**Samples:**

# **MongoDB CRUD Operations**

# CRUD Operations

https://www.mongodb.com/docs/manual/crud/

## Create Operations

* Let's create a new database called employee\_db.

use employee\_db

use creates a new database, if one does not already exist.

* Next, let's create a collection called employee\_info.

db.createCollection("employee\_info")

* Let's confirm that the collection was successfully created.

show collections

| **fname** | **lname** | **salary** | **departments** | **hiredate** |
| --- | --- | --- | --- | --- |
| john | doe | 70000 | sales, admin | 2018-08-29 |

Converting this table into an object, a set of key:value pairs surrounded by {} looks like this:

{

"fname": "john",

"lname": "doe",

"salary": 70000,

"departments": ["sales", "admin"],

"hiredate": "2018-08-29"

}

Now JSON object inserted it into collection.

**query:**

db.employee\_info.insertOne({

"fname": "john",

"lname": "doe",

"salary": 70000,

"departments": ["sales", "admin"],

"hiredate": "2018-08-29"})

**result:**

{

"acknowledged" : true,

"insertedId" : ObjectId("5d8e6f9ccaa4f8ddbe27296f")

}

The data is first shown in tabular format followed by JSON object format to ease you into thinking about data non-relationally.

| **empno** | **fname** | **lname** | **role** | **salary** | **departments** | **hiredate** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | charlie | rodgers | manager |  | sales, marketing |  |
| 2 | sunil | chakraborty | team lead |  | marketing, finance |  |
| 3 | sally | jones | team lead |  | hr, admin |  |
| 4 | ben | bradley | manager |  | legal |  |
| 5 | radha | desai | worker |  |  |  |
| 6 | shruti | patel | worker |  |  |  |
| 7 | mahesh | iyer | manager |  |  |  |

**InsertMany()**

insertMany() takes an array as input so enlcose all your documents in [] as shown in the query below.

**query:**

db.employee\_info.insertMany([

{

"empno": 1,

"fname": "charlie",

"lname": "rodgers",

"role": "manager",

"departments": ["sales", "marketing"]

},

{

"empno": 2,

"fname": "sunil",

"lname": "chakraborty",

"role": "team lead",

"departments": ["marketing", "finance"]

},

{"empno": 3,

"fname": "sally",

"lname": "jones",

"role": "team lead",

"departments": ["hr", "admin"]

},

{

"empno": 4,

"fname": "ben",

"lname": "bradley",

"role": "manager",

"departments": ["legal"]

},

{

"empno": 5,

"fname": "radha",

"lname": "desai",

"role": "worker"

},

{

"empno": 6,

"fname": "shruti",

"lname": "patel",

"role": "worker"

},

{

"empno": 7,

"fname": "mahesh",

"lname": "iyer",

"role": "manager"

}

])

### **Using the "find()" method**

To display an entire document, use the find() method. When using find() with no additional parameters, every document in the database will be returned.

**Update**

The MongoDB query language supports 3 update operations:

* updateOne()
* updateMany()
* replaceOne()

**Delete Operations**

The MongoDB query language supports 2 delete operations:

* deleteOne()
* deleteMany()

movieDetails collection, there is only 1 movie from the year 2018 with a title "QQ Speed". To perform the exercise, delete this record.

* **query:**
* db.movieDetails.deleteOne({"title": "QQ Speed"})
* Alternatively, you could use:
* db.movieDetails.deleteOne({"year": 2018})

MongoDB offers multiple operators that can be used in its update operations. In the exercise above, we looked at the $set operator.

| **Name** | **Description** |
| --- | --- |
| $set | Sets the value of a field in a document |
| $unset | Removes the specified field from a document |
| $currentDate | Sets the value of a field to current date, either as a Date or a Timestamp |
| $inc | Increments the value of a field by a specified amount |
| $mul | Multiplies the value of the field by the specified amount |
| $min | Only updates the field if the specified value is less than the existing field value. |
| $max | Only updates the field if the specified value is greater than the existing field value. |
| $setOnInsert | Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents. |
| $rename | Renames a field |

**Comparison Query Operators**

| **Name** | **Description** |
| --- | --- |
| $eq | Matches values that are equal to a specified value. |
| $gt | Matches values that are greater than a specified value. |
| $gte | Matches values that are greater than or equal to a specified value. |
| $in | Matches any of the values specified in an array. |
| $lt | Matches values that are less than a specified value. |
| $lte | Matches values that are less than or equal to a specified value. |
| $ne | Matches all values that are not equal to a specified value. |
| $nin | Matches none of the values specified in an array. |

Example

movies which were made in the year 1970 or before.

**query:**

db.movieDetails.find({year: {"$lte": 1970}},

{title: 1, \_id: 0})

movies which were made in or before 1970 but after 1965?

**query:**

db.movieDetails.find({year: {"$lte": 1970, "$gt": 1965}},

{title: 1, \_id: 0})

### **Working with multiple fields**

movies which had a runtime longer than 3 hours and had a imdb.rating higher than 8.

**query:**

db.movieDetails.find({"runtime": {"$gt": 180},

"imdb.rating": {"$gt":8}},

{"title": 1, "\_id": 0})

### The "$ne" query operator

ne stands for **not equal**. Therefore the use of this operator filters out records where the matching condition is not true.

Filter out records where the rating field does not contain the value of 'NOT RATED'.

**query:**

db.movieDetails.find({"rated": {"$ne": "NOT RATED"}},

{"\_id":0, "title":1, "rated":1})

### The "$in" query operator

The $in operator allows us to specify 1 or more values in an array. If any 1 of those filter conditions is matched, a resulting document is returned.

Find movies which were tagged as one of the following genres: Sport, Talk-Show or News. Pretty unusual genres for a movie!

**query:**

db.movieDetails.find({"genres": {"$in": ["Sport", "Talk-Show", "News"]}},

{"\_id":0, "title":1, "genres":1})

Use the $nin query operator (**not in** operator) to match **none** of the values specified in the array.

## Logical Operators

These operators perform one of the following logical operations on the fields:

| **Name** | **Description** |
| --- | --- |
| $and | Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. |
| $or | Joins query clauses with a logical OR returns all documents that match the conditions of either clause. |
| $not | Inverts the effect of a query expression and returns documents that do not match the query expression. |
| $nor | Joins query clauses with a logical NOR returns all documents that fail to match both clauses. |

## Array Operators

| **Name** | **Description** |
| --- | --- |
| $all | Matches arrays that contain all elements specified in the query. |
| $elemMatch | Selects documents if element in the array field matches all the specified $elemMatch conditions. |
| $size | Selects documents if the array field is a specified size. |

### The "$all" query operator

$all matches fields against an array of elements. A document is returned when all the elements listed in the query are found in the document's array field.

Use the $all operator to find movies which were both: History and War genres.

**query:**

db.movieDetails.find({"genres": {"$all": ['History', 'War']}},

{"\_id": 0, "title": 1, "genres": 1})

**Aggregation Operations**

https://www.mongodb.com/docs/manual/aggregation/