Class 1 : Introduction to MongoDB

MongoDB Installation Download and Install MongoDB on windows using

https://www.mongodb.com/download-center/community

Download MongoDB shell using

https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-7.0.11-signed.msi

MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently. It is categorized under the NoSQL database because the storage and retrieval of data in the MongoDB are not in the form of tables.

- **Structured Data:**

The information is typically organized in a specific format, often using tables with rows and columns. This makes it easier to search, filter, and analyze the data.

- **Database Management System (DBMS):**

This is the software that acts like the filing cabinet manager. It allows you to store, retrieve, update, and manage all the data within the database.

- **Data Types:**

Databases can hold various kinds of information, including text, numbers, images, videos, and more.

- **MongoDB vs. MySQL**

MySQL (link resides outside IBM.com) uses a structured query language to access stored data. In this format, schemas are used to create database structures, utilizing tables as a way to standardize data types so that values are searchable and can be queried properly. A mature solution, MySQL is useful for a variety of situations including website databases, applications and commercial product management.

- **Advantage**

 Schema Flexibility:

- MongoDB stores data in flexible, JSON-like documents. This allows for the storage of semi-structured data and the ability to change the data structure over time without significant downtime or reengineering.

 Scalability:

- MongoDB is designed to scale out horizontally by sharding, distributing data across multiple servers. This makes it easier to

manage large volumes of data and handle high-throughput operations.

- High Performance:
  - With its ability to index and query data efficiently, MongoDB can deliver high performance for both read and write operations. It supports rich queries, indexing, and aggregation.
- Geospatial Support:
  - MongoDB has built-in support for geospatial data and queries, making it suitable for applications that require location-based features.
- Aggregation Framework:
  - MongoDB provides a powerful aggregation framework that allows for complex data processing and transformation tasks within the database, reducing the need for additional processing in the application layer.
- Document-Based Model:
  - The document model maps naturally to objects in application code, making it easy for developers to work with data. This is particularly beneficial for object-oriented programming.

 Open Source:

- MongoDB is open-source, which means it is free to use and has a large, active community contributing to its development and providing support.

 Strong Ecosystem:

- MongoDB has a strong ecosystem with a wide range of tools, libraries, and integrations. This includes support for various programming languages, data visualization tools, and cloud services.

- **Disadvantage**

 Data Size and Memory Usage:

- MongoDB can consume a significant amount of memory and storage due to its schema-less design and the storage format of its BSON documents, which might include redundant data and larger document sizes.

 Limited Support for Complex Transactions:

- Although MongoDB supports multi-document ACID transactions starting from version 4.0, its transaction support is still not as robust as traditional relational databases. Complex transactions involving many documents can be less efficient and more challenging to manage.

- Consistency Concerns:
  - By default, MongoDB is configured for eventual consistency, which might not be suitable for applications requiring strong consistency. This means that there can be a delay before all nodes in a distributed system reflect a write operation.
- Indexing Limitations:
  - While MongoDB supports indexing, there are limitations on the number of indexes per collection and the size of indexed fields. Poor indexing strategies can lead to significant performance issues.
- Join Limitations:
  - MongoDB does not support joins in the traditional sense found in relational databases. While it has the $lookup operation to perform joins, these operations can be less efficient and more cumbersome for complex queries.
- Learning Curve:
  - For teams accustomed to relational databases, transitioning to MongoDB can involve a steep learning curve. Developers need to understand NoSQL principles and best practices to effectively use MongoDB.

 Data Duplication:

- The lack of normalization can lead to data duplication, which might result in increased storage requirements and challenges with data consistency and updates.

- **Applications**

  Web Application

  Big Data

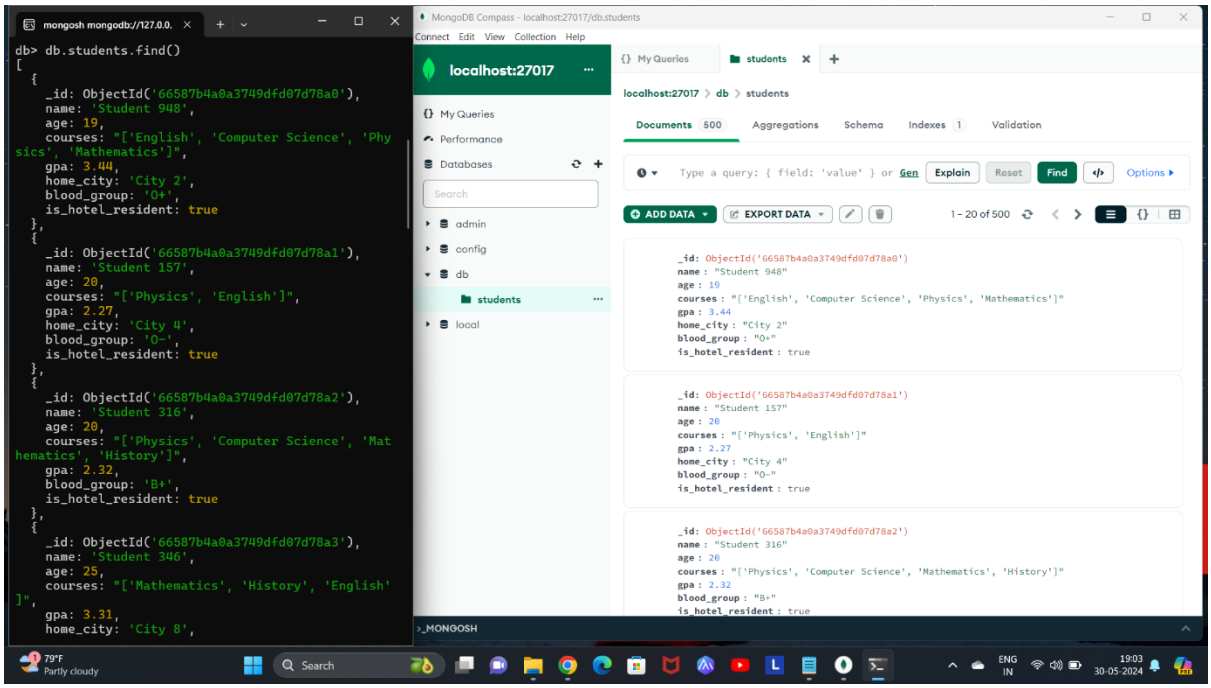
  Demographic and Biometric Data

  Synchronization

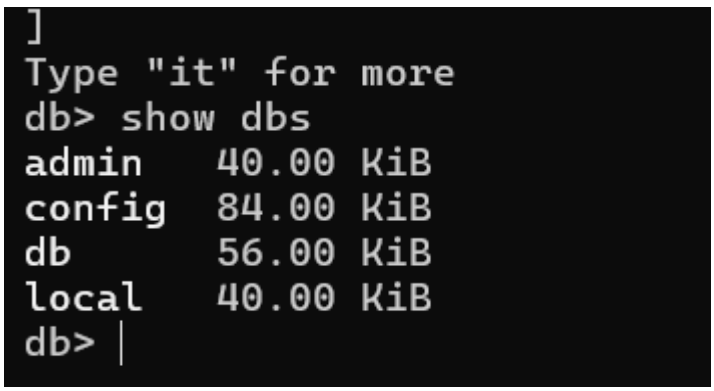
Class 2 : Add ,Update and Delete

To find the data present in the collections ,we can use the command “db.collection_name.find()” .

In this the collection name is “students”.



“show dbs” command shows all the database.



- Collections :

A collection is a group of documents.

If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

- Database:

MongoDB groups collections into databases.

A single instance of MongoDB can host several databases, each grouping together zero or more collections.

- Document:

At the heart of MongoDB is the document:

an ordered set of keys with associated values.

The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

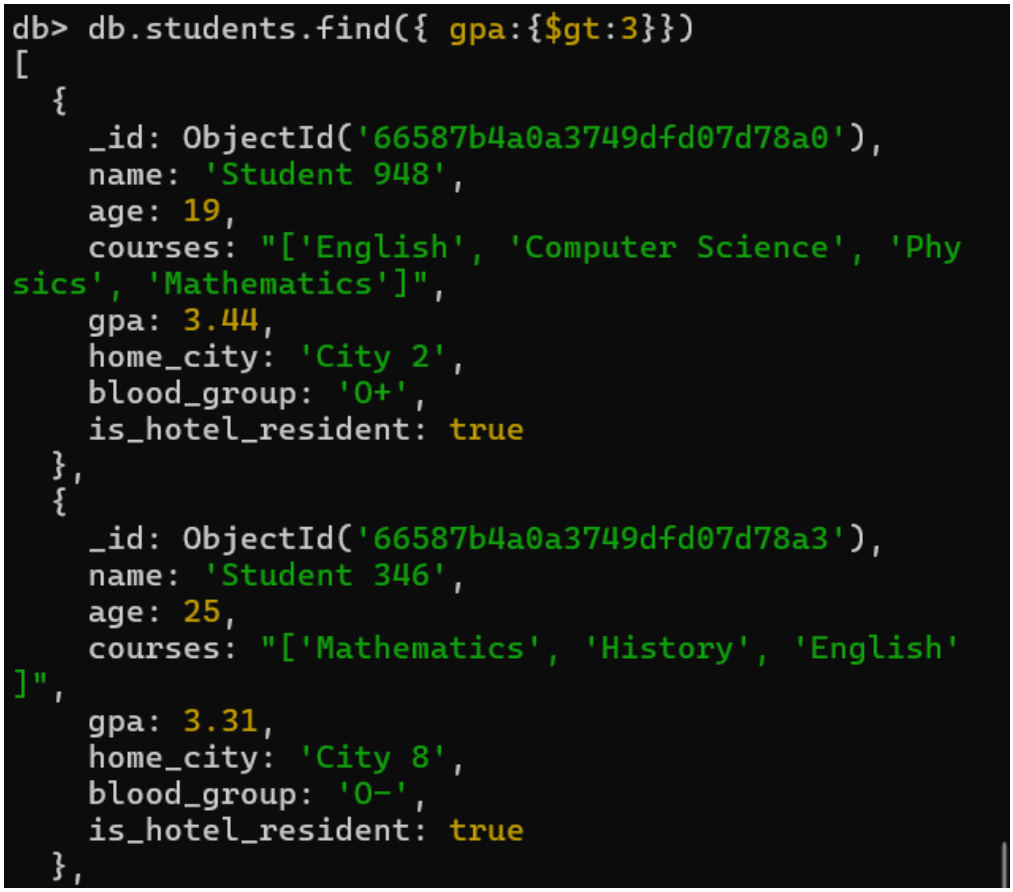
Class 3 : WHERE,AND,OR & CRUD

- **WHERE**:

WHERE is used when we want to filter a subset based on a condition.

To find the collection with gpa greater than 3.0 we use a command

db.students.find({gpa:{$gt:3.0}});



We use the condition gpa greater than 3. Here it shows students gpa greater than 3.The result is shown is based on this condition.

- **AND:**

AND is used when, in a given collection we want to filter a subset based on multiple conditions.

To find students who are lived in city 1 and having a blood group "O-"we use

db.students.find({$and :

[{home_city:"City1"},{blood_group:"O-"}]});

home_city : City1’ and ‘blood_group : O-‘ are the conditions on which it is filtered.

- **OR:**

OR is used when , in a given collection we want to filter a subset based on multiple conditions but any one is sufficient .

Here we are checking for students who are hotel resident and scored gpa less than 3.0 we use **db.students.find({$or:[{is_hotel _resident:true},{gpa:{$lt:3.000}}]});**

```
db> db.students.find({$or:[{is_hotel_resident:true},{gpa:{$lt:3.0}}]});
[
  {
    _id: ObjectId('6663dac4f24355f2c2a837e5'),
    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('6663dac4f24355f2c2a837e6'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837e7'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837e8'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
```

- **CRUD:**

  C – Create / Insert

  R – Remove

  U – Update

  D – Delete

This is applicable for a collection (table) or a document (row)

- Update:

'$set' command is used to update the data present in the collection.

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

- Delete:

It is used to delete the data present in the collection.

```
db> db.students.deleteOne({ name:"Sam" })
{ acknowledged: true, deletedCount: 1 }
db>
```

- Update many:

update some aspects(saved information) of a students

To update data of students with a gpa greater than 3.5 by increasing 0.5 we use

db.students.updateMany({gpa:3.5}},{$inc:{gpa:0.5}});

```
db> db.students.updateMany({gpa:{$gt:3.5}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 124,
  modifiedCount: 124,
  upsertedCount: 0
}
db>
```

- Delete many:

To delete data of students who are hotel_residents we use a command

db.students.deleteMany({is_hotel_resident:true});

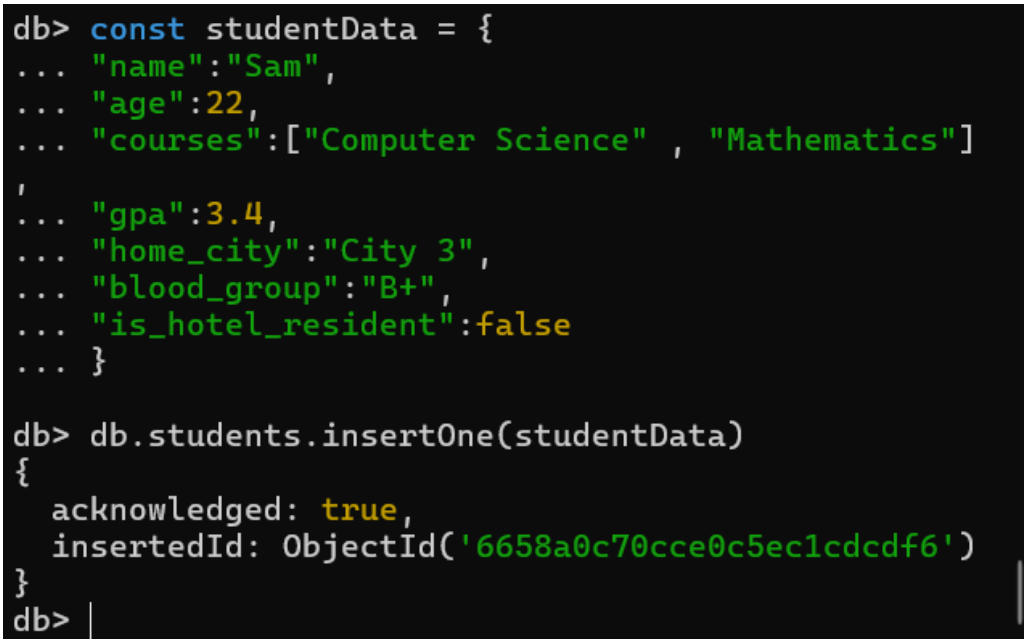
and the countings of deleted students is 246.

```
db> db.students.deleteMany({is_hotel_resident:true});
{ acknowledged: true, deletedCount: 246 }
db>
```

Delete all the students who's blood group "A-" we us **db.students.deleteMany({blood_group:"A-"});**

- Insert:

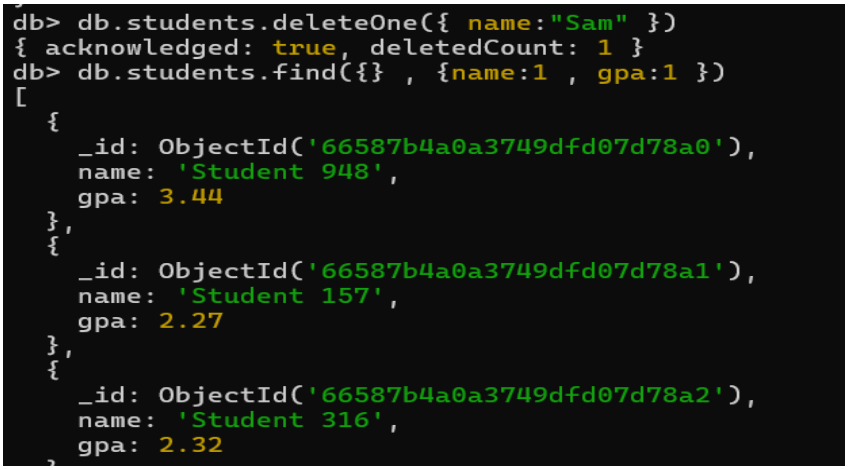
Here we are inserting the student details name 'Sam' and other information to the collection 'students'.the insertion is done one time.



Projection:

This is used when we don't need all columns / attributes.

- Benefits of Projection:
  - ✓ Reduced 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.



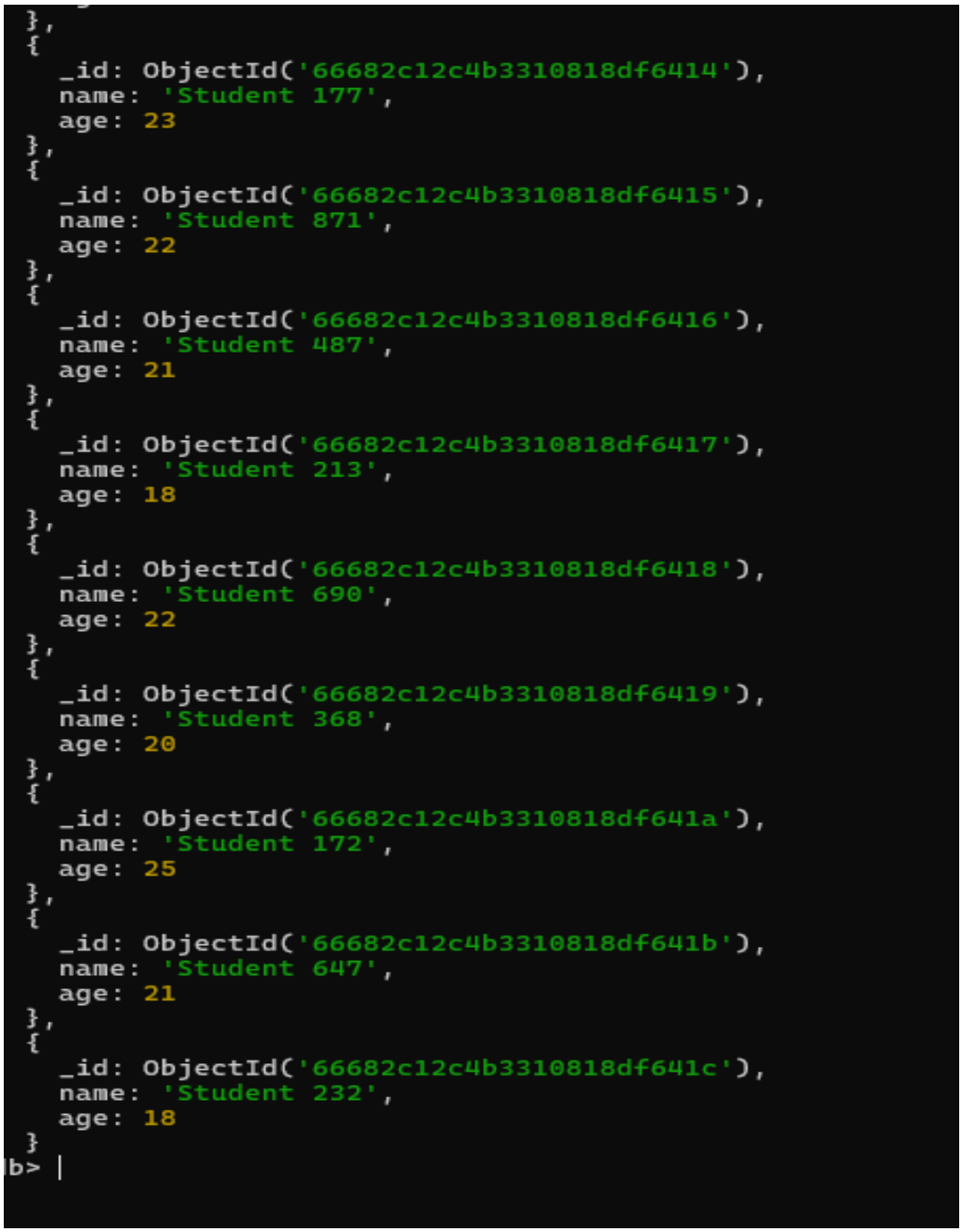
Here it only shows the name and gpa . Because the command is give as ‘name:1’ and ‘gpa:1’.

Class 4 : Projection, Limit & Selectors

Get Selected Attributes:

Given a Collection you want to FILTER a subset of attributes. That is the place Projection is used.

```
db> db.students.find({},{name:1,age:1});
[
  {
    _id: ObjectId('66682c12c4b3310818df6409'),
    name: 'Student 948',
    age: 19
  },
  {
    _id: ObjectId('66682c12c4b3310818df640a'),
    name: 'Student 157',
    age: 20
  },
  {
    _id: ObjectId('66682c12c4b3310818df640b'),
    name: 'Student 316',
    age: 20
  },
  {
    _id: ObjectId('66682c12c4b3310818df640c'),
    name: 'Student 346',
    age: 25
  },
  {
    _id: ObjectId('66682c12c4b3310818df640d'),
    name: 'Student 930',
    age: 25
  },
  {
    _id: ObjectId('66682c12c4b3310818df640e'),
    name: 'Student 305',
    age: 24
  },
  {
    _id: ObjectId('66682c12c4b3310818df640f'),
    name: 'Student 268',
    age: 21
  },
  {
    _id: ObjectId('66682c12c4b3310818df6410'),
    name: 'Student 563',
    age: 18
  },
  {
    _id: ObjectId('66682c12c4b3310818df6411'),
    name: 'Student 440',
    age: 21
  },
  {
    _id: ObjectId('66682c12c4b3310818df6412'),
    name: 'Student 536',
    age: 20
  },
  {
    _id: ObjectId('66682c12c4b3310818df6413'),
    name: 'Student 256',
    age: 19
  },
```

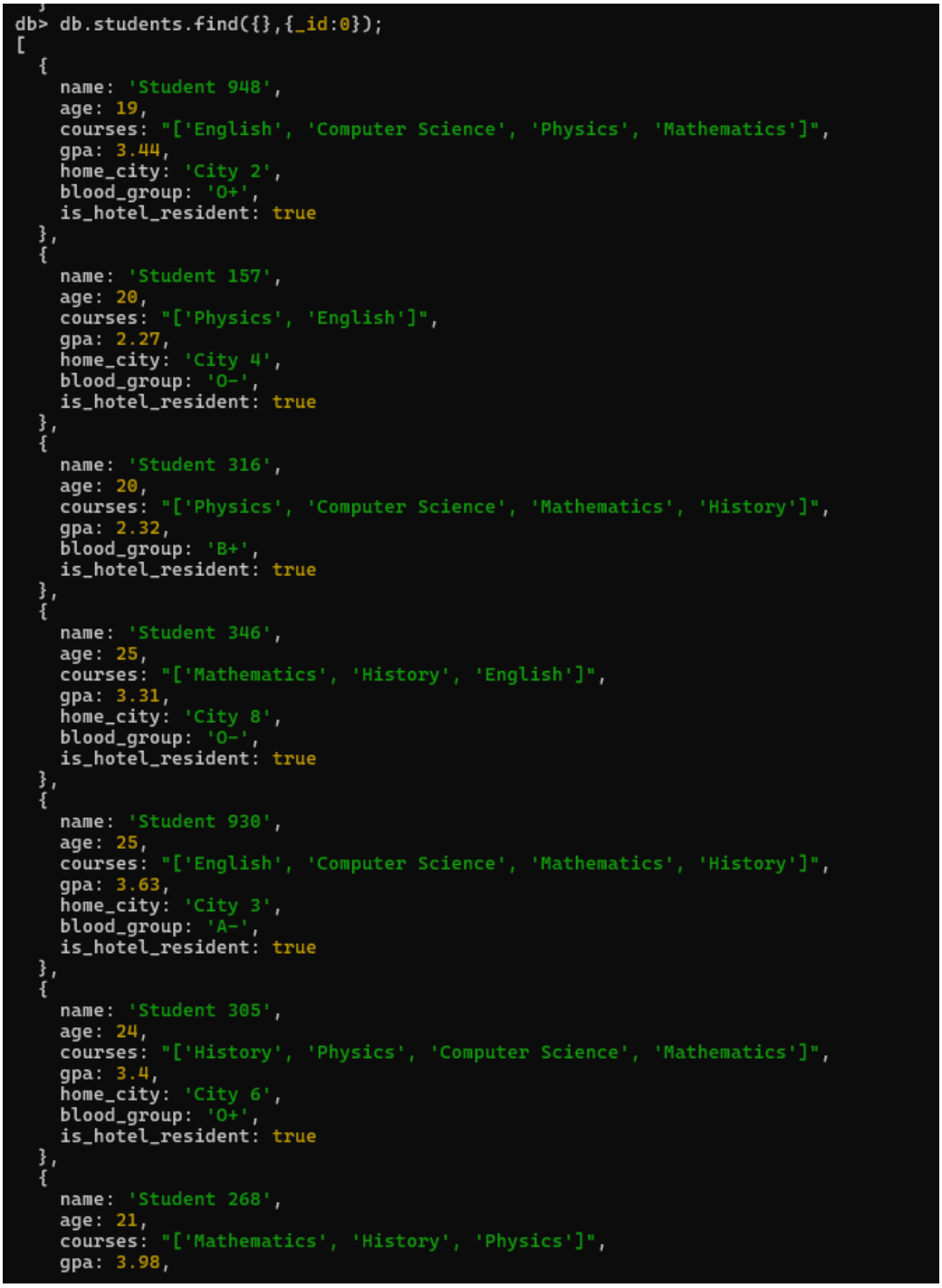


Here we are displaying only name and age. We use the command **db.students.find({},{name:1, age:1});**

Ignore Attributes :

To get all students data but excluding the id .We use the command

db.students.find({},{-id:0});



```
db> db.students.find({},{_id:0});
[
  {
    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
  },
  {
    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
  },
  {
    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
  },
  {
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
```

Here students data is shown without id.

Retrieving specific Fields from Nested Objects:

```
Type "it" for more
db> db.students.find({}, { name: 1, courses: { $slice: 2 } });
[
  {
    _id: ObjectId('66682c12c4b3310818df6409'),
    name: 'Student 948',
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df640a'),
    name: 'Student 157',
    courses: "['Physics', 'English']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df640b'),
    name: 'Student 316',
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df640c'),
    name: 'Student 346',
    courses: "['Mathematics', 'History', 'English']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df640d'),
    name: 'Student 930',
    courses: "['English', 'Computer Science', 'Mathematics', 'History']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df640e'),
    name: 'Student 305',
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df640f'),
    name: 'Student 268',
    courses: "['Mathematics', 'History', 'Physics']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df6410'),
    name: 'Student 563',
    courses: "['Mathematics', 'English']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df6411'),
    name: 'Student 440',
    courses: "['History', 'Physics', 'Computer Science']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df6412'),
    name: 'Student 536',
    courses: "['History', 'Physics', 'English', 'Mathematics']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df6413'),
    name: 'Student 256',
    courses: "['Computer Science', 'Mathematics', 'History', 'English']"
  },
  {
    _id: ObjectId('66682c12c4b3310818df6414'),
    name: 'Student 177',
```

Here only students name and the second course is obtained.

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)

To get first 5 document:

```
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
  },
  {
    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
  },
  {
    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
  }
]
db>
```

In this snapshot we got first five documents of the students data.

Selectors

Selectors in MongoDB are used to specify criteria for querying, updating, or deleting documents in a collection. They are part of MongoDB's powerful query language, enabling you to precisely target documents based on various conditions.

Comparsion gt It:

To find all the students with age greater than 10. We use the command

db.students.find({age:{$gt:10}});

```
]
db> db.students.find({age:{$gt:10}});
[
  {
    _id: ObjectId('66682c12c4b3310818df6409'),
    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('66682c12c4b3310818df640a'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66682c12c4b3310818df640b'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66682c12c4b3310818df640c'),
    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('66682c12c4b3310818df640d'),
    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('66682c12c4b3310818df640e'),
    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
  },
```

Operators

Logical operators:

$and operator:

Each condition is defined that must meet for the document ti be selected.

$or:

Each condition is defined atleast one of its must be matched to return a document.

AND Operator:

To find the students from city 3 with blood group A+.

```
db> db.students.find({
... $and:[
... {home_city:"City 3"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('66682c12c4b3310818df641a'),
    name: 'Student 172',
    age: 25,
    courses: "['English', 'History', 'Physics', 'Mathematics']",
    gpa: 2.46,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66682c12c4b3310818df643e'),
    name: 'Student 959',
    age: 24,
    courses: "['History', 'Computer Science']",
    gpa: 3.43,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66682c12c4b3310818df64ce'),
    name: 'Student 918',
    age: 19,
    courses: "['Physics', 'Computer Science']",
    gpa: 3.92,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66682c12c4b3310818df6515'),
    name: 'Student 728',
    age: 24,
    courses: "['Mathematics', 'Physics', 'English']",
    gpa: 3.95,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
```

Here we got the details of the students from city 3 with blood group A+.

OR Operator:

To find the students who are hotel residents OR have a gpa less than 2.

```
]
db> db.students.find({
... $or:[
... {is_hotel_resident:true},
... {gpa:{$lt:2.0}}
... ]
... });
[
  {
    _id: ObjectId('66682c12c4b3310818df6409'),
    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('66682c12c4b3310818df640a'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66682c12c4b3310818df640b'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66682c12c4b3310818df640c'),
    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('66682c12c4b3310818df640d'),
    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('66682c12c4b3310818df640e'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
```

Here students with gpa less than 2.0 is obtained.

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

Querying in MongoDB involves using selectors to filter documents in a collection.

To find students with both lobby and campus permission

db.students_permission.find({permissions:{$bitsAllSet:[LOBBY_PER

MISSION, CAMPUS-PERMISSION]}});

First we need to define bit positions for permisssions (collection)so we defines

const LOBBY_PERMISSION=1;

const CAMPUS-PERMISSION=2

```
b> db.students_permission.find({permissions:{$bitsAllSet:[LOBBY_PERMISS
ON,CAMPUS_PERMISSION]}});
[
  {
    _id: ObjectId('6663ff4286ef416122dcfcd5'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('6663ff4286ef416122dcfcd6'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('6663ff4286ef416122dcfcd7'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
b>
```

Geospatial Query

Geospatial queries in MongoDB allow you to work with location-based data, such as finding documents within a certain distance from a point, within a polygon, or near other shapes. MongoDB supports various geospatial operations and indexing types to efficiently handle these queries.

Here to find a location:

```
db> db.locations.find({
... location:{
... $geoWithin:{
... $centerSphere:[[-74.005,40.712],0.00621376]
... }
... }
... });
[
  {
    _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 ] }
  }
]
```