**Mongo DB**

* MongoDB is an open-source document database and leading NoSQL database(written in C++). There is no concept of relationship.
* MongoDB is a cross-platform, document oriented database and works on concept of collection and document.
* Database is a physical container for collections. Each database gets its own set of files on the file system. MongoDB is a document database in which one collection holds different documents
* Collection is a group of MongoDB documents which is equivalent to RDBMS table. 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.
* A document is a set of key-value pairs. Documents have dynamic schema 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.

|  |  |
| --- | --- |
| **RDBMS** | **MongoDB** |
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| Column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key \_id provided by mongodb itself) |

* Advantages of MongoDB over RDBMS
  + **Schema less**
  + Conversion/mapping of application objects to database objects not needed.
  + Structure of a single object is clear
  + No complex joins
  + MongoDB supports dynamic queries on documents using a document-based query language
  + Uses internal memory for storing the (windowed) working set, enabling faster access of data
  + Tuning

## 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
* Starting mongo DB
  + D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set up\mongodb\data"
  + D:\set up\mongodb\bin>mongo.exe

Few startup statements:

MongoDB starting : pid=528 port=27017 dbpath=E:\data\db 64-bit host=INFANTA

MongoDB shell version v4.0.3

connecting to: mongodb://127.0.0.1:27017

the active storage engine to 'wiredTiger'.

[WiredTiger](https://docs.mongodb.com/manual/core/wiredtiger/) is the default storage engine starting in MongoDB 3.2. WiredTiger provides a document-level concurrency model, checkpointing, and compression, among other features

wiredtiger\_open config: create,cache\_size=3524M,session\_max=20000,eviction=(threads\_min=4,threads\_max=4),config\_base=false,statistics=(fast),

log=(enabled=true,archive=true,path=journal,compressor=snappy),file\_manager=(close\_idle\_time=100000),

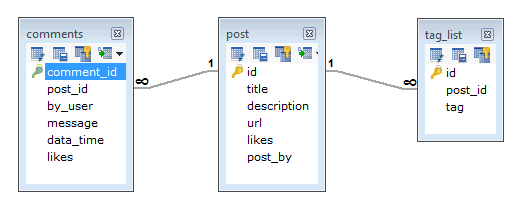
statistics\_log=(wait=0),verbose=(recovery\_progress)

\*\* WARNING: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted.

## Some considerations while designing Schema in MongoDB

* + Design your schema according to user requirements.
  + Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
  + Duplicate the data (but limited) because disk space is cheap as compare to compute time.
  + Do joins while write, not on read.
  + Optimize your schema for most frequent use cases.
  + Do complex aggregation in the schema.

RDBMS



MongoDB

{

\_id: POST\_ID

title: TITLE\_OF\_POST,

description: POST\_DESCRIPTION,

by: POST\_BY,

url: URL\_OF\_POST,

tags: [TAG1, TAG2, TAG3],

likes: TOTAL\_LIKES,

comments: [

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

},

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

}

]

}

Basic Commands

* Create Database - The command will create a new database if it doesn't exist, otherwise it will return the existing database.

**use DATABASE\_NAME**

>use mydb

switched to db mydb

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

* check currently selected database - To display database, you need to insert at least one document into it.

>db

mydb

* check databases list

>show dbs

local 0.78125GB

test 0.23012GB

* drop a existing database - This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

>use mydb

switched to db mydb

>db.dropDatabase()

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

>

* create collection – db.createCollection(name, options)

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

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

**createCollection()** method with few important options –

>db.createCollection("mycol", { capped : true, autoIndexId : true, size :

6142800, max : 10000 } )

{ "ok" : 1 }

>

In 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

>

* **show collections**

>show collections

mycollection

system.indexes

* drop a collection - db.COLLECTION\_NAME.drop()

>use mydb

switched to db mydb

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

>db.mycollection.drop()

true

* MongoDB Datatypes
  + String
  + Integer
  + Boolean
  + Double
  + Min / Max keys - This type is used to compare a value against the lowest and highest BSON(BinaryJSON) elements
  + Arrays
  + Timestamp
  + Object - This datatype is used for embedded documents
  + Null
  + Symbol - it's generally reserved for languages that use a specific symbol type
  + Date
  + Object ID - This datatype is used to store the document’s ID
  + Binary Data
  + Code - This datatype is used to store JavaScript code into the document
  + Regular Expression
* **insert data into MongoDB collection** - db.COLLECTION\_NAME.insert(document)
  + 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.

>db.mycol.insert({

\_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

})

* + To insert multiple documents in a single query, you can pass an array of documents in insert() command.

>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

}

]

}

])

* + To insert the document you can use **db.COLLECTION\_NAME.save(document)** also
* **query data from MongoDB collection** - db.COLLECTION\_NAME.find()
  + find() method will display all the documents in a non-structured way
  + To display the results in a formatted way, you can use pretty() method.
  + findOne() method, that returns only one document.

>db.mycol.find().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"

}

>

* **RDBMS Where Clause Equivalents in MongoDB**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

## AND in MongoDB - if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

## Example

>db.mycol.find({$and:[{"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"

}

## OR in MongoDB

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

## Example

>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"

}

>

## AND and OR together

>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"

}

>

## update document into a collection - update() and save()

## 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

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

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

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

## 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})

## remove a document from the collection - db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA, justOne)

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

>db.mycol.find()

## Projection

## Projection means selecting only the necessary data rather than selecting whole of the data of a document

## 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

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

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

{"title":"Tutorials Point Overview"}

>

## Limit Records

## limit() method accepts one number type argument, which is the number of documents that you want to be displayed

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

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

>

## Skip Records - skip the number of documents

## The default value in skip() method is 0.

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

{"title":"NoSQL Overview"}

>

## Sort

## sort() method accepts a document containing a list of fields along with their sorting order.

## 1 is used for ascending order while -1 is used for descending order.

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

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

{"title":"Tutorials Point Overview"}

{"title":"NoSQL Overview"}

{"title":"MongoDB Overview"}

>

## Indexing

## 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

## To create an index you need to use ensureIndex() method

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

## ensureIndex() method you can pass multiple fields, to create index on multiple fields.

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

## Aggregation and Pipeline

## Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result

## 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)

## Create Dump

>mongodump

## Restore backup

>mongorestore

## Replication - MongoDB achieves replication by the use of replica set. A replica set is a group of mongod instances that host the same data set.

* + Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
  + In a replica set, one node is primary node and remaining nodes are secondary.
  + All data replicates from primary to secondary node.
  + At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
  + After the recovery of failed node, it again join the replica set and works as a secondary node
* 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.