MongoDB Interview questions

**MongoDB :-**

[https://docs.mongodb.com/manual/tutorial/](https://docs.mongodb.com/manual/tutorial/query-array-of-documents/)

<https://www.guru99.com/what-is-mongodb.html>

**What is MongoDB?**

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s.

MongoDB Features

**Schema-less Database:** It is the great feature provided by the MongoDB. A Schema-less database means one collection can hold different types of documents in it. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size. It is not necessary that the one document is similar to another document like in the relational databases. Due to this cool feature, MongoDB provides great flexibility to databases.

Document Oriented: In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.

**Scalabiltiy:** MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers, here a large amount of 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. It will also add new machines to a running database.

**Replication:** MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.

**Aggregation:** It allows to perform operations on the grouped data and get a single result or computed result. It is similar to the SQL GROUPBY clause. It provides three different aggregations i.e, aggregation pipeline, map-reduce function, and single-purpose aggregation methods

**High Performance:** The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

Ad hoc queries : MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.

Indexing : Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.

**Load balancing :** MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

**Key Components of MongoDB Architecture**

Below are a few of the common terms used in MongoDB

**\_id –** This is a field required in every MongoDB document. The \_id field represents a unique value in the MongoDB document. The \_id field is like the document's primary key. If you create a new document without an \_id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table,

Mongo DB will add a 24 digit unique identifier to each document in the collection. ObjectId

ObjectIds are small, likely unique, fast to generate, and ordered. ObjectId values are 12 bytes in length, consisting of:

4-byte timestamp value, representing the ObjectId's creation, measured in seconds since the Unix epoch

5-byte random value

3-byte incrementing counter, initialized to a random value

| \_Id | CustomerID | CustomerName | OrderID |
| --- | --- | --- | --- |
| 563479cc8a8a4246bd27d784 | 11 | Guru99 | 111 |
| 563479cc7a8a4246bd47d784 | 22 | Trevor Smith | 222 |
| 563479cc9a8a4246bd57d784 | 33 | Nicole | 333 |

**Collection –** This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.

**Cursor –** This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.

**Database –** This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.

**Document -** A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.

**Field -** A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair's defined in the document.

[Diagram

Description automatically generated](https://www.guru99.com/images/MongoDB/112015_1051_Introductio2.png)

JSON – This is known as[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the \_id field and a normal collection field. The \_id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

How mongoDB is different from RDBMS ?  
Some major differences in between MongoDB and the RDBMS are as follows:

|  |  |
| --- | --- |
| MongoDB | RDBMS |
| It is a non-relational and document-oriented database. | It is a relational database. |
| It is suitable for hierarchical data storage. | It is not suitable for hierarchical data storage. |
| It has a dynamic schema. | It has a predefined schema. |
| It centers around the CAP theorem (Consistency, Availability, and Partition tolerance). | It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability). |
| In terms of performance, it is much faster than RDBMS. | In terms of performace, it is slower than MongoDB. |

Table

Description automatically generated

Advantages of MongoDB :

It is a schema-less NoSQL database. You need not to design the schema of the database when you are working with MongoDB.

It does not support join operation.

It provides great flexibility to the fields in the documents.

It contains heterogeneous data.

It provides high performance, availability, scalability.

It supports Geospatial efficiently.

It is a document oriented database and the data is stored in BSON documents.

It also supports multiple document ACID transition(string from MongoDB 4.0).

It does not require any SQL injection.

It is easily integrated with Big Data Hadoop

Disadvantages of MongoDB :

It uses high memory for data storage.

You are not allowed to store more than 16MB data in the documents.

The nesting of data in BSON is also limited you are not allowed to nest data more than 100 levels.

What is NoSQL?

NoSQL Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

NoSQL database stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data. Let's understand about NoSQL with a diagram in this NoSQL database tutorial:

Chart

Description automatically generated

Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

[Diagram

Description automatically generated](https://www.guru99.com/images/1/101818_0537_NoSQLTutori2.png)

NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Features of NoSQL

Non-relational

NoSQL databases never follow the [relational model](https://www.guru99.com/relational-data-model-dbms.html)

Never provide tables with flat fixed-column records

Work with self-contained aggregates or BLOBs

Doesn't require object-relational mapping and data normalization

No complex features like query languages, query planners,

referential integrity joins, ACID

Schema-free

NoSQL databases are either schema-free or have relaxed schemas

Do not require any sort of definition of the schema of the data

Offers heterogeneous structures of data in the same domain

[Diagram

Description automatically generated](https://www.guru99.com/images/1/101818_0537_NoSQLTutori3.png)

NoSQL is Schema-Free

Types of NoSQL Databases

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

Types of NoSQL Databases:

Key-value Pair Based

Column-oriented Graph

Graphs based

Document-oriented

[Graphical user interface, diagram, application

Description automatically generated](https://www.guru99.com/images/1/101818_0537_NoSQLTutori5.png)

**What is the CAP Theorem?**

CAP theorem is also called brewer's theorem. It states that is impossible for a distributed data store to offer more than two out of three guarantees

Consistency

Availability

Partition Tolerance

**Consistency:**

The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.

**Availability:**

The database should always be available and responsive. It should not have any downtime.

**Partition Tolerance:**

Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

The use Command

MongoDB use DATABASE\_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of use DATABASE statement is as follows −

use DATABASE\_NAME

Example

If you want to use a database with name <mydb>, then use DATABASE statement would be as follows −

>use mydb

switched to db mydb

To check your currently selected database, use the command db

>db

mydb

If you want to check your databases list, use the command show dbs.

>show dbs

local 0.78125GB

test 0.23012GB

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

>db.movie.insert({"name":"tutorials point"})

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

The dropDatabase() Method

MongoDB db.dropDatabase() command is used to drop a existing database.

Syntax

Basic syntax of dropDatabase() command is as follows −

db.dropDatabase()

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Example

First, check the list of available databases by using the command, show dbs.

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

>

If you want to delete new database <mydb>, then dropDatabase() command would be as follows −

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>

Now check list of databases.

>show dbs

local 0.78125GB

test 0.23012GB

The createCollection() Method

MongoDB db.createCollection(name, options) is used to create collection.

Syntax

Basic syntax of createCollection() command is as follows −

db.createCollection(name, options)

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Name | String | Name of the collection to be created |
| Options | Document | (Optional) Specify options about memory size and indexing |

Options parameter is optional, so you need to specify only the name of the collection. Following is the list of options you can use −

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| capped | Boolean | (Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also. |
| autoIndexId | Boolean | (Optional) If true, automatically create index on \_id field.s Default value is false. |
| size | number | (Optional) Specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also. |
| max | number | (Optional) Specifies the maximum number of documents allowed in the capped collection. |

While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

Examples

Basic syntax of createCollection() method without options is as follows −

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

You can check the created collection by using the command show collections.

>show collections

mycollection

system.indexes

The following example shows the syntax of createCollection() method with few important options −

> db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } ){

"ok" : 0,

"errmsg" : "BSON field 'create.autoIndexID' is an unknown field.",

"code" : 40415,

"codeName" : "Location40415"

}

>

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

>db.tutorialspoint.insert({"name" : "tutorialspoint"}),

WriteResult({ "nInserted" : 1 })

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

The drop() Method :-

MongoDB's db.collection.drop() is used to drop a collection from the database.

Basic syntax of drop() command is as −db.COLLECTION\_NAME.drop()

MongoDB supports many datatypes. Some of them are −

String − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.

Integer − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

Boolean − This type is used to store a boolean (true/ false) value.

Double − This type is used to store floating point values.

Min/ Max keys − This type is used to compare a value against the lowest and highest BSON elements.

Arrays − This type is used to store arrays or list or multiple values into one key.

Timestamp − ctimestamp. This can be handy for recording when a document has been modified or added.

Object − This datatype is used for embedded documents.

Null − This type is used to store a Null value.

Symbol − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

Date − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

Object ID − This datatype is used to store the document’s ID.

Binary data − This datatype is used to store binary data.

Code − This datatype is used to store JavaScript code into the document.

Regular expression − This datatype is used to store regular expression.

Add MongoDB Array using insert() with Example

The "insert" command can also be used to insert multiple documents into a collection at one time. The below code example can be used to insert multiple documents at a time.

The following example shows how this can be done,

Step 1) Create a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)variable called myEmployee to hold the array of documents

Step 2) Add the required documents with the Field Name and values to the variable

Step 3) Use the insert command to insert the array of documents into the collection

var myEmployee=

[

{

"Employeeid" : 1,

"EmployeeName" : "Smith"

},

{

"Employeeid" : 2,

"EmployeeName" : "Mohan"

},

{

"Employeeid" : 3,

"EmployeeName" : "Joe"

},

];

db.Employee.insert(myEmployee);

What is Primary Key in MongoDB?

In MongoDB, \_id field as the primary key for the collection so that each document can be uniquely identified in the collection. The \_id field contains a unique ObjectID value.

By default when inserting documents in the collection, if you don't add a field name with the \_id in the field name, then MongoDB will automatically add an Object id field as shown below

[Text

Description automatically generated](https://www.guru99.com/images/MongoDB/112115_0607_Introductio11.png)

When you query the documents in a collection, you can see the ObjectId for each document in the collection.

If you want to ensure that MongoDB does not create the \_id Field when the collection is created and if you want to specify your own id as the \_id of the collection, then you need to explicitly define this while creating the collection.

When explicitly creating an id field, it needs to be created with \_id in its name.

Let's look at an example on how we can achieve this.

db.Employee.insert({\_id:10, "EmployeeName" : "Smith"})

The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

Syntax

The basic syntax of insert() command is as follows −

>db.COLLECTION\_NAME.insert(document)

Example

> db.users.insert({

... \_id : ObjectId("507f191e810c19729de860ea"),

... title: "MongoDB Overview",

... description: "MongoDB is no sql database",

... by: "tutorials point",

... url: "http://www.tutorialspoint.com",

... tags: ['mongodb', 'database', 'NoSQL'],

... likes: 100

... })

WriteResult({ "nInserted" : 1 })

>

Here mycol is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the \_id parameter, then MongoDB assigns a unique ObjectId for this document.

\_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

\_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

You can also pass an array of documents into the insert() method as shown below:.

> db.createCollection("post")

> db.post.insert([

{

title: "MongoDB Overview",

description: "MongoDB is no SQL database",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 100

},

{

title: "NoSQL Database",

description: "NoSQL database doesn't have tables",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 20,

comments: [

{

user:"user1",

message: "My first comment",

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

BulkWriteResult({

"writeErrors" : [ ],

"writeConcernErrors" : [ ],

"nInserted" : 2,

"nUpserted" : 0,

"nMatched" : 0,

"nModified" : 0,

"nRemoved" : 0,

"upserted" : [ ]

})

>

To insert the document you can use db.post.save(document) also. If you don't specify \_id in the document then save() method will work same as insert() method. If you specify \_id then it will replace whole data of document containing \_id as specified in save() method.

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

The basic syntax of insert() command is as follows −

>db.COLLECTION\_NAME.insertOne(document)

Example

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

> db.createCollection("empDetails")

{ "ok" : 1 }

> db.empDetails.insertOne(

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

})

{

"acknowledged" : true,

"insertedId" : ObjectId("5dd62b4070fb13eec3963bea")

}

>

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Example

Following example inserts three different documents into the empDetails collection using the insertMany() method.

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5dd631f270fb13eec3963bed"),

ObjectId("5dd631f270fb13eec3963bee"),

ObjectId("5dd631f270fb13eec3963bef")

]

}

The find() Method

To query data from MongoDB collection, you need to use MongoDB's find() method.

Syntax

The basic syntax of find() method is as follows −

>db.COLLECTION\_NAME.find()

find() method will display all the documents in a non-structured way.

Example

Assume we have created a collection named mycol as −

> use sampleDB

switched to db sampleDB

> db.createCollection("mycol")

{ "ok" : 1 }

>

And inserted 3 documents in it using the insert() method as shown below −

> db.mycol.insert([

{

title: "MongoDB Overview",

description: "MongoDB is no SQL database",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 100

},

{

title: "NoSQL Database",

description: "NoSQL database doesn't have tables",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 20,

comments: [

{

user:"user1",

message: "My first comment",

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

Following method retrieves all the documents in the collection −

> db.mycol.find()

{ "\_id" : ObjectId("5dd4e2cc0821d3b44607534c"), "title" : "MongoDB Overview", "description" : "MongoDB is no SQL database", "by" : "tutorials point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }

{ "\_id" : ObjectId("5dd4e2cc0821d3b44607534d"), "title" : "NoSQL Database", "description" : "NoSQL database doesn't have tables", "by" : "tutorials point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 20, "comments" : [ { "user" : "user1", "message" : "My first comment", "dateCreated" : ISODate("2013-12-09T21:05:00Z"), "like" : 0 } ] }

>

The pretty() Method

To display the results in a formatted way, you can use pretty() method.

Syntax

>db.COLLECTION\_NAME.find().pretty()

Example

Following example retrieves all the documents from the collection named mycol and arranges them in an easy-to-read format.

> db.mycol.find().pretty()

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534c"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534d"),

"title" : "NoSQL Database",

"description" : "NoSQL database doesn't have tables",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 20,

"comments" : [

{

"user" : "user1",

"message" : "My first comment",

"dateCreated" : ISODate("2013-12-09T21:05:00Z"),

"like" : 0

}

]

}

The findOne() method

Apart from the find() method, there is findOne() method, that returns only one document.

Syntax

>db.COLLECTIONNAME.findOne()

Example

Following example retrieves the document with title MongoDB Overview.

> db.mycol.findOne({title: "MongoDB Overview"})

{

"\_id" : ObjectId("5dd6542170fb13eec3963bf0"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations.

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Syntax | Example | RDBMS Equivalent |
| Equality | {<key>:{$eg;<value>}} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |
| Values in an array | {<key>:{$in:[<value1>, <value2>,……<valueN>]}} | db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty() | Where name matches any of the value in :["Raj", "Ram", "Raghu"] |
| Values not in an array | {<key>:{$nin:<value>}} | db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty() | Where name values is not in the array :["Ramu", "Raghav"] or, doesn’t exist at all |

AND in MongoDB

Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND −

>db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

> db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534c"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

>

For the above given example, equivalent where clause will be ' where by = 'tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

Using AND and OR Together

Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},

{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

NOR in MongoDB

Syntax

To query documents based on the NOT condition, you need to use $not keyword. Following is the basic syntax of NOT −

>db.COLLECTION\_NAME.find(

{

$not: [

{key1: value1}, {key2:value2}

]

}

)

Example

Assume we have inserted 3 documents in the collection empDetails as shown below −

db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Age: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Age: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Age: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

Following example will retrieve the document(s) whose first name is not "Radhika" and last name is not "Christopher"

> db.empDetails.find(

{

$nor:[

{"First\_Name": "Radhika"},

{"Last\_Name": "Christopher"}

]

}

).pretty()

{

"\_id" : ObjectId("5dd631f270fb13eec3963bef"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Age" : "24",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

NOT in MongoDB

Syntax

To query documents based on the NOT condition, you need to use $not keyword following is the basic syntax of NOT −

>db.COLLECTION\_NAME.find(

{

$NOT: [

{key1: value1}, {key2:value2}

]

}

).pretty()

Example

Following example will retrieve the document(s) whose age is not greater than 25

> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )

{

"\_id" : ObjectId("5dd6636870fb13eec3963bf7"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Age" : "24",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

MongoDB's update() and save() methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of update() method is as follows −

>db.COLLECTION\_NAME.update(SELECTION\_CRITERIA, UPDATED\_DATA)

Example

Consider the mycol collection has the following data.

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

MongoDB Save() Method

The save() method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB save() method is shown below −

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

Example

Following example will replace the document with the \_id '5983548781331adf45ec5'.

>db.mycol.save(

{

"\_id" : ObjectId("507f191e810c19729de860ea"),

"title":"Tutorials Point New Topic",

"by":"Tutorials Point"

}

)

WriteResult({

"nMatched" : 0,

"nUpserted" : 1,

"nModified" : 0,

"\_id" : ObjectId("507f191e810c19729de860ea")

})

>db.mycol.find()

{ "\_id" : ObjectId("507f191e810c19729de860e6"), "title":"Tutorials Point New Topic",

"by":"Tutorials Point"}

{ "\_id" : ObjectId("507f191e810c19729de860e6"), "title":"NoSQL Overview"}

{ "\_id" : ObjectId("507f191e810c19729de860e6"), "title":"Tutorials Point Overview"}

>

MongoDB findOneAndUpdate() method

The findOneAndUpdate() method updates the values in the existing document.

Syntax

The basic syntax of findOneAndUpdate() method is as follows −

>db.COLLECTION\_NAME.findOneAndUpdate(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

Example

Assume we have created a collection named empDetails and inserted three documents in it as shown below −

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Age: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Age: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Age: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

Following example updates the age and email values of the document with name 'Radhika'.

> db.empDetails.findOneAndUpdate(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

{

"\_id" : ObjectId("5dd6636870fb13eec3963bf5"),

"First\_Name" : "Radhika",

"Last\_Name" : "Sharma",

"Age" : "30",

"e\_mail" : "radhika\_newemail@gmail.com",

"phone" : "9000012345"

}

MongoDB updateOne() method

This methods updates a single document which matches the given filter.

Syntax

The basic syntax of updateOne() method is as follows −

>db.COLLECTION\_NAME.updateOne(<filter>, <update>)

Example

> db.empDetails.updateOne(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }

>

MongoDB updateMany() method

The updateMany() method updates all the documents that matches the given filter.

Syntax

The basic syntax of updateMany() method is as follows −

>db.COLLECTION\_NAME.update(<filter>, <update>)

Example

> db.empDetails.updateMany(

{Age:{ $gt: "25" }},

{ $set: { Age: '00'}}

)

{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }

You can see the updated values if you retrieve the contents of the document using the find method as shown below −

> db.empDetails.find()

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf5"), "First\_Name" : "Radhika", "Last\_Name" : "Sharma", "Age" : "00", "e\_mail" : "radhika\_newemail@gmail.com", "phone" : "9000012345" }

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf6"), "First\_Name" : "Rachel", "Last\_Name" : "Christopher", "Age" : "00", "e\_mail" : "Rachel\_Christopher.123@gmail.com", "phone" : "9000054321" }

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf7"), "First\_Name" : "Fathima", "Last\_Name" : "Sheik", "Age" : "24", "e\_mail" : "Fathima\_Sheik.123@gmail.com", "phone" : "9000054321" }

>

The remove() Method

MongoDB's remove() method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

deletion criteria − (Optional) deletion criteria according to documents will be removed.

justOne − (Optional) if set to true or 1, then remove only one document.

Syntax

Basic syntax of remove() method is as follows −

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

Example

Consider the mycol collection has the following data.

{\_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},

{\_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},

{\_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}

Following example will remove all the documents whose title is 'MongoDB Overview'.

>db.mycol.remove({'title':'MongoDB Overview'})

WriteResult({"nRemoved" : 1})

> db.mycol.find()

{"\_id" : ObjectId("507f191e810c19729de860e2"), "title" : "NoSQL Overview" }

{"\_id" : ObjectId("507f191e810c19729de860e3"), "title" : "Tutorials Point Overview" }

Remove Only One

If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.

> db.mycol.remove({})

WriteResult({ "nRemoved" : 2 })

> db.mycol.find()

>

MongoDB – Projection :-

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

The find() Method

MongoDB's find() method, explained in [MongoDB Query Document](https://www.tutorialspoint.com/mongodb/mongodb_query_document.htm#_blank) accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute find() method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

Syntax

The basic syntax of find() method with projection is as follows −

>db.COLLECTION\_NAME.find({},{KEY:1})

Example

Consider the collection mycol has the following data −

{\_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},

{\_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},

{\_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}

Following example will display the title of the document while querying the document.

>db.mycol.find({},{"title":1,\_id:0})

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

{"title":"Tutorials Point Overview"}

>

Please note \_id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

In this chapter, we will learn how to limit records using MongoDB.

The Limit() Method

To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of limit() method is as follows −

>db.COLLECTION\_NAME.find().limit(NUMBER)

Example

Consider the collection myycol has the following data.

{\_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},

{\_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},

{\_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}

Following example will display only two documents while querying the document.

>db.mycol.find({},{"title":1,\_id:0}).limit(2)

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

>

If you don't specify the number argument in limit() method then it will display all documents from the collection.

MongoDB Skip() Method

Apart from limit() method, there is one more method skip() which also accepts number type argument and is used to skip the number of documents.

Syntax

The basic syntax of skip() method is as follows −

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

Example

Following example will display only the second document.

>db.mycol.find({},{"title":1,\_id:0}).limit(1).skip(1)

{"title":"NoSQL Overview"}

>

Please note, the default value in skip() method is 0.

The sort() Method

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

The basic syntax of sort() method is as follows −

>db.COLLECTION\_NAME.find().sort({KEY:1})

Example

Consider the collection myycol has the following data.

{\_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"}

{\_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"}

{\_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}

Following example will display the documents sorted by title in the descending order.

>db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

{"title":"Tutorials Point Overview"}

{"title":"NoSQL Overview"}

{"title":"MongoDB Overview"}

>

Please note, if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

MongoDB – Indexing:-

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

The createIndex() Method

To create an index, you need to use createIndex() method of MongoDB.

Syntax

The basic syntax of createIndex() method is as follows().

>db.COLLECTION\_NAME.createIndex({KEY:1})

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

Example

>db.mycol.createIndex({"title":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

>

In createIndex() method you can pass multiple fields, to create index on multiple fields.

>db.mycol.createIndex({"title":1,"description":-1})

>

This method also accepts list of options (which are optional). Following is the list −

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| background | Boolean | Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is false. |
| unique | Boolean | Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is false. |
| name | string | The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order. |
| sparse | Boolean | If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is false. |
| expireAfterSeconds | integer | Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection. |
| weights | document | The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score. |
| default\_language | string | For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is English. |
| language\_override | string | For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language. |

The dropIndex() method

You can drop a particular index using the dropIndex() method of MongoDB.

Syntax

The basic syntax of DropIndex() method is as follows().

>db.COLLECTION\_NAME.dropIndex({KEY:1})

Here key is the name of the file on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

Example

> db.mycol.dropIndex({"title":1})

{

"ok" : 0,

"errmsg" : "can't find index with key: { title: 1.0 }",

"code" : 27,

"codeName" : "IndexNotFound"

}

The dropIndexes() method

This method deletes multiple (specified) indexes on a collection.

Syntax

The basic syntax of DropIndexes() method is as follows() −

>db.COLLECTION\_NAME.dropIndexes()

Example

Assume we have created 2 indexes in the named mycol collection as shown below −

> db.mycol.createIndex({"title":1,"description":-1})

Following example removes the above created indexes of mycol −

>db.mycol.dropIndexes({"title":1,"description":-1})

{ "nIndexesWas" : 2, "ok" : 1 }

>

The getIndexes() method

This method returns the description of all the indexes int the collection.

Syntax

Following is the basic syntax od the getIndexes() method −

db.COLLECTION\_NAME.getIndexes()

Example

Assume we have created 2 indexes in the named mycol collection as shown below −

> db.mycol.createIndex({"title":1,"description":-1})

Following example retrieves all the indexes in the collection mycol −

> db.mycol.getIndexes()

[

{

"v" : 2,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test.mycol"

},

{

"v" : 2,

"key" : {

"title" : 1,

"description" : -1

},

"name" : "title\_1\_description\_-1",

"ns" : "test.mycol"

}

]

>

MongoDB - Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(\*) and with group by is an equivalent of MongoDB aggregation.

The aggregate() Method

For the aggregation in MongoDB, you should use aggregate() method.

Syntax

Basic syntax of aggregate() method is as follows −

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

Example

In the collection you have the following data −

{

\_id: ObjectId(7df78ad8902c)

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by\_user: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

},

{

\_id: ObjectId(7df78ad8902d)

title: 'NoSQL Overview',

description: 'No sql database is very fast',

by\_user: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 10

},

{

\_id: ObjectId(7df78ad8902e)

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Neo4j',

url: 'http://www.neo4j.com',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

},

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method −

> db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

{ "\_id" : "tutorials point", "num\_tutorial" : 2 }

{ "\_id" : "Neo4j", "num\_tutorial" : 1 }

>

Sql equivalent query for the above use case will be select by\_user, count(\*) from mycol group by by\_user.

In the above example, we have grouped documents by field by\_user and on each occurrence of by user previous value of sum is incremented. Following is a list of available aggregation expressions.

|  |  |  |
| --- | --- | --- |
| Expression | Description | Example |
| $sum | Sums up the defined value from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | Gets the minimum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | Gets the maximum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | Inserts the value to an array in the resulting document. | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

Pipeline Concept

In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework −

$project − Used to select some specific fields from a collection.

$match − This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.

$group − This does the actual aggregation as discussed above.

$sort − Sorts the documents.

$skip − With this, it is possible to skip forward in the list of documents for a given amount of documents.

$limit − This limits the amount of documents to look at, by the given number starting from the current positions.

$unwind − This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

MongoDB - Sharding

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Why Sharding?

In replication, all writes go to master node

Latency sensitive queries still go to master

Single replica set has limitation of 12 nodes

Memory can't be large enough when active dataset is big

Local disk is not big enough

Vertical scaling is too expensive

Sharding in MongoDB

The following diagram shows the Sharding in MongoDB using sharded cluster.

Diagram

Description automatically generated

In the following diagram, there are three main components −

Shards − Shards are used to store data. They provide high availability and data consistency. In production environment, each shard is a separate replica set.

Config Servers − Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment, sharded clusters have exactly 3 config servers.

Query Routers − Query routers are basically mongo instances, interface with client applications and direct operations to the appropriate shard. The query router processes and targets the operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally, a sharded cluster have many query routers.

MongoDB - Relationships

Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modeled via Embedded and Referenced approaches. Such relationships can be either 1:1, 1:N, N:1 or N:N.

Let us consider the case of storing addresses for users. So, one user can have multiple addresses making this a 1:N relationship.

Following is the sample document structure of user document −

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"name": "Tom Hanks",

"contact": "987654321",

"dob": "01-01-1991"

}

Following is the sample document structure of address document −

{

"\_id":ObjectId("52ffc4a5d85242602e000000"),

"building": "22 A, Indiana Apt",

"pincode": 123456,

"city": "Los Angeles",

"state": "California"

}

Modeling Embedded Relationships

In the embedded approach, we will embed the address document inside the user document.

> db.users.insert({

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin",

"address": [

{

"building": "22 A, Indiana Apt",

"pincode": 123456,

"city": "Los Angeles",

"state": "California"

},

{

"building": "170 A, Acropolis Apt",

"pincode": 456789,

"city": "Chicago",

"state": "Illinois"

}

]

}

})

This approach maintains all the related data in a single document, which makes it easy to retrieve and maintain. The whole document can be retrieved in a single query such as −

>db.users.findOne({"name":"Tom Benzamin"},{"address":1})

Note that in the above query, db and users are the database and collection respectively.

The drawback is that if the embedded document keeps on growing too much in size, it can impact the read/write performance.

<https://www.tutorialspoint.com/mongodb/>