Connect with Adnan Manna on LinkedIn!

MongoDB CRUD Operation Simplified

Open to connect with anyone who sees this post!

1. Create

```
db.people.insert({name: 'Tom', age: 28});
```

Or

```
db.people.save({name: 'Tom', age: 28});
```

The difference with save is that if the passed document contains an _id field, if a document already exists with that _id it will be updated instead of being added as new.

Two new methods to insert documents into a collection, in MongoDB 3.2.x:

Use insertOne to insert only one record:

```
db.people.insertOne({name: 'Tom', age: 28});
```

Use insertMany to insert multiple records:

```
db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}, {name: 'Kathy', age: 23}])
```

Note that insert is highlighted as deprecated in every official language driver since version 3.0. The full distinction being that the shell methods actually lagged behind the other drivers in implementing the method. The same thing applies for all other CRUD methods

2. Update

Update the entire object:

```
db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace only first matching document.

db.people.updateMany({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace all matching documents.
```

Or just update a single field of a document. In this case age:

```
db.people.update({name: 'Tom'}, {$set: {age: 29}})
```

You can also update multiple documents simultaneously by adding a third parameter. This query will update all documents where the name equals Tom:

```
db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{$set:{age: 30}) //Will update only first matching document.

db.people.updateMany({name: 'Tom'},{$set:{age: 30}}) //Will update all matching documents.
```

If a new field is coming for update, that field will be added to the document.

```
db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})// Document will have `salary` field as well.
```

If a document is needed to be replaced,

```
db.collection.replaceOne({name:'Tom'}, {name:'Lakmal',age:25,address:'Sri Lanka'})
```

can be used.

Note: Fields you use to identify the object will be saved in the updated document. Field that are not defined in the update section will be removed from the document.

3. Delete

Deletes all documents matching the query parameter:

```
// New in MongoDB 3.2
db.people.deleteMany({name:
'Tom'})

// All versions
db.people.remove({name: 'Tom'})
```

Or just one

```
// New in MongoDB 3.2
db.people.deleteOne({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'},
true)
```

MongoDB's remove() method. If you execute this command without any argument or without empty argument it will remove all documents from the collection.

```
db.people.remove();
```

or

```
db.people.remove({});
```

4. Read

Query for all the docs in the people collection that have a name field with a value of 'Tom':

```
db.people.find({name: 'Tom'})
```

Or just the first one:

```
db.people.findOne({name: 'Tom'})
```

You can also specify which fields to return by passing a field selection parameter. The following will exclude the _id field and only include the age field:

```
db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```

Note: by default, the _id field will be returned, even if you don't ask for it. If you would like not to get the _id back, you can just follow the previous example and ask for the _id to be excluded by specifying _id: 0 (or _id: false).If you want to find sub record like address object contains country, city, etc.

```
db.people.find({'address.country': 'US'})
```

& specify field too if required

```
db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})Remember that the result has a `.pretty()` method that pretty-prints resulting JSON:

db.people.find().pretty()
```