CLASS 2

ADD, UPDATE AND DELETE

ADD Operation:

The $add operator is used to perform an addition operation on numbers, dates, or arrays. It is commonly used in aggregation pipelines or within an update operation to increment values.

Syntax:

json

{ $add:

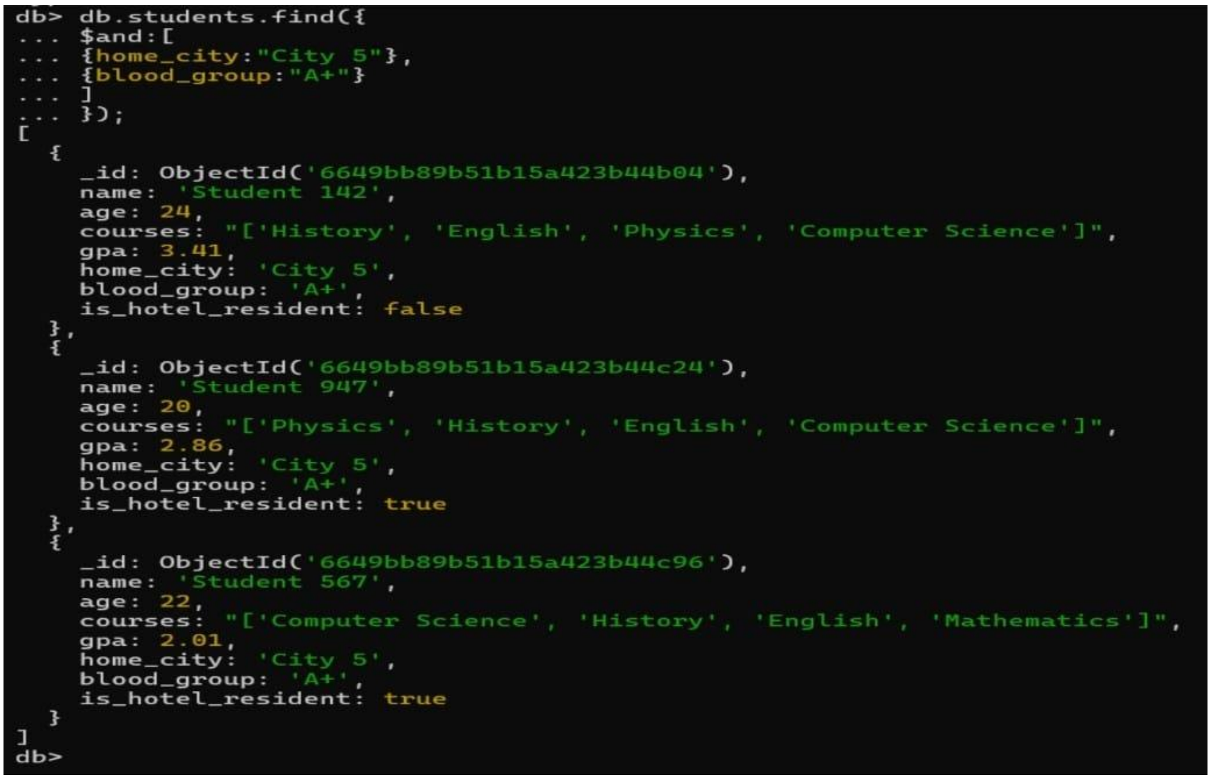
[<expression1>,

<expression2>,

...

] }

Example:



```
db> db.students.find({
... $and:[
... {home_city:"City 5"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44b04'),
    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('6649bb89b51b15a423b44c24'),
    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('6649bb89b51b15a423b44c96'),
    name: 'Student 567',
    age: 22,
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",
    gpa: 2.01,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
db>
```

UPDATE OPREATION:

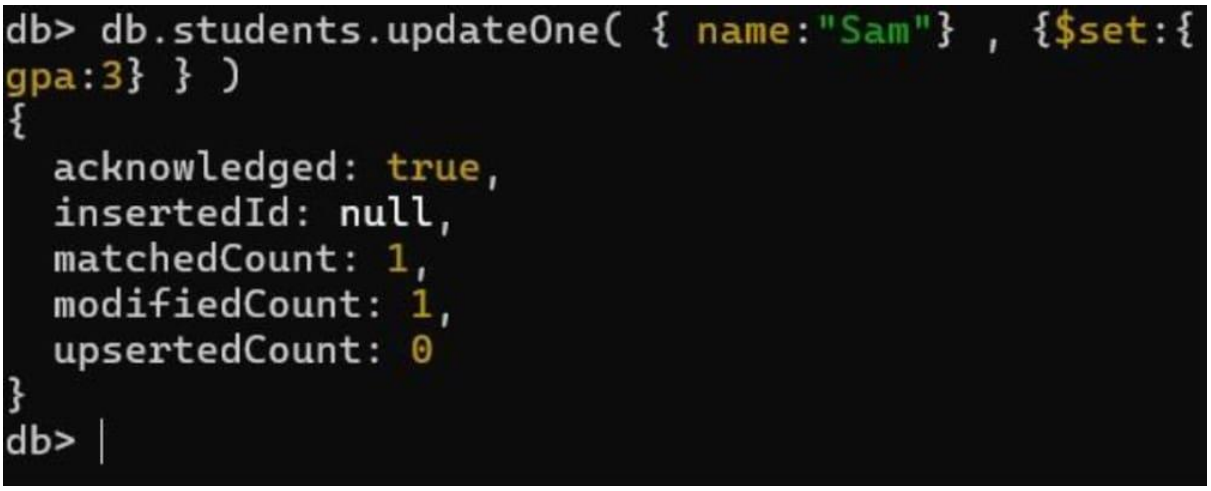
The update operation in MongoDB is used to modify documents in a collection. The operation can update specific fields, add new fields, or even remove fields in the matched documents.

Syntax:

```
db.collection.updateMany(
 <filter>,
<update>,
 <options>
 )
```

Example:

using sets



```
db> db.students.updateOne( { name:"Sam"} , {$set:{
gpa:3} } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
db> |
```

DELETE:

The delete operation in MongoDB is used to remove documents from a collection. This can be done using methods such as 'deleteOne' or 'deleteMany', depending on whether you want to delete a single document or multiple documents that match a given filter.

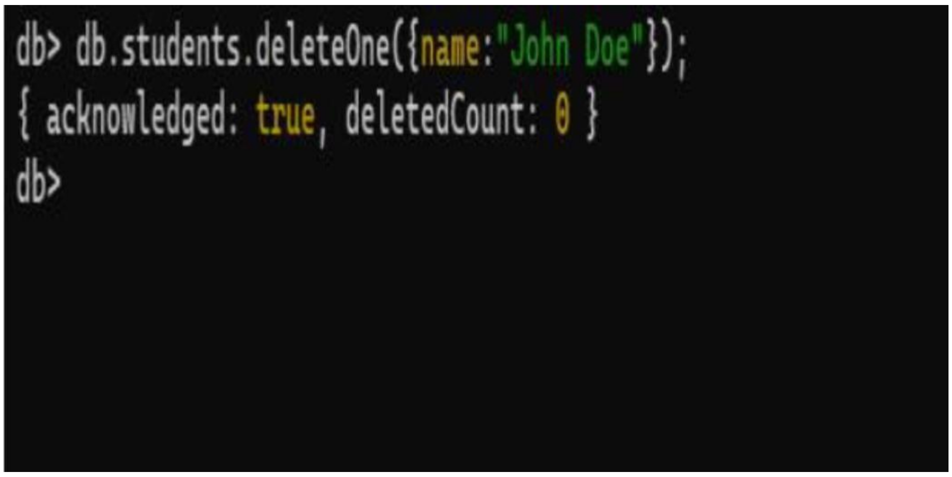
Syntax:

Basic syntax of **remove()** method is as follows –

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

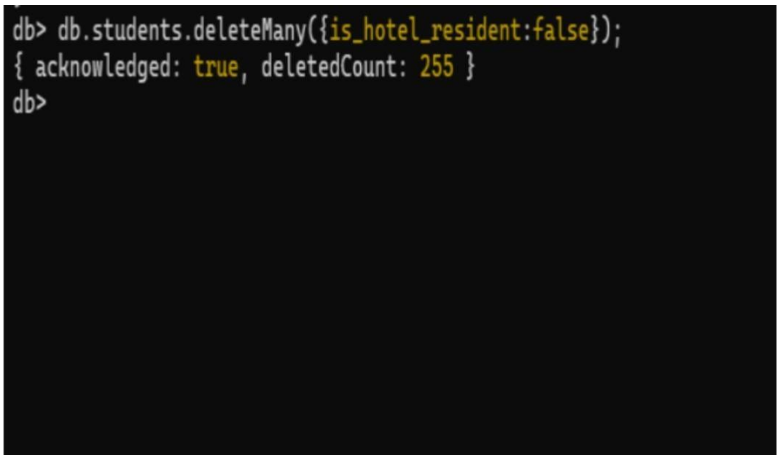
Example:

Deleting a single document:



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

DeleteMany :



```
db> db.students.deleteMany({is_hotel_resident:false});
{ acknowledged: true, deletedCount: 255 }
db>
```

DOCUMENTS, COLLECTIONS

Documents:

In MongoDB, the data records are stored as BSON documents. Here, BSON stands for binary representation of JSON documents, although BSON contains more data types as compared to JSON. The document is created using field-value pairs or key-value pairs and the value of the field can be of any BSON type.

Syntax:

```
{
field1: value1
field2: value2
....
fieldN: valueN
}
```

Collections:

Collections are just like tables in relational databases, they also store data, but in the form of documents. A single database is allowed to store multiple collections.

Syntax:

```
db.collection_name.insertOne({..})
```

CLASS 3

WHERE, CRUD

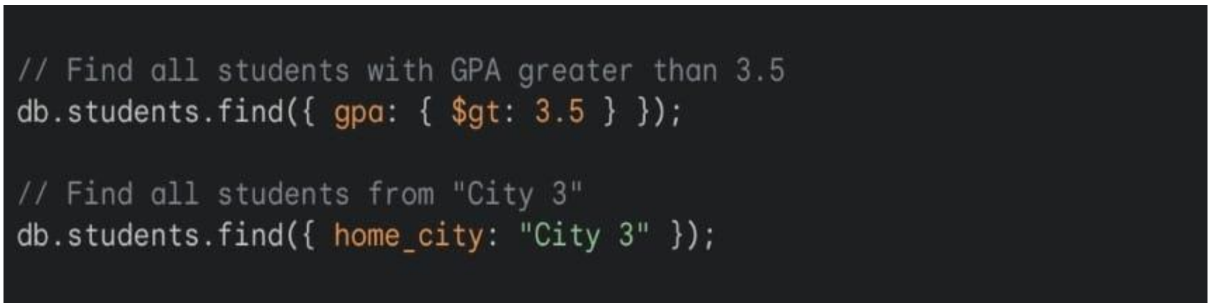
Where:

$where operator allows you to execute JavaScript code directly within a query. This operator can be used to query documents based on criteria that cannot be expressed using the regular MongoDB query language syntax.

Syntax:

```
db.collection.find({
  $where: function() {
    // JavaScript code to evaluate
  }
});
```

Example:



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

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

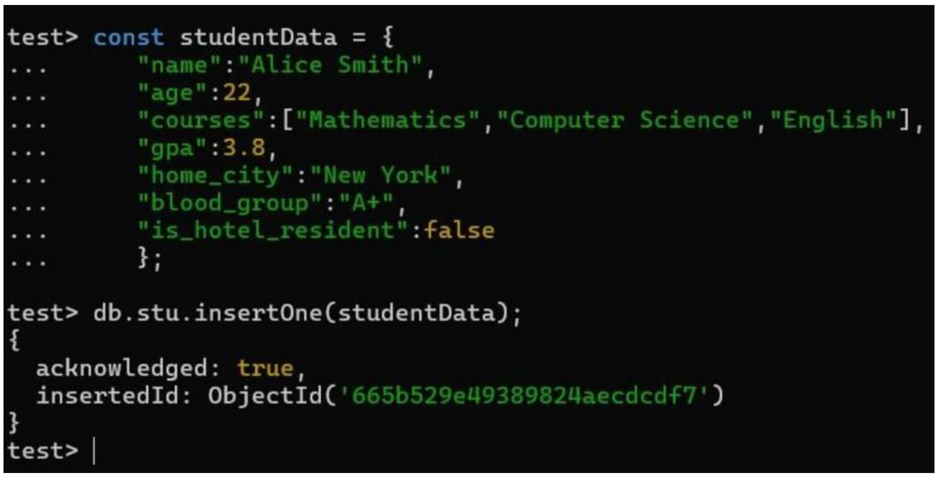
CRUD:

(Create, Read, Update, and Delete) are the basic set of operations that allow users to interact with the MongoDB server. As we know, to use MongoDB we need to interact with the MongoDB server to perform certain operations like entering new data into the application, updating data into the application, deleting data from the application, and reading the application data.

$$\begin{tikzcd} C \arrow[r] & \text{Create} \\ R \arrow[r] & \text{Read} \\ U \arrow[r] & \text{Update} \\ D \arrow[r] & \text{Delete} \end{tikzcd}$$

1. Create (Insert)

To add new documents to a collection.



```
test> 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
...       };

test> db.stu.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('665b529e49389824aecdcdf7')
}
test> |
```

CLASS 4

PROJECTION, LIMITS, SELECTOR

Projection:

In MongoDB refers to the process of selecting which fields to include or exclude in the result set when querying a collection. This can help optimize performance and reduce the amount of data sent over the network by only returning necessary fields.

The projection work based on:

1 indicates that the field should be included name to be returned. 0 indicates that the field should be excluded them from the result

Example:

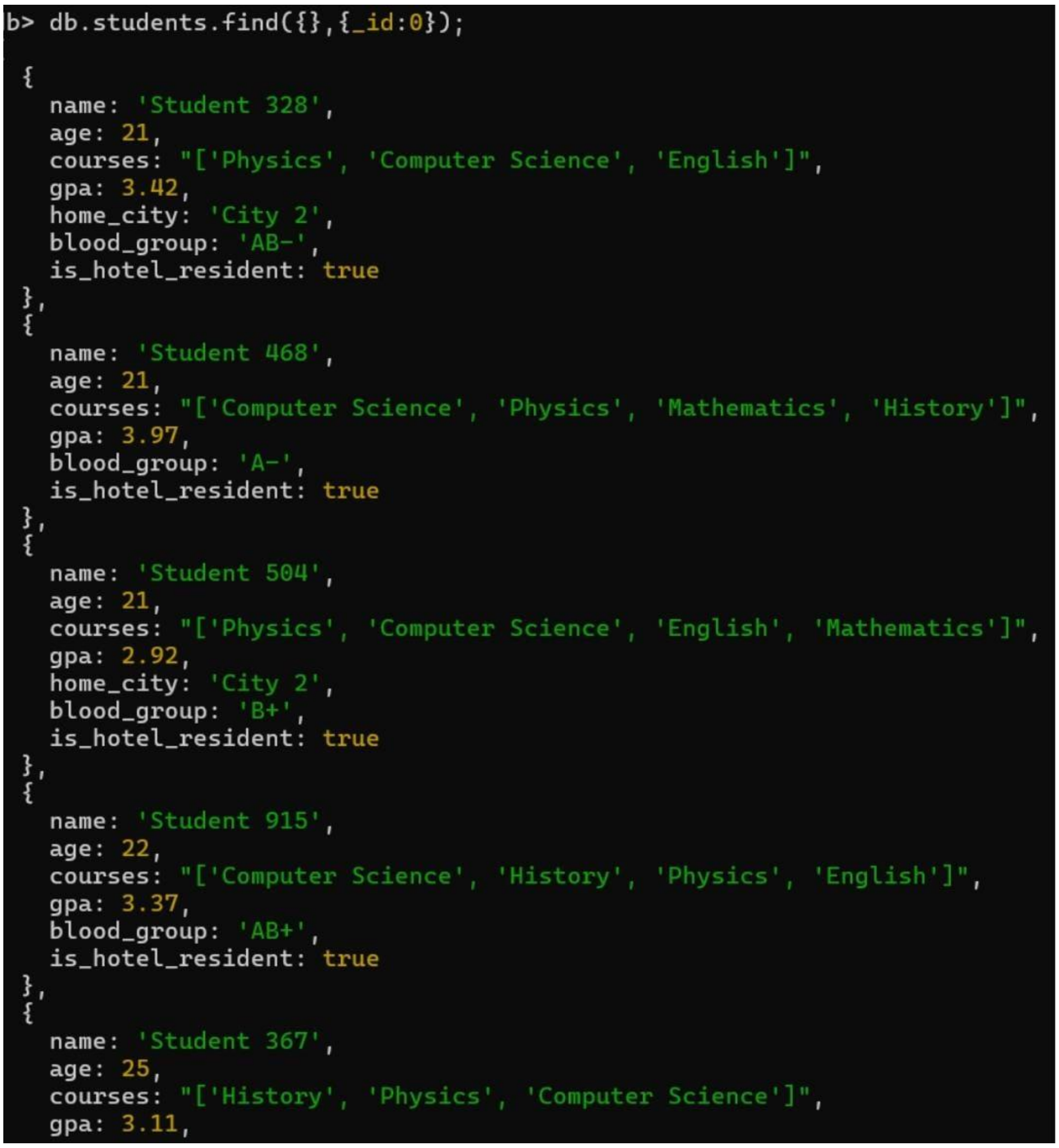
```
{
 "_id": 1,
"name": "Alice",
 "age": 30,
"email": "alice@example.com",
 "address": {
 "city":
"New York",
"zipcode": "10001"
 }
}
```

Get Selected Attributes:

To select specific fields in a MongoDB query, you use the find() method with a projection document. The projection document specifies which fields to include (using 1) or exclude (using 0).

Syntax:

```
db. students. find ({}, { _id: 0 })
```



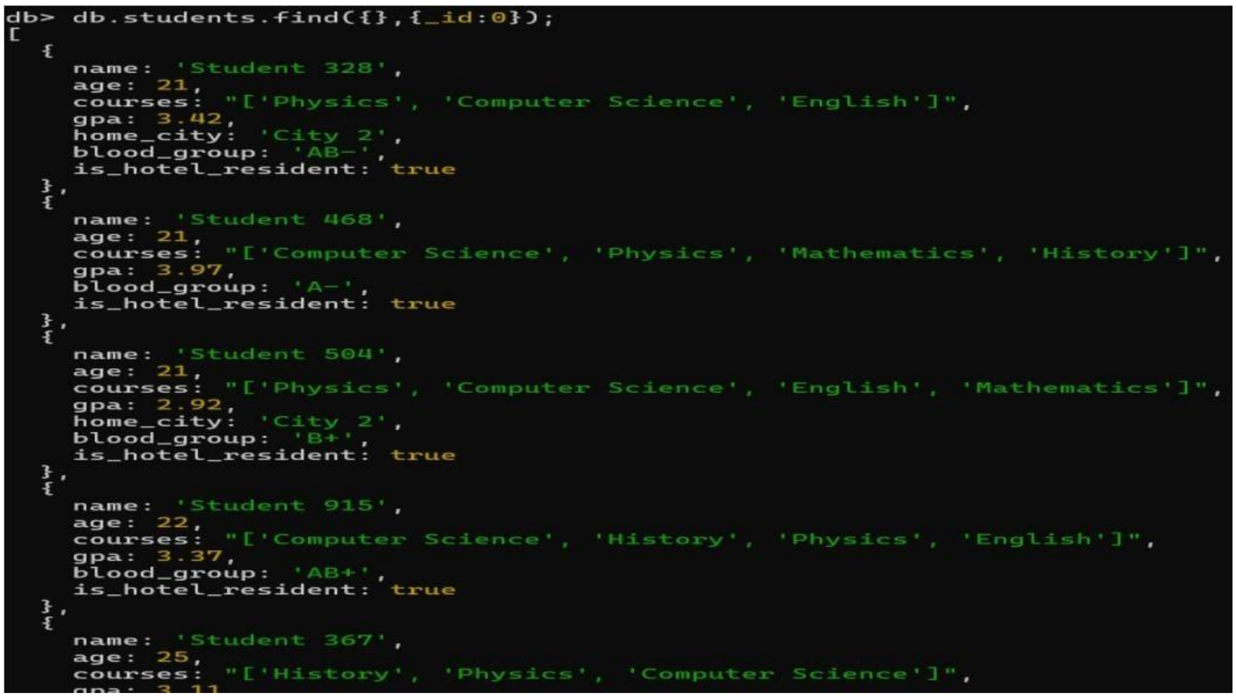
```
b> db.students.find({},{_id:0});

 {
   name: 'Student 328',
   age: 21,
   courses: "['Physics', 'Computer Science', 'English']",
   gpa: 3.42,
   home_city: 'City 2',
   blood_group: 'AB-',
   is_hotel_resident: true
 },
 {
   name: 'Student 468',
   age: 21,
   courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
   gpa: 3.97,
   blood_group: 'A-',
   is_hotel_resident: true
 },
 {
   name: 'Student 504',
   age: 21,
   courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
   gpa: 2.92,
   home_city: 'City 2',
   blood_group: 'B+',
   is_hotel_resident: true
 },
 {
   name: 'Student 915',
   age: 22,
   courses: "['Computer Science', 'History', 'Physics', 'English']",
   gpa: 3.37,
   blood_group: 'AB+',
   is_hotel_resident: true
 },
 {
   name: 'Student 367',
   age: 25,
   courses: "['History', 'Physics', 'Computer Science']",
   gpa: 3.11,
```

Ignore attributes:

In MongoDB, ignoring an attribute (or field) in query results is done using projection. By setting the value of a field to 0 in the projection document, you can exclude that field from the returned documents.

```
Db. students. find({},{_id:0});
```



```
db> db.students.find({},{_id:0});
[
  {
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
    name: 'Student 367',
    age: 25,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 3.11
```

Limits:

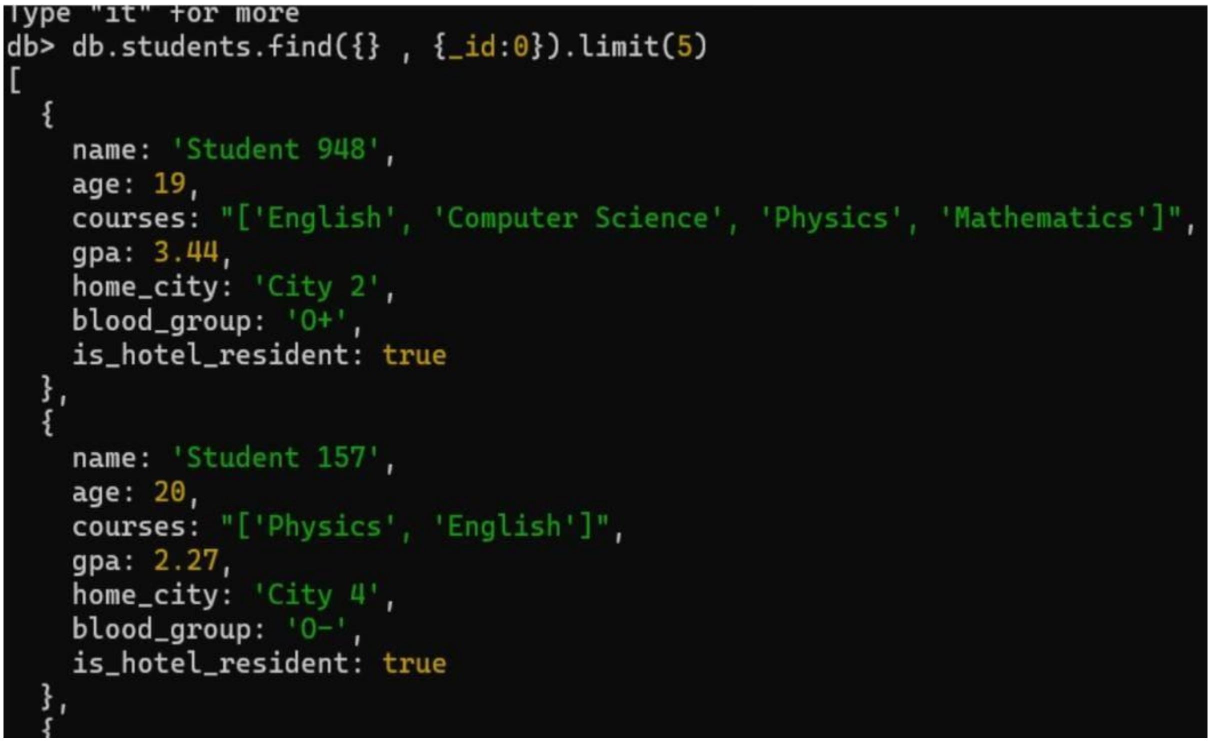
In MongoDB, the **limit()** method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want.

Syntax:

db.collectionName.find(<query>).limit(<number>)

Coding:

Get 5 documents



```
Type "it" for more
db> db.students.find({} , {_id:0}).limit(5)
[
  {
    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
  },
  {
```

Selector:

- AND operation
- OR operation
- Comparison between gt and lt:

Grater than:

The $gt operator in MongoDB is used to specify a query condition where the field value must be greater than a specified value. It stands for "greater than."

Lesser than:

the less than operation is performed using the $lt operator. This operator allows you to query documents where a specific field's value is less than a given value.

Greater Than ($gt):

• **Functionality:** The $gt operator selects documents where the value of a specified field is greater than (but not equal to) a given value.

• **Example:** db. collection. find({ field: { $gt: value } }) selects documents where the field value is greater than value.

• **Use Cases:** Filtering documents with values exceeding a certain threshold, selecting the latest entries based on a timestamp, or retrieving documents with numerical values above a specific limit.

Less Than ($lt):

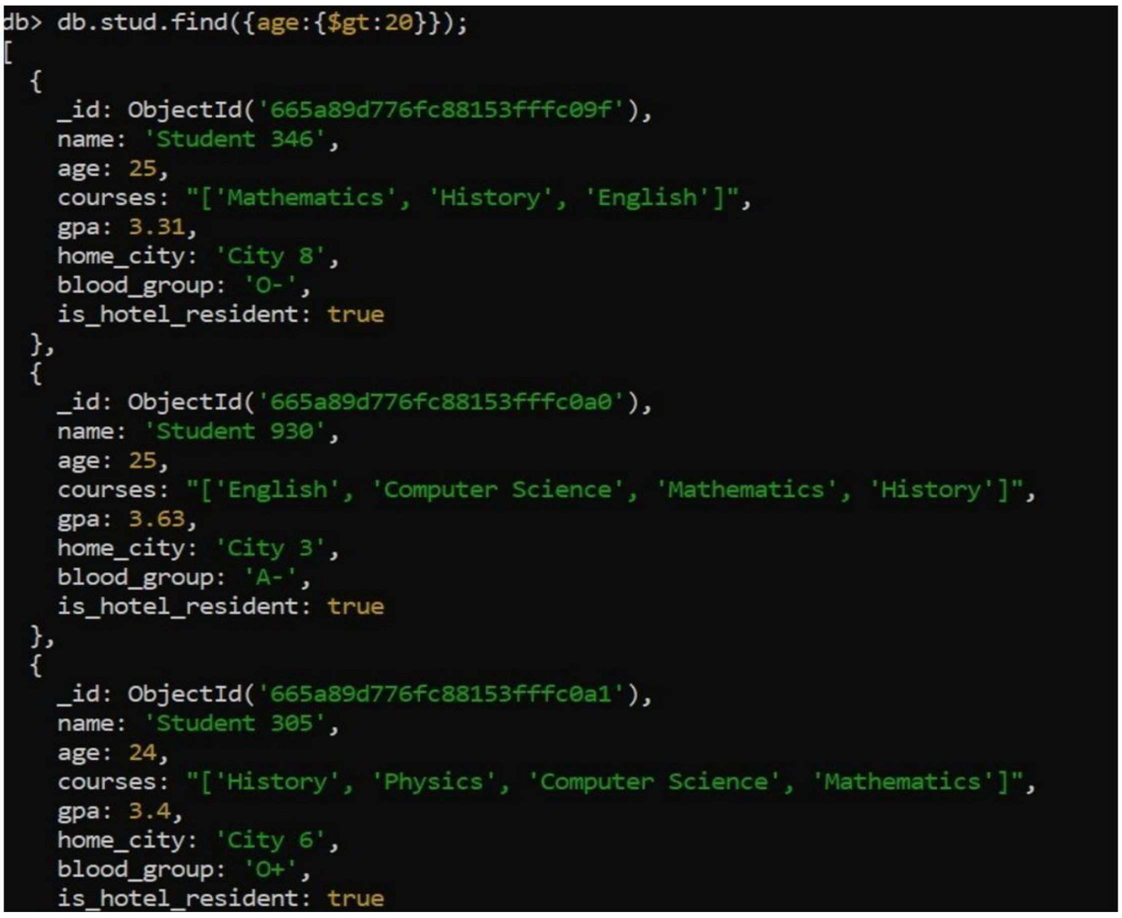
• **Functionality**: The $lt operator selects documents where the value of a specified field is less than (but not equal to) a given value.

• **Example:**

db. collection. find({ field: { $lt: value } }) selects documents where the field value is less than value.

• **Use Cases**: Filtering documents with values below a certain threshold, selecting older entries based on a timestamp, or retrieving documents with numerical values below a specific limit.

Example:db. students. find({age:{$gt:20} });



```
db> db.stud.find({age:{$gt:20}});
[
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a1'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
```