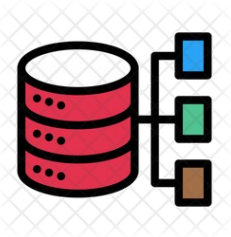
MongoDB Operations

Tushar B. Kute,
http://tusharkute.com



Create Database

- There is no create database command in MongoDB. Actually, MongoDB do not provide any command to create database.
- It may be look like a weird concept, if you are from traditional SQL background where you need to create a database, table and insert values in the table manually.
- Here, in MongoDB you don't need to create a database manually because MongoDB will create it automatically when you save the value into the defined collection at first time.

Create Database

- If there is no existing database, the following command is used to create a new database.
- Syntax:

```
use DATABASE_NAME
```

- If the database already exists, it will return the existing database.
- To check the database list, use the command show dbs:

```
> show dbs
```

Drop Database

- The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database.
- Syntax:

```
db.dropDatabase()
```

- This syntax will delete the selected database. In the case you have not selected any database, it will delete default "test" database.

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.
- Syntax:

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

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

Create Collection: Options

- Capped Boolean (Optional)
  - If it is set to true, enables a capped collection. Capped collection is a fixed size collecction 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 it is set to true, automatically create index on ID field. Its default value is false.

Create Collection: Options

- Size Number (Optional)
  - It specifies a maximum size in bytes for a capped collection. Ifcapped is true, then you need to specify this field also.
- Max Number (Optional)
  - It specifies the maximum number of documents allowed in the capped collection.

Create Collection: Example

- Let's take an example to create collection. In this example, we are going to create a collection name mydata.

```
>use test
switched to db test
>db.createCollection("mydata")
{ "ok" : 1 }
```

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

```
> show collections
```

Create Collection: Example

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

```
>db.mydata.insert({"name" : "tushar"})
>show collections
mydata
```

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

```
db.collection_name.find()
```

Drop Collection

- In MongoDB, db.collection.drop() method is used to drop a collection from a database.
- It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.
- The db.collection.drop() method does not take any argument and produce an error when it is called with an argument.
- This method removes all the indexes associated with the dropped collection.
- Syntax:

  db.COLLECTION_NAME.drop()

Insert document

- In MongoDB, the db.collection.insert() method is used to add or insert new documents into a collection in your database.
- Upsert
  - There are also two methods "db.collection.update()" method and "db.collection.save()" method used for the same purpose. These methods add new documents through an operation called upsert.
  - Upsert is an operation that performs either an update of existing document or an insert of new document if the document to modify does not exist.

Insert document

- Syntax

```
>db.COLLECTION_NAME.insert(document)
```

Insert document

```
db.mydata.insert(
  {
    course: "BDA",
    details: {
      duration: "6 months",
      Trainer: "Tushar Kute"
    },
    Batch: [ { size: "Small", qty: 10 }, { size: "Medium", qty: 30 } ],
    category: "Data Science"
  }
)
```

Check the document

- If the insertion is successful, you can view the inserted document by the following query.

```
> db.mydata.find()
```

- You will get the inserted document in return.

Query Document

- The find() Method:
- To query data from MongoDB collection, you need to use MongoDB's find() method.
- Syntax:
  - Basic syntax of find() method is as follows

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

Query Document

- The pretty() Method:
- To display the results in a formatted way, you can use pretty() method.
- Syntax:

```
db.collection_name.find().pretty()
```

- Apart from find() method there is findOne() method, that reruns only one document.

RDBMS Clauses equivalent to MongoDB

- Equality:
  - Syntax: `{<key>:<value>}`
  - Example:

    ```
    db.institute.find({"by":"Mitu Research"}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where by = 'Mitu Research'
    ```

- Less than:
  - Syntax: `{<key>:{$lt:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$lt:50}}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where likes < 50
    ```

RDBMS Clauses equivalent to MongoDB

- Less than or equal to:
  - Syntax: `{<key>:{$lte:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$lte:50}}).pretty()
    ```
  - RDBMS equivalent:

    ```
    where likes <= 50
    ```
- Greater than:
  - Syntax: `{<key>:{$gt:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$gt:50}}).pretty()
    ```
  - RDBMS equivalent:

    ```
    where likes > 50
    ```

RDBMS Clauses equivalent to MongoDB

- Greater than or equal to:
  - Syntax: `{<key>:{$gte:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$gte:50}}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where likes >= 50
    ```

- Not equal to:
  - Syntax: `{<key>:{$ne:<value>}}`
  - Example:

    ```
    db.institute.find({"likes":{$ne:50}}).pretty()
    ```

  - RDBMS equivalent:

    ```
    where likes != 50
    ```

AND and OR Operations

- In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it AND condition. Basic syntax of AND is shown below:

```
db.mycol.find({key1:value1, key2:value2}).pretty()
```

- To query documents based on the OR condition, you need to use $or keyword. Basic syntax of OR is shown below:
- `db.mycol.find({$or: [{key1: value1}, {key2:value2}]}`

AND Operation Example

```
> db.institute.find({"by":"Mitu Research","title": "NoSQL Database"}).pretty()
{
        "_id" : ObjectId("579b0030329b317d9a0ab0bb"),
        "title" : "NoSQL Database",
        "description" : "NoSQL database doesn't have tables",
        "by" : "Mitu Research",
        "url" : "http://www.mitu.co.in",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 20
}
```

OR Operation Example

```
> db.institute.find({$or:[{"by":"Mitu Research","title": "NoSQL Database"}]}).pretty()
{
        "_id" : ObjectId("579b0030329b317d9a0ab0bb"),
        "title" : "NoSQL Database",
        "description" : "NoSQL database doesn't have tables",
        "by" : "Mitu Research",
        "url" : "http://www.mitu.co.in",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 20
}
```

Insert multiple documents

- If you want to insert multiple documents in a collection, you have to pass an array of documents to the db.collection.insert() method.
- Create an array of documents
- Define a variable named Allcourses that hold an array of documents to insert.

Insert multiple documents

```
var Allcourses =
   [
    {
      Course: "Java",
      details: { Duration: "6 months", Trainer: "Tushar Kute" },
      Batch: [ { size: "Medium", qty: 25 } ],
      category: "Programming Language"
    },
    {
      Course: "Python",
      details: { Duration: "3 months", Trainer: "Rashmi Thorave" },
      Batch: [ { size: "Small", qty: 10 }, { size: "Large", qty: 30 } ],
      category: "Programming Language"
    }
   ];
```

Insert multiple documents

- Inserts the documents

- Pass this Allcourses array to the db.collection.insert() method to perform a bulk insert.

```
> db.mydata.insert( Allcourses );
```

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)

Update Documents

- Update the existing course "java" into "python":

  > db.mydata.update({'course':'java'},{$set: {'course':'python'}})

- Check the updated document in the collection:

  > db.mydata.find()

Update Documents

- By default mongodb will update only single document, to update multiple you need to set a paramter 'multi' to true.
- `db.institute.update({"by":"Mitu Research"}, {$set:{"title":"MongoDB Tutorial"}}, {multi:true})`

```
> db.institute.update({"by":"Mitu Research"},{$set:{"title":"MongoDB Tutorial"}})
> db.institute.find()
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no s
ql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [  "mongodb",  "database",
  "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
```

Update Documents

- MongoDB Save() Method:

  The save() method replaces the existing document with the new document passed in save() method
- Syntax:

```
db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Update Documents - Examples

- db.institute.save({"_id":ObjectId("5798d311e102bcdf8d057376"),"name":"rashmi thorave"})

```
> db.institute.find()
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no sql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn't have tables", "likes" : 20, "tags" : [ "mongodb", "database", "NoSQL" ], "title" : "MongoDB Tutorial", "url" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
> db.institute.save({"_id":ObjectId("5798d311e102bcdf8d057376"),"name":"rashmi thorave"})
> db.institute.find()
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no sql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn't have tables", "likes" : 20, "tags" : [ "mongodb", "database", "NoSQL" ], "title" : "MongoDB Tutorial", "url" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
```

Delete Documents

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

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

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

- Syntax:

```
db.COLLECTION_NAME.remove(DELETION_CRITERIA)
```

Delete Documents

- `db.institute.remove({'title':'MongoDB Overview'})`

```
> db.institute.find()
{ "_id" : ObjectId("579b0030329b317d9a0ab0ba"), "title" : "MongoDB Overview", "description" : "MongoDB is no s
ql database", "by" : "Mitu Skillologies", "url" : "http://www.mitu.co.in", "tags" : [  "mongodb",  "database",
  "NoSQL" ], "likes" : 1000 }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
> db.institute.remove({'title':'MongoDB Overview'})
> db.institute.find()
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
```

Delete Documents

- Remove only one
  - If there are multiple records and you want to delete only 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

- In mongodb projection meaning is selecting only 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 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 list of fields with value 1 or 0. 1 is used to show the filed while 0 is used to hide the field.
- Syntax:

```
db.COLLECTION_NAME.find({},{KEY:1})
```

Projection – Example

```
> db.institute.find({},{"title":1,_id:0})
{  }
{ "title" : "MongoDB Tutorial" }
{ "title" : "MongoDB Tutorial" }
```

Limit Records

- The limit() Method:

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

- Syntax:

```
db.COLLECTION_NAME.find().limit(NUMBER)
```

Limit Records

```
> db.institute.find({},{"title":1,_id:0})
{  }
{ "title" : "MongoDB Tutorial" }
{ "title" : "MongoDB Tutorial" }
> db.institute.find({},{"title":1,_id:0}).limit(2)
{  }
{ "title" : "MongoDB Tutorial" }
```

Skip Records

- The skip() method:

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

- Syntax:

```
db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Skip Records

```
> db.institute.find({},{"title":1,_id:0})
{  }
{ "title" : "MongoDB Tutorial" }
{ "title" : "MongoDB Tutorial" }
> db.institute.find({},{"title":1,_id:0}).limit(2).skip(2)
{ "title" : "MongoDB Tutorial" }
```

Sort Documents

- The sort() Method:

  To sort documents in MongoDB, you need to use sort() method. sort() method accepts a document containing 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:

```
db.COLLECTION_NAME.find().sort({KEY:1})
```

Sort Documents

- `db.institute.find().sort({"title":-1})`

```
> db.institute.find().sort({"title":-1})
{ "_id" : ObjectId("579b0030329b317d9a0ab0bb"), "by" : "Mitu Research", "description" : "NoSQL database doesn'
t have tables", "likes" : 20, "tags" : [  "mongodb",  "database",  "NoSQL" ], "title" : "MongoDB Tutorial", "u
rl" : "http://www.mitu.co.in" }
{ "_id" : ObjectId("5798d3b6e102bcdf8d057377"), "name" : "tushar kute", "title" : "MongoDB Tutorial" }
{ "_id" : ObjectId("5798d311e102bcdf8d057376"), "name" : "rashmi thorave" }
```

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:

  Syntax:

  ```
  db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
  ```

Aggregation

```
> db.institute.insert(
... {
... 'title': 'MongoDB Overview',
... 'description': 'MongoDB is no sql database',
... 'by_user': 'rashmi thorave',
... 'url': 'http://www.mitu.co.in',
... 'tags': ['mongodb', 'database', 'NoSQL'],
... 'likes': 100
... },
... {
... title: 'NoSQL Overview',
... description: 'No sql database is very fast',
... by_user: 'tushar kute',
... url: 'http://www.tusharkute.com',
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 10
... },
... {
... title: 'Neo4j Overview',
... description: 'Neo4j is no sql database',
... by_user: 'Neo4j',
... url: 'http://www.neo4j.com',
... tags: ['neo4j', 'database', 'NoSQL'],
... likes: 750
... })
```

Aggregation

```
> db.institute.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
        "result" : [
                {
                        "_id" : "rashmi thorave",
                        "num_tutorial" : 1
                }
        ],
        "ok" : 1
}
```

Sql equivalent query for the above use case will be

```
select by_user, count(*) from mycol group by by_user
```

Aggregation functions

- `db.institute.aggregate([{$group : {_id: "$by_user", num_tutorial : {$sum :"$likes"}}}])`

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"}}}])

Distinct

- Finds the distinct values for a specified field across a single collection.
- distinct returns a document that contains an array of the distinct values.
- The return document also contains an embedded document with query statistics and the query plan.

Distinct

- The following example returns the distinct values for the field dept from all documents in the mydata collection:

- db.runCommand ({ distinct: "mydata", key: "dept" })

- db.runCommand(
  {
  distinct: "restaurants",
  key: "rating",
  query: { cuisine: "italian" },
  }
)

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com