**MONGO DB**

MongoDB is an open-source document database and leading NoSQL database.

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.

## Sample Document:

{

\_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,

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.

## Why Use MongoDB?

* 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

## 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. {use DATABASE\_NAME }

* To check your currently selected database, use the command **db**
* 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

## The dropDatabase() Method

>use mydb

switched to db mydb

>db.dropDatabase()

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

>

## The createCollection() Method:

db.createCollection(name, options)

**example:**

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

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" : 1 }

>

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

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

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

## The drop() Method:

**db.collection.drop()**

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

>

## The find() Method

>db.COLLECTION\_NAME.find()

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>:<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 |

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

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

**Ex:**

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

## The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection.

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

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

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

>db.mycol.find()

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

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

>

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()

>db.mycol.find()

>

## The find() Method

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

## 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.

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

## MongoDB Skip() Method

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

## The sort() Method

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

## The aggregate() Method

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

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

**PHP\_MONGODB**

* make a connection, you need to specify the database name, if the database doesn't exist then MongoDB creates it automatically.
* Following is the code snippet to connect to the database −
* <?php
* // connect to mongodb
* $m = new MongoClient();
* echo "Connection to database successfully";
* // select a database
* $db = $m->mydb;
* echo "Database mydb selected";
* ?>
* When the program is executed, it will produce the following result −
* Connection to database successfully
* Database mydb selected

## Create a Collection

* Following is the code snippet to create a collection −
* <?php
* // connect to mongodb
* $m = new MongoClient();
* echo "Connection to database successfully";
* // select a database
* $db = $m->mydb;
* echo "Database mydb selected";
* $collection = $db->createCollection("mycol");
* echo "Collection created succsessfully";
* ?>
* When the program is executed, it will produce the following result −
* Connection to database successfully
* Database mydb selected
* Collection created succsessfully

## Insert a Document

To insert a document into MongoDB, **insert()** method is used.

Following is the code snippet to insert a document −

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->mycol;

echo "Collection selected succsessfully";

$document = array(

"title" => "MongoDB",

"description" => "database",

"likes" => 100,

"url" => "http://www.tutorialspoint.com/mongodb/",

"by" => "tutorials point"

);

$collection->insert($document);

echo "Document inserted successfully";

?>

When the program is executed, it will produce the following result −

Connection to database successfully

Database mydb selected

Collection selected succsessfully

Document inserted successfully

## Find All Documents

To select all documents from the collection, find() method is used.

Following is the code snippet to select all documents −

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->mycol;

echo "Collection selected succsessfully";

$cursor = $collection->find();

// iterate cursor to display title of documents

foreach ($cursor as $document) {

echo $document["title"] . "\n";

}

?>

When the program is executed, it will produce the following result −

Connection to database successfully

Database mydb selected

Collection selected succsessfully {

"title": "MongoDB"

}

## Update a Document

To update a document, you need to use the update() method.

In the following example, we will update the title of inserted document to **MongoDB Tutorial**. Following is the code snippet to update a document −

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->mycol;

echo "Collection selected succsessfully";

// now update the document

$collection->update(array("title"=>"MongoDB"),

array('$set'=>array("title"=>"MongoDB Tutorial")));

echo "Document updated successfully";

// now display the updated document

$cursor = $collection->find();

// iterate cursor to display title of documents

echo "Updated document";

foreach ($cursor as $document) {

echo $document["title"] . "\n";

}

?>

When the program is executed, it will produce the following result −

Connection to database successfully

Database mydb selected

Collection selected succsessfully

Document updated successfully

Updated document {

"title": "MongoDB Tutorial"

}

## Delete a Document

To delete a document, you need to use remove() method.

In the following example, we will remove the documents that has the title **MongoDB Tutorial**. Following is the code snippet to delete a document −

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->mycol;

echo "Collection selected succsessfully";

// now remove the document

$collection->remove(array("title"=>"MongoDB Tutorial"),false);

echo "Documents deleted successfully";

// now display the available documents

$cursor = $collection->find();

// iterate cursor to display title of documents

echo "Updated document";

foreach ($cursor as $document) {

echo $document["title"] . "\n";

}

?>

When the program is executed, it will produce the following result −

Connection to database successfully

Database mydb selected

Collection selected succsessfully

Documents deleted successfully

In the above example, the second parameter is boolean type and used for **justOne** field of **remove()** method.

Remaining MongoDB methods **findOne(), save(), limit(), skip(), sort()**etc. works same as explained above.