
MongoDB®

MongoDB.

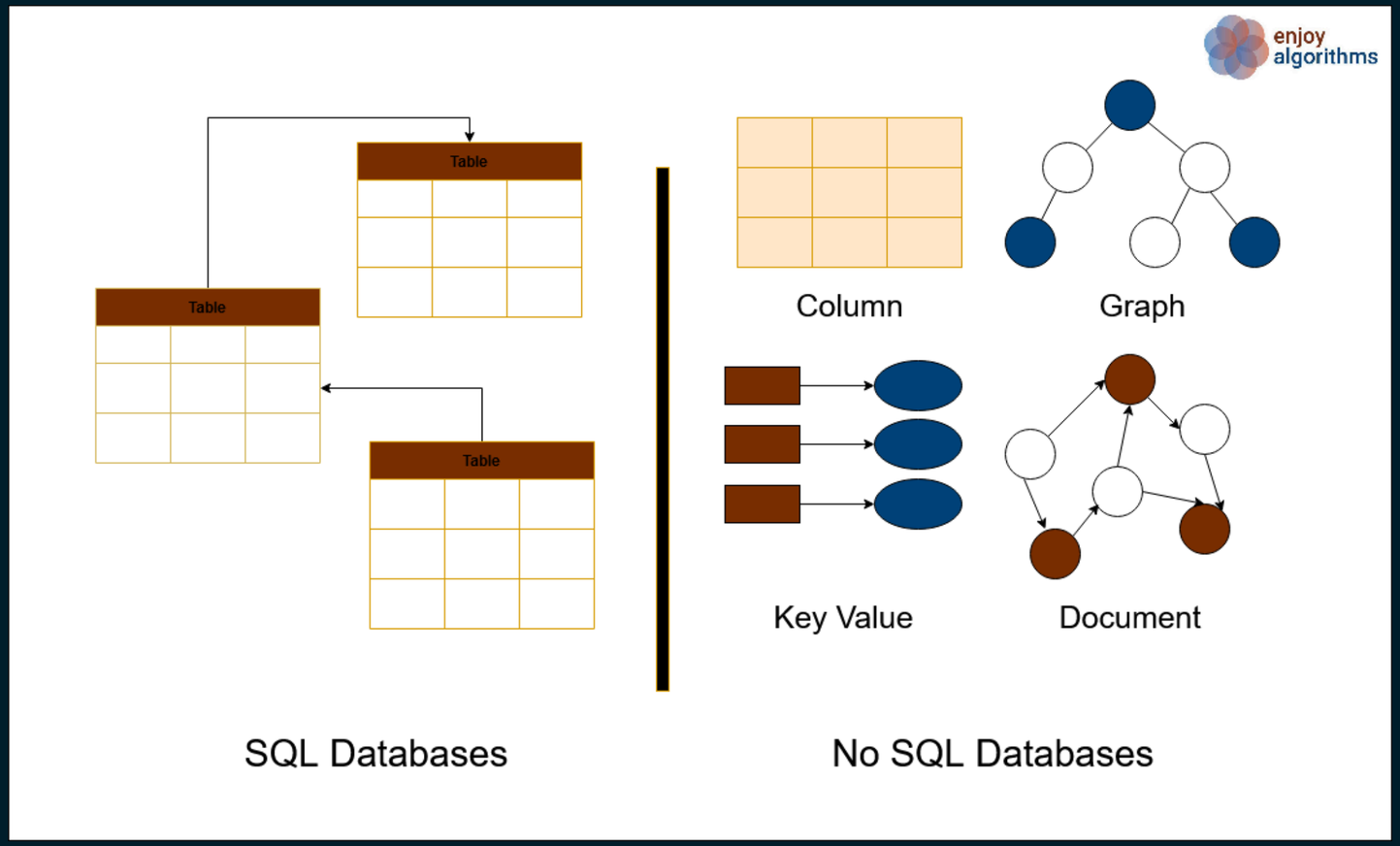
OUTLINE

1. Introduction and fundamentals of MongoDB
2. SQL vs MongoDB (NoSQL)
3. MongoDB Installation and Setup
4. Commands & CRUD operation
5. Practice
6. Queries
7. Indexes

MongoDB.

Introduction to MongoDB

- MongoDB is a NoSQL, document-oriented database designed for high performance, high availability, and easy scalability.



MongoDB.

JSON:

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.

```
{
  "city": "New York",
  "name": {
            ”first”: ”John”,
            “last”: “Cena”
          },
  “gender”: “M”,
  “friends”: [”joseph”,”jack”]
}
```

//accessing: ‘name.first’, ‘name.last’

MongoDB.

Remember! MongoDB works with objects (JSON) or array.
Every operation, query etc are written as an object.
So it’s important to understand JSON.

Also, remember that MongoDB is case-sensitive, unlike SQL. For example, ‘studentName’ and ‘studentname’ are treated differently.

Mongodb's operations are just like calling a function in programming.

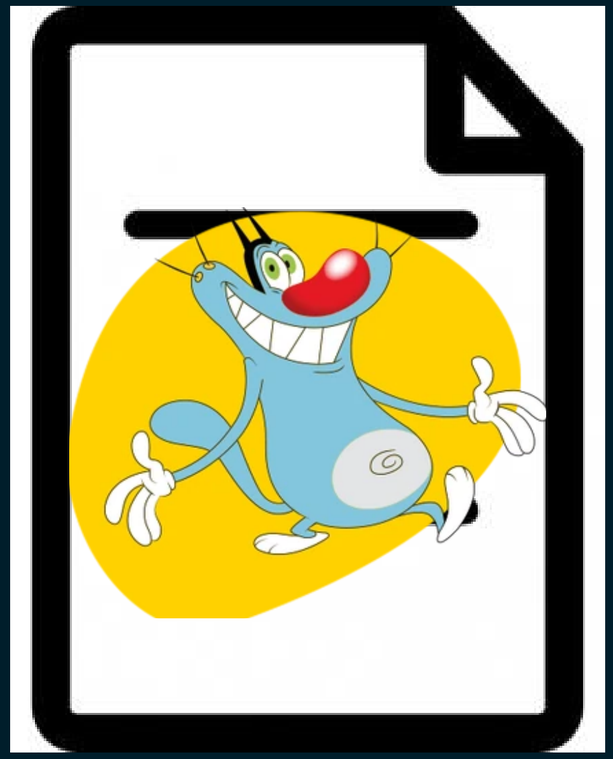
MongoDB.

- Document

  Definition: A record in MongoDB, similar to a row in a SQL database, represented in a JSON-like format (BSON).

Example:

```
{
  "_id": ObjectId("60c72b2f9af1f040a5f073b8"),
  "name": "Oggy",
  "species": "Cat",
  "color": "Blue",
  "personality": "Lazy but good-hearted",
  "friends": ["Jack"],
  "enemies": ["Joey", "Dee Dee", "Marky"]
}
```



Keys are like column names in a table, and the corresponding values are the data entries for each document.

MongoDB.

- Collection

  Definition: A group (array) of MongoDB documents, similar to a table in a SQL database.

Cats

MongoDB.

- Database

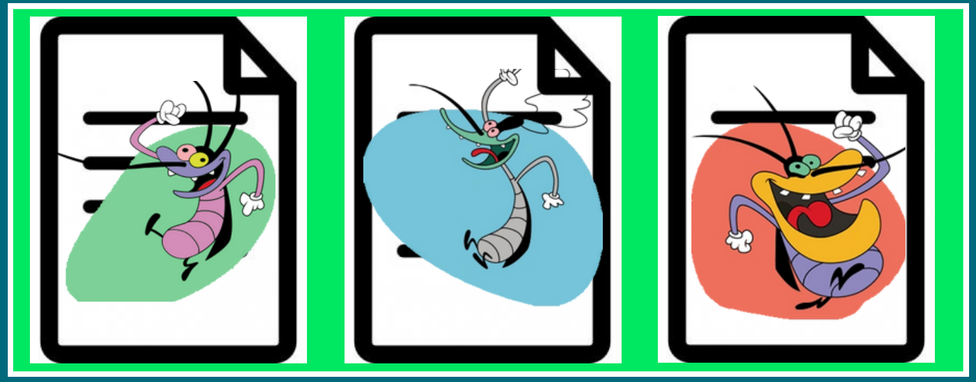
  Definition: A container for collections, similar to a database in SQL.

oggy_and_cockroaches

Cats

Humans

Cockroaches



MongoDB.

Key Features

Document-Oriented: Stores data in JSON-like documents, making it flexible and easy to work with.

Schema-Less: Allows for dynamic schema, meaning documents in the same collection can have different fields.

Scalability: Supports horizontal scaling through sharding, distributing data across multiple servers.

Replication: Provides high availability through replica sets, which replicate data across multiple servers.

Indexing: Supports various types of indexes to improve query performance.

Aggregation Framework: Powerful framework for data aggregation and processing.

MongoDB.

SQL vs NoSQL (Not Only SQL)

- Uses tables with rows and columns.
- Fixed schema; requires predefined structure.
- Scales vertically by upgrading hardware.
- Best for structured data and complex queries (e.g., financial systems).
- Examples: MySQL, PostgreSQL, Oracle.

- Uses various models like document, key-value and graph.
- Dynamic schema; allows flexible data structures.
- Scales horizontally by adding more servers.
- Best for large volumes of unstructured data and real-time applications (e.g., social media, IoT).
- Examples: MongoDB, Redis, Cassandra.

MongoDB.

NoSQL(document)

SQL

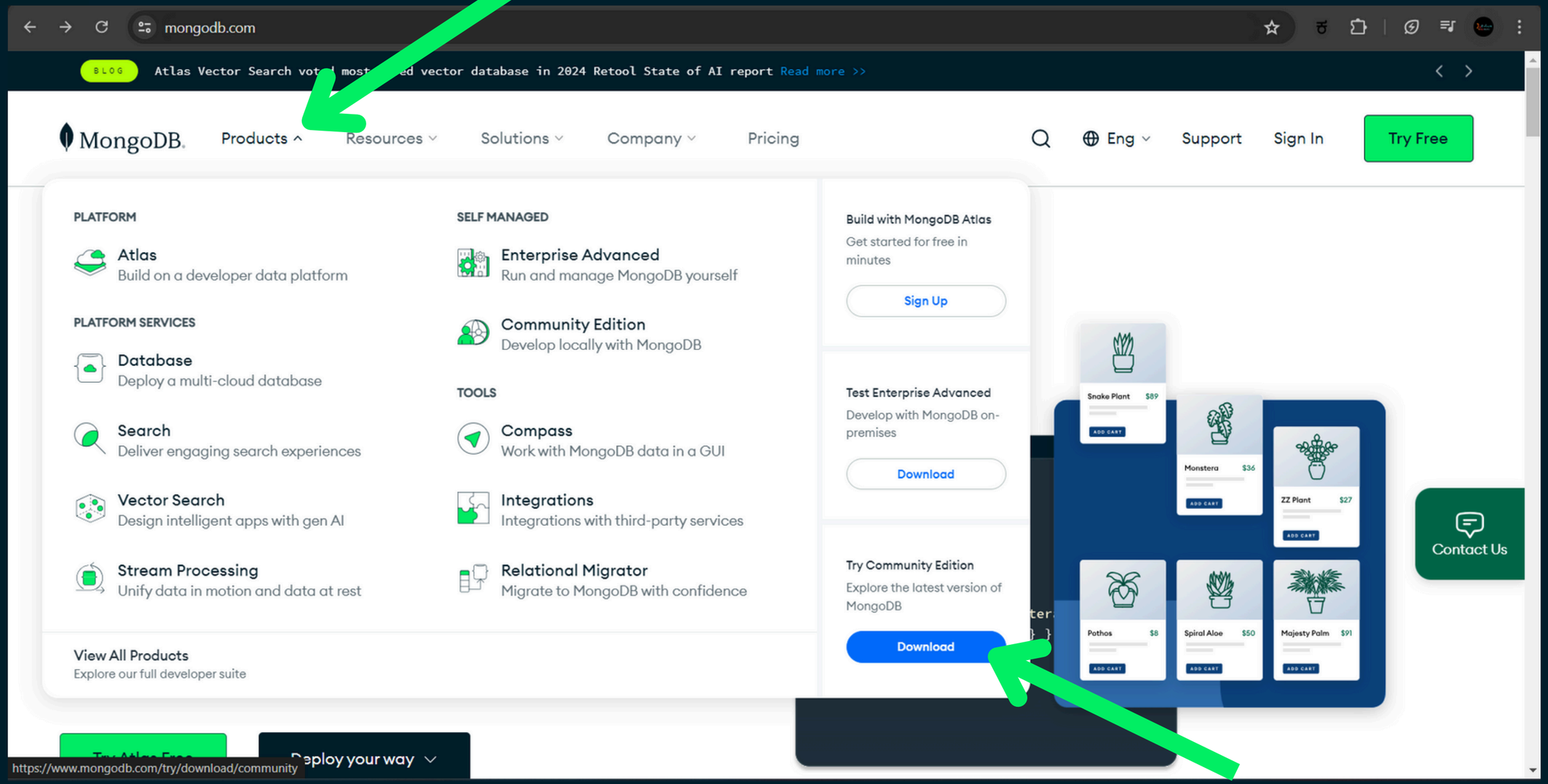
ID	Fruit_Name	Fruit_Color
1	Banana	Yellow
2	Apple	Red
3	Lemon	Yellow
4	Strawberry	Red
5	Watermelon	Green
6	Lime	Green


```
[
 {
  "_id": ObjectId("60c72b2f9af1f040a5f073b1"),
  "ID": 1,
  "Fruit_Name": "Banana",
  "Fruit_Color": "Yellow"
 },
 {
  "_id": ObjectId("60c72b2f9af1f040a5f073b2"),
  "ID": 2,
  "Fruit_Name": "Apple",
  "Fruit_Color": "Red"
 },
 {
  "_id": ObjectId("..."),
  "ID": "...",
  "Fruit_Name": "...",
  "Fruit_Color": "..."
 }
]
```

MongoDB.

MongoDB Installation and Setup

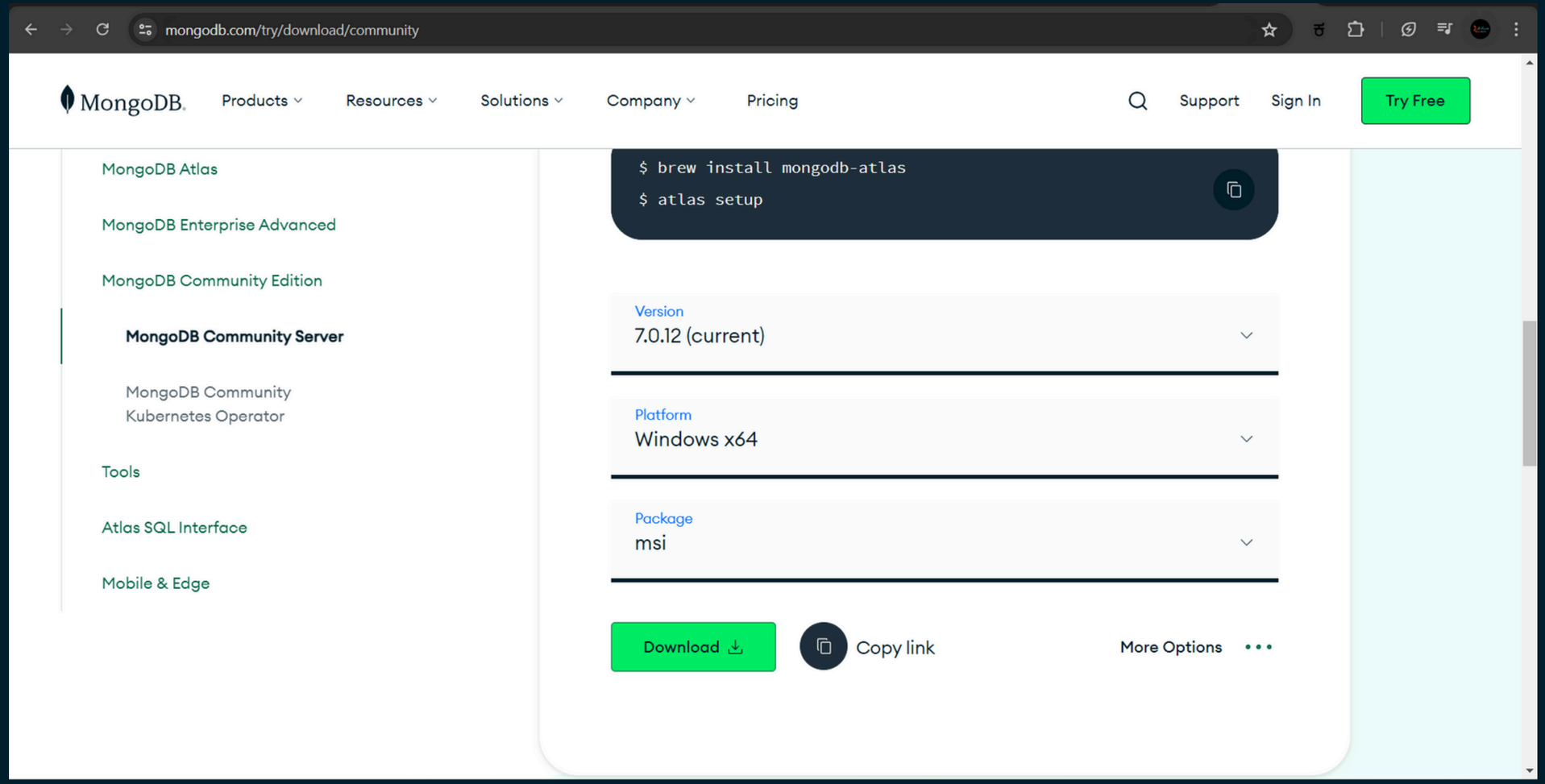
MongoDB



visit **mongodb.com,**

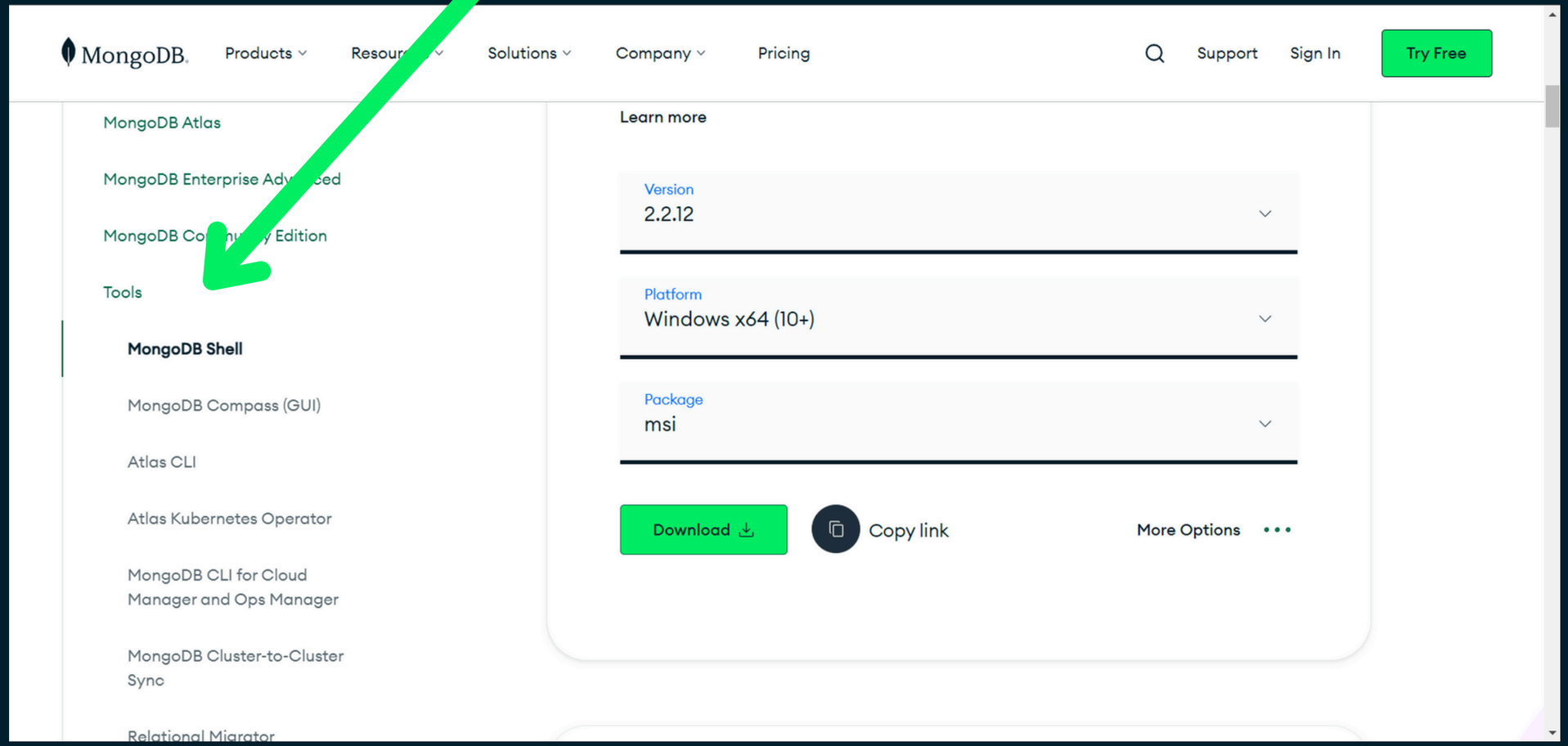
click on **download** button under **products> Try community Edition**

MongoDB



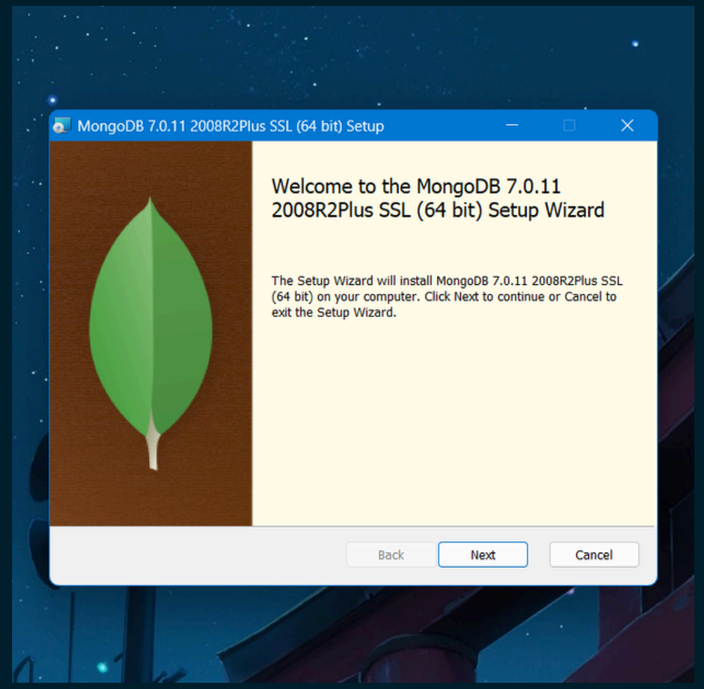
(scroll) Make sure to select '**msi**' in **package,** and click **download**

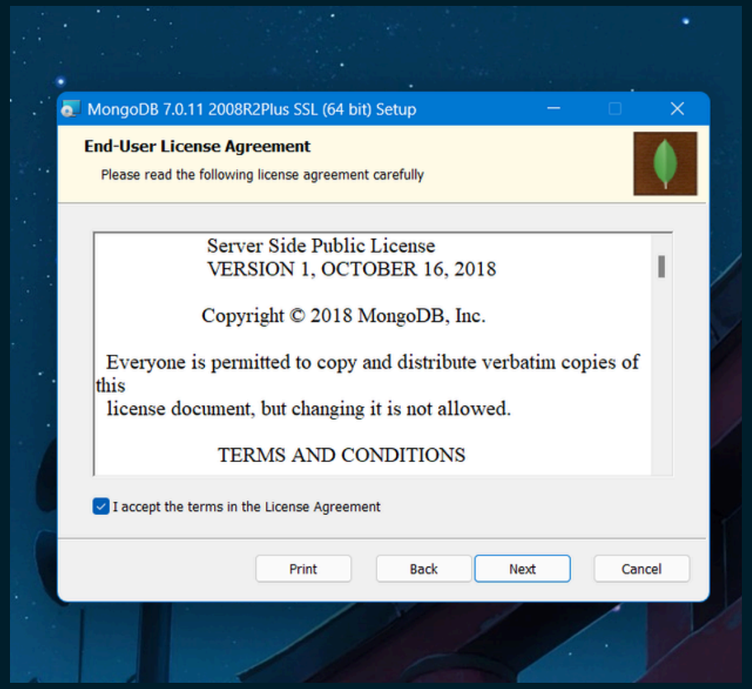
MongoDB

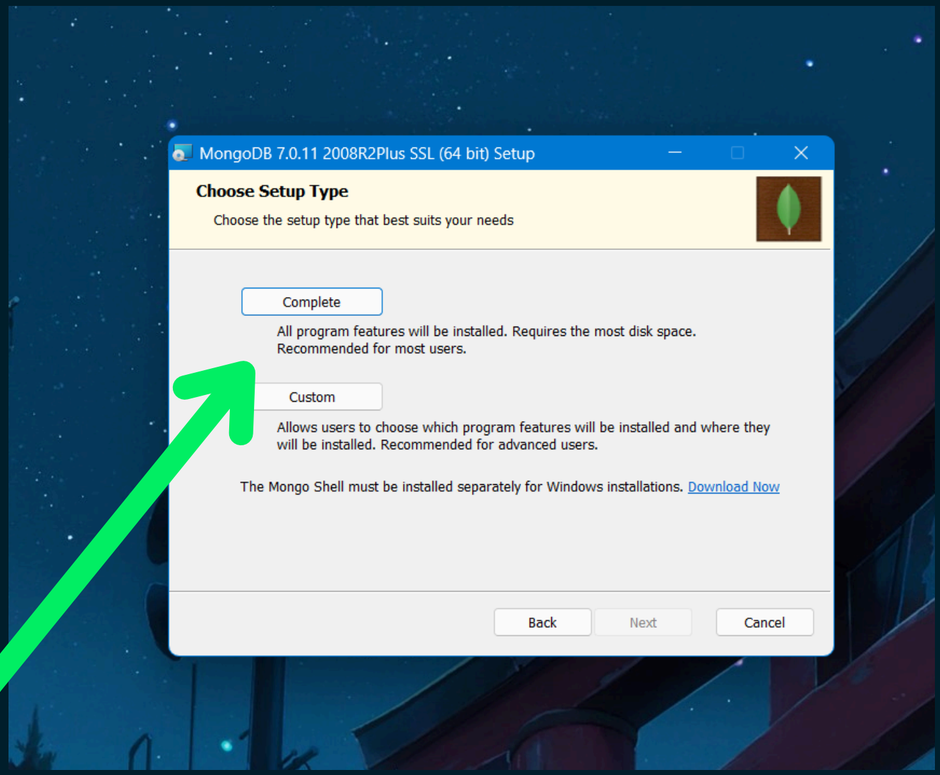


Also download **Mongodb shell** from **tools,** Make sure that **msi** is selected under package

MongoDB

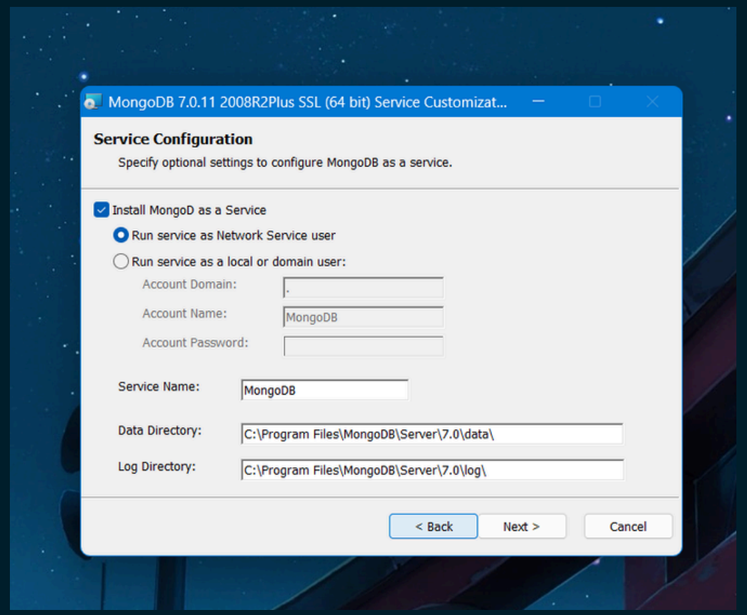


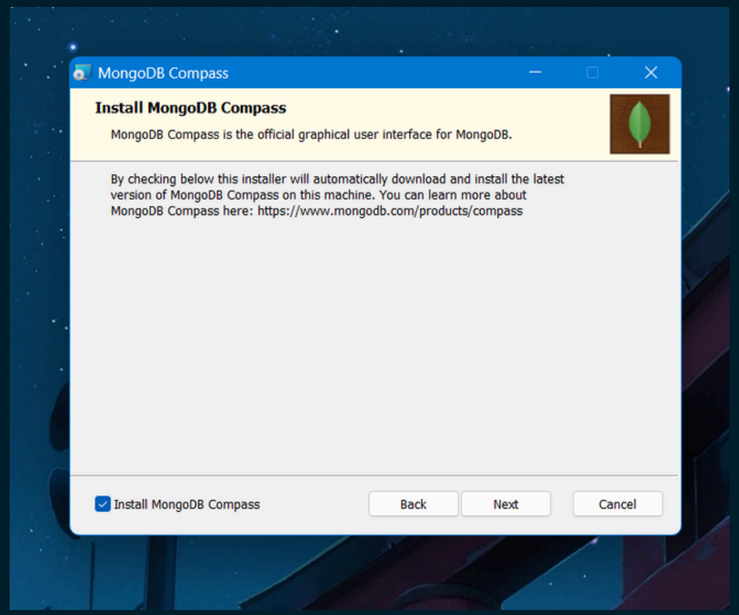


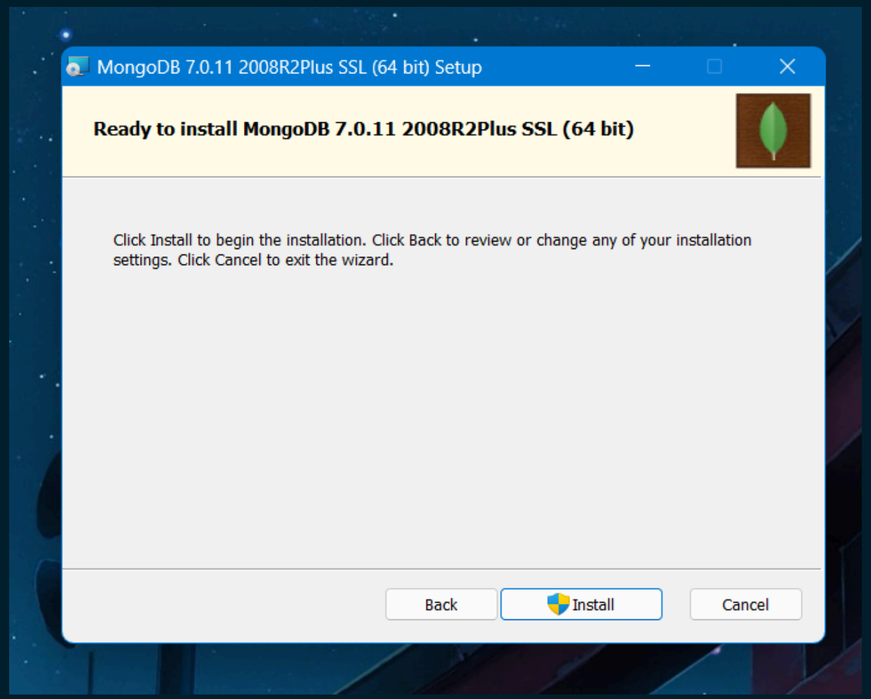


Next > accept > next > complete

MongoDB



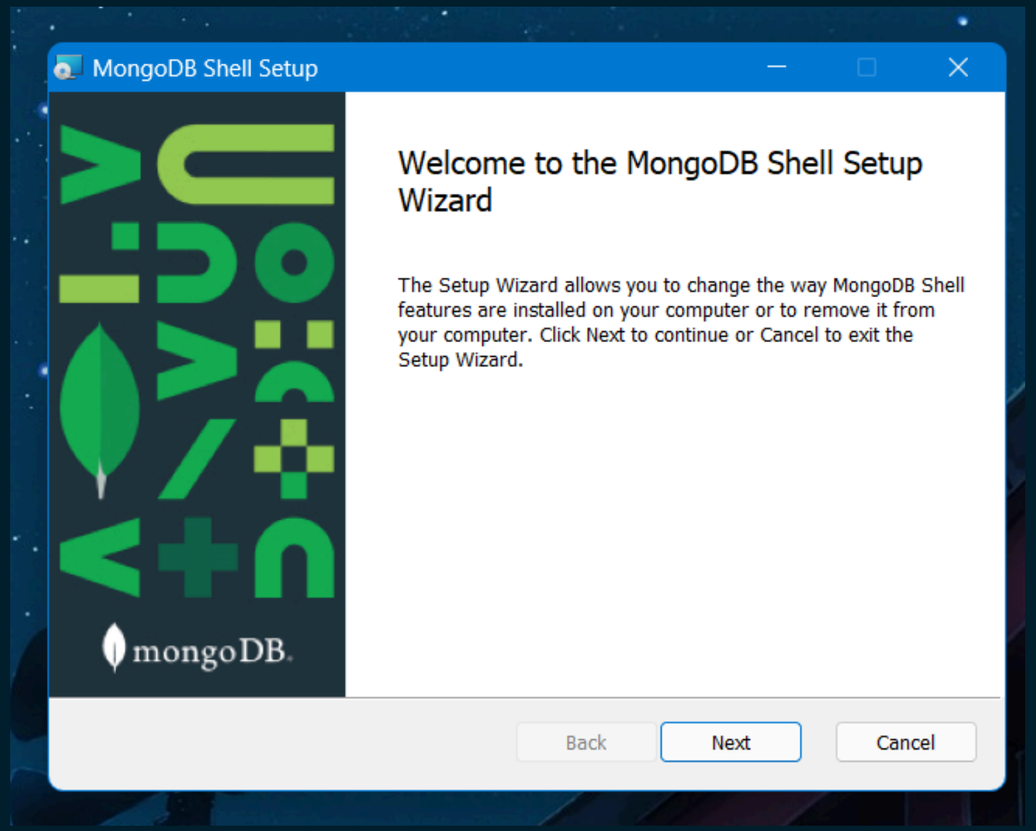




Next > select install mongodb compass > next > install

MongoDB

Similarly install mongosh (mongodb shell)



MongoDB.

Commands

Syntax:

<command> <parameters>

ex: use collegeDb;

db.<method>(<parameters>);

ex: db.createCollection(“students”)

db.<collection-name>.<method>(<parameters>);

ex: db.students.insertOne({ usn: ”4JK22CS0XX”, name: ”X” })

MongoDB.

Helpers:

Syntax:

<command> <parameters>

Show databases/collections

```
show dbs/collections   //lists databases/collections
db                     // prints the current database
```

Switch Databse

```
use <dbName>
use collegeDb
```

MongoDB.

DDL commands

Examples:

db.createCollection("<collectionName>", { <options> })

db.<collection>.drop()

DML commands

Examples:

db.<collection>.insertOne(<document>)

db.<collection>.insertMany([<document1>,<document2>,...])

db.<collection>.updateOne(<filter>,<update>)

db.<collection>.deleteOne(<filter>)

MongoDB.

Query commands

Examples:

db.<collection>.find(<query>)
db.<collection>.findOne(<query>)

Indexing Commands

Examples:

db.<collection>.createIndex({ <field>: <order> }, { <options> })
db.<collection>.dropIndex("<indexName>")

MongoDB.

Other commands

- Aggregation Commands
- Administrative Commands
- User Management Commands

MongoDB.

CRUD

Insert (Create)

db.<collection>.insertOne(<document>)

db.students.insertOne({'name':"X", 'usn':"4JK22CS0XX"})

SQL: INSERT INTO students VALUES(...)

db.<collection>.insertMany([<document1>,<document2>,...])

```
db.students.insertMany([
    { 'name':"X", 'rollNo': 11 },
    { 'name':"Y", 'rollNo': 12 }
])
```

MongoDB.

Find (Read)

db.<collection>.find(<query>, <projection>)

db.students.find({rollNo:11},{ name: true })

SQL: SELECT name FROM students WHERE rollno = 11

db.<collection>.findOne(<query>, <projection>)

db.students.findOne({ ‘age’:18 })

MongoDB.

Update

db.<collection>.updateOne(<query>, <update>)

db.students.updateOne({rollNo:11},{ $set: {name: “John” }})

db.<collection>.updateMany(<query>, <update>)

db.students.updateMany({ ‘age’:18 }, { $set: { name: “John” } })

MongoDB.

Delete

db.<collection>.deleteOne(<query>)

db.students.deleteOne({’rollNo’:11})

db.<collection>.deleteMany(<query>)

db.students.deleteMany({ ‘age’:18 })

SQL: DELETE FROM students WHERE age = 18;

MongoDB.

LET'S TRY BASIC COMMANDS

Database: collegDb

Collection: students

id	name	age	email	major
1	Alice Johnson	20	alice.johnson@example.com	Computer Science
2	Bob Smith	21	bob.smith@example.com	Information Science
3	Charlie Brown	50	charlie.brown@example.com	Computer Science
4	David Lee	35	david.lee@example.com	Civil
5	Eve Davis	18	eve.davis@example.com	Computer Science
6	Bob Lee	32	bob.lee@example.com	Information Science

try insertOne, insertMany

update the name of id 4 to David Luck

delete document with major = Civil

delete all documents with major Information Science

MongoDB.

Operators

Every operator preceds with **$** ex: $or, $eq, $lt, $gt etc.

SYNTAX

exactly 2 operands(1 operand being field value)

{ property: { $op: operand } }

{ name: { $ne: “Eve Devis” } }

//select all documents where name is not “Eve Devis”

Multiple (2 or more) operands

{ $op: [operand1, operand2, ...] }

{ $or: [{ age: 18 }, { age: 20 }] }

//select all documents where age is 18 or 20

MongoDB.

Comparison Ops $ne, $eq, $lt, $gt, $gte, $lte

db.students.find({ age: {$lt: 40, $gt: 25} }) note: implicit ‘and’

select * from students where age<40 and age>25

db.students.find({ age: {$eq: 21 } }); (same as { age: 21 })

select * from students where age=21

db.students.find({ age: {$ne: 21 } });

SELECT * FROM students WHERE age !=21;

MongoDB.

Logical Ops $and, $or, $not, $nor

```
db.students.find({ $and: [ { age: { $lt: 40 } }, { age: { $gt: 25 } } ] });
```

SELECT * FROM students WHERE age < 40 AND age > 25;

```
db.students.find({ $or: [ { age: { $lt: 18 } }, { age: { $gt: 25 } } ] });
```

SELECT * FROM students WHERE age < 18 OR age > 25;

```
db.students.find({ age: { $not: { $gt: 25 } } });
```

SELECT * FROM students WHERE NOT (age > 25);

MongoDB.

Element operators $exists, $type

```
{ age: { $exists: true } }
{ age: { $type: "int" } }
```

Update Operators $set, $unset, $inc, $push, $pull

```
{ $set: { age: 26 } }
{ $unset: { age: "" } }
{ $inc: { age: 1 } }
{ $pull: { interests: "hiking" } }
{ $push: { interests: "coding" } }
```

MongoDB.

Array operators

```
{ age: { $in: [20, 25, 30] } }

{ age: { $nin: [20, 25, 30] } }

{ interests: { $all: ["reading", "hiking"] } }

{ interests: { $size: 2 } }
```

MongoDB.

Example

//Select all documents having age<21 or age>40
SQL: select * from students where age<21 or age>40

```
db.students.find({
        $or:  [
            { age:   { $lt: 21}   } ,
            { age:   { $gt: 40} }
         ]
})
```

MongoDB.

employee collection

id	name	age	email
1	John Doe	30	john.doe@example.com
2	Jane Smith	25	jane.smith@example.com
3	Michael Johnson	35	michael.johnson@example.com
4	Emily Brown	28	emily.brown@example.com
5	Alex Williams	32	alex.williams@example.com

address.street	address.city	address.state	address.zip	active	interests
123 Main St	Anytown	CA	12345	TRUE	hiking, reading
456 Elm St	Otherville	NY	54321	FALSE	photography, gardening
789 Oak Ave	Smalltown	TX	67890	TRUE	cooking, traveling
321 Pine Rd	Villageton	FL	13579	FALSE	painting, dancing
654 Cedar Ln	Metropolis	IL	97531	TRUE	sports, music

NOTE:

```
{
   ...,
   address: {...},
   intrests: [...]
}
```

MongoDB.

1. Find all documents where active status is true.
2. Find all documents where age is greater than 30.
3. Find all documents where address.state is "CA".

   NOTE: syntax {’address.state’ : ”CA”}
4. Update the age of "Jane Smith" to 26.
5. Add a new interest "coding" to the interests array for "John Doe".
6. Find all documents where age is less than or equal to 30.
7. Find all documents where active is true or age is greater than 30.

MongoDB.

Indexes

Indexes in MongoDB are special data structures that store a small portion of the collection's data set in an easy-to-traverse form. Indexes support the efficient execution of queries and can significantly improve the performance of read operations. Without indexes, MongoDB must perform a collection scan, which can be slow and resource-intensive.

- **Single Field Index:**

```
db.<collection>.createIndex({ age: 1 })        // 1 = ascending, -1 = descending
db.<collection>.createIndex({ id: 1 }, { unique: true })
                                                // with unique constraint
```

MongoDB.

- **Compound Index**

db.collection.createIndex({ id: 1, age: 1 })

db.collection.createIndex({ key1: 1, key2: 1 })

db.collection.createIndex({ key1: 1, key2: 1 }, { unique: true })

Methods

db.collection.createIndex(<fields>,<options>)

db.collection.getIndexes();

db.collection.dropIndex(<fields>);

db.collection.dropIndex("<index_name>");

MongoDB.

THANK YOU