**Mongo Database**

**What is Mongo DB?**

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++

MongoDB is a cross-platform, document-oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

**Database**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

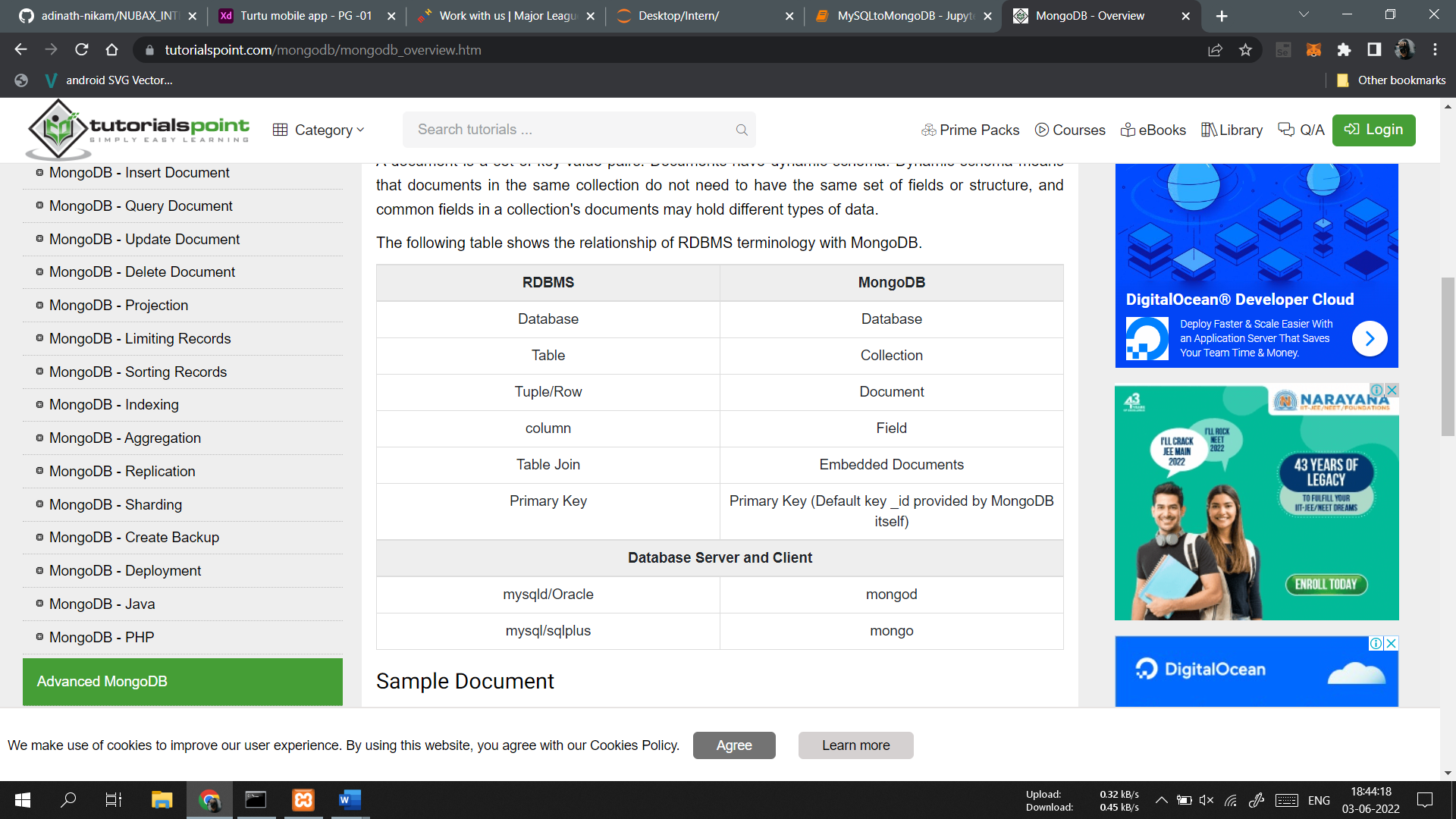
**Collection**

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

**Document**

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.



**Sample Document**

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

{

\_id: ObjectId(7df78ad8902c)

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: 'tutorials point',

url: 'https://www.example.com',

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

likes: 100,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2011,1,20,2,15),

like: 0

},

{

user:'user2',

message: 'My second comments',

dateCreated: new Date(2011,1,25,7,45),

like: 5

}

]

}

\_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide \_id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

**Advantages of MongoDB over RDBMS**

Schema less − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.

Structure of a single object is clear.

No complex joins.

Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.

Tuning.

Ease of scale-out − MongoDB is easy to scale.

Conversion/mapping of application objects to database objects not needed.

Uses internal memory for storing the (windowed) working set, enabling faster access of data.

**Why Use MongoDB?**

Document Oriented Storage − Data is stored in the form of JSON style documents.

Index on any attribute

Replication and high availability

Auto-Sharding

Rich queries

Fast in-place updates

Professional support by MongoDB

**Where to Use MongoDB?**

Big Data

Content Management and Delivery

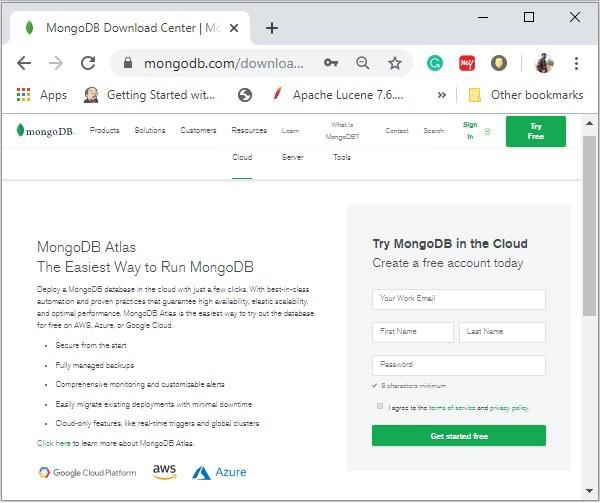
Mobile and Social Infrastructure

User Data Management

Data Hub

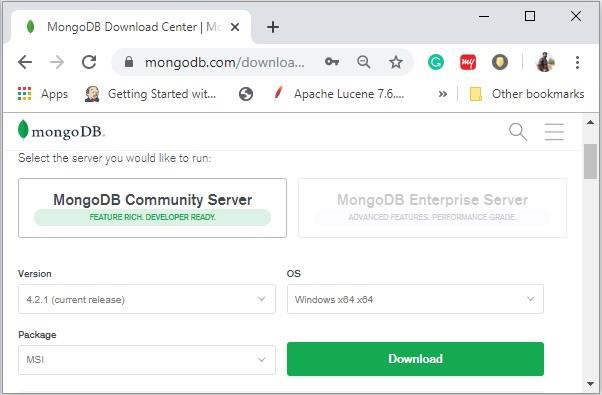
Install MongoDB On Windows

To install MongoDB on Windows, first download the latest release of MongoDB from https://www.mongodb.com/download-center.



Mongodb Cloud

Enter the required details, select the Server tab, in it you can choose the version of MongoDB, operating system and, packaging as:



Mongo DB Community

Now install the downloaded file, by default, it will be installed in the folder

**C:\Program Files\.**

MongoDB requires a data folder to store its files. The default location for the MongoDB data directory is c:\data\db. So, you need to create this folder using the Command Prompt. Execute the following command sequence.

**C:\>md data**

**C:\md data\db**

Then you need to specify set the dbpath to the created directory in mongod.exe. For the same, issue the following commands.

In the command prompt, navigate to the bin directory current in the MongoDB installation folder. Suppose my installation folder is C:\Program Files\MongoDB

**C:\Users\XYZ>d:cd C:\Program Files\MongoDB\Server\4.2\bin**

**C:\Program Files\MongoDB\Server\4.2\bin>mongod.exe --dbpath "C:\data"**

This will show waiting for connections message on the console output, which indicates that the mongod.exe process is running successfully.

Now to run the MongoDB, you need to open another command prompt and issue the following command.

**C:\Program Files\MongoDB\Server\4.2\bin>mongo.exe**

**MongoDB shell version v4.2.1**

**connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb**

**Implicit session: session { "id" : UUID("4260beda-f662-4cbe-9bc7-5c1f2242663c") }**

**MongoDB server version: 4.2.1**

**>**

This will show that MongoDB is installed and run successfully. Next time when you run MongoDB, you need to issue only commands.

**C:\Program Files\MongoDB\Server\4.2\bin>mongod.exe --dbpath "C:\data"**

**C:\Program Files\MongoDB\Server\4.2\bin>mongo.exe**

**Install MongoDB on Ubuntu**

Run the following command to import the MongoDB public GPG key −

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

Create a /etc/apt/sources.list.d/mongodb.list file using the following command.

echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen'

| sudo tee /etc/apt/sources.list.d/mongodb.list

Now issue the following command to update the repository −

sudo apt-get update

Next install the MongoDB by using the following command −

apt-get install mongodb-10gen = 4.2

In the above installation, 2.2.3 is currently released MongoDB version. Make sure to install the latest version always. Now MongoDB is installed successfully.

**Start MongoDB**

sudo service mongodb start

Stop MongoDB

sudo service mongodb stop

Restart MongoDB

sudo service mongodb restart

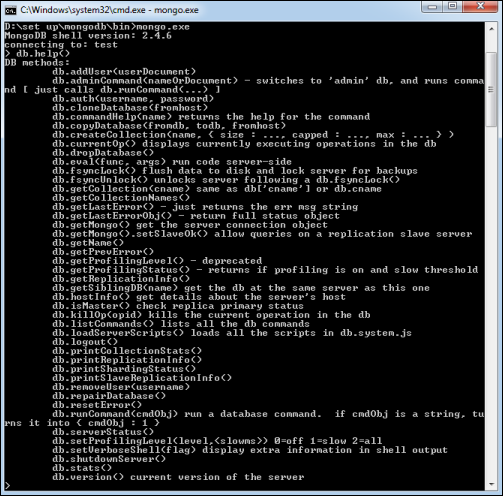
To use MongoDB run the following command.

mongo

This will connect you to running MongoDB instance.

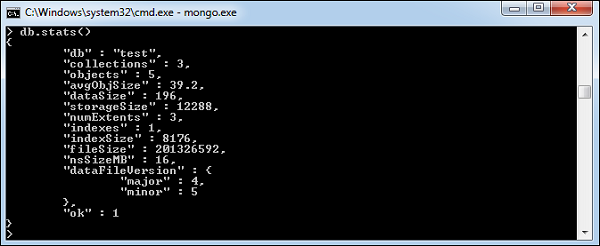
**MongoDB Help**

To get a list of commands, type db.help() in MongoDB client. This will give you a list of commands as shown in the following screenshot.



**MongoDB Statistics**

To get stats about MongoDB server, type the command db.stats() in MongoDB client. This will show the database name, number of collection and documents in the database. Output of the command is shown in the following screenshot.



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.

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.

Syntax

Basic syntax of **drop()** command is as follows −

db.COLLECTION\_NAME.drop()

### Example

First, check the available collections into your database **mydb**.

>use mydb

switched to db mydb

>show collections

mycol

mycollection

system.indexes

tutorialspoint

Now drop the collection with the name **mycollection**.

>db.mycollection.drop()

true

Again check the list of collections into database.

>show collections

mycol

system.indexes

tutorialspoint

drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

**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.

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 } ] }

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"}

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" }

**MongoDB with Python and Pandas**

# Import Python Mongo Library

import pymongo

# Establish Connection with MongoDB

mongodbConn = pymongo.MongoClient("mongodb://localhost:27017")

# Create Database

mongoDB = mongodbConn["TempDB"]

# List Databases

mongodbConn.list\_database\_names()

# Collection == Table

mongoCollection = mongoDB["TempCollection"]

# List Collections

mongoDB.list\_collection\_names();

# Insert Single Value

collData = { "id": "this\_is\_id", "name": ["this\_is\_value"]}

insertResult = mongoCollection.insert\_one(collData)

# Insert Multiple Values

collData2 = [

{"id": "this\_is\_id\_1", "value": "this\_is\_value\_1"},

{"id": "this\_is\_id\_2", "value": "this\_is\_value\_2"},

{"id": "this\_is\_id\_3", "value": "this\_is\_value\_3"}

]

insertResult2 = mongoCollection.insert\_many(collData2)

# Find Single Value

mongoCollection.find\_one()

# Find Multiple Values

for i in mongoCollection.find():

print(i)

# Retrieve Specific Values

for x in mongoCollection.find({},{"name":1}):

print(x)

# Deletion – Single Delete

mongoQuery = { "id": "this\_is\_id" }

mongoCollection.delete\_one(mongoQuery)

#Multiple Delete

mongoCollection.delete\_many(mongoQuery)

# Drop Collection

mongoCollection = mongoDB["tempCollection"]

mongoCollection.drop()

# Update Document

mongoQuery = {"id": "this\_is\_id"}

newvalue = {"$set": {"id": "this\_is\_new\_id"}}

mongoCollection.update\_one(mongoQuery, newvalue)

# Limit the Result

result = mongoCollection.find().limit(5)

**Migration from MySQL to MongoDB without Pandas**

#import MySQL Library in Python

import mysql.connector as connection

# MySQL Connnection

mysqlConn = connection.Connect(host="localhost", user="root", passwd="", database="beneficiary\_db")

# Defining Cursor

cursor = mysqlConn.cursor(dictionary=True)

# Query

cursor.execute("SELECT id, age FROM beneficiaries LIMIT 1")

mysqlResult = cursor.fetchall()

# Insert into MongoDB Without Pandas

mongoResult = mongoCollection.insert\_many(mysqlResult)

print(len(mongoResult.inserted\_ids))

**Using Pandas**

# Import pandas

from pandas.io import sql

import json

# Read from MySQL

query = "SELECT \* FROM beneficiaries LIMIT 5"

df = sql.read\_sql(query, mysqlConn)

# Insert into MongoDB

# mongoResult = mongoCollection.insert\_many(df.to\_json())

records = json.loads(df.T.to\_json()).values()

res = mongoCollection.insert\_many(records)