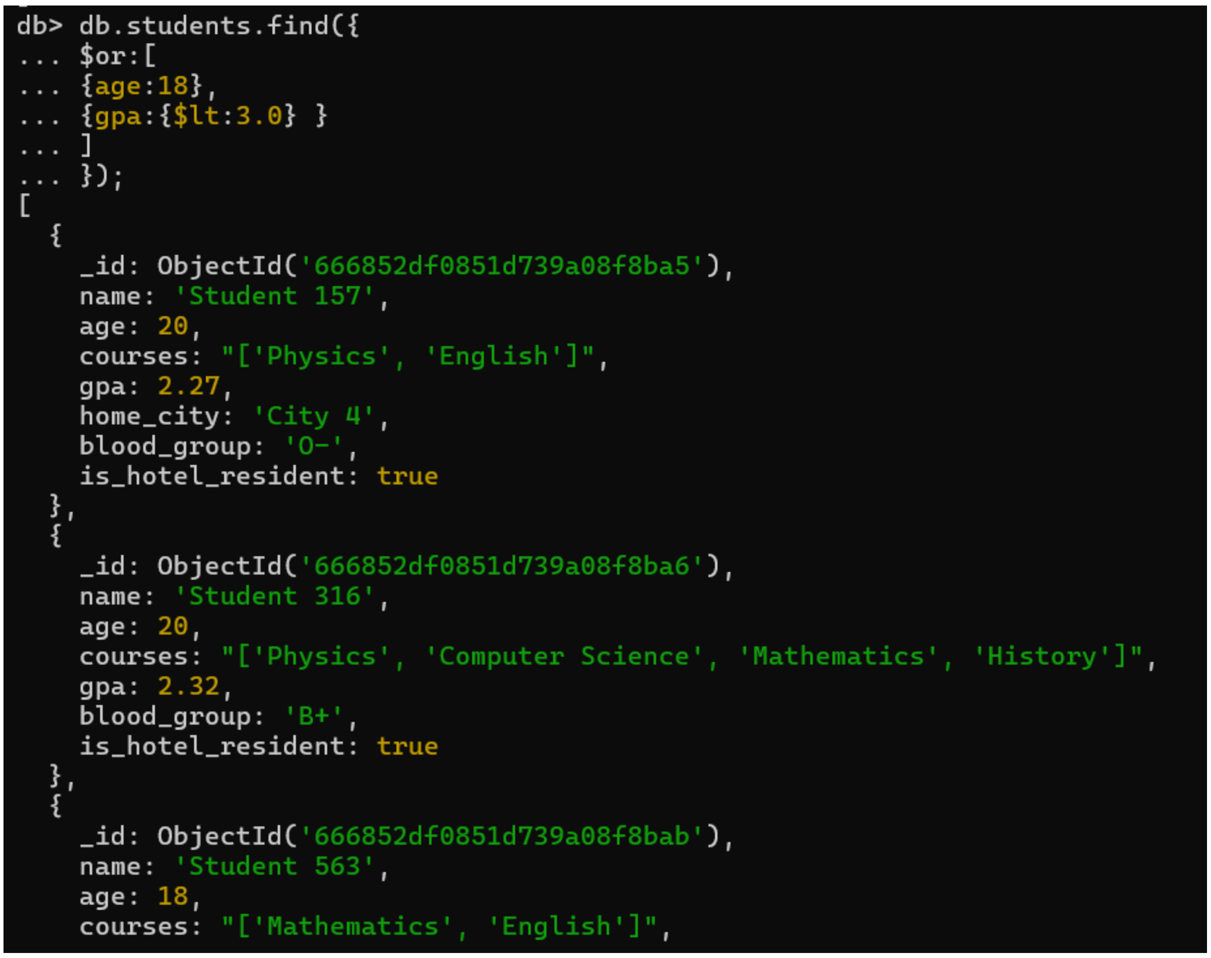
The Collections you want to FILTER a subset based on multiple conditions,this is where we use AND operator.



```
db> db.students.find({
... $and: [
... { home_city:"City 5"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('666852df0851d739a08f8bdb'),
    name: 'Student 142',
    age: 24,
    courses: "['History', 'English', 'Physics', 'Computer Science']",
    gpa: 3.41,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8cfb'),
    name: 'Student 947',
    age: 20,
    courses: "['Physics', 'History', 'English', 'Computer Science']",
    gpa: 2.86,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666852df0851d739a08f8d6d'),
    name: 'Student 567',
    age: 22,
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",
    gpa: 2.01,
    home_city: 'City 5',
    blood_group: 'A+',
```

OR:

The Collection you want to FILTER a subset based on multiple conditions but Any One is Sufficient.



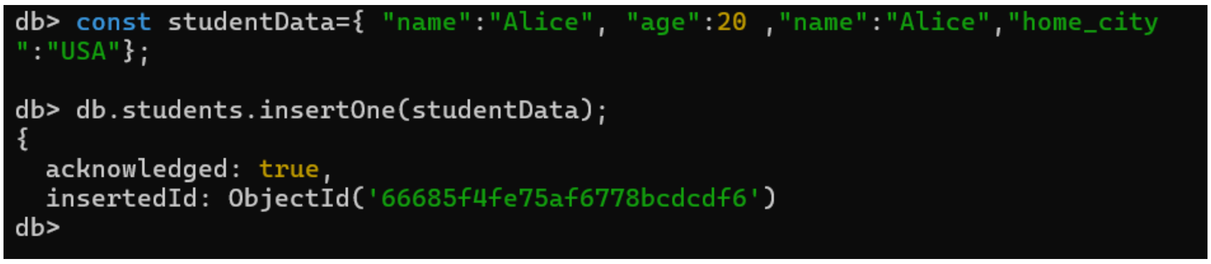
```
db> db.students.find({
... $or:[
... {age:18},
... {gpa:{$lt:3.0} }
... ]
... });
[
  {
    _id: ObjectId('666852df0851d739a08f8ba5'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666852df0851d739a08f8ba6'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666852df0851d739a08f8bab'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
```

CRUD OPERATIONS:

- C - Create / Insert
- R - Remove
- U - update
- D - Delete

INSERT:

Inserts a document or documents into a collection.

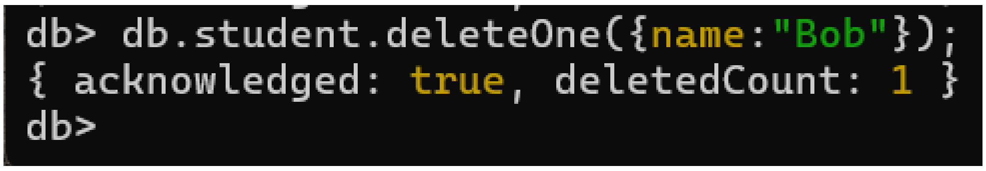


```
db> const studentData={ "name":"Alice", "age":20 ,"name":"Alice","home_city
":"USA"};

db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('66685f4fe75af6778bcdcdf6')
db>
```

DELETE:

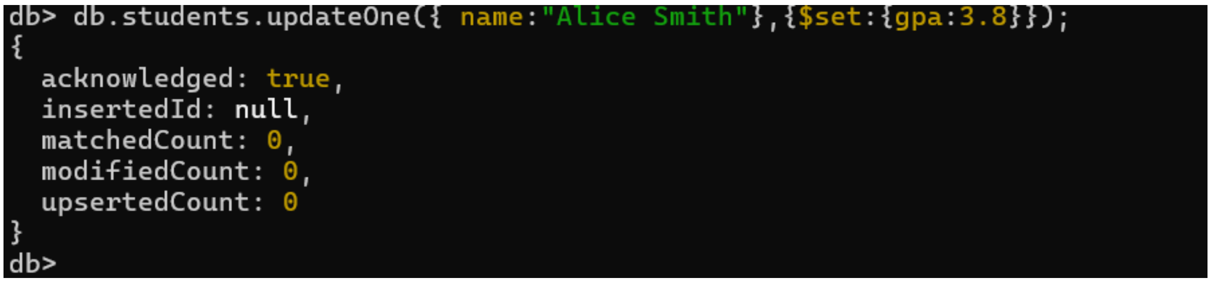
Removes/Deletes a single document from a collection.



```
db> db.student.deleteOne({name:"Bob"});
{ acknowledged: true, deletedCount: 1 }
db>
```

UPDATE:

Modifies an existing document or documents in a collection. The method can modify specific fields of an existing document or documents or replace an existing document entirely, depending on the updated parameter.

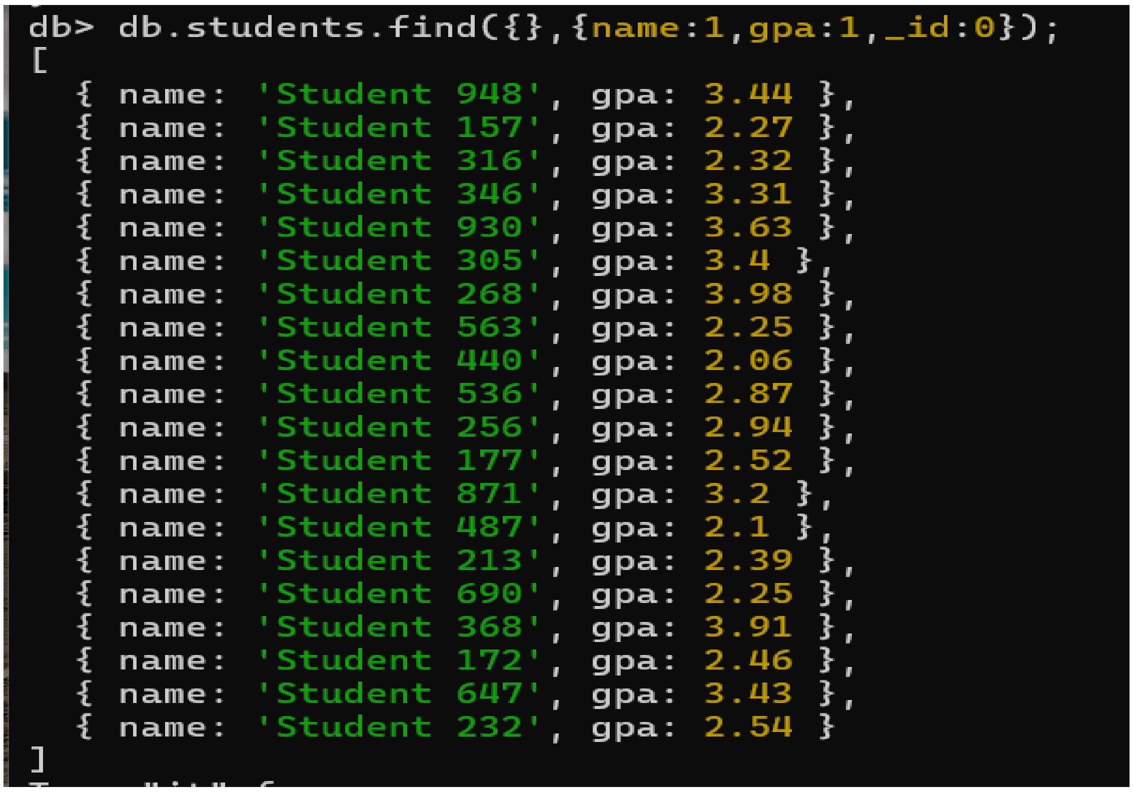


```
db> db.students.updateOne({ name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
db>
```

PROJECTIONS:

It allows you to select only the necessary data rather than selecting whole data from the document.

To get the name and gpa of the students with no object id we use (_id:0), the result is shown below.



```
db> db.students.find({},{name:1,gpa:1,_id:0});
[
  { name: 'Student 948', gpa: 3.44 },
  { name: 'Student 157', gpa: 2.27 },
  { name: 'Student 316', gpa: 2.32 },
  { name: 'Student 346', gpa: 3.31 },
  { name: 'Student 930', gpa: 3.63 },
  { name: 'Student 305', gpa: 3.4 },
  { name: 'Student 268', gpa: 3.98 },
  { name: 'Student 563', gpa: 2.25 },
  { name: 'Student 440', gpa: 2.06 },
  { name: 'Student 536', gpa: 2.87 },
  { name: 'Student 256', gpa: 2.94 },
  { name: 'Student 177', gpa: 2.52 },
  { name: 'Student 871', gpa: 3.2 },
  { name: 'Student 487', gpa: 2.1 },
  { name: 'Student 213', gpa: 2.39 },
  { name: 'Student 690', gpa: 2.25 },
  { name: 'Student 368', gpa: 3.91 },
  { name: 'Student 172', gpa: 2.46 },
  { name: 'Student 647', gpa: 3.43 },
  { name: 'Student 232', gpa: 2.54 }
]
```

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.

SLICING FROM NESTED ARRAYS:

MongoDB provides different types of array update operators to update the values of the array fields in the documents and $slice modifier is one of them.

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

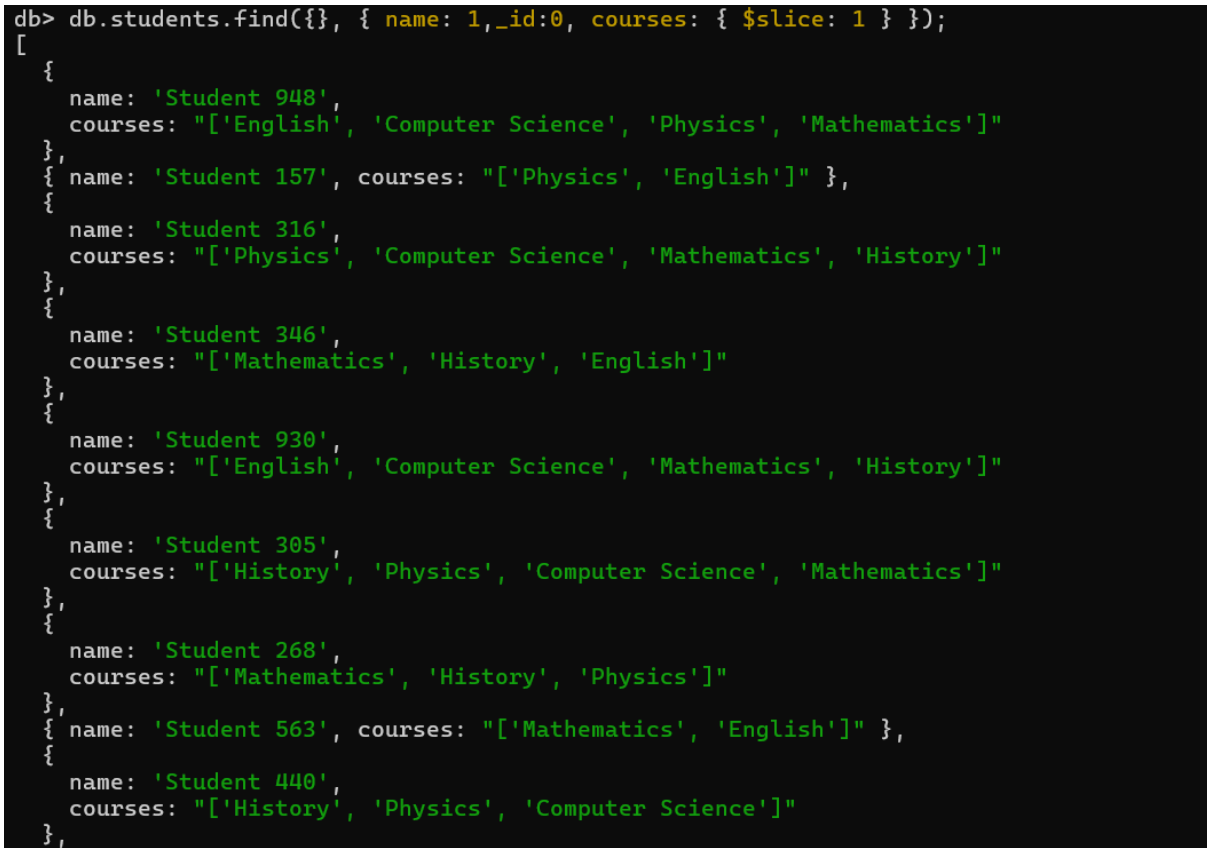
Syntax:

db.collection.find(

  <query>,

  { <arrayField>: { $slice: <number> } }

);



```
db> db.students.find({}, { name: 1,_id:0, courses: { $slice: 1 } });
[
  {
    name: 'Student 948',
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']"
  },
  { name: 'Student 157', courses: "['Physics', 'English']" },
  {
    name: 'Student 316',
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']"
  },
  {
    name: 'Student 346',
    courses: "['Mathematics', 'History', 'English']"
  },
  {
    name: 'Student 930',
    courses: "['English', 'Computer Science', 'Mathematics', 'History']"
  },
  {
    name: 'Student 305',
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']"
  },
  {
    name: 'Student 268',
    courses: "['Mathematics', 'History', 'Physics']"
  },
  { name: 'Student 563', courses: "['Mathematics', 'English']" },
  {
    name: 'Student 440',
    courses: "['History', 'Physics', 'Computer Science']"
  },
```

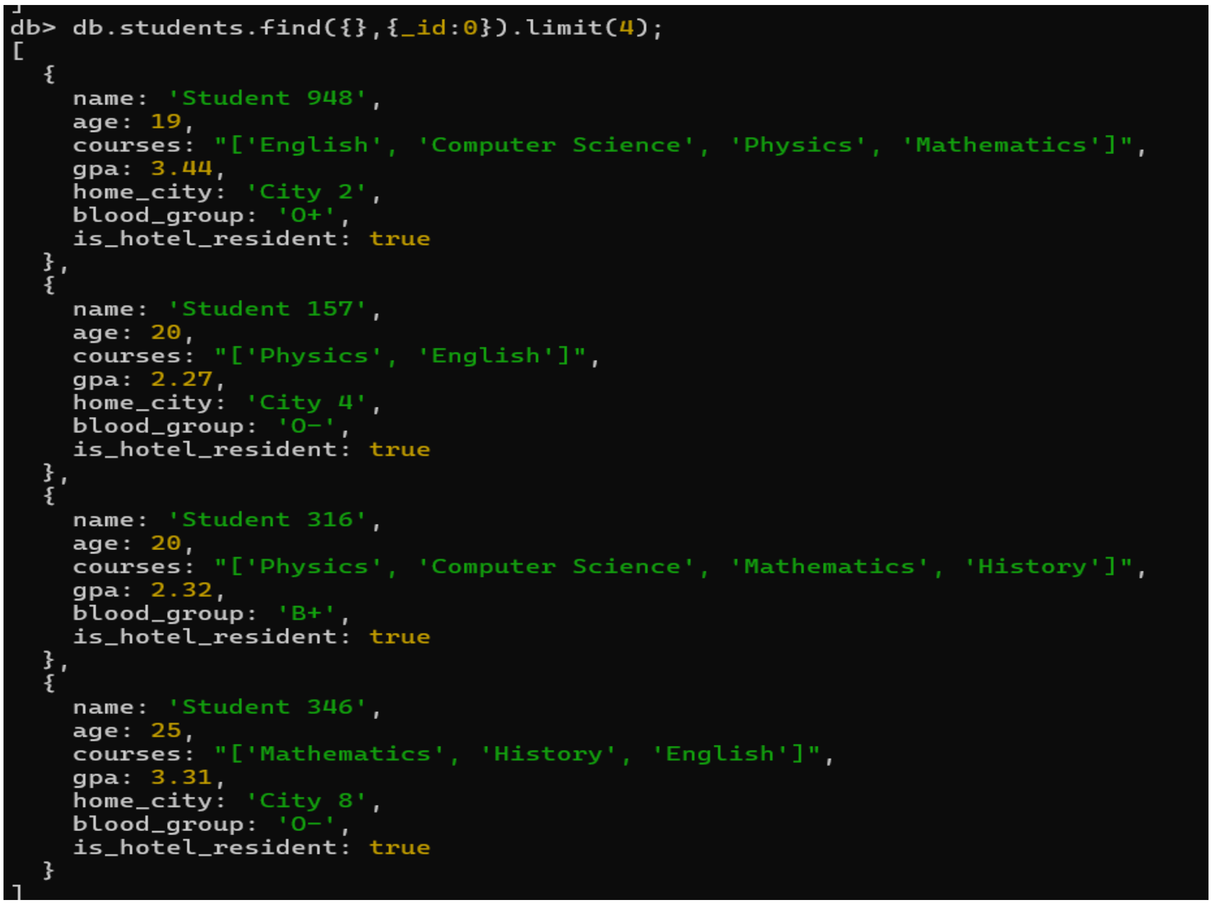
LIMIT:

1.The limit operator is used with the find method.

2.It's chained after the filter criteria or any sorting operations.

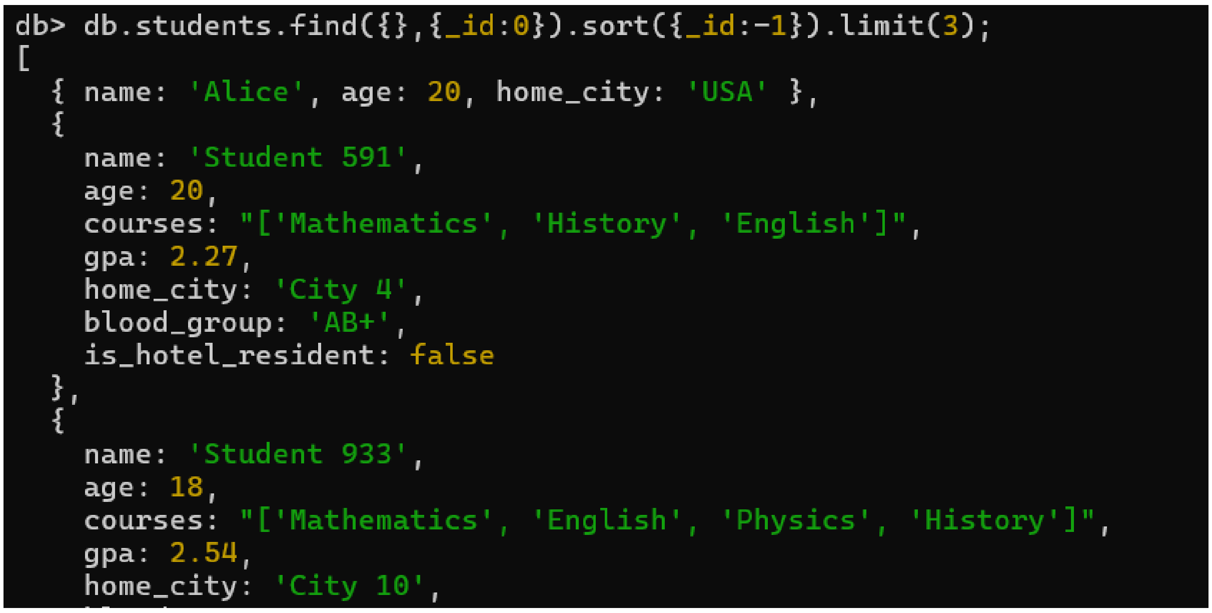
Syntax: db.collection.find({filter}, {projection}).limit(number)

For getting first 4 documents we use the below command:



```
db> db.students.find({},{_id:0}).limit(4);
[
  {
    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
  },
  {
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  }
]
```

For getting the top 10 results we use the following command:

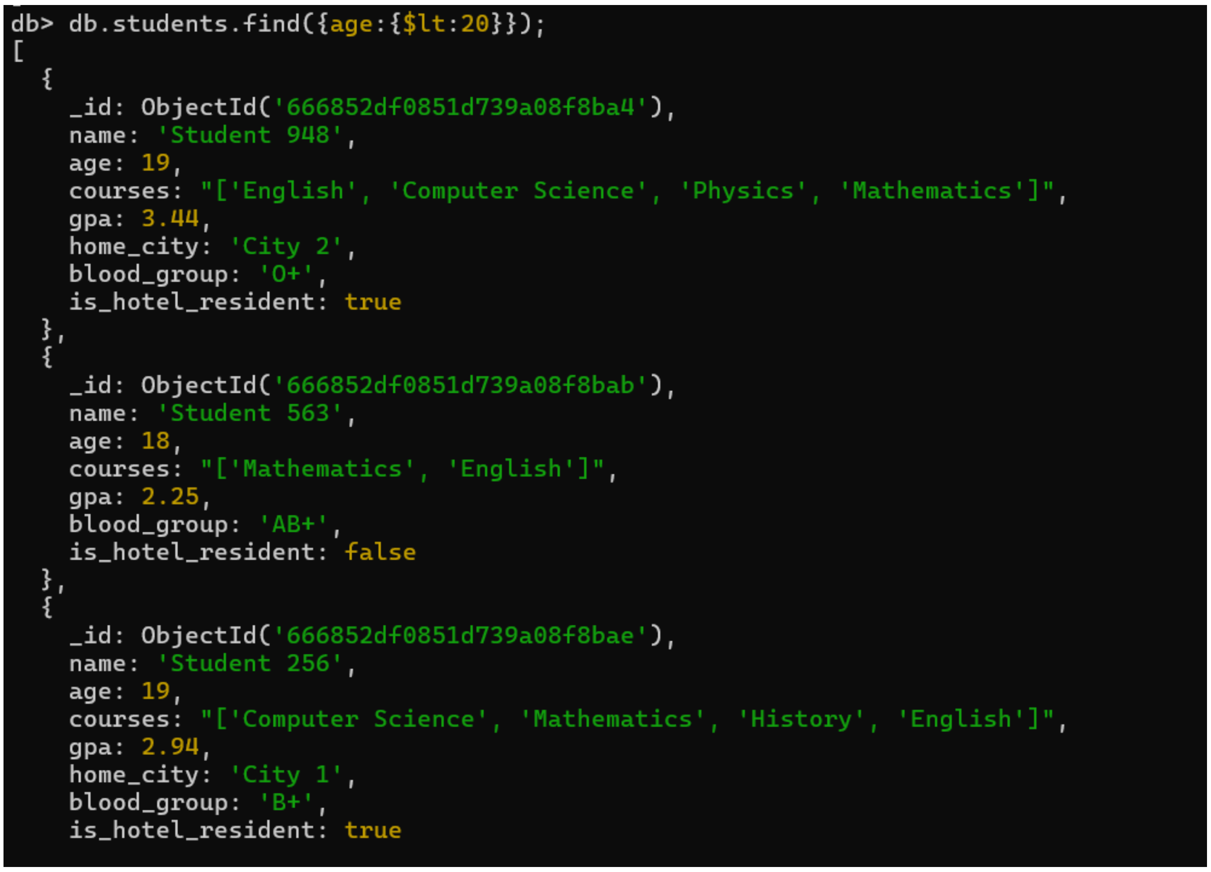


```
db> db.students.find({},{_id:0}).sort({_id:-1}).limit(3);
[
  { name: 'Alice', age: 20, home_city: 'USA' },
  {
    name: 'Student 591',
    age: 20,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    name: 'Student 933',
    age: 18,
    courses: "['Mathematics', 'English', 'Physics', 'History']",
    gpa: 2.54,
    home_city: 'City 10',
```

SELECTORS:

Comparing the greater than and less than operators by using the operators such as $gt,$lt.

In the below example we got data of the students age less than 20.



```
db> db.students.find({age:{$lt:20}});
[
  {
    _id: ObjectId('666852df0851d739a08f8ba4'),
    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('666852df0851d739a08f8bab'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8bae'),
    name: 'Student 256',
    age: 19,
    courses: "['Computer Science', 'Mathematics', 'History', 'English']",
    gpa: 2.94,
    home_city: 'City 1',
    blood_group: 'B+',
    is_hotel_resident: true
```

$gt=greater than

$lt=lesser than

BITWISE TYPES

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`.

```
}
```

GEOSPATIAL:

In MongoDB, you can store geospatial data as GeoJSON objects or as legacy coordinate pairs.

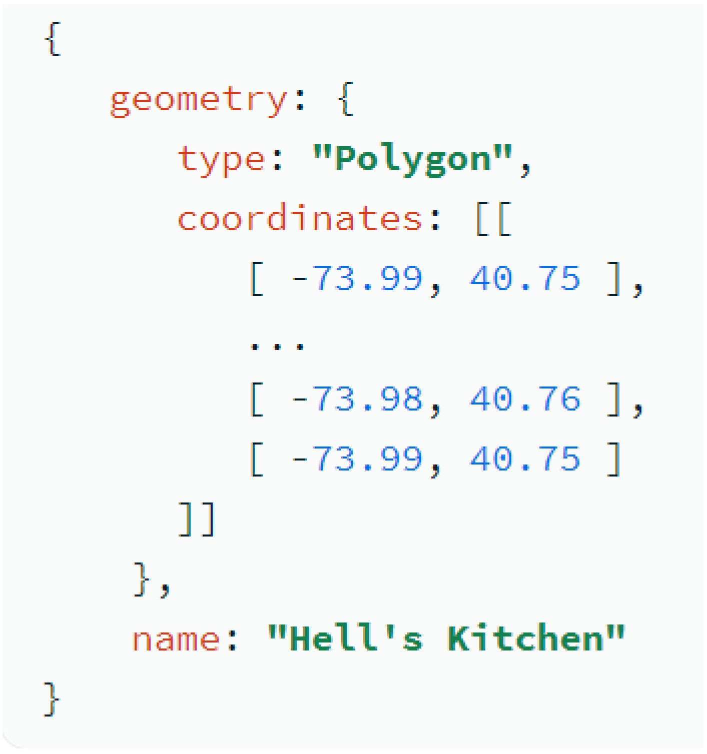
To specify GeoJSON data, use an embedded document with:

- a field named type that specifies the GeoJSON object type and
- a field named coordinates that specifies the object's coordinates.
  If specifying latitude and longitude coordinates, list the **longitude** first and then **latitude**:
  - Valid longitude values are between -180 and 180, both inclusive.
  - Valid latitude values are between -90 and 90, both inclusive.

Example:

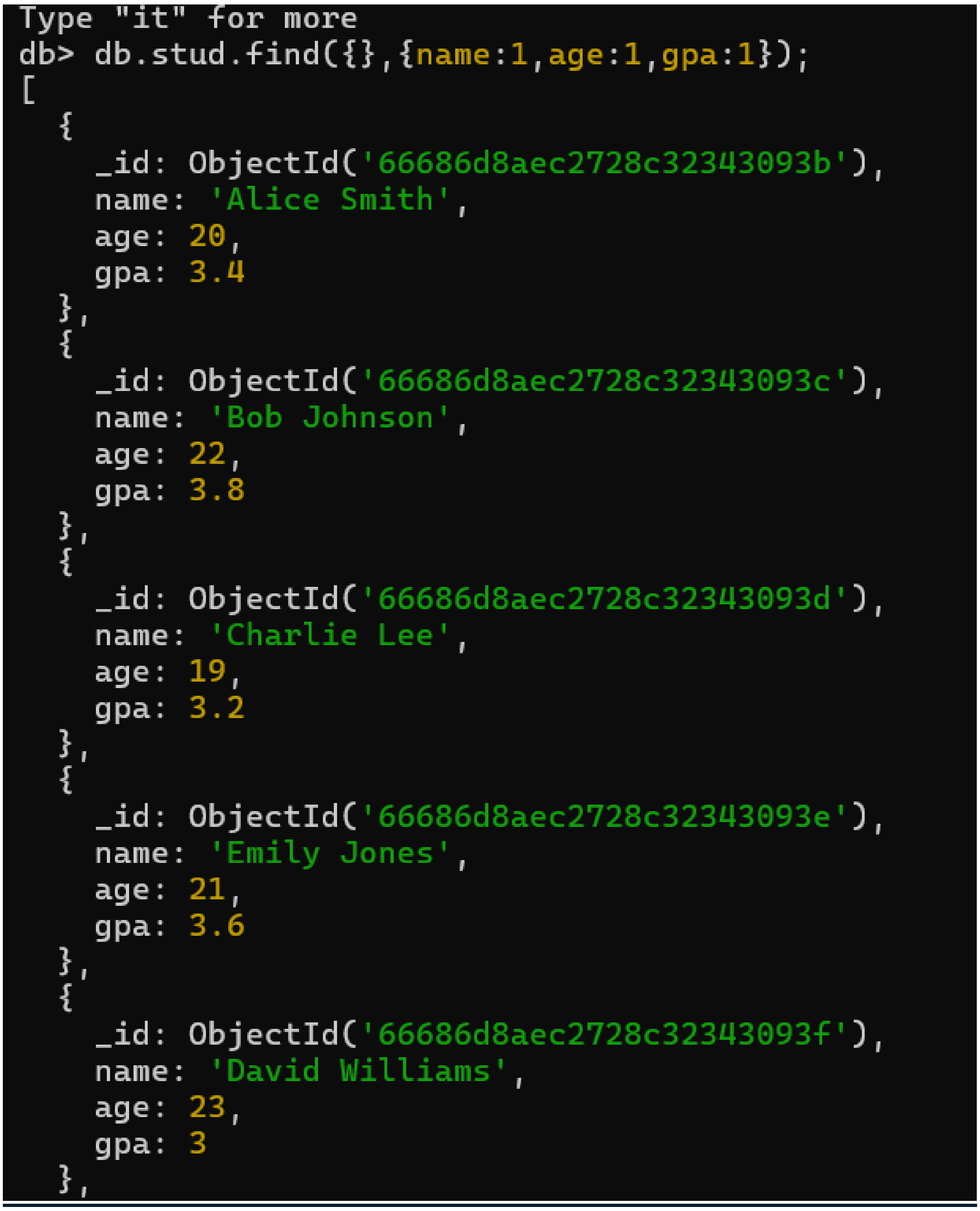
```
db.neighborhoods.findOne()
```

This query will return a document like the following:



```
{
   geometry: {
      type: "Polygon",
      coordinates: [[
         [ -73.99, 40.75 ],
         ...
         [ -73.98, 40.76 ],
         [ -73.99, 40.75 ]
      ]]
    },
    name: "Hell's Kitchen"
}
```

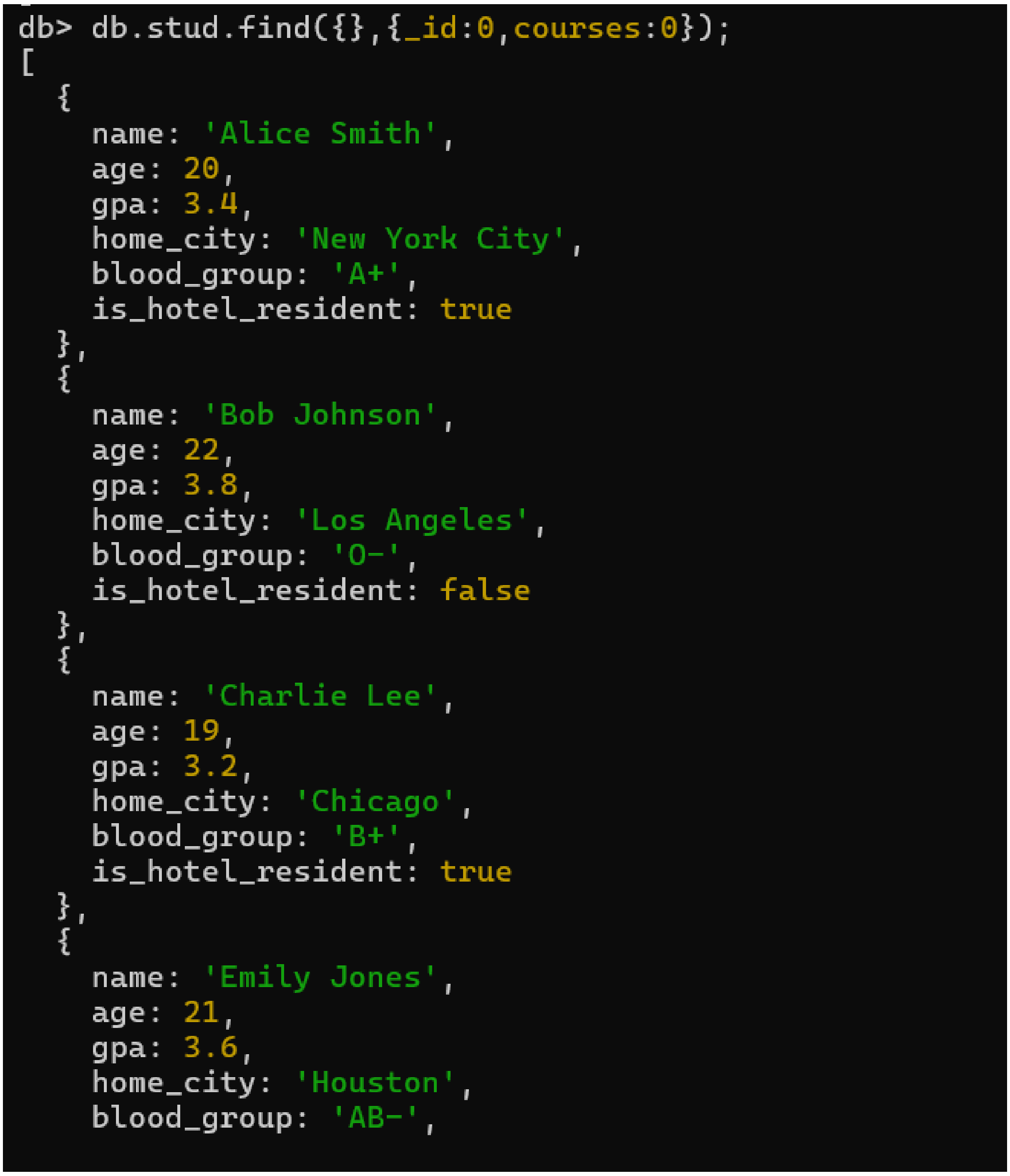
RETRIEVING NAME,AGE,GPA USING PROJECTIONS:



```
Type "it" for more
db> db.stud.find({},{name:1,age:1,gpa:1});
[
  {
    _id: ObjectId('66686d8aec2728c32343093b'),
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4
  },
  {
    _id: ObjectId('66686d8aec2728c32343093c'),
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8
  },
  {
    _id: ObjectId('66686d8aec2728c32343093d'),
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2
  },
  {
    _id: ObjectId('66686d8aec2728c32343093e'),
    name: 'Emily Jones',
    age: 21,
    gpa: 3.6
  },
  {
    _id: ObjectId('66686d8aec2728c32343093f'),
    name: 'David Williams',
    age: 23,
    gpa: 3
  },
```

In the above example we have retrieved the data of the following candidates name,age,gpa with parameter 1 for getting it i.e true and 0 for false for getting the data.

EXCLUDING THE FIELDS IN THE DATA:



```
db> db.stud.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-',
```

In the above example we are not getting the data i.e of object id and courses such that the following parameters are given by the value 0.