Mongo DB

Report

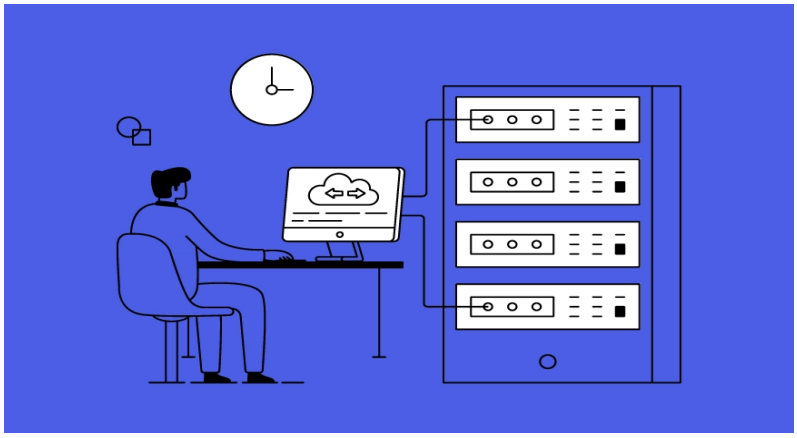
Introduction:

What is Database?

`A database is a collection of data that is organized, which is also called structured data. It can be accessed or stored in a computer system. It can be managed through a Database Management System (DBMS), a software used to manage data. Database refers to related data in a structured form.

How do Document Databases Work?

A document database has information retrieved or stored in the form of a document or other words semi-structured database. Since they are non-relational, so they are often referred to as NoSQL data.



The document database fetches and accumulates data in forms of key-value pairs but here, the values are called as Documents. A document can be stated as a complex data structure. Document here can be a form of text, arrays, strings, JSON, XML, or any such format. The use of nested documents is also very common. It is very effective as most of the data created is usually in the form of JSON and is unstructured.

What Is Database Management?

A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database.

DBMS provides an environment to store and retrieve the data in coinvent and efficient manner.

Key Features of DBMS

- **Data modeling:** A DBMS provides tools for creating and modifying data models, which define the structure and relationships of the data in a database.
- **Data storage and retrieval:** A DBMS is responsible for storing and retrieving data from the database, and can provide various methods for searching and querying the data.
- **Concurrency control:** A DBMS provides mechanisms for controlling concurrent access to the database, to ensure that multiple users can access the data without conflicting with each other.
- **Data integrity and security:** A DBMS provides tools for enforcing data integrity and security constraints, such as constraints on the values of data and access controls that restrict who can access the data.
- **Backup and recovery:** A DBMS provides mechanisms for backing up and recovering the data in the event of a system failure.
- **DBMS can be classified into two types:** Relational Database Management System (RDBMS) and Non-Relational Database Management System (NoSQL or Non-SQL)
- **RDBMS:** Data is organized in the form of tables and each table has a set of rows and columns. The data are related to each other through primary and foreign keys.

- **NoSQL:** Data is organized in the form of key-value pairs, documents, graphs, or column-based. These are designed to handle large-scale, high-performance scenarios.

MongoDB:

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).

Features of Mongo DB:

- **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.

- **Replication and High Availability**: MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.

- **Aggregation**: Aggregation operations process data records and return the computed results. It is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

Installation:

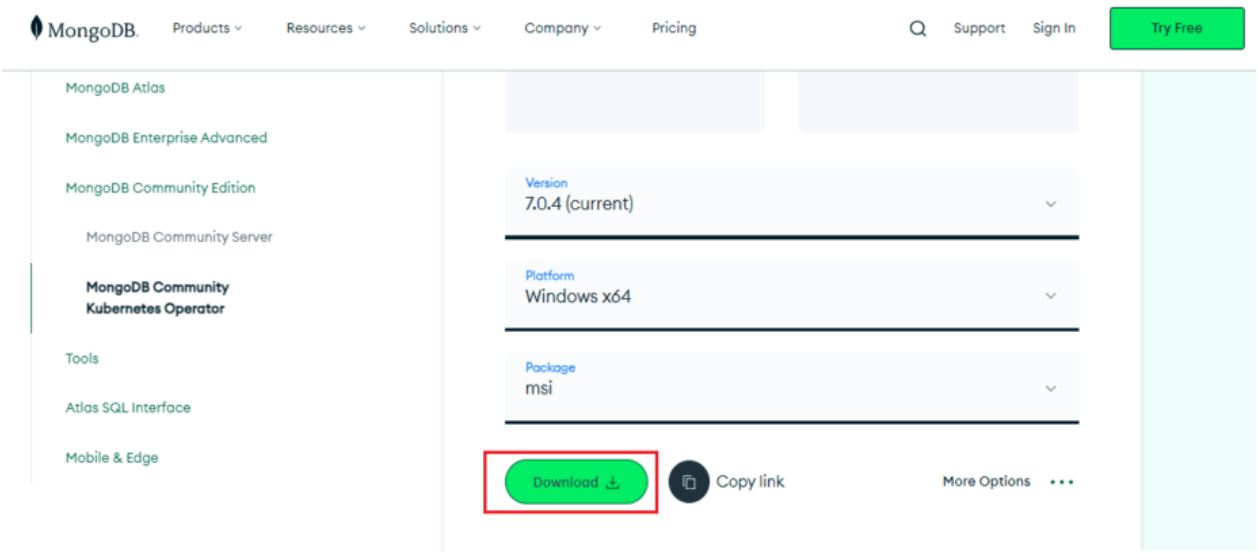
Requirements to Install MongoDB on Windows:

- MongoDB 4.4 and later only support 64-bit versions of Windows.
- MongoDB 7.0 Community Edition supports the following 64-bit versions of Windows on x86_64 architecture:
  - Windows Server 2022
  - Windows Server 2019
  - Windows 11
- Ensure that the user is running mongod and mongos has the necessary permissions from the following groups:
  - Performance Monitor Users
  - Performance Log Users

Steps to Install MongoDB on Windows using MSI:

To install MongoDB on Windows, first, download the MongoDB server and then install the MongoDB shell. The Steps below explain the installation process in detail and provide the required resources for the smooth download and install MongoDB.

Step 1: Go to the MongoDB Download Center to download the MongoDB Community Server.

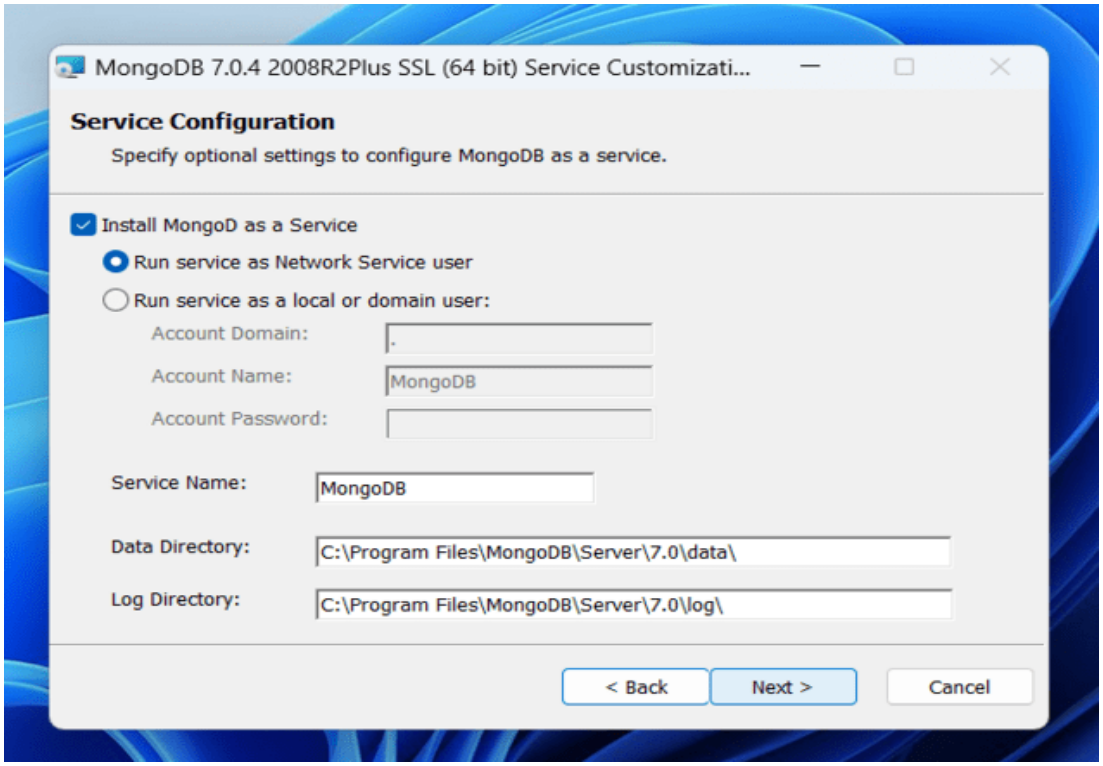


Here, You can select any version, Windows, and package according to your requirement. For Windows, we need to choose:

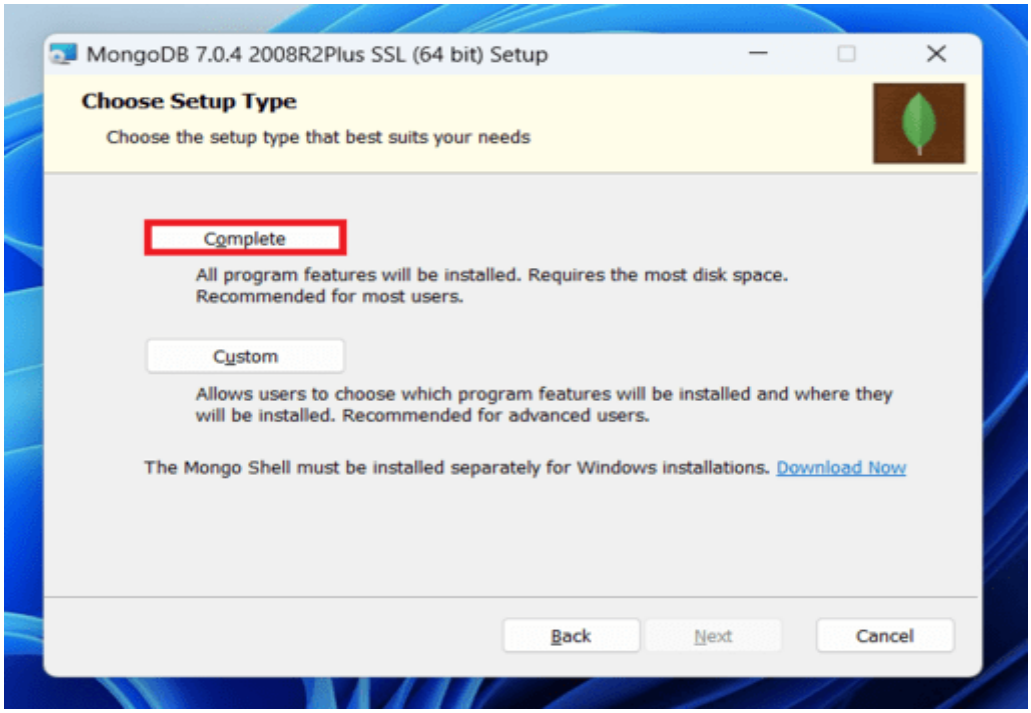
- Version: 7.0.4
- OS: Windows x64
- Package: msi

Step 2: When the download is complete open the msi file and click the *next button* in the startup screen:

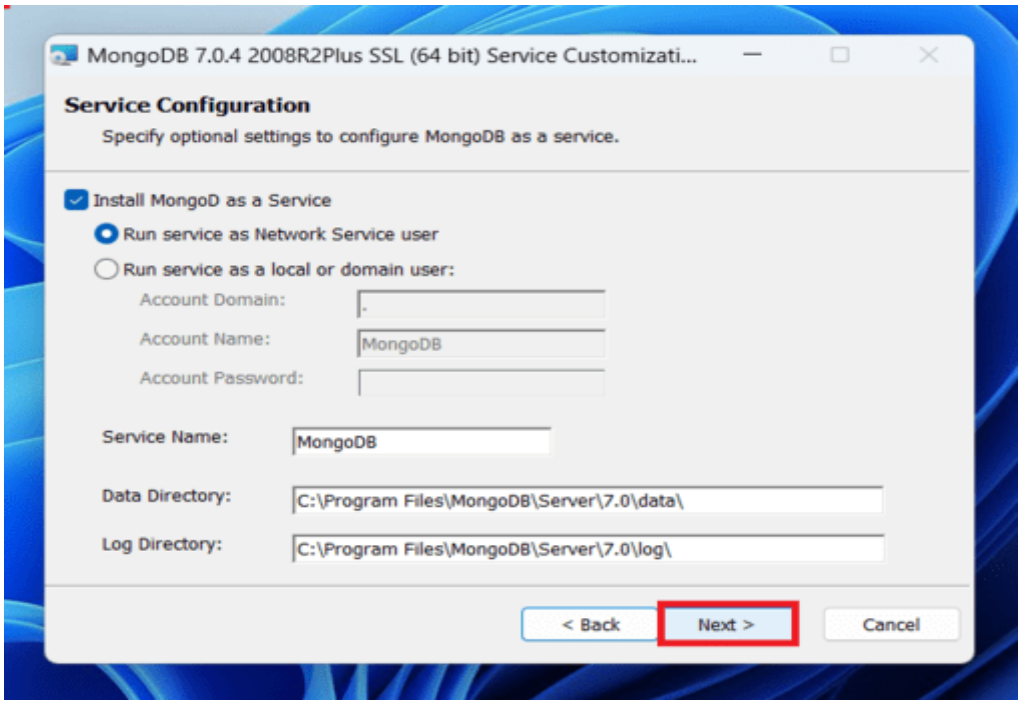
Step 3: Now accept the End-User License Agreement and click the next button:



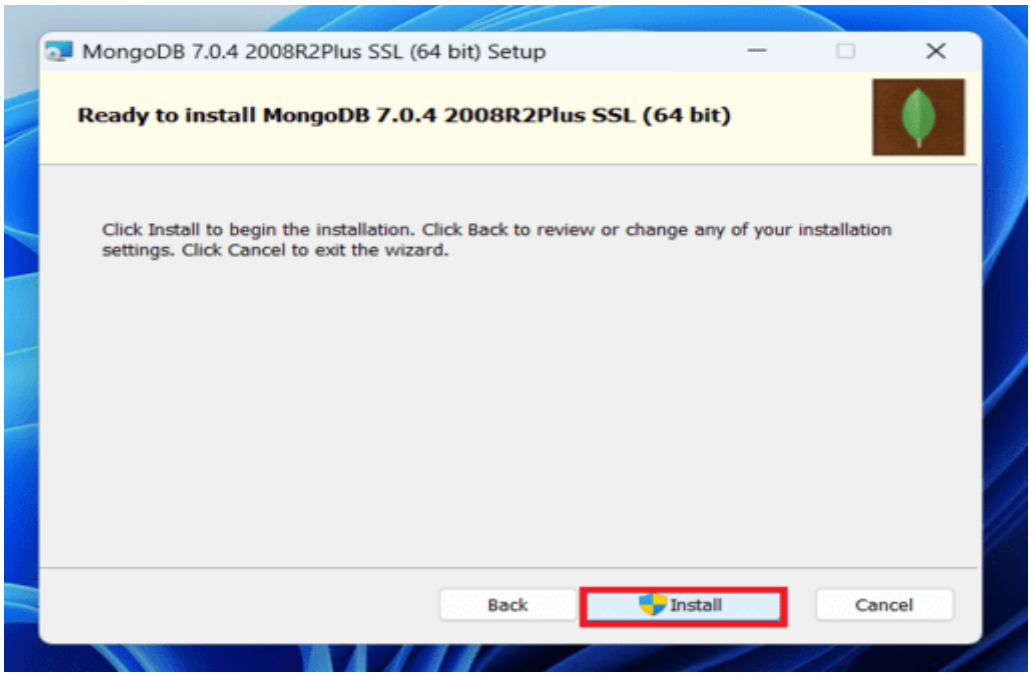
Step 4: Now select the *complete option* to install all the program features. Here, if you can want to install only selected program features and want to select the location of the installation, then use the *Custom option:*



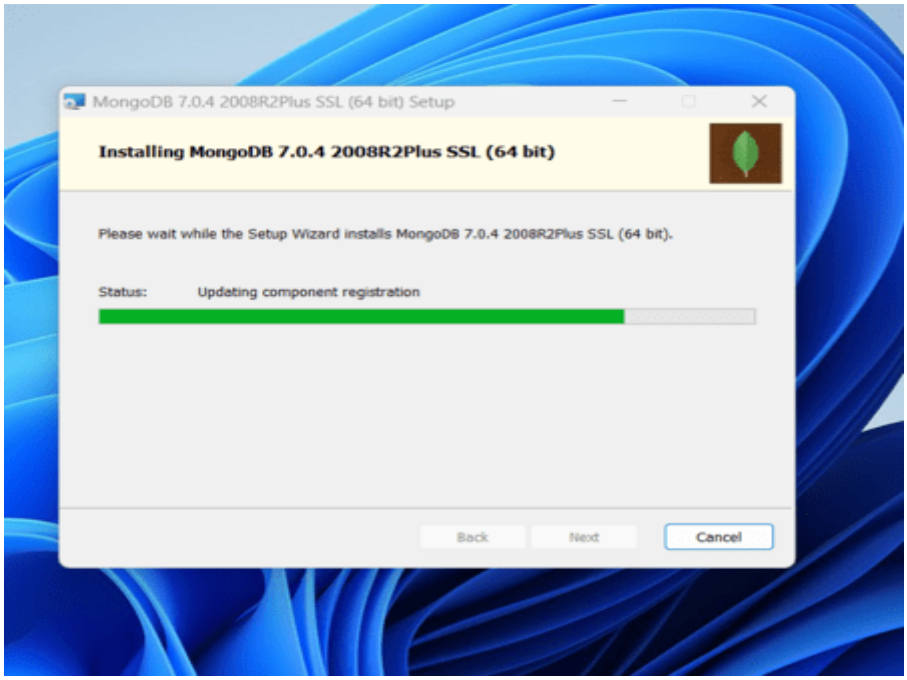
Step 5: Select "Run service as Network Service user" and copy the path of the data directory. Click Next:



Step 6: Click the *Install button* to start the MongoDB installation process:

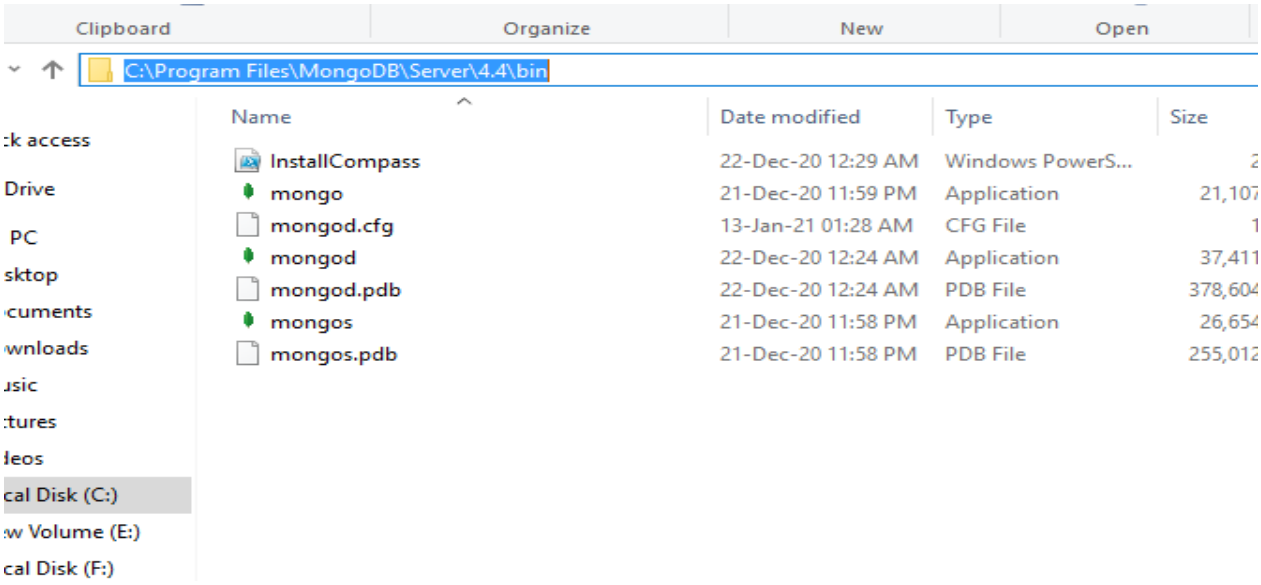


Step 7: After clicking on the install button installation of MongoDB begins:

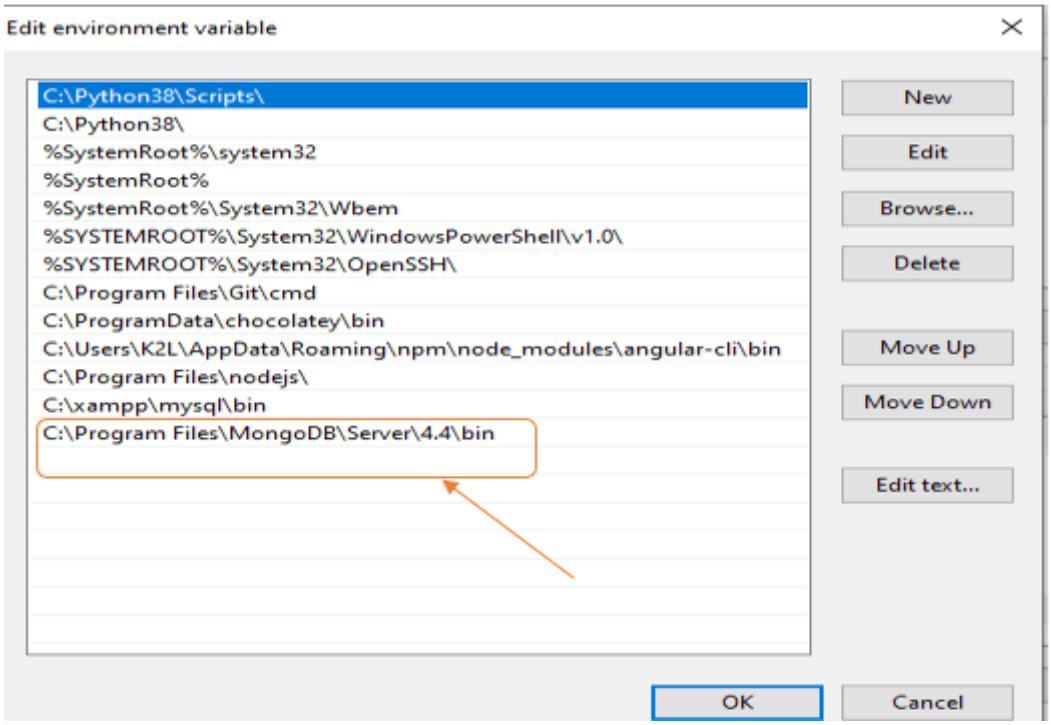


Step 8: Now click the *Finish button* to complete the MongoDB installation process:

Step 9: Now we go to the location where MongoDB installed in step 5 in your system and copy the bin path:



Step 10: 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:



Step 11: After setting the environment variable, we will run the MongoDB server, i.e. mongod. So, open the command prompt and run the following command:

mongod

When you run this command you will get an error i.e. *C:/data/db/ not found*.

Step 12: Now, Open C drive and create a folder named "data" inside this folder create another folder named "db". After creating these folders. Again open the command prompt and run the following command:

mongod

Now, this time the MongoDB server(i.e., mongod) will run successfully.

Run mongo Shell:

Step 13: 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.

Run MongoDB:

Now you can make a new database, collections, and documents in your shell

Run Commands:

Show Databases:

To display the database you are using, type db:

show dbs

The operation should return test, which is the default database.

Switch Databases:

To switch databases, issue the use <db> helper :

use<database>

Show All Tables:

To show all tables in database ,issue the following command:

show collections

Insertion Of Record Or Data:

Insert a data into collection or table and also create collection if it is not exist by issuing following command:

<database_name>.<collection_name>.insert({attribute:value})

Insertion Of Multiple Record Or Data:

To insert multiple data into collection ,we issue following command:

<database_name>.<collection_name>.batchInsert([{attribute1:value1},{attribute2:value2}……..,{attribute-n:value-n}])

Print All The Rows:

To print the rows of tables following command is issued as follows:

<database_name>.<collection_name>.find()

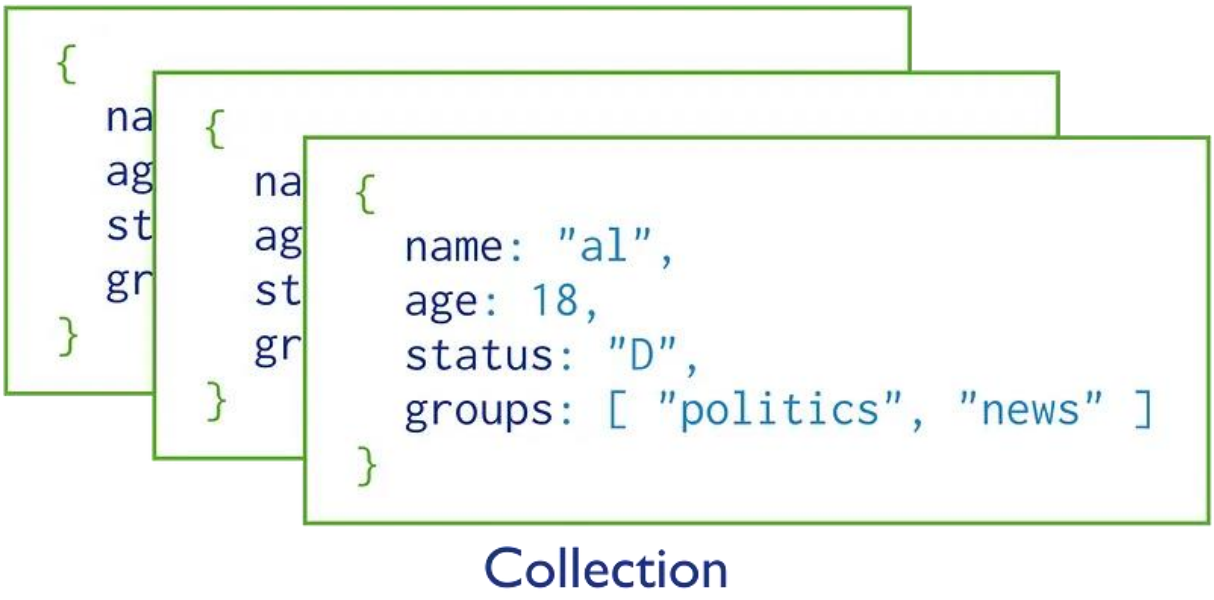
This function also can be used to get particular row from table later on we will come across this.

Databases:

In MongoDB, databases hold one or more collections of documents. To select a database to use, in `mongosh`, issue the `use <db>` statement

Collections:

MongoDB stores documents in collections. Collections are analogous to tables in relational databases.



Datatypes:

MongoDB Server stores data using the JSON format which supports some additional data types that are not available using the JSON format.

Date:

`mongosh` provides various methods to return the date, either as a string or as a `Date` object:

- `Date()` method which returns the current date as a string.
- `new Date()` :constructor which returns a `Date` object using the `ISODate()` wrapper.
- `ISODate()` :constructor which returns a `Date` object using the `ISODate()` wrapper.

Int32:

If a number can be converted to a 32-bit integer, `mongosh` will store it as `Int32`. If not, `mongosh` defaults to storing the number as a `Double`. Numerical values that are stored as `Int32` in `mongosh` would have been stored by default as `Double` in the `mongo` shell.

Long:

The Long() constructor can be used to explicitly specify a 64-bit integer.

Decimal128:

Decimal128() values are 128-bit decimal-based floating-point numbers that emulate decimal rounding with exact precision.

Dataset

The below dataset consists of students data which consists of information about students's age , gpa, home city ,courses etc.,

Load the student data set here link

WHERE

Given a Collection you want to FILTER a subset based on a condition. That is the place WHERE is used .For queries that cannot be done any other way, there are "$where" clauses, which allow you to execute arbitrary JavaScript as part of your query. This allows you to do (almost) anything within a query. For security, use of "$where" clauses should be highly restricted or eliminated. End users should never be allowed to execute arbitrary "$where" clauses.

Following example query demonstrates the data retrival based on condition ,here we are retrieving all the students whose age is greater than 19:

```
db> db.student.find({age:{$gt:19}});
```

We can also see the count of the data or students details retrieved using count() function and below query is example for such query:

```
db> db.student.find({age:{$gt:19}}).count();
376
```

AND:

Given a Collection you want to FILTER a subset based on multiple conditions.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.

Following example query demonstrates the data retrieval from collection **student** of database **db** using $and operator, which retrieves the data of those students whose blood group id A+ and home city is City 5:

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

OR:

Given a Collection you want to FILTER a subset based on multiple conditions but Any One is Sufficient .You can use this operator in methods like find(), update(), etc. according to your requirements. You can also use this operator with text queries, GeoSpatial queries, and sort operations. When MongoDB evaluating the clauses in the $or expression, it performs a collection scan.

Following example query demonstrates the data retrieval from collection **student** of database **db** using $or operator, which retrieves the data of those students either whose gpa is 3 or home city is City 5:

```
db> db.student.find(
... { $or:[
... {gpa:3.0},
... {home_city:"City 5"}
... ]
... });
```

CRUD Operations:

CRUD operations create, read, update, and delete documents.

Create Operation:

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

- db.collection.insertOne()
- db.collection.insertMany()

In MongoDB, insert operations target a single collection.

For example, here insertOne() is used to insert single data into collection “student” of database db :

```
db> db.student.insertOne({
... "name":"Sumith",
... "age":22,
... "courses":["Mathematics","Computer science","History"],
... "gpa":3.0,
... "home_city":"bengaluru",
... "blood_group":"A+",
... "is_hotel_resident":false
... });
{
  acknowledged: true,
  insertedId: ObjectId('66869c618e63eb0464cc8988')
}
```

For example, here insertMany() is used to insert multiple student data into collection "student" of database db :

```
db> db.student.insertMany({
... "name": "ranjith",
... "age": 22,
... "courses": ["Mathematics", "Computer science", "History"],
... "gpa": 3.9,
... "home_city": "bengaluru",
... "blood_group": "O+",
... "is_hotel_resident": true
... },
... {
... "name": "sujith",
... "age": 19,
... "courses": ["Mathematics", "Computer science", "Political Science"],
... "gpa": 3.4,
... "home_city": "chikkamagaluru",
... "blood_group": "O-",
... "is_hotel_resident": true
... });
```

```
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('66869fd78e63eb0464cc8989') }
}
```

Read operation:

Read operations retrieve documents from a collection i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:

- db.collection.find()

You can specify query filters or criteria that identify the documents to return.

Following example shows the query which is used to read data ,here using find() function is used to retrieve data :

```
db> db.student.find({age:{$gt:19}});
```

Update Operation:

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- db.collection.updateOne()
- db.collection.updateMany()

In MongoDB, update operations target a single collection.

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

Following example query shows how updateOne() function is used to update values of student data of collection:

```
db> db.student.updateOne(
... {age:19},{$set:{gpa:3.6}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Following example query shows how updateMany() function is used to update values of student data of collection:

```
db> db.student.updateMany(
... {age:19,gpa:{$lt:3}},{$set:{blood_group:"O+"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 33,
  modifiedCount: 30,
  upsertedCount: 0
}
```

Delete Operation:

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

- db.collection.deleteOne()
- db.collection.deleteMany()

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

Following example query demonstrates deletion operation using deleteOne() :

```
db> db.student.deleteOne({name:"sujith"});
{ acknowledged: true, deletedCount: 0 }
```

Following example query demonstrates deletion operation using deleteMany() :

```
db> db.student.deleteMany({home_city:"City 5"});
{ acknowledged: true, deletedCount: 40 }
```

PROJECTION:

Given a Collection you want to FILTER a subset of attributes. That is the place Projection is used. By default, queries in MongoDB return all fields in matching documents. To limit the amount of data that MongoDB sends to applications, you can include a projection document to specify or restrict fields to return.

- Use the projection document as the second argument to the find method
- A projection can explicitly include several fields by setting the <field> to 1 in the projection document.
- Instead of listing the fields to return in the matching document, you can use a projection to exclude specific fields

For example ,we want to retrieve only name and age of students we use projection:

```
db> db.student.find({},{name:1,age:1});
```

Return All But the Excluded Fields:

Instead of listing the fields to return in the matching document, you can use a projection to exclude specific fields.

We have to note tht, With the exception of the _id field, you cannot combine inclusion and exclusion statements in projection documents.

For example ,we want to retrieve only name and age of students and also want to exclude id we use projection with exclusion:

```
db> db.student.find({},{name:1,age:2,_id:0});
```

Also if we don’t want only _id field in such case we use exclusion to ignore _id attribute, for example:

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

$slice Operator:

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

The slice operation is used to retrieves the specific attributes from nested objects.

Following example shows how $slice operator is used to retrieve specific data from nested array,

Here we want to retrive first course of all students :

```
db> db.student.find( {}, { name:1, courses:{$slice:1}});
```

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)

For example ,we want to limit the number of results to five we use limit as following:

```
db> db.student.find({},{name:1}).limit(5);
```

Lets say we want to fetch top 5 student who scored highest gpa, the following query retrieves the data:

```
db> db.student.find({},{name:1}).sort({gpa:1}).limit(5);
```

SELECTORS:

MongoDB provides various query selectors for complex queries:

- Comparison Operators ($eq, $gt, $lt, etc.)
- Logical Operators ($and, $or, $not, $nor)
- Element Operators ($exists, $type)
- Array Operators ($in, $all, $elemMatch)

For example ,we want to retrieve users with age greater than 20 :

```
db> db.student.find({age: { $gt:20}});
```

For example ,we want to retrieve users with age less than 20 :

```
db> db.student.find({age: { $lt:20}});
```

For example ,if we want to find student from city 2 and blood group B+:

```
db> db.student.find({
... $and:[
... { home_city:"City 2"},
... {blood_group:"B+"},
... ]
... });
```

For example ,if we want to find student who are hotel resident or whose gpa is less than 3:

```
db> db.student.find({
... $or:[
... { is_hotel_resident:true},
... { gpa:{$lt:30}}
... ]
... });
```

$elemMatch and $eq:

The $elemMatch operator limits the contents of an <array> field from the query results to contain only the first element matching the $elemMatch condition.

The $eq operator matches documents where the value of a field equals the specified value which specifies equality condition.

Lets take new data set , create a new collection called **candidates** :link

For example, we want to find the candidates who enrolled in Mathematics with specific projection:

```
db> db.candidates.find({courses: {$elemMatch:{$eq :"Mathematics"}}},{name:1,"courses.$":1});
```

Explanation: Collection name: students_permission

- **name:** Student's name (string)
- **age:** Student's age (number)
- **permissions:** Bitmask representing user permissions (number)

Bitwise Operators:

Bitwise operators return data based on bit position conditions.

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`.

- In our example its a 32 bit each bit representing different things
- Bitwise value 7 means all access 7 -> 111

Bit 1	Bit 2	Bit 3
cafe	campus	lobby

Here we are defining bit positions for permissions :

```
db> const lobby_permission=1;

db> const campus_permission=2;
```

The below query will find students who has both campus and lobby permission bitsAllset:

```
db> db.students_permission.find({
...  permissions:{$bitsAllSet:[lobby_permission,campus_permission]}
... });
```

Geospatial Query Operators:

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

MongoDB provides the following geospatial query operators:

Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The `2dsphere` and `2d` indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.

Create collection called “locations” using this json data set:link

If we want to fetch data within 1 km radian we use the following geospatial query:

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

AGGREGATION OPERATIONS:

Aggregation operations process multiple documents and return computed results. You can use aggregation operations to:

- Group values from multiple documents together.
- Perform operations on the grouped data to return a single result.
- Analyze data changes over time.

To perform aggregation operations, you can use:

- **Aggregation pipelines**, which are the preferred method for performing aggregations.
- **Single purpose aggregation methods**, which are simple but lack the capabilities of an aggregation pipeline.

Syntax:

db.collection.aggregate(<AGGREGATE OPERATION>)

Expression Type	**Description**	**Syntax**
Accumulators	Perform calculations on entire groups of documents	
$sum	Calculates the sum of all values in a numeric field within a group.	"$fieldName": { $sum: "$fieldName" }
$avg	Calculates the average of all values in a numeric field within a group.	"$fieldName": { $avg: "$fieldName" }
$min	Finds the minimum value in a field within a group.	"$fieldName": { $min: "$fieldName" }
$max	Finds the maximum value in a field within a group.	"$fieldName": { $max: "$fieldName" }

$push	Creates an array containing all unique or duplicate values from a field	"$arrayName": { $push: "$fieldName" }
$addToSet	Creates an array containing only unique values from a field within a group.	"$arrayName": { $addToSet: "$fieldName" }
$first	Returns the first value in a field within a group (or entire collection).	"$fieldName": { $first: "$fieldName" }
$last	Returns the last value in a field within a group (or entire collection).	"$fieldName": { $last: "$fieldName" }