MongoDb

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

{

Name:”sumit”,

Age:26,

Status:”A”,

Groups:[“news”,”sports”]

}

The advantages of using documents are:

* Documents (i.e. objects) correspond to native data types in many programming languages.
* Embedded documents and arrays reduce need for expensive joins.
* Dynamic schema supports fluent polymorphism.

## Key Features

* High Performance
* Rich Query Language
* High Availability
* Horizontal Scalability
* Support for Multiple Storage Engines(WiredTiger Storage Engine, MMAPv1 Storage Engine)

# Install MongoDB

Download from mongodb.com

### Common Terms in MongoDB

1. **\_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. (24 digit)
2. **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.
3. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
4. **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.
5. **Document** - A record in a MongoDB collection is basically called a document. The document in turn will consist of field name and values.
6. **Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.
7. **JSON** – This is known as[JavaScript](http://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.

### Data Modelling

Sort of flexibility is what makes MongoDB so powerful.

When modeling data in Mongo, keep the following things in mind

1. What are the needs of the application – Look at the business needs of the application and see what data and the type of data needed for the application. Based on this, ensure that the structure of the document is decided accordingly.
2. What are data retrieval patterns – If you foresee a heavy query usage then consider the use of indexes in your data model to improve the efficiency of queries.
3. Are frequent insert's, updates and removals happening in the database – Reconsider the use of indexes or incorporate sharding if required in your data modeling design to improve the efficiency of your overall MongoDB environment.

|  |  |  |
| --- | --- | --- |
| **RDBMS** | **MongoDB** | **Difference** |
| Table | Collection | In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs. |
| Row | Document | In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents. |
| Column | Field | In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields. |
| Joins | Embedded documents | In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in Mongodb. |

### Creating a database

use dbname

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

1. >db

dbname

To **check the database list**, use the command show dbs:

>show dbs

**To delete databse**

>db.dropDatabase()

# MongoDB Create Collection

In MongoDB, db.createCollection(name, options) is used to create collection. But usually you don?t need to create collection. MongoDB creates collection automatically when you insert some documents. It will be explained later. First see how to create collection:

**Syntax:**

db.createCollection(**name**, options)

Here,

**Name:** is a string type, specifies the name of the collection to be created.

**Options:** is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

>db.createCollection("Emp")

To **check the created collection**, use the command "show collections".

>show collections

How does MongoDB create collection automatically

MongoDB creates collections automatically when you insert some documents. For example: Insert a document named seomount into a collection named SSSIT. The operation will create the collection if the collection does not currently exist.

>db.Emp.**insert**({"name" : "sumit"})

>show collections

Emp

If you want to see the inserted document, use the find() command.

Syntax:

db.collection\_name.find()

# Datatype

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.

# MongoDB Drop collection

db.COLLECTION\_NAME.**drop**()

CRUD: Documents

# MongoDB insert documents

In MongoDB, the**db.collection.insert()** method is used to add or insert new documents into a collection in your database.

db.COLLECTION\_NAME.**insert**(document)

Example

db.Emp.**insert**(

   {

     name: "Sumit",

     details: {

        city: "Noida",

        company: "Ducat"

     },

    Technology: "Mean"

   }

)

Check the inserted documents

db. Emp.find()

MongoDB insert multiple documents

Create an array of documents

var details=[{

     name: "anuj",

     details: {

        city: "Delhi",

        company: "Ducat"

     },

    Technology: "Php"  ,

Age:30

   } ,

{

     name: "Swasti",

     details: {

        city: "Noida",

        company: "Innova"

     },

    Technology: "Sap"  ,

Age:20

   },

{

     name: "Sushil",

     details: {

        city: "Delhi",

        company: "XYZ"

     },

    Technology: "UI"  ,

Age:34

   }];

db.Emp.**insert**(details)

# MongoDB update documents

In MongoDB, update() method is used to update or modify the existing documents of a collection.

**Syntax:**

db.COLLECTION\_NAME.**update**(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

Example

db.Emp.**insert**(

   {

     name: "Sumit",

     details: {

        city: "Noida",

        company: "Ducat"

     },

    Technology: "Mean"

   }

)

**Update the existing name "Sumit" into "sunil":**

db.Emp.**update**({'name':'Sumit'},{$**set**:{'name':'sunil'}})

>db.mycol.save(

{

"\_id" : ObjectId(5983548781331adf45ec7), "title":"php",

"by":"Sumit"

}

)

# MongoDB Delete documents

In MongoDB, the db.colloction.remove() method is used to delete documents from a collection. The remove() method works on two parameters.

**1. Deletion criteria:** With the use of its syntax you can remove the documents from the collection.

**2. JustOne:** It removes only one document when set to true or 1.

**Syntax:**

db.collection\_name.remove (DELETION\_CRITERIA)

Remove all documents

db.Emp.remove({})

Remove all documents that match a condition

db.Emp.remove( { city : "noida" } )

db.Emp.remove( { “details.city” : "noida" } )

# MongoDB Query documents

In MongoDB, the **db.collection.find()** method is used to retrieve documents from a collection. This method returns a cursor to the retrieved documents.

The db.collection.find() method reads operations in mongoDB shell and retrieves documents containing all their fields.

db.COLLECTION\_NAME.find({})

Select all documents in a collection:

db.Emp.find()

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

## The pretty() Method

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

### Syntax

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

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

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 |

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

{"title": "bac"}]}).pretty()

>db.mycol.find({$and:[{"by":"abc"},{"title": "Php"}]}).pretty()

# MongoDB limit() Method

In MongoDB, limit() method is used to limit the fields of document that you want to show. Sometimes, you have a lot of fields in collection of your database and have to retrieve only 1 or 2. In such case, limit() method is used.

The MongoDB limit() method is used with find() method.

**Syntax:**

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

**eg:**

db.Emp.find().limit(1)

MongoDB skip() method

In MongoDB, skip() method is used to skip the document. It is used with find() and limit() methods.

Syntax

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

**eg:**

db.Emp.find().limit(1).skip(2)

# MongoDB sort() method

In MongoDB, sort() method is used to sort the documents in the collection. This method accepts a document containing list of fields along with their sorting order.

The sorting order is specified as 1 or -1.

* 1 is used for ascending order sorting.
* -1 is used for descending order sorting.

**Syntax:**

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

Execute the following query to display the documents in descending order.

db.Emp.find().sort({"city":-1})

## 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)

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

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

|  |  |  |
| --- | --- | --- |
| **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"}}}]) |
| $push | Inserts the value to an array in the resulting document. | db.Emp.aggregate([{$group : {name : "Sushil", url : {$push: "ABCD"}}}]) |
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate([{$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"}}}]) |

Npm install mongoose --save