Full Stack Web Programming

Seven Advanced Academy

MongoDB Part II

Lesson 80

MongoDB Insert Document

- 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)
```

MongoDB Insert Document

- Example:

```
>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
})
```

- Here mycol is our collection name, as created in the previous part. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

MongoDB Insert Document

- 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)
```

- 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: 'Seven Academy',
      tags: ['mongodb', 'database', 'NoSQL'],
      likes: 100
   },

   {
      title: 'NoSQL Database',
      description: "NoSQL database doesn't have tables",
      by: 'Seven Academy',
      tags: ['mongodb', 'database', 'NoSQL'],
      likes: 20,
      comments: [
         {
            user:'user1',
            message: 'My first comment',
            dateCreated: new Date(2013,11,10,2,35),
            like: 0
         }
      ]
   }
])
```

MongoDB Insert Document

- 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.

MongoDB Query Document

- 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.

MongoDB Query Document

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

```
>db.COLLECTION_NAME.find().pretty()
```

```
>db.mycol.find().pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "Seven Academy",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

MongoDB Query Document

- 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": Seven Academy"}).pretty()	where by = Seven Academy
Less Than	{<key>: {$lt: <value>}}	db.mycol.find({"likes": {$lt:50}}).pretty()	where likes < 50

RDBMS Where Clause Equivalents in MongoDB

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

- **Syntax**
- In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND –

```
>db.mycol.find(
   {
      $and: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

AND in MongoDB

- **Example**
- Following example will show all the tutorials written by 'Seven Academy' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({$and:[{"by":"seven Academy"},{"title": "MongoDB Overview"}]}).pretty() {
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "Seven Academy",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

AND in MongoDB

- For the above given example, equivalent where clause will be ' where by = 'seven Academy' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

OR in MongoDB

- **Syntax**
- To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR –

```
>db.mycol.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

- **Example**
- Following example will show all the tutorials written by 'SevenAcademy' or whose title is 'MongoDB Overview'.

OR in MongoDB

```
>db.mycol.find({$or:[{"by":"Seven Academy"},{"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "Seven Academy",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

Using OR and AND Together

- **Example**
- The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'Seven Academy'. Equivalent SQL where clause is **'where likes>10 AND (by = 'Seven Academy' OR title = 'MongoDB Overview')'**

Using OR and AND Together

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "Seven Academy"},
   {"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "Seven Academy",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

MongoDB Update Document

- 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.

MongoDB Update 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'.

MongoDB Update Document

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

- 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'.

MongoDB Update Document

```
>db.mycol.save(
   {
      "_id" : ObjectId(5983548781331adf45ec7), "title":"Seven Academy New Topic",
         "by":"Seven Academy"
   }
)
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"Seven Academy New Topic",
   "by":"Seven Academy"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

MongoDB Delete Document

- The remove() Method
- MongoDB's remove() method is used to remove a document from the collection. The remove() method accepts two parameters. One is deletion criteria and second is justOne flag.
- **deletion criteria** – (Optional) deletion criteria according to the documents that will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.

MongoDB Delete 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(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Seven Academy Overview"}
```

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

MongoDB Delete Document

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Seven Academy 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(DELLETION_CRITTERIA, 1)
```

MongoDB Delete Document

- **Remove All Documents**
- If you don't specify deletion criteria, then MongoDB will delete the whole documents from the collection. This is equivalent of SQL's **truncate command**.

```
>db.mycol.remove()
>db.mycol.find()
>
```

MongoDB Projection

- In MongoDB, **projection** means selecting only the 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.

MongoDB Projection

- The find() Method
- MongoDB's find() method, explained in MongoDB Query Document 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 a list of fields with value **1** or **0**. 1 is used to show the field while 0 is used to hide the fields.

MongoDB Projection

- **Syntax**
- The basic syntax of find() method with projection is as follows –

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

- **Example**
- Consider the collection mycol has the following data –

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Seven Academy Overview"}
```

MongoDB Projection

- Following example will display the title of the document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Seven Academy Overview"}
```

- Please note _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as **0**.

MongoDB Limit Records

- 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.

- **Syntax**
- The basic syntax of limit() method is as follows −

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

MongoDB Limit Records

- **Example**
- Consider the collection mycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Seven Academy Overview"}
```

- Following example will display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

MongoDB Limit Records

- **Note:** If you don't specify the number argument in limit() method then it will display all documents from the collection.
- MongoDB Skip() Method
- Apart from limit() method, there is one more method skip() which also accepts number type argument and is used to skip the number of documents.

- **Syntax**
- The basic syntax of skip() method is as follows −

MongoDB Limit Records

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

- Following example will display only the second document.

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

- Please note, the default value in skip() method is 0.

MongoDB Sort Records

- The sort() Method
- To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a 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**
- The basic syntax of sort() method is as follows −

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

MongoDB Sort Records

- **Example**
- Consider the collection mycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Seven Academy Overview"}
```

- Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Seven Academy Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
```

MongoDB Sort Records

- **Note**, if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

Congratulation!