MONGO DB

Introduction to MongoDB:

MongoDB is a popular open-source NoSQL database program. It uses a document-oriented data model, making it especially useful for applications with evolving and complex data structures. MongoDB stores data in flexible, JSONlike documents, which can vary in structure, offering a more dynamic and scalable approach compared to traditional relational databases. It's widely used in modern web development for its scalability, flexibility, and ease of use.

Why to study mongoDB ?

- Market Demand
- Flexible Data Model
- Scalability Introduction to MongoDB
- Performance Optimization
- Real-time Analytics
- Big Data Trends
- Diversity in Database Systems.

Applications Of MongoDB:

- Web and Mobile Applications
- Content Management Systems (CMS.
- E-commerce Platforms
- Real-time Analytics and Logging
- Internet of Things (IoT)
- Social Networks and Web Collaboration Platforms.

Database :

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. It is designed to efficiently manage, retrieve, and manipulate data according to various criteria. Databases are commonly used in a wide range of applications, from simple ones like address books to complex ones like airline reservation systems or financial records.
Databases are essential components of modern computing, used in web applications, mobile apps, enterprise systems, and many other scenarios where structured data needs to be stored, managed, and accessed efficiently.

Collection:

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.

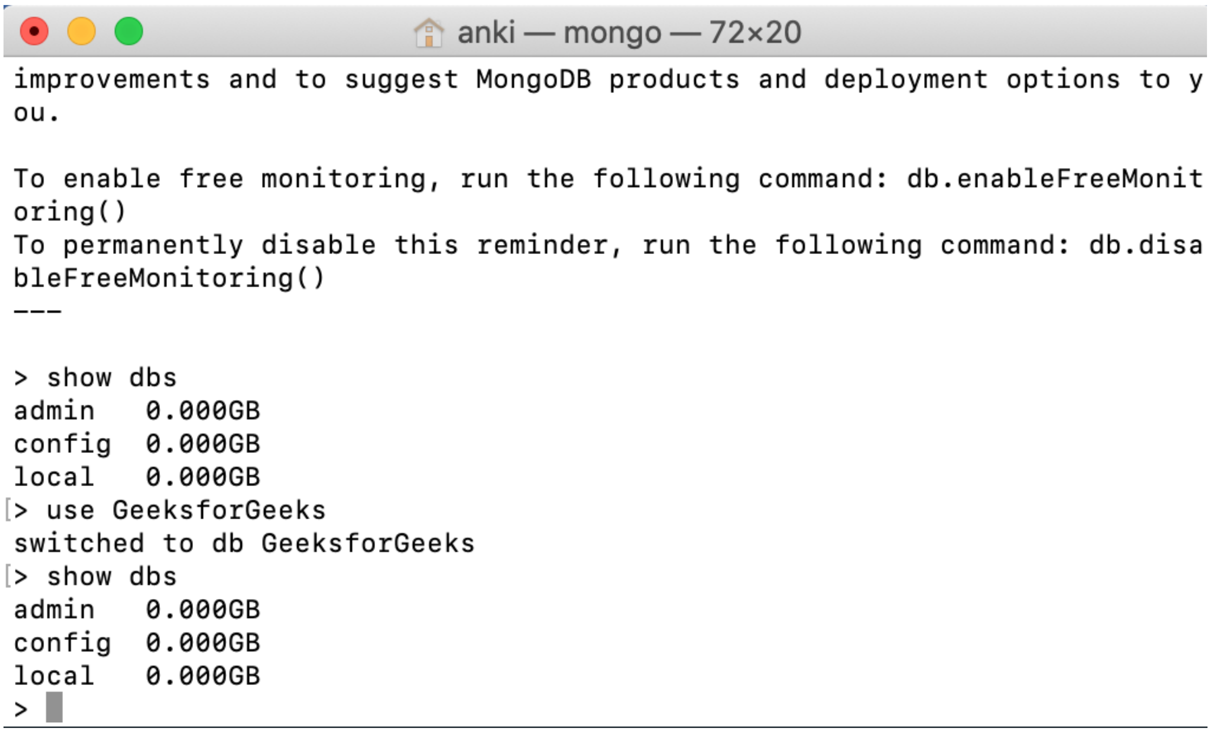
Naming Restrictions for Collection:

Before creating a collection you should first learn about the naming restrictions for collections:

- Collection name must starts with an underscore or a character.
- Collection name does not contain $, empty string, null character and does not begin with system. Prefix.

Examples:

For creating a database using the name of database in mongoDb Compass.

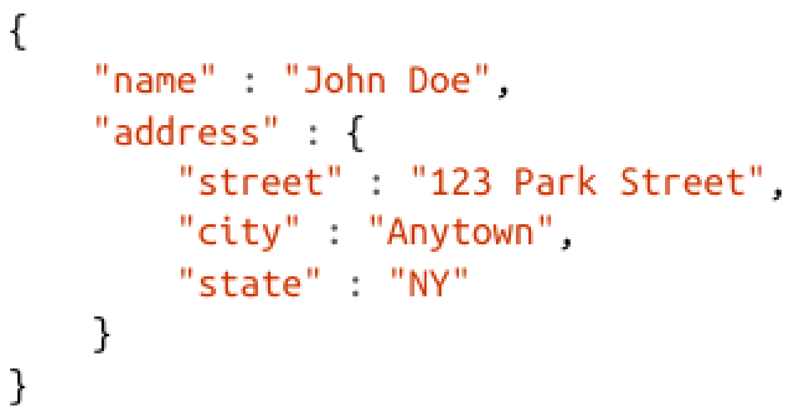


```
improvements and to suggest MongoDB products and deployment options to y
ou.

To enable free monitoring, run the following command: db.enableFreeMonit
oring()
To permanently disable this reminder, run the following command: db.disa
bleFreeMonitoring()
---

> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use GeeksforGeeks
switched to db GeeksforGeeks
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

DATATYPE: Basically each document will be in JSON format which will be as follows. Where each attributes inside can be of multiple data types.



```
{
    "name" : "John Doe",
    "address" : {
        "street" : "123 Park Street",
        "city" : "Anytown",
        "state" : "NY"
    }
}
```

Document:

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.

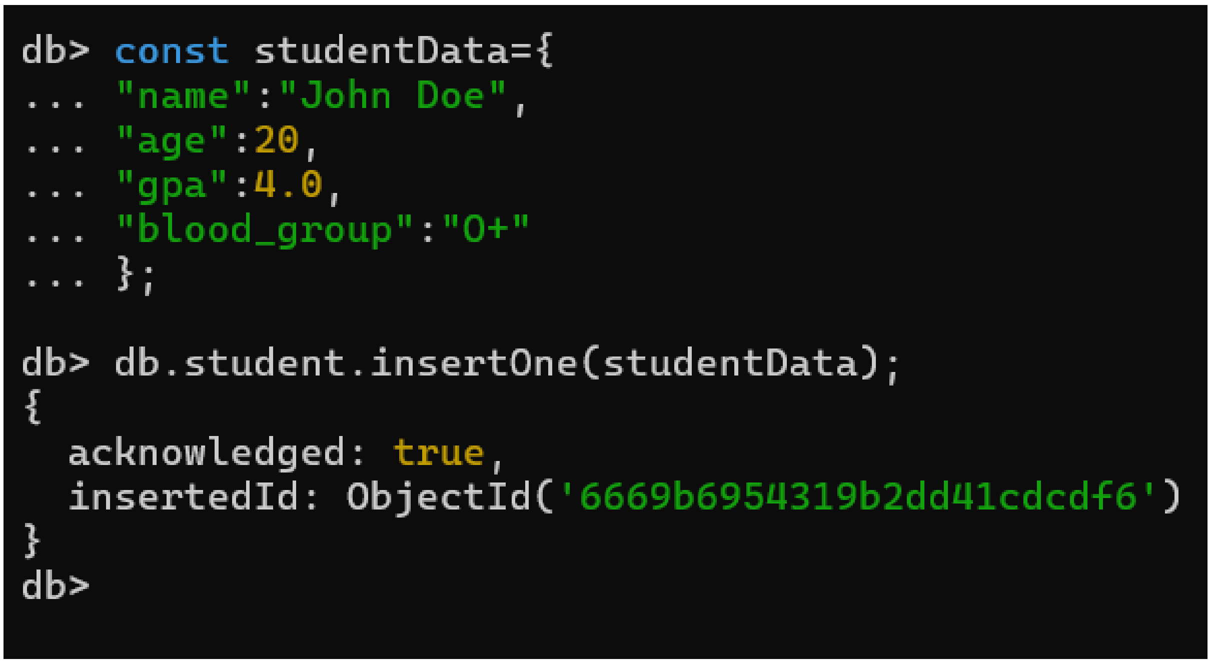
CRUD OPERATIONS:

C
Create
R
Read
U
Update
D
Delete

---

1. Create Operations:

The **create or insert operations** are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.

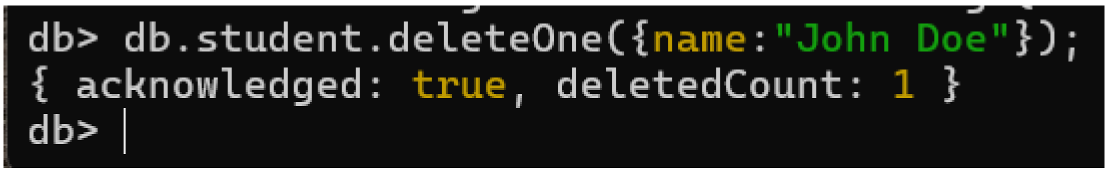


```
db> const studentData={
... "name":"John Doe",
... "age":20,
... "gpa":4.0,
... "blood_group":"O+"
... };

db> db.student.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6669b6954319b2dd41cdcdf6')
}
db>
```

2. Delete Operations:

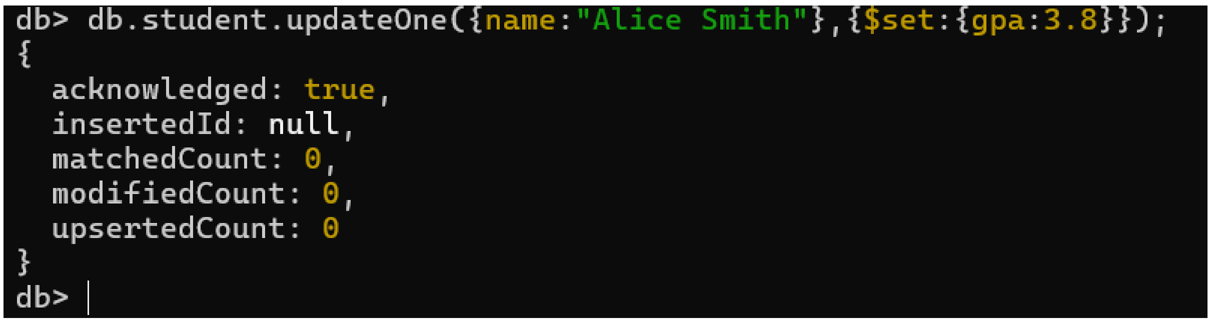
The delete operation are used to delete or remove the documents from a collection.



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

3. Update Operations

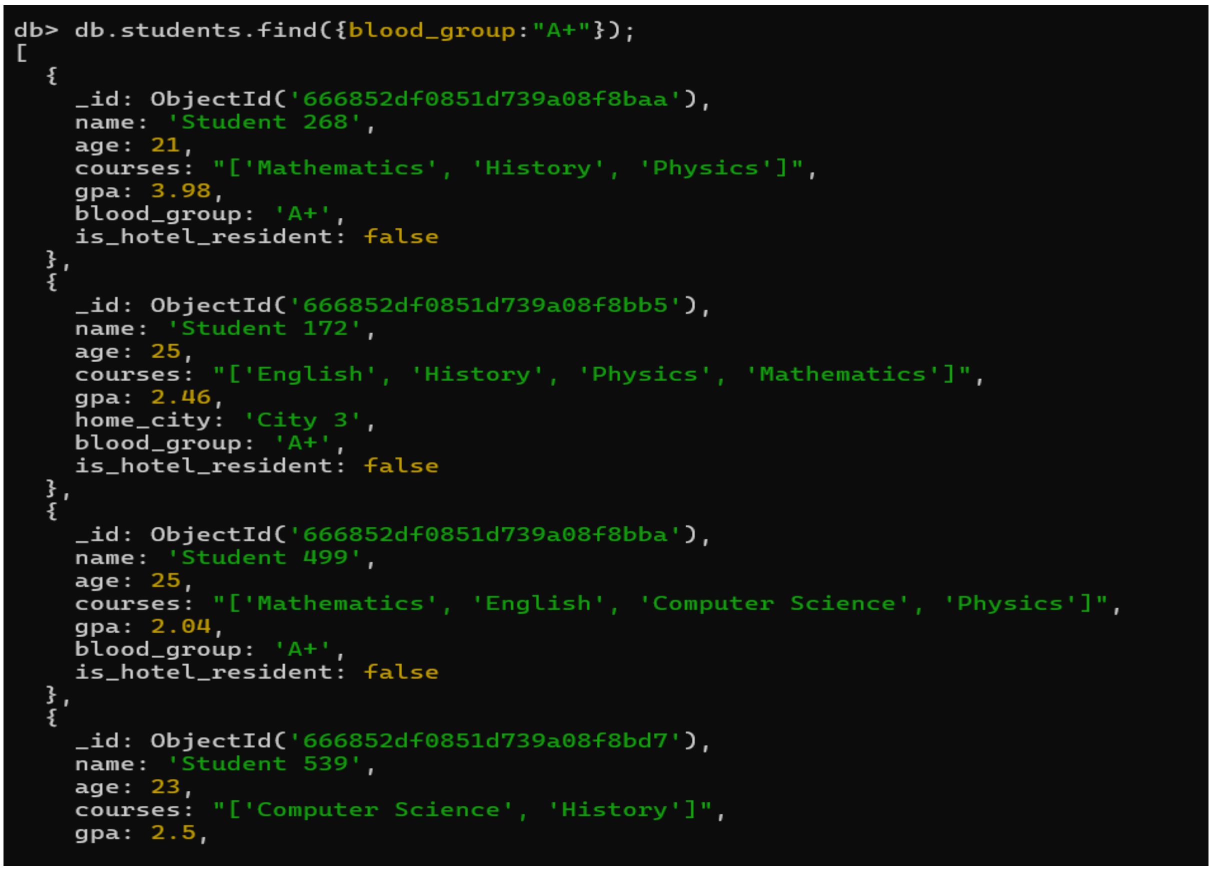
The update operations are used to update or modify the existing document in the collection.



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

WHERE:

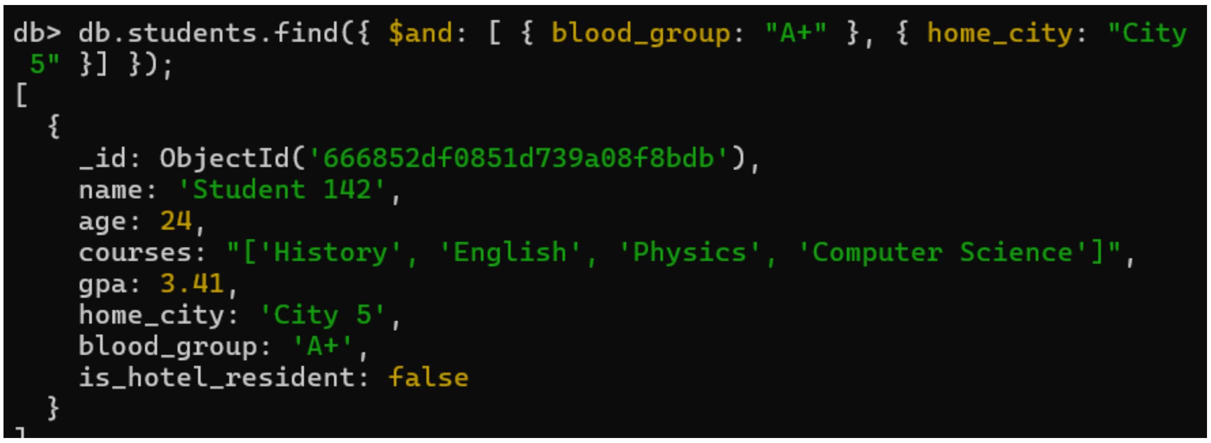
Given a Collection you want to FILTER a subset based on a condition. That is the place WHERE is used.



```
db> db.students.find({blood_group:"A+"});
[
  {
    _id: ObjectId('666852df0851d739a08f8baa'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8bb5'),
    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('666852df0851d739a08f8bba'),
    name: 'Student 499',
    age: 25,
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']",
    gpa: 2.04,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8bd7'),
    name: 'Student 539',
    age: 23,
    courses: "['Computer Science', 'History']",
    gpa: 2.5,
```

AND operator ($and):

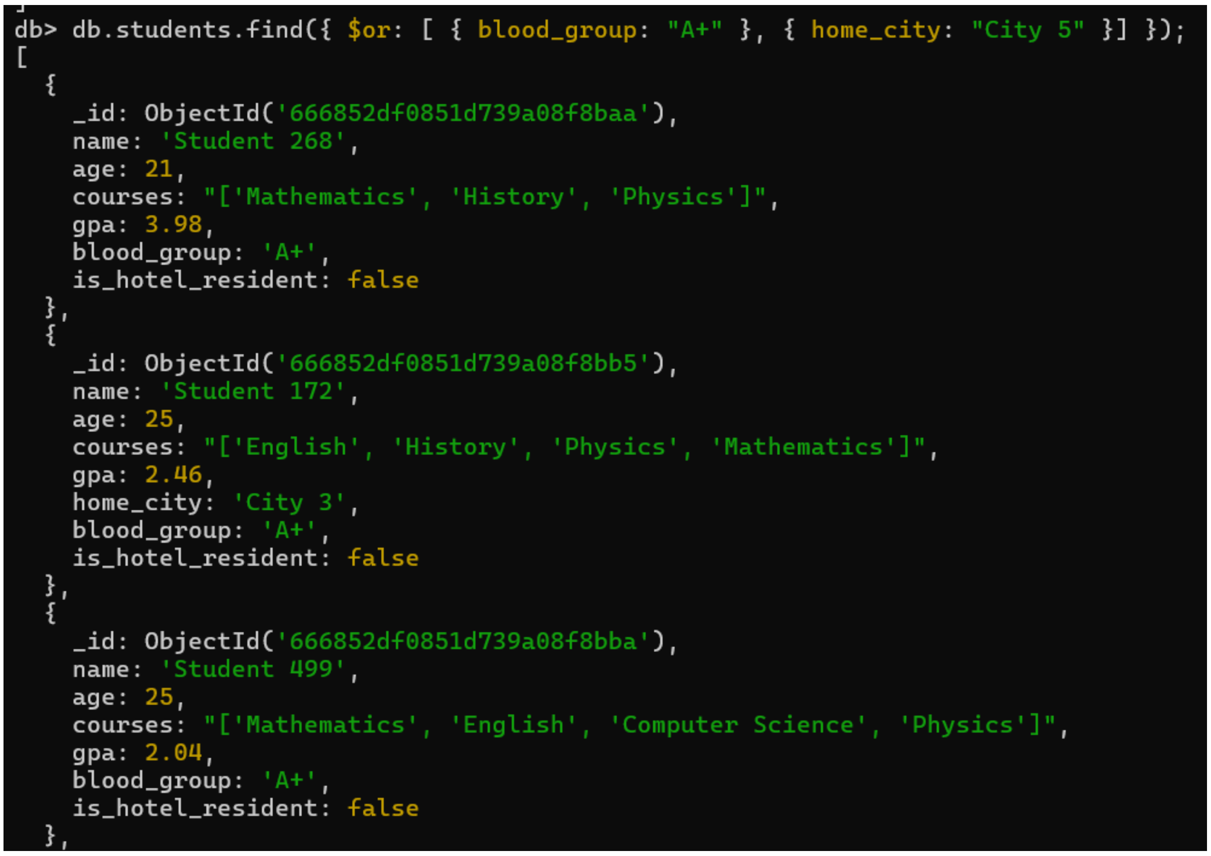
This operator is used to perform logical AND operation on the array of one or more expressions and select or retrieve only those documents that match all the given expression in the array.



```
db> db.students.find({ $and: [ { blood_group: "A+" }, { home_city: "City
 5" }] });
[
  {
    _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
  }
]
```

OR operator ($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.



```
db> db.students.find({ $or: [ { blood_group: "A+" }, { home_city: "City 5" }] });
[
  {
    _id: ObjectId('666852df0851d739a08f8baa'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8bb5'),
    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('666852df0851d739a08f8bba'),
    name: 'Student 499',
    age: 25,
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']",
    gpa: 2.04,
    blood_group: 'A+',
    is_hotel_resident: false
  },
```

ObjectId in MongoDB:

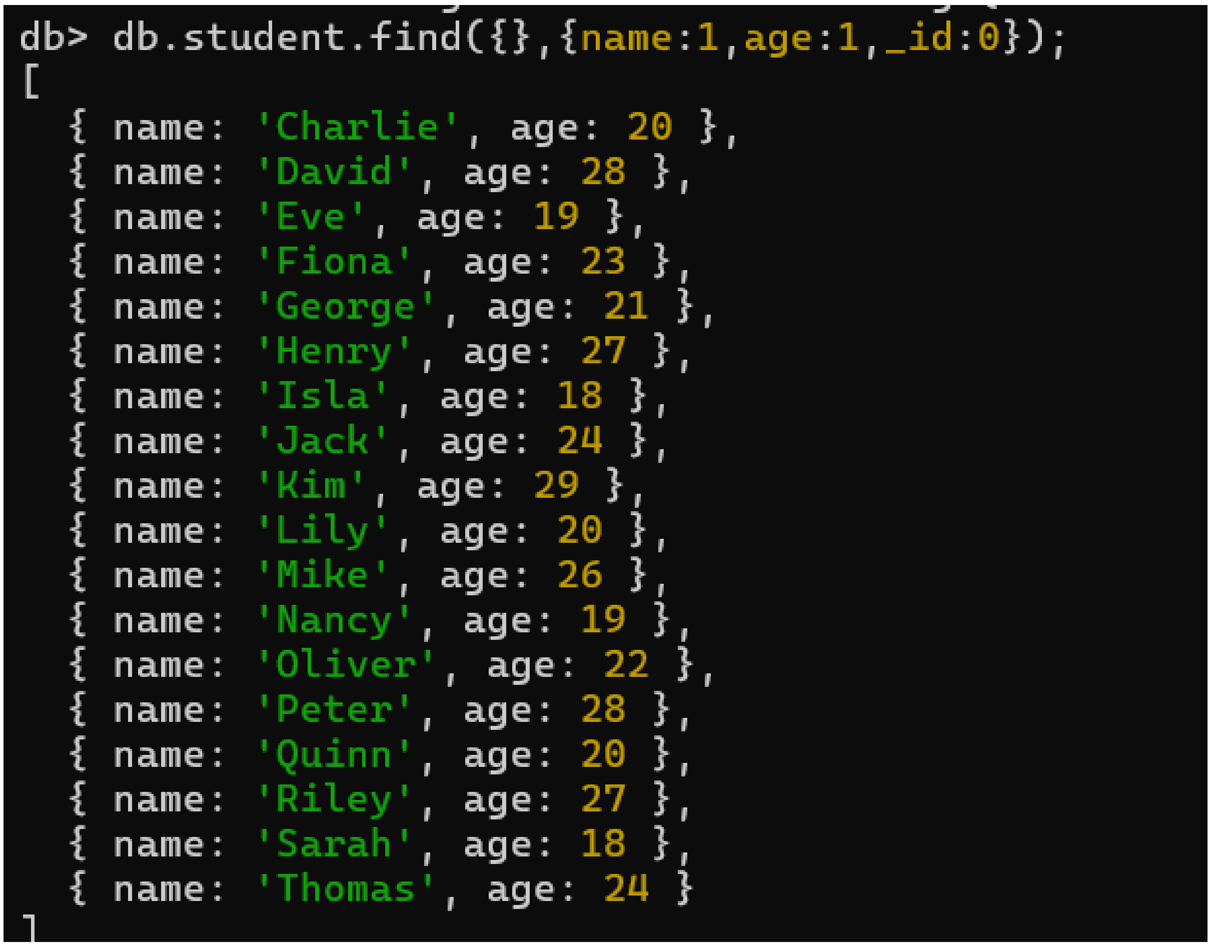
Every document in the collection has an "_id" field that is used to uniquely identify the document in a particular collection it acts as the primary key for the documents in the collection. "_id" field can be used in any format and the default format is ObjectId of the document.
An ObjectID is a 12-byte Field Of BSON type

- The first 4 bytes representing the Unix Timestamp of the document
- The next 3 bytes are the machine Id on which the MongoDB server is running.
- The next 2 bytes are of process id

- The last Field is 3 bytes used for increment the objectid.

Projection:

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



```
db> db.student.find({},{name:1,age:1,_id:0});
[
  { name: 'Charlie', age: 20 },
  { name: 'David', age: 28 },
  { name: 'Eve', age: 19 },
  { name: 'Fiona', age: 23 },
  { name: 'George', age: 21 },
  { name: 'Henry', age: 27 },
  { name: 'Isla', age: 18 },
  { name: 'Jack', age: 24 },
  { name: 'Kim', age: 29 },
  { name: 'Lily', age: 20 },
  { name: 'Mike', age: 26 },
  { name: 'Nancy', age: 19 },
  { name: 'Oliver', age: 22 },
  { name: 'Peter', age: 28 },
  { name: 'Quinn', age: 20 },
  { name: 'Riley', age: 27 },
  { name: 'Sarah', age: 18 },
  { name: 'Thomas', age: 24 }
]
```

For Ignoring Attributes:

We use the following below command food ignoring the attributes.

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

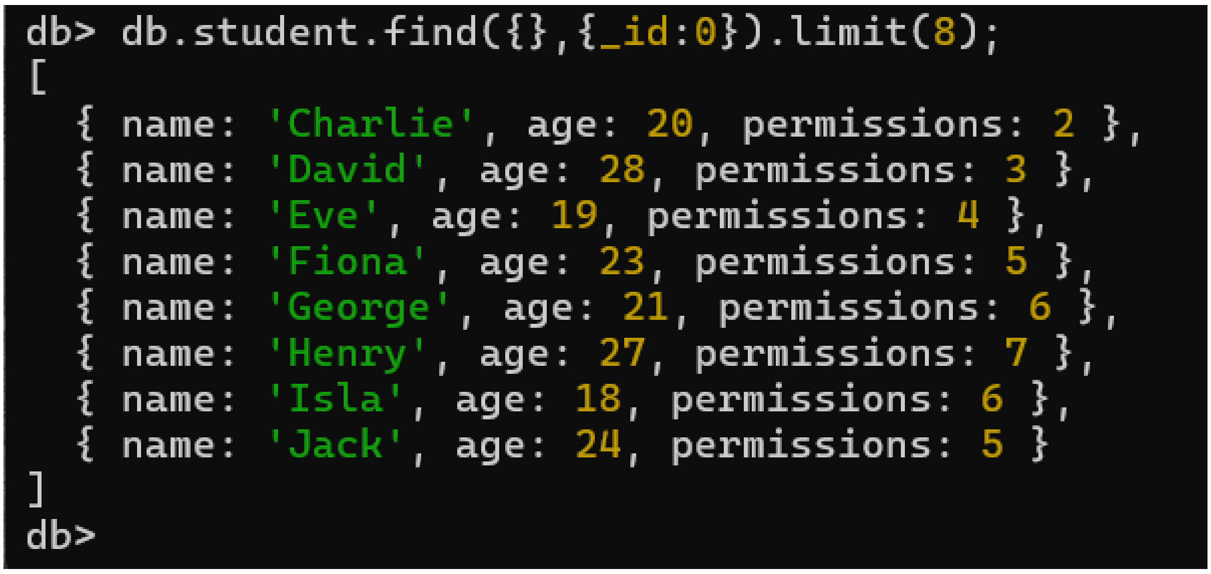
Benefits of Projection:

- 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:

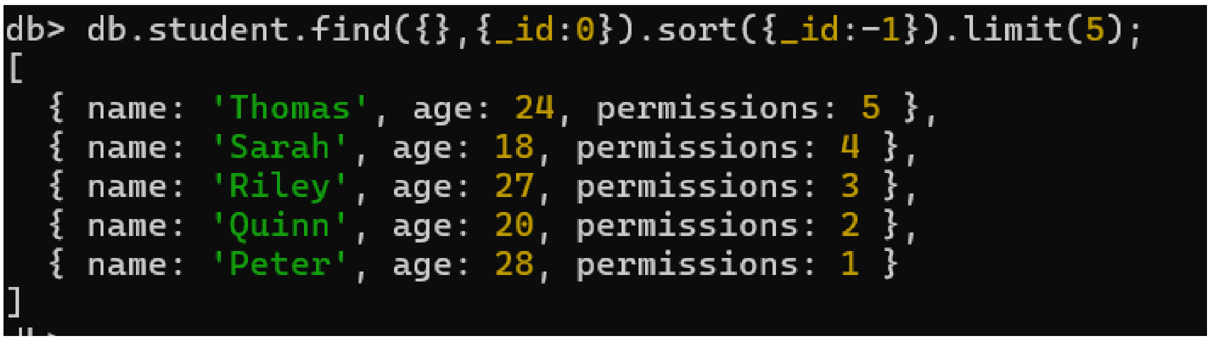
The limit() method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want. Or in other words, this method uses on cursor to specify the maximum number of documents/ records the cursor will return.

Getting first 8 documents:



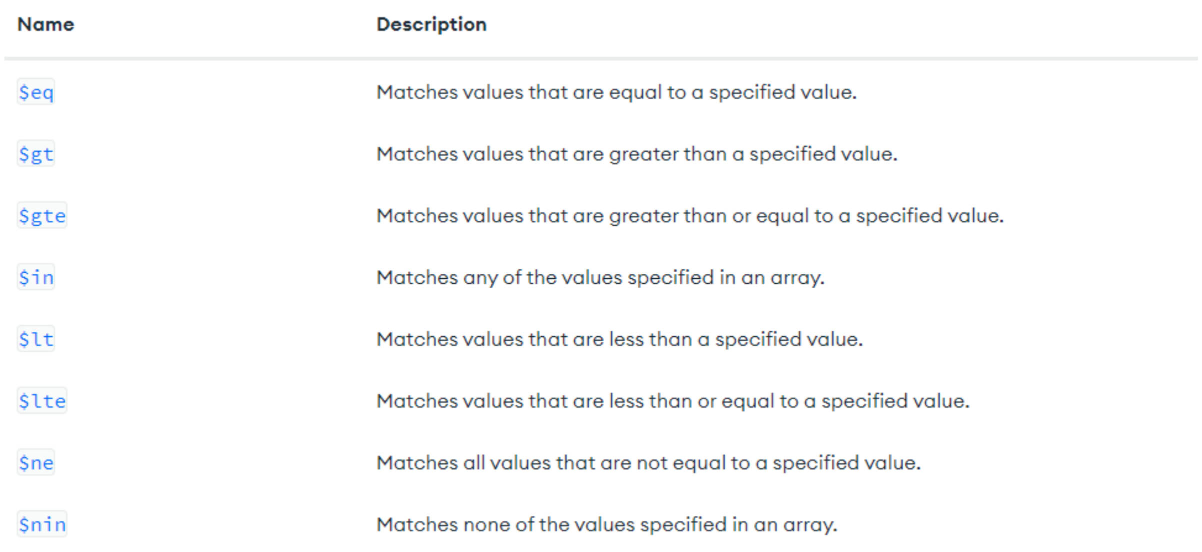
```
db> db.student.find({},{_id:0}).limit(8);
[
  { name: 'Charlie', age: 20, permissions: 2 },
  { name: 'David', age: 28, permissions: 3 },
  { name: 'Eve', age: 19, permissions: 4 },
  { name: 'Fiona', age: 23, permissions: 5 },
  { name: 'George', age: 21, permissions: 6 },
  { name: 'Henry', age: 27, permissions: 7 },
  { name: 'Isla', age: 18, permissions: 6 },
  { name: 'Jack', age: 24, permissions: 5 }
]
db>
```

To get Top 10 results in the Data:



```
db> db.student.find({},{_id:0}).sort({_id:-1}).limit(5);
[
  { name: 'Thomas', age: 24, permissions: 5 },
  { name: 'Sarah', age: 18, permissions: 4 },
  { name: 'Riley', age: 27, permissions: 3 },
  { name: 'Quinn', age: 20, permissions: 2 },
  { name: 'Peter', age: 28, permissions: 1 }
]
```

SELECTORS:



Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

- $gt selects those documents where the value of the specified field is greater than (i.e. >) the specified value.

SYNTAX:

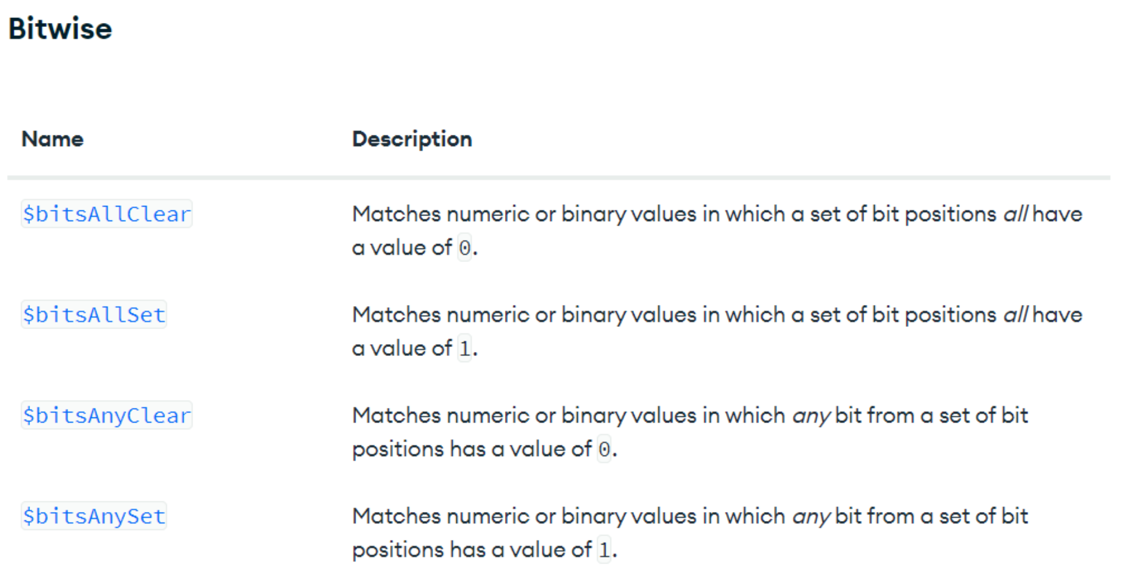
{ field: { $gt: value } }

- $lt selects the documents where the value of the field is less than (i.e. <) the specified value.

SYNTAX:

{ field: { $lt: value } }

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`.

GEOSPATIAL:

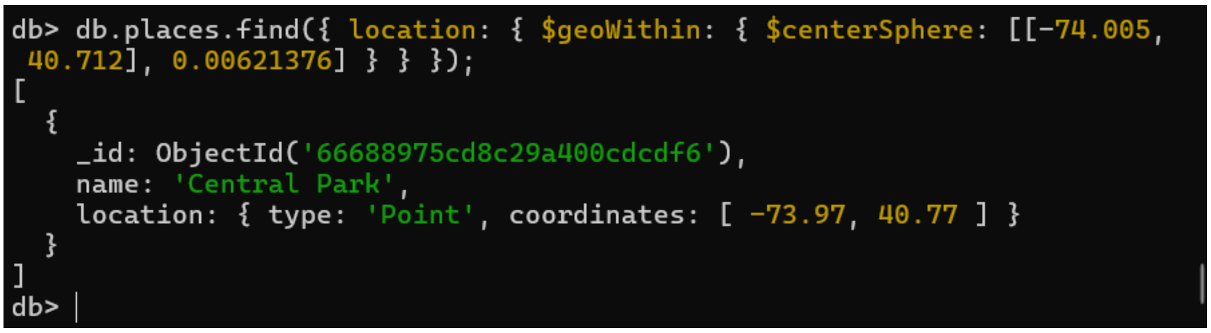
Geospatial Data:

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

GeoJSON Objects:

To calculate geometry over an Earth-like sphere, store your location data as GeoJSON objects.
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.



```
db> db.places.find({ location: { $geoWithin: { $centerSphere: [[-74.005,
 40.712], 0.00621376] } } });
[
  {
    _id: ObjectId('66688975cd8c29a400cdcdf6'),
    name: 'Central Park',
    location: { type: 'Point', coordinates: [ -73.97, 40.77 ] }
  }
]
db> 
```

CONCLUSION:

MongoDB is a popular NoSQL database that offers flexibility, scalability, and performance for modern applications. Its key features include document-oriented storage, dynamic schema, high availability, and horizontal scalability through sharding.

In conclusion, MongoDB is well-suited for use cases such as real-time analytics, content management, mobile apps, and IoT applications where flexibility, scalability, and fast iteration are essential. However, like any technology, it's important to carefully consider your specific requirements and use cases before choosing MongoDB or any other database solution.