MongoDB®

INTRODUCTION

What is MongoDB?

MongoDB is a source-available, cross-platform, document-oriented database program. Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and current versions are licensed under the Server Side Public License (SSPL). MongoDB is a member of the MACH Alliance.

What is Database?

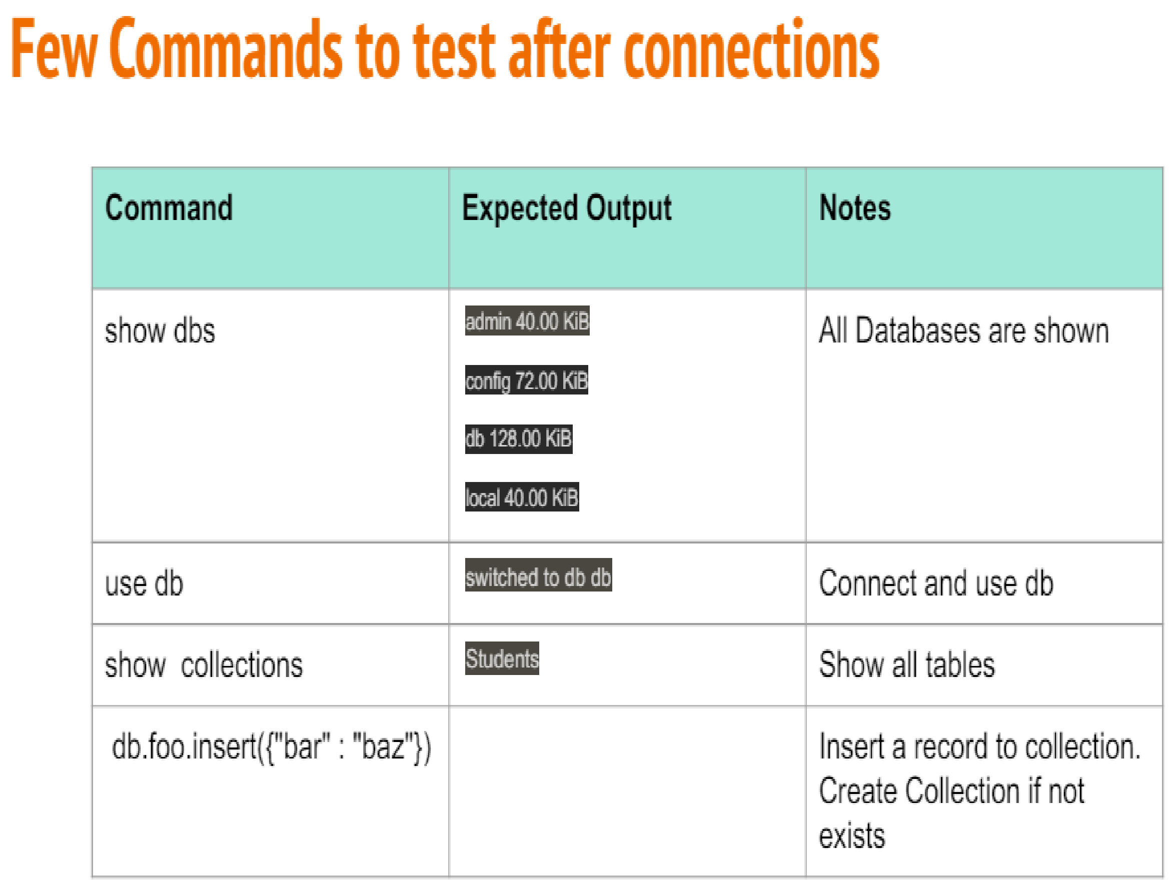
In computing, a database is an organized collection of data or a type of data store based on the use of a database management system (DBMS), the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a database systems.

SET UP:

https://www.geeksforgeeks.org/how-to-install-mongodb-on-windows/?ref=ml_lbp

DATABASE MANAGEMENT SYSTEMS

Database Management Systems are software systems used to store, retrieve, and run queries on data. A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.



Few Commands to test after connections

Command	Expected Output	Notes
show dbs	admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB	All Databases are shown
use db	switched to db db	Connect and use db
show collections	Students	Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists

DOCUMENTS, COLLECTIONS AND DATATYPES

DOCUMENTS

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

```
{"greetings":" Hello World!"}
```

COLLECTIONS

A Collection is a group of documents . MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

DATATYPES

Basically each document will be stored in JSON format , where each attributes is of multiple data types

```
{
"name":  "Edward Bill",
"skills":  "Software Development",
"salary":  7500,
"status":  true,
}
```

Experiment 1

WHERE, AND, OR & CRUD

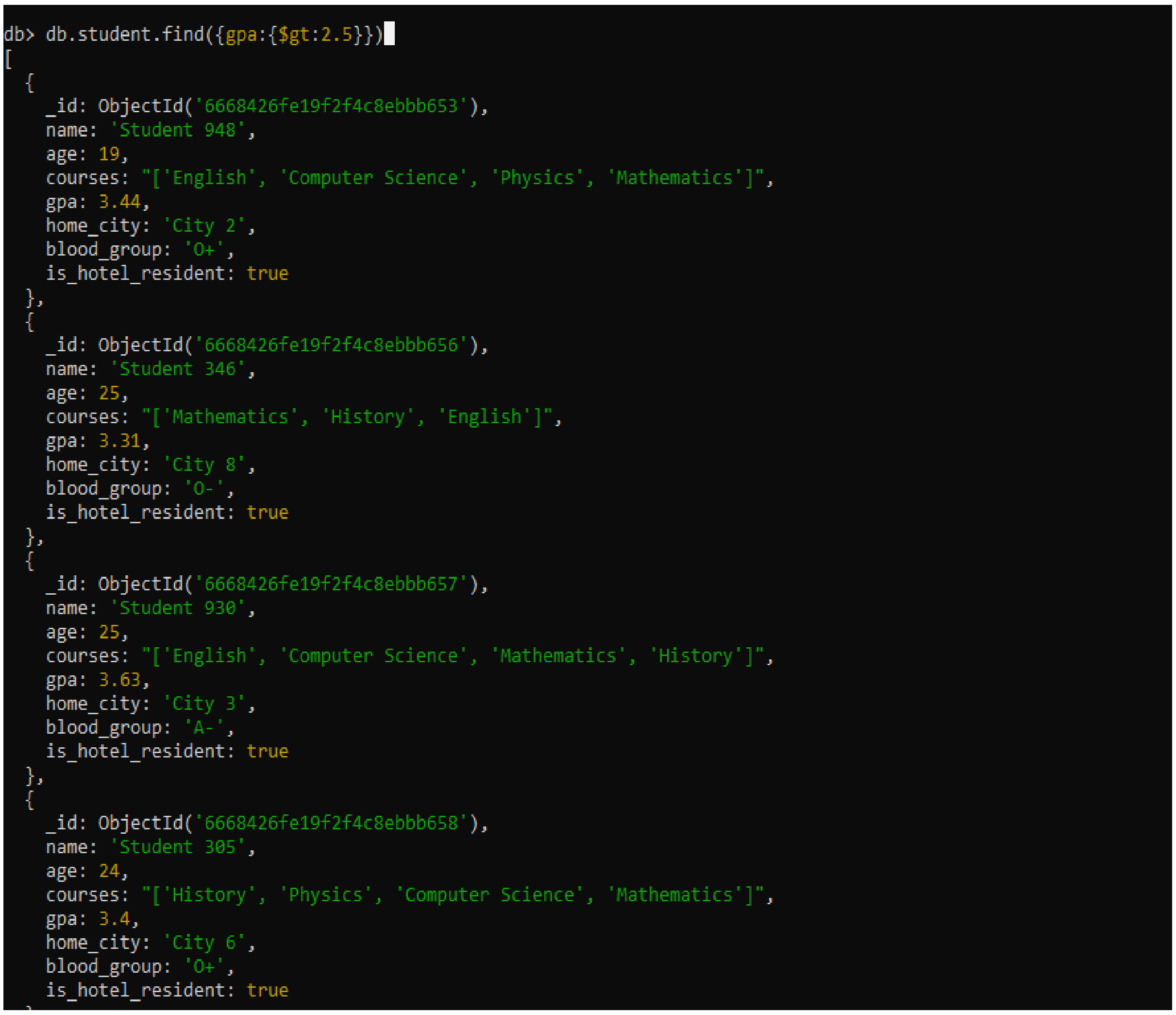
WHERE

● Use the $where operator to pass either a string containing a JavaScript expression or a full JavaScript function to the query system. The $where provides greater flexibility, but requires that the database processes the JavaScript expression or function for each document in the collection.

Example:

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

Output:



```
db> db.student.find({gpa:{$gt:2.5}})
[
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb653'),
    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('6668426fe19f2f4c8ebbb656'),
    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('6668426fe19f2f4c8ebbb657'),
    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('6668426fe19f2f4c8ebbb658'),
    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
```

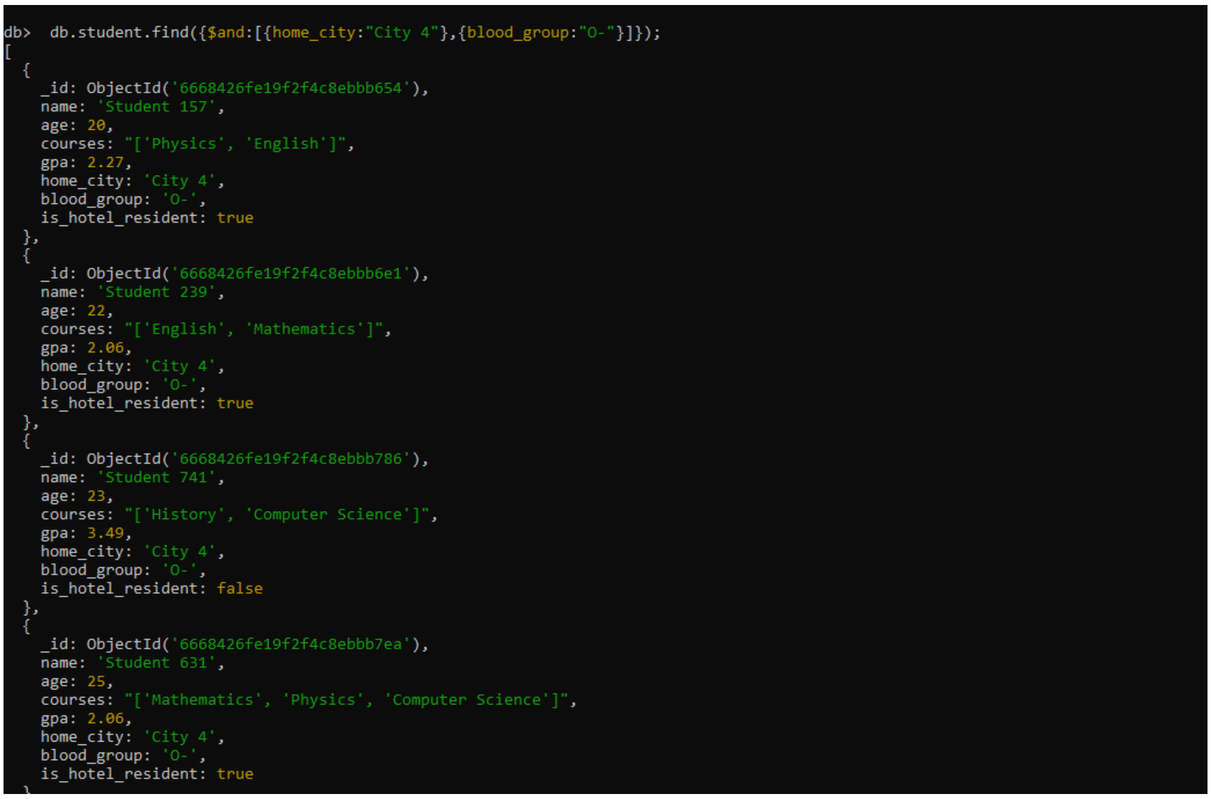
AND

● $and performs a logical AND operation on an array of one or more expressions (<expression1>, <expression2>, and so on) and selects the documents that satisfy all the expressions.

Example:

```
      db.student.find({
       $and:[
       { home_city: “Ciyt 4”},
       { blood_group : “O-“ }
    ]
     });
```

Output:



```
db>  db.student.find({$and:[{home_city:"City 4"},{blood_group:"O-"}]});
[
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb654'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb6e1'),
    name: 'Student 239',
    age: 22,
    courses: "['English', 'Mathematics']",
    gpa: 2.06,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb786'),
    name: 'Student 741',
    age: 23,
    courses: "['History', 'Computer Science']",
    gpa: 3.49,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb7ea'),
    name: 'Student 631',
    age: 25,
    courses: "['Mathematics', 'Physics', 'Computer Science']",
    gpa: 2.06,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
```

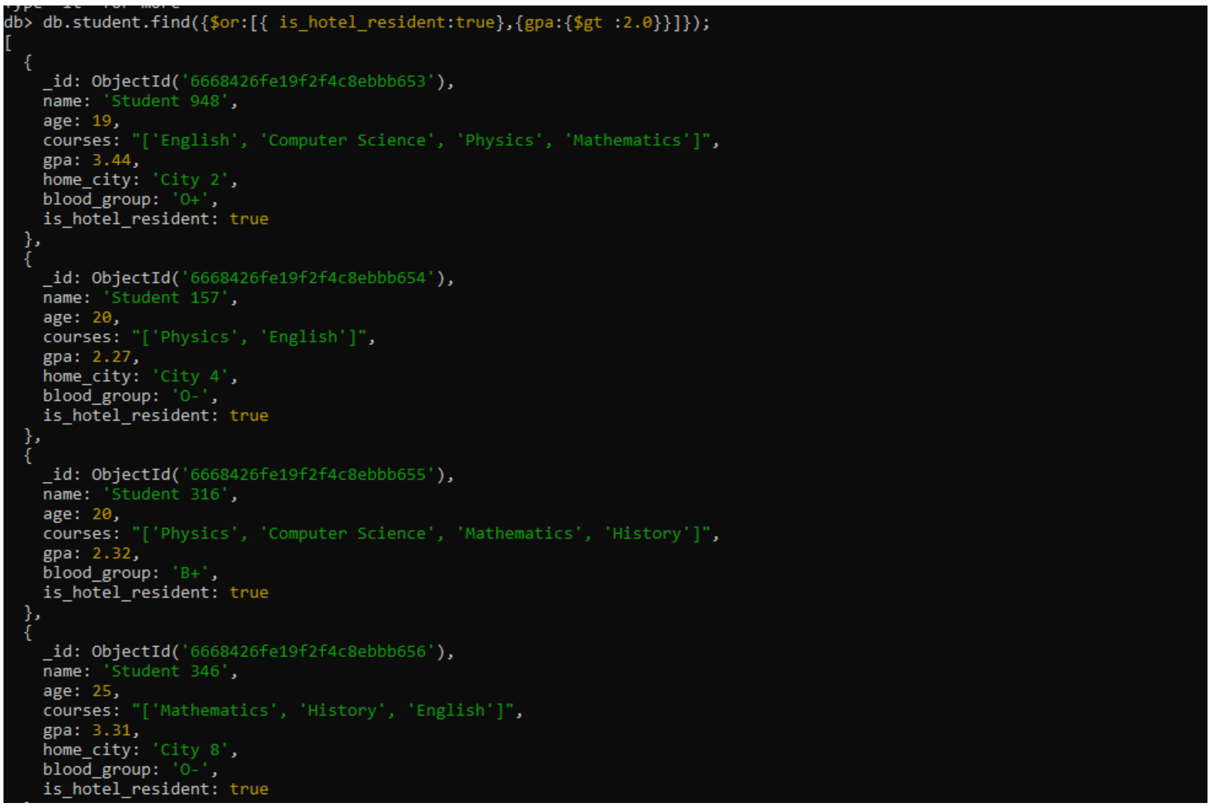
OR

● This operator is used to perform logical OR operation on the array of two or more expressions and select or retrieve only those documents that match at least one of the given expression in the array. You can use this operator in methods like find(), update(), etc.

Example:

```
 db.student.find({
$or:[
     { is_hotel_resident: true},
     { gpa: {$gt :2.0}}
 ]
});
```

Output:



```
db> db.student.find({$or:[{ is_hotel_resident:true},{gpa:{$gt :2.0}}]});
[
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb653'),
    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('6668426fe19f2f4c8ebbb654'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb655'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb656'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
```

CRUD

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

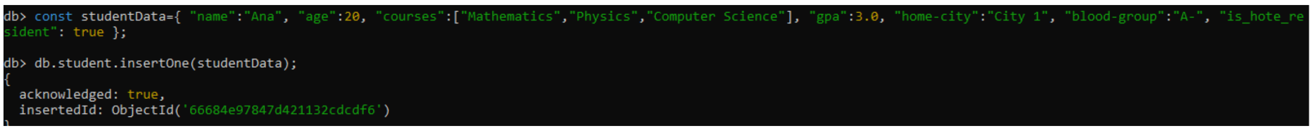
This is applicable for a Collection (Table) or a Document (Row)

Insert

Example:

```
 const studentData ={
“name”:”Ana”,
“age”:20,
 “courses”: [“Mathematics”,”Physics”,”Computer
Science”],
“gpa”:3.0,
“home_city”: “City 1”,
“blood_group”: “A-“,
“is_hotel_resident”:true };
```

Output:



```
db> const studentData={ "name":"Ana", "age":20, "courses":["Mathematics","Physics","Computer Science"], "gpa":3.0, "home-city":"City 1", "blood-group":"A-", "is_hote_re
sident": true };

db> db.student.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('66684e97847d421132cdcdf6')
```

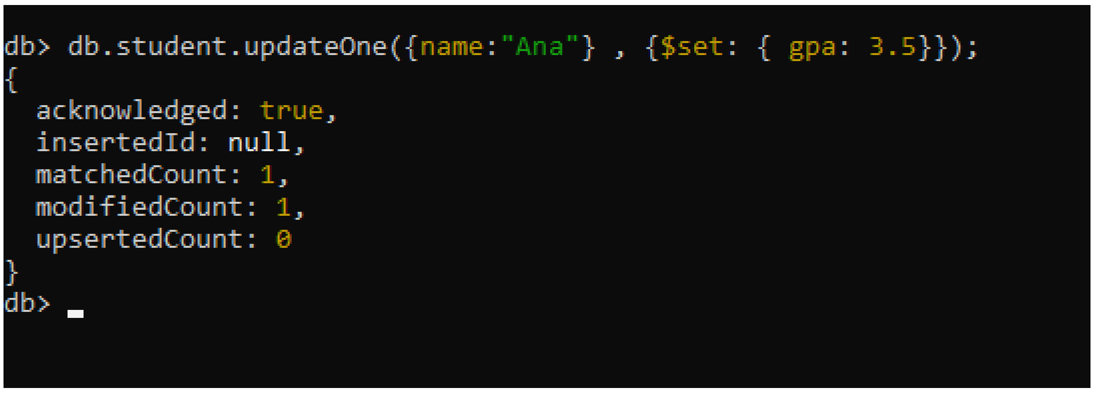
Update

Example:

```
db.student.updateOne(
{name : “Ana”,
{$set: {gpa :3.5}
```

});

Output:



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

Delete

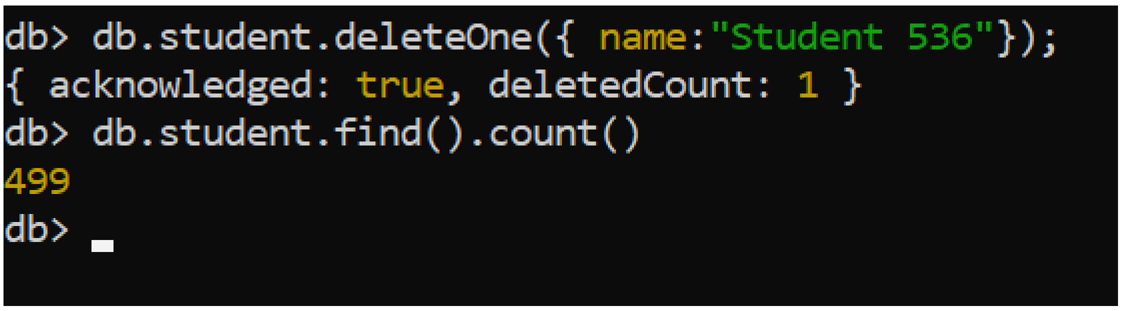
Example:

db.student.deleteOne(

{ name : “Student 536”}

);

Output:



```
db> db.student.deleteOne({ name:"Student 536"});
{ acknowledged: true, deletedCount: 1 }
db> db.student.find().count()
499
db> _
```

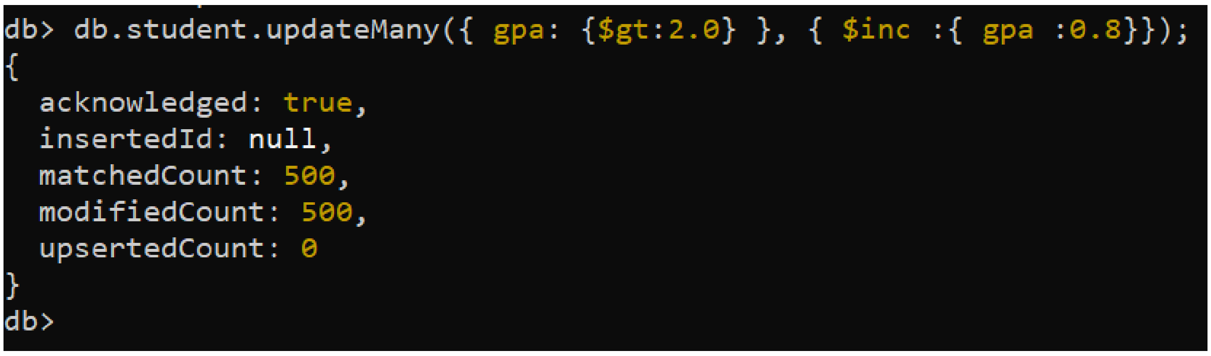
Update many

Example:

db.student.updateMany({ gpa :{$gt:2.0 } },

{$inc :{ gpa :0.8 } });

Output:



```
db> db.student.updateMany({ gpa: {$gt:2.0} }, { $inc :{ gpa :0.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 500,
  modifiedCount: 500,
  upsertedCount: 0
}
db>
```

Delete many

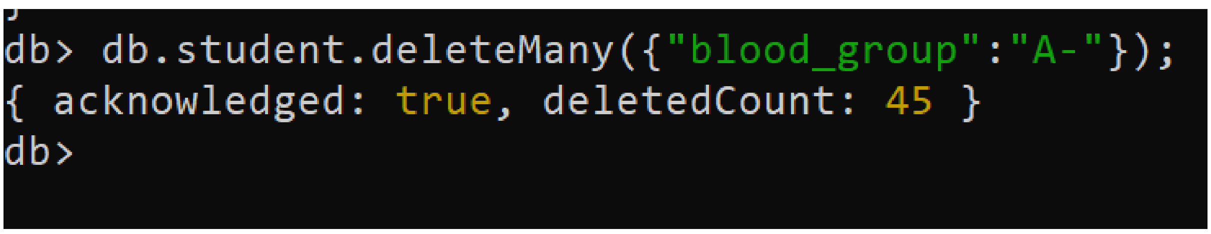
Example:

db.student.deleteMany(

{ “blood_group”: “A- “}

);

Output:



```
db> db.student.deleteMany({"blood_group":"A-"});
{ acknowledged: true, deletedCount: 45 }
db>
```

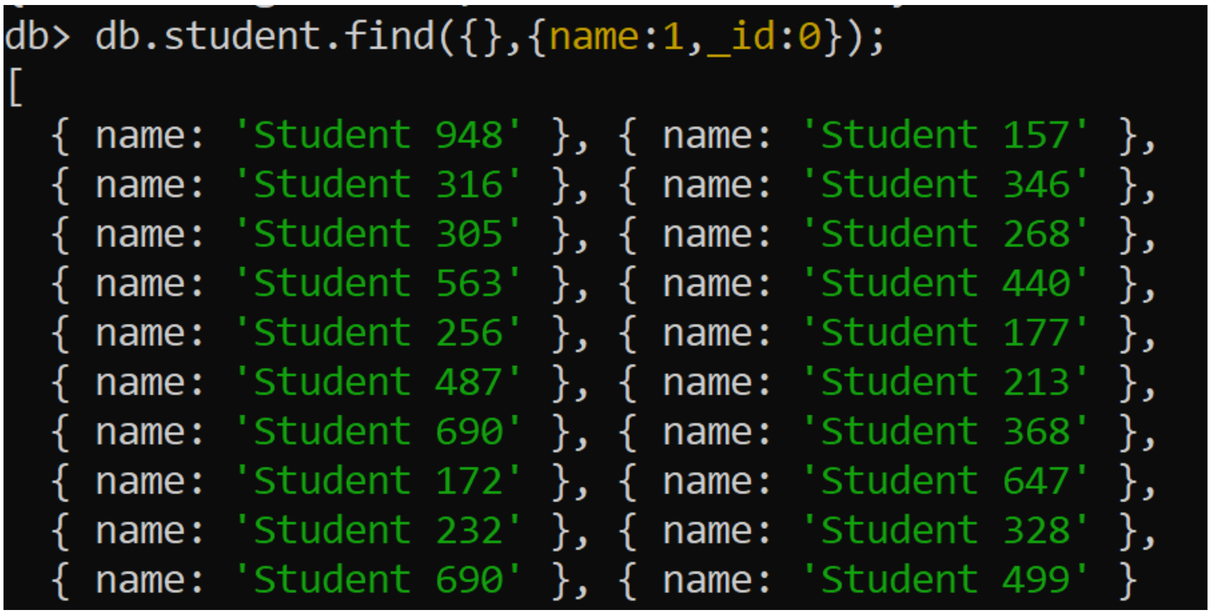
Projection

This is used when we don’t need specific column.

Example:

db.student.find({},{name :1 ,_id :0});

Output:



```
db> db.student.find({},{name:1,_id:0});
[
  { name: 'Student 948' }, { name: 'Student 157' },
  { name: 'Student 316' }, { name: 'Student 346' },
  { name: 'Student 305' }, { name: 'Student 268' },
  { name: 'Student 563' }, { name: 'Student 440' },
  { name: 'Student 256' }, { name: 'Student 177' },
  { name: 'Student 487' }, { name: 'Student 213' },
  { name: 'Student 690' }, { name: 'Student 368' },
  { name: 'Student 172' }, { name: 'Student 647' },
  { name: 'Student 232' }, { name: 'Student 328' },
  { name: 'Student 690' }, { name: 'Student 499' }
```

Class 4: Projection, Limit & Selectors

Experiment 2 & 3

Agenda

● Understand Projection

○ MongoDB allows you to select only the necessary data rather than selecting whole data from the document. For example, a document contains 5 fields.

● Understand LIMIT

○ Develop a MongoDB query to display the first 5 documents from the results obtained.

PROJECTION

Projection

● MongoDB projection solves this problem by enabling the find() function to be used with data filtering arguments, which allow users to extract only the necessary data fields from a document.

● Used to extract only the fields you need from a document—not all fields.

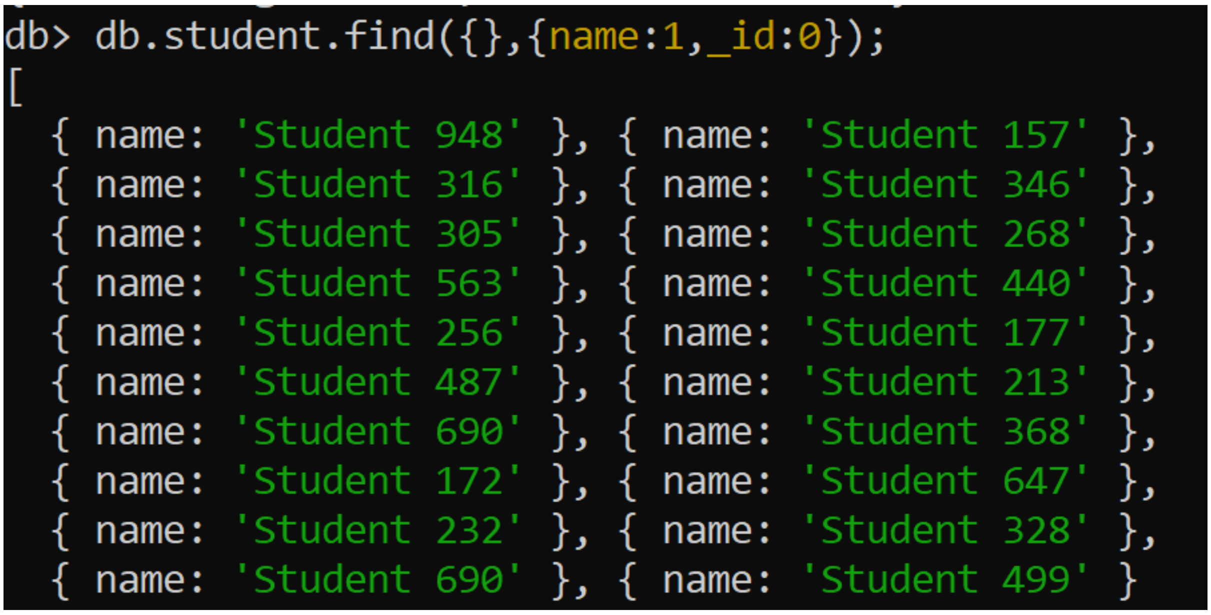
Get Selected Attributes

● To retrieve only “name” attribute from a collection called “student”.

Example:

```
db.student.find({},{name :1,_id:0});
```

Output:



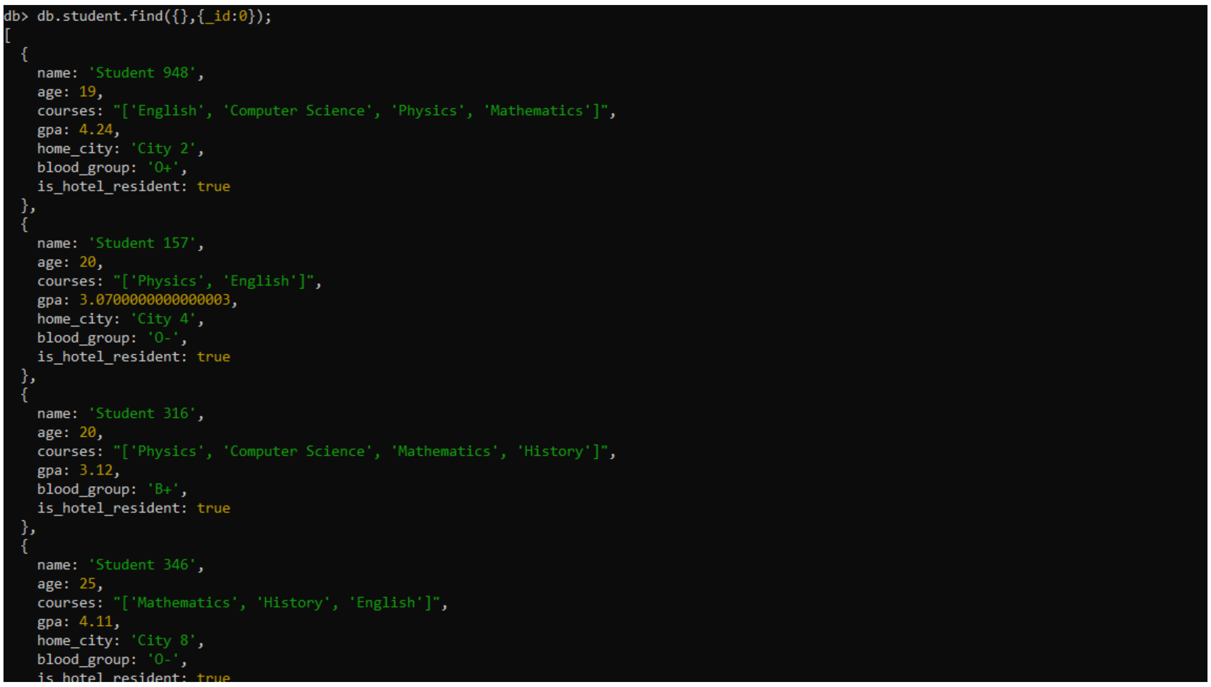
```
db> db.student.find({},{name:1,_id:0});
[
  { name: 'Student 948' }, { name: 'Student 157' },
  { name: 'Student 316' }, { name: 'Student 346' },
  { name: 'Student 305' }, { name: 'Student 268' },
  { name: 'Student 563' }, { name: 'Student 440' },
  { name: 'Student 256' }, { name: 'Student 177' },
  { name: 'Student 487' }, { name: 'Student 213' },
  { name: 'Student 690' }, { name: 'Student 368' },
  { name: 'Student 172' }, { name: 'Student 647' },
  { name: 'Student 232' }, { name: 'Student 328' },
  { name: 'Student 690' }, { name: 'Student 499' }
```

Ignore Attributes

Example:

db.student.find({ } ,{ _id:0});

Output:



```
db> db.student.find({},{_id:0});
[
  {
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 4.24,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 3.0700000000000003,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.12,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 4.11,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
```

Benefits of Projection

- Project concise and transparent data.
- Filter the data set without impacting the overall database performance.
- Simplifies your code.

LIMIT

Limit

● 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.

● this method uses on cursor to specify the maximum number of documents/ records the cursor will return.

● . We can use this method after the find() method and find() will give you all the records or documents in the collection.

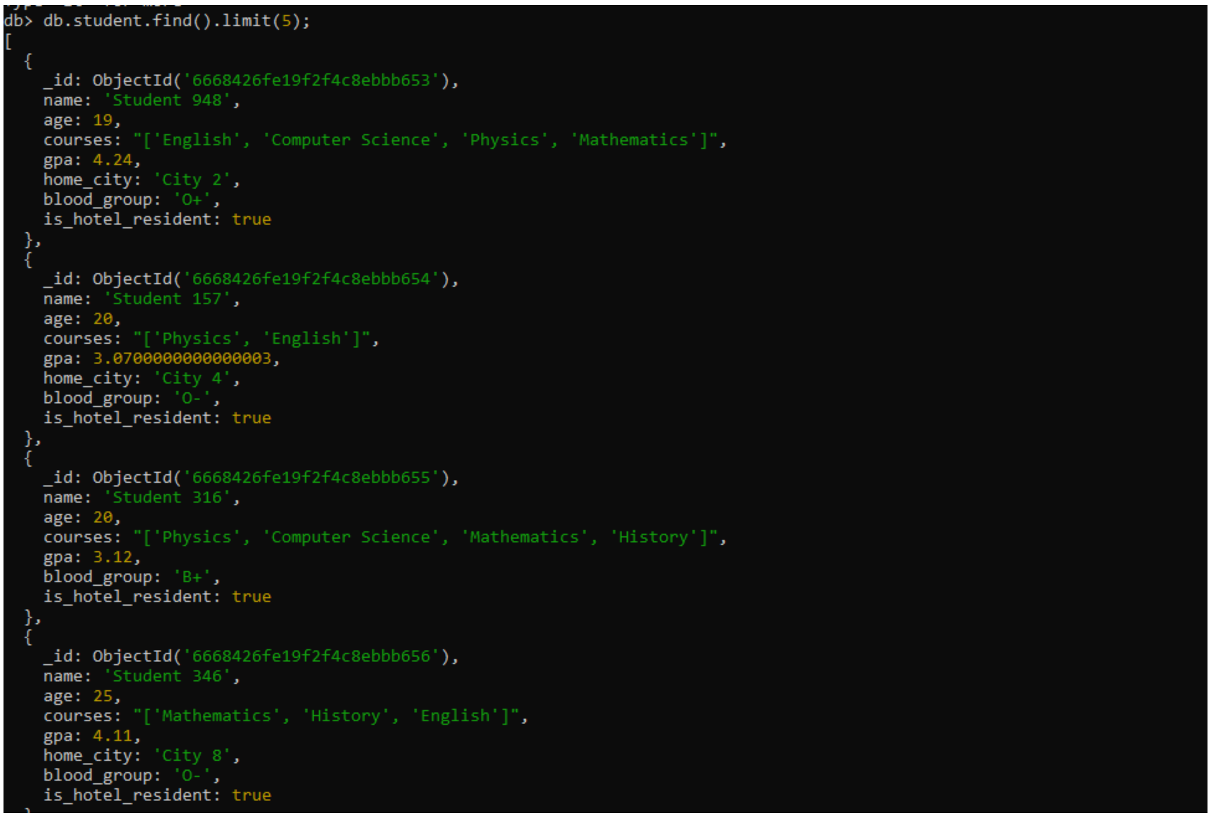
Syntax: db.collection.find({filter}, {projection}).limit(number)

Get First 5 document

Example:

db.student.find().limit(5);

Output:



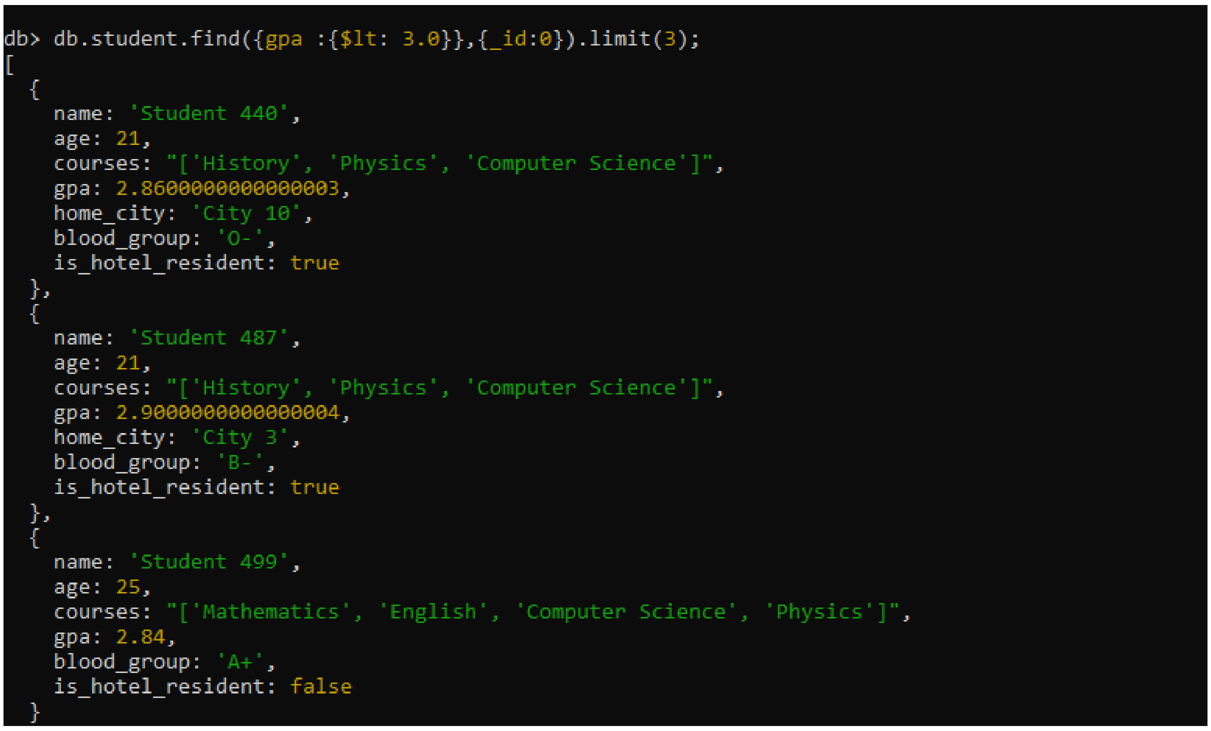
```
db> db.student.find().limit(5);
[
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb653'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 4.24,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb654'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 3.0700000000000003,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb655'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.12,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb656'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 4.11,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
```

Limiting Results

Example:

db.student.find({gpa: {$lt : 3.0 } },{_id:0}).limit(3);

Output:



```
db> db.student.find({gpa :{$lt: 3.0}},{_id:0}).limit(3);
[
  {
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.8600000000000003,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 487',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.9000000000000004,
    home_city: 'City 3',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    name: 'Student 499',
    age: 25,
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']",
    gpa: 2.84,
    blood_group: 'A+',
    is_hotel_resident: false
  }
```

Experiment 3 - SELECTORS

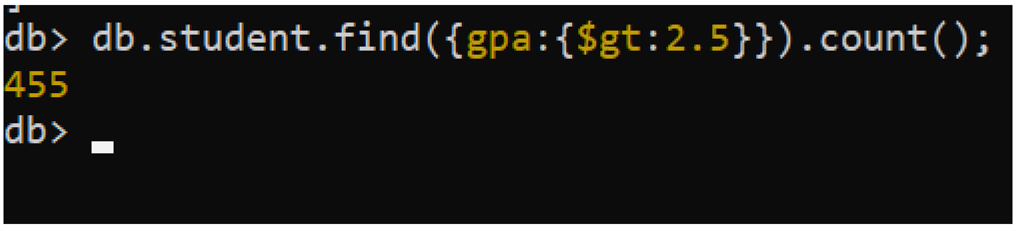
Comparison gt lt

‘gt’ stands for “GREATER THAN” and ‘lt’ stands for “LESSER THAN” .They are comparison operators used in queries to filter documents based on the values of specific fields.

Example:

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

Output:



```
db> db.student.find({gpa:{$gt:2.5}}).count();
455
db> 
```

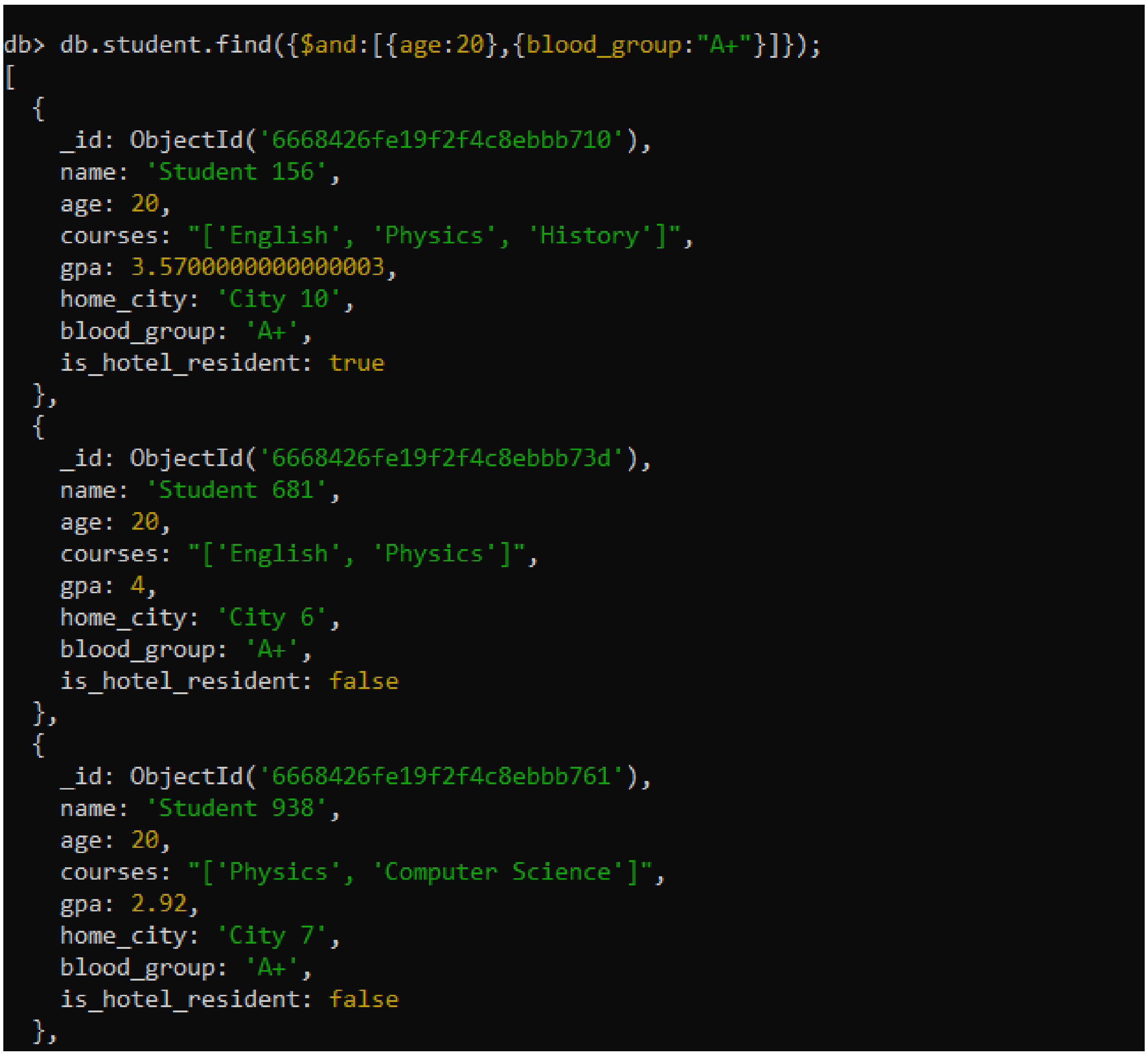
AND operator

Example:

```
db.student.find({
$and : [
{age:20},
{blood_group} ]
```

});

Output:



```
db> db.student.find({$and:[{age:20},{blood_group:"A+"}]});
[
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb710'),
    name: 'Student 156',
    age: 20,
    courses: "['English', 'Physics', 'History']",
    gpa: 3.5700000000000003,
    home_city: 'City 10',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb73d'),
    name: 'Student 681',
    age: 20,
    courses: "['English', 'Physics']",
    gpa: 4,
    home_city: 'City 6',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb761'),
    name: 'Student 938',
    age: 20,
    courses: "['Physics', 'Computer Science']",
    gpa: 2.92,
    home_city: 'City 7',
    blood_group: 'A+',
    is_hotel_resident: false
  },
```

OR operator

Example:

```
db.student.find({
$or :[
```

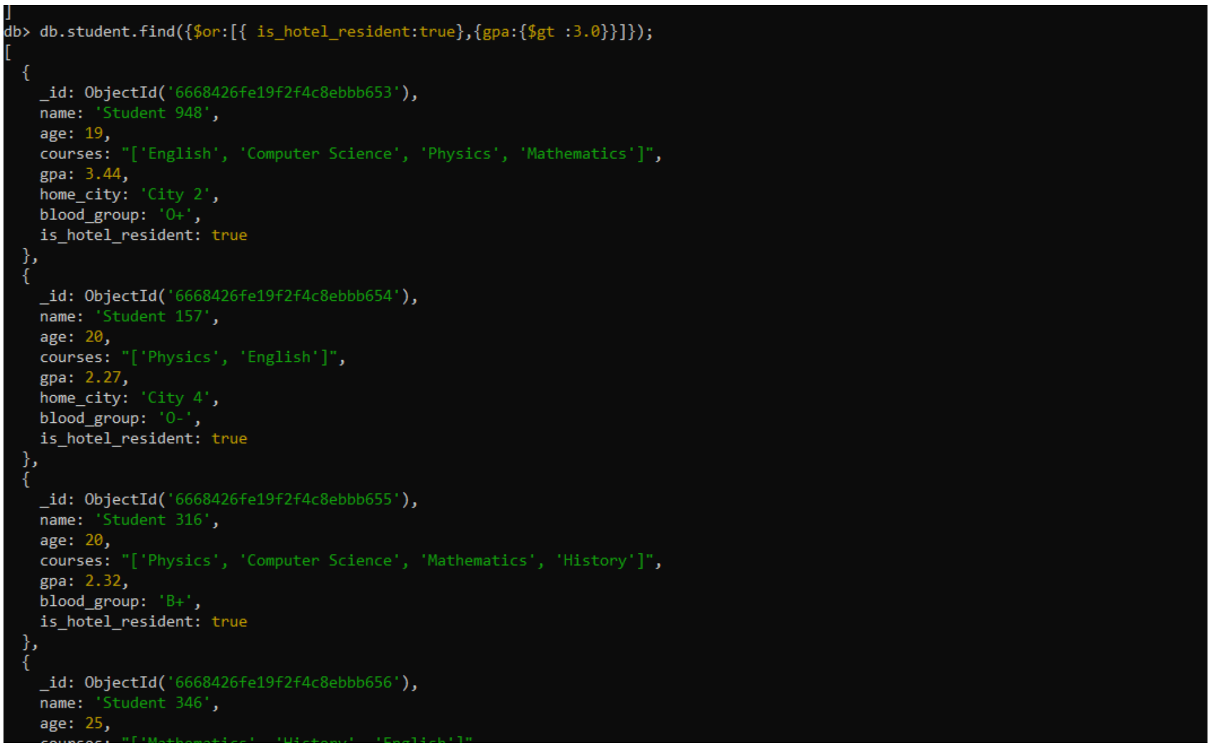
{is_hotel_resident: true},

{gpa :{$gt : 3.0}

]

}):

Output:



```
]
db> db.student.find({$or:[{ is_hotel_resident:true},{gpa:{$gt :3.0}}]});
[
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb653'),
    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('6668426fe19f2f4c8ebbb654'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb655'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6668426fe19f2f4c8ebbb656'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
```

GEOSPATIAL

Geospatial indexing and querying in MongoDB enables efficient storage and retrieval of geospatial data, such as points, lines, and polygons. MongoDB supports various geospatial queries and operators.

Geospatial Query

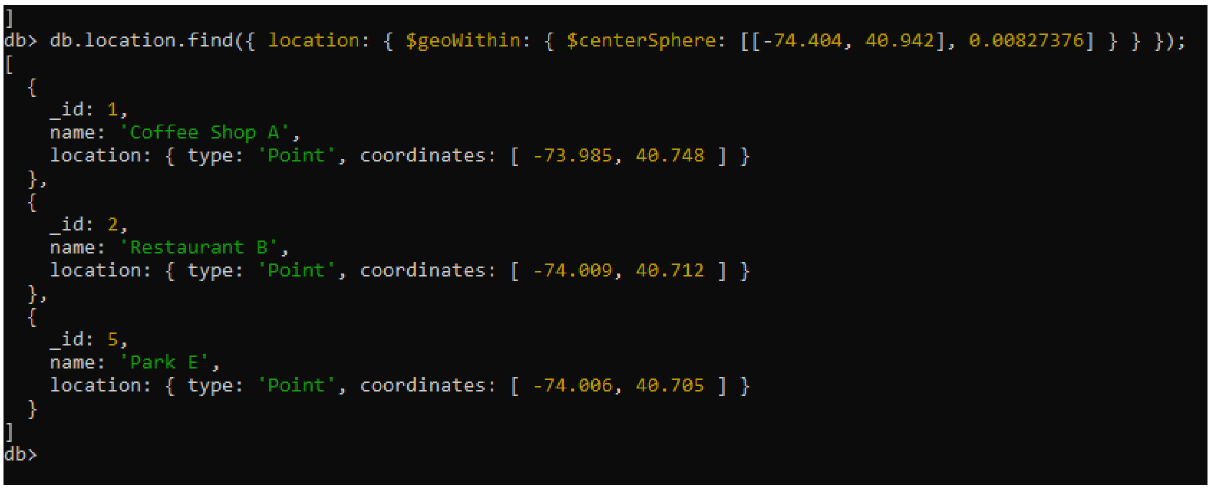
$geoWithin : operator is used to query for documents That are within a specified geometric shape.

$centerSphere : specifically looking to find the documents within tha spherical region .

Example:

```
db.location.find({
location:[
$geoWithin:{
$centerSphere:[[-74.404 , 40.942],0.00827376}
 }
 });
```

Output



```
]
db> db.location.find({ location: { $geoWithin: { $centerSphere: [[-74.404, 40.942], 0.00827376] } } });
[
  {
    _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>
```

Data types and Operations

- DataType
    - Line String
    - Polygon
    - Point

Data types and Operations

Some commonly used datatype operations are as follows:

1. Comparison Operators: $eq, $ne, $gt, $gte, $lt, $lte, $in, $nin, etc., used for comparison.

2. Geospatial Operators: $geoWithin, $geoIntersects, $near, $nearSphere, for geospatial queries.

3. Array Operators: $all, $elemMatch, $size, for querying arrays.

4. Logical Operators: $and, $or, $not, $nor, for logical operations.