CLASS 03: Where, AND, OR & CURD

WHERE:

The where is the parameter of the MongoDB find() method allows us to specify a JavaScript expression that is used to filter the documents returned by the query. This expression can reference the fields of the documents in the collection, as well as any functions defined in the expression.

The query here is specified with the find() method in MongoDB. Then uses the $where operator to find all documents in the student collection where the given field is equal to the specified value. The $where operator is set with a function() that is executed for each document in the collection. The function() is deployed with the expression. This expression uses the this keyword to refer to the field name of the current document being evaluated.

Comparison 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.

AND:

MongoDB provides a variety of logical query operators. The $and operator is one of those operators. The $and operator is used to perform logical AND operations on an array of one or more expressions. The user can use this operator in methods like find(), update(), etc., as per their requirements.The user can also use "AND operation" with the help of a comma (,). The $and operator uses short-circuit evaluation. When the user specifies the same field or operator in more than one expression, then the user can use the "$AND operation".

Syntax:

{ $and: [{ Expression 1 }, { Expression 2 }, { Expression 3 }, ..., { Expression N }] }

Or

{ { Expression 1 }, { Expression 2 }, { Expression 3 },..., { Expression N }}

For example, suppose two expressions (e1 and e2). If the first expression (e1) is evaluated to be false, then the second expression (e2) will not be evaluated.

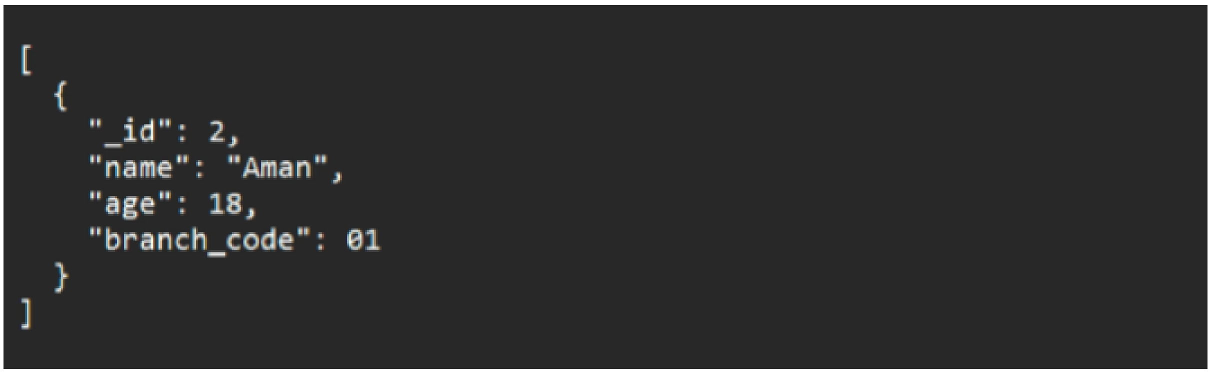
Example:

```
[
  {
    "_id": 1,
    "name": "Adarsh",
    "age": 22,
    "branch_code": 02
  },
  {
    "_id": 2,
    "name": "Aman",
    "age": 18,
    "branch_code": 01
  },
  {
    "_id": 3,
    "name": "Bhavya",
    "age": 19,
    "branch_code": 03
  },
  {
    "_id": 4,
    "name": "Mohak",
    "age": 19,
    "branch_code": 04
  },
  {
    "_id": 5,
    "name": "Sneha",
    "age": 21,
    "branch_code": 02
  }
]
```

Let's consider a simple example where you want to find a student named "Aman" whose branch code is 01 and want to retrieve students who have the branch code 03 or 04 and are younger than 20. You can use the $and operator for this query:

```
db.students.find({
  $and: [
    { name: "Aman" },
    { branch_code: 01 }
  ]
})
```

It retrieves the student with the name "Aman" who has the branch code of 01, the only student in the collection that matches both criteria and retrieves users who satisfy the above two conditions: having a branch code of 03 or 04 and age is less than 20.This query will return the following result from the student's collection.



```
[
  {
    "_id": 2,
    "name": "Aman",
    "age": 18,
    "branch_code": 01
  }
]
```

OR:

In MongoDB, the $or operator is a type of logical operator that performs a logical OR operation on two or more expressions and returns the documents that match any of the expressions. Let’s begin with the syntax of $or operator in MongoDB and using $or in queries.

Syntax:

```
db.collection.find({ $or: [{ cond1 }, { cond2 }, ...{ condN } ]})
```

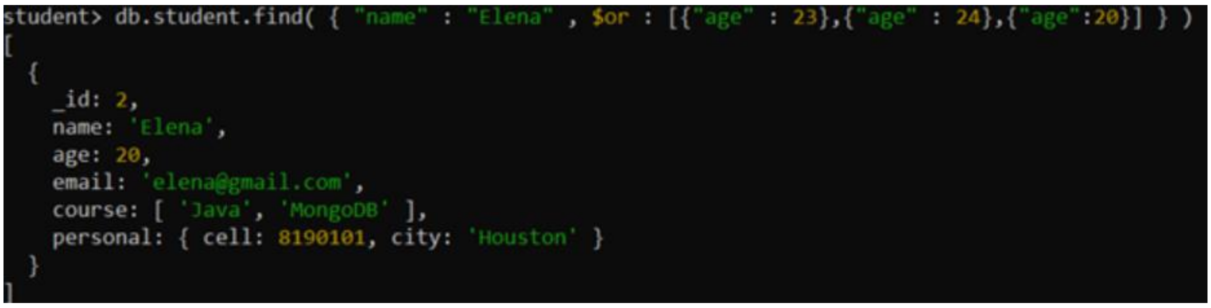
$or operator with an example, lets create a collection with few documents.

```
db.student.insertMany([
      {
         _id: 1,
         name: "Mickel",
         age: 24,
         email: "mickel@gmail.com",
         course: ["Python", "MongoDB"],
         personal: {"cell": 10918129, "city":"Austin"}

      },
      {
         _id: 2,
         name: "Elena",
         age: 20,
         email: "elena@gmail.com",
          course: ["Java", "MongoDB"],
         personal: {"cell": 8190101, "city":"Houston"}
      },
      {
         _id: 3,
         name: "Caroline",
         age: 25,
         email: "caroline@gmail.com",
         course: ["C++", "SQL"],
         personal: {"cell": 3627791, "city":"New York"}
      },
      {
         _id: 4,
         name: "Jimmy",
         age: 23,
         email: "jimmy@gmail.com",
         course: ["Java", "PHP"],
         personal: {"cell": 1337691, "city":"Austin"}
      }
   ]
)
```

For example the first condition is that the name field should be equal to Elena, and the second condition is that the age field should be equal to either 23, 24, or 20. For the second condition, we deployed the $or operator, which indicates the logical OR operation between the three age conditions.

```
db.student.find( { "name" : "Elena" , $or : [{"age" : 23},{"age" : 24},{"age":20}] } )
```

output:



```
student> db.student.find( { "name" : "Elena" , $or : [{"age" : 23},{"age" : 24},{"age":20}] } )
[
  {
    _id: 2,
    name: 'Elena',
    age: 20,
    email: 'elena@gmail.com',
    course: [ 'Java', 'MongoDB' ],
    personal: { cell: 8190101, city: 'Houston' }
  }
]
```

What is CRUD in MongoDB?

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

MongoDB documents are modified by connecting to a server, querying the proper documents, and then changing the setting properties before sending the data back to the database to be updated. CRUD is data-oriented, and it's standardized according to HTTP action verbs.

When it comes to the individual CRUD operations:

- The Create operation is used to insert new documents in the MongoDB database.
- The Read operation is used to query a document in the database.
- The Update operation is used to modify existing documents in the database.
- The Delete operation is used to remove documents in the database.

How to Perform CRUD Operations

Now that we've defined MongoDB CRUD operations, we can take a look at how to carry out the individual operations and manipulate documents in a MongoDB database. Let's go into the processes of creating, reading, updating, and deleting documents, looking at each operation in turn.

Create Operations

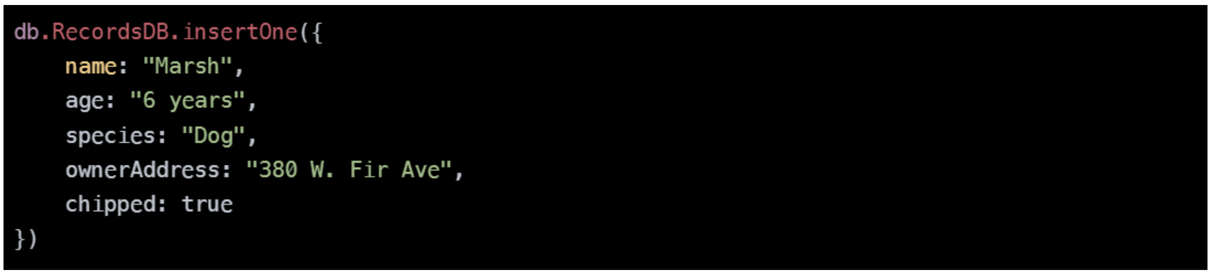
For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed. Create operations in MongoDB target a single collection, not multiple collections. Insert operations in MongoDB are atomic on a single document level.

MongoDB provides two different create operations that you can use to insert documents into a collection:

- db.collection.insertOne()
- db.collection.insertMany()

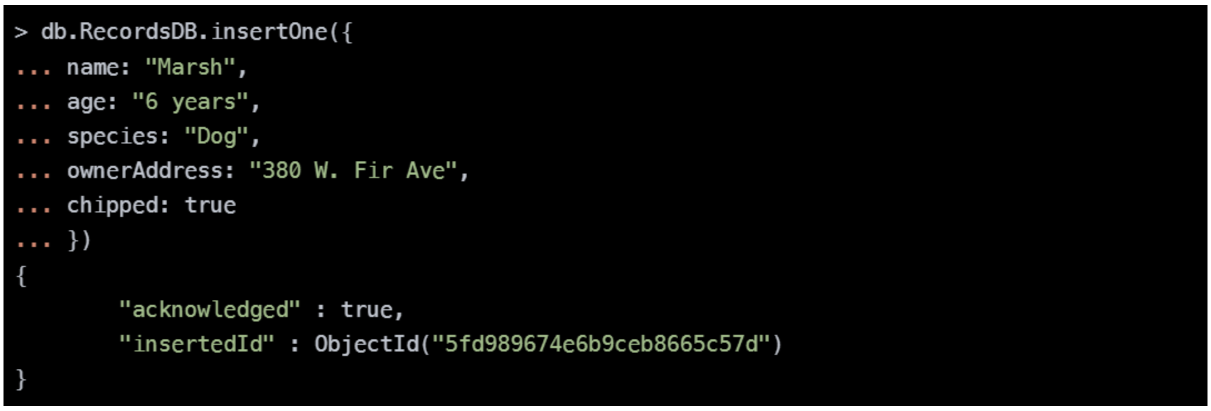
insertOne()

As the namesake, insertOne() allows you to insert one document into the collection. For this example, we're going to work with a collection called RecordsDB. We can insert a single entry into our collection by calling the insertOne() method on RecordsDB. We then provide the information we want to insert in the form of key-value pairs, establishing the schema.



```
db.RecordsDB.insertOne({
    name: "Marsh",
    age: "6 years",
    species: "Dog",
    ownerAddress: "380 W. Fir Ave",
    chipped: true
})
```

If the create operation is successful, a new document is created. The function will return an object where "acknowledged" is "true" and "insertID" is the newly created "ObjectId."



```
> db.RecordsDB.insertOne({
... name: "Marsh",
... age: "6 years",
... species: "Dog",
... ownerAddress: "380 W. Fir Ave",
... chipped: true
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fd989674e6b9ceb8665c57d")
}
```

insertMany()

It's possible to insert multiple items at one time by calling the *insertMany()* method on the desired collection. In this case, we pass multiple items into our chosen collection (*RecordsDB*) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.

```
db.RecordsDB.insertMany([{
    name: "Marsh",
    age: "6 years",
    species: "Dog",
    ownerAddress: "380 W. Fir Ave",
    chipped: true},
      {name: "Kitana",
      age: "4 years",
      species: "Cat",
      ownerAddress: "521 E. Cortland",
      chipped: true}])
```

```
db.RecordsDB.insertMany([{ name: "Marsh", age: "6 years", species: "Dog",
ownerAddress: "380 W. Fir Ave", chipped: true}, {name: "Kitana", age: "4 years",
species: "Cat", ownerAddress: "521 E. Cortland", chipped: true}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5fd98ea9ce6e8850d88270b4"),
                ObjectId("5fd98ea9ce6e8850d88270b5")
        ]
}
```

Update Operations

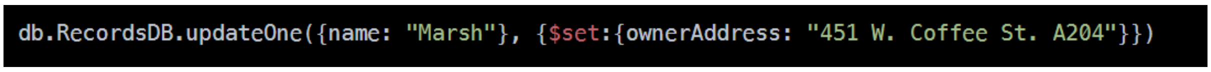
Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

For MongoDB CRUD, there are three different methods of updating documents:

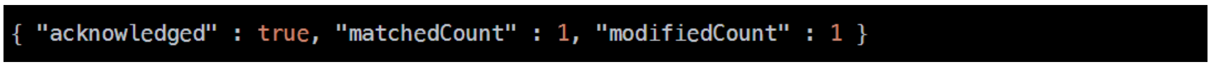
- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

updateOne()

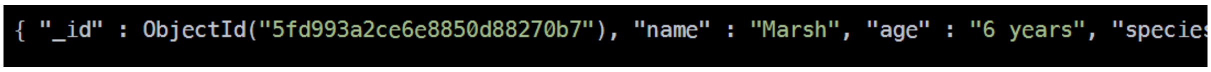
We can update a currently existing record and change a single document with an update operation. To do this, we use the updateOne() method on a chosen collection, which here is “RecordsDB.” To update a document, we provide the method with two arguments: an update filter and an update action.The update filter defines which items we want to update, and the update action defines how to update those items.



```
db.RecordsDB.updateOne({name: "Marsh"}, {$set:{ownerAddress: "451 W. Coffee St. A204"}})
```



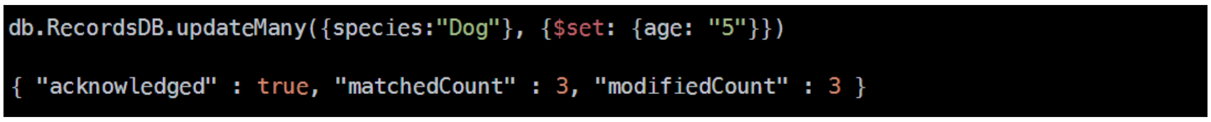
```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```



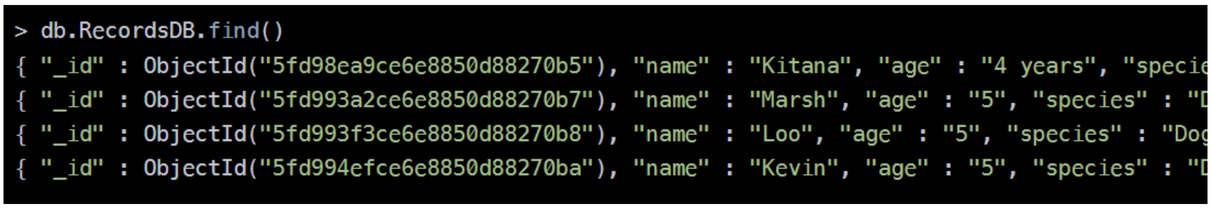
```
{ "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "6 years", "specie
```

updateMany()

updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.



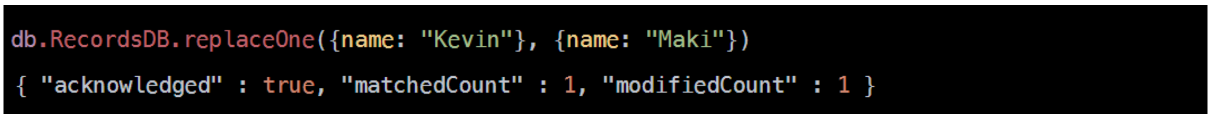
```
db.RecordsDB.updateMany({species:"Dog"}, {$set: {age: "5"}})

{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```



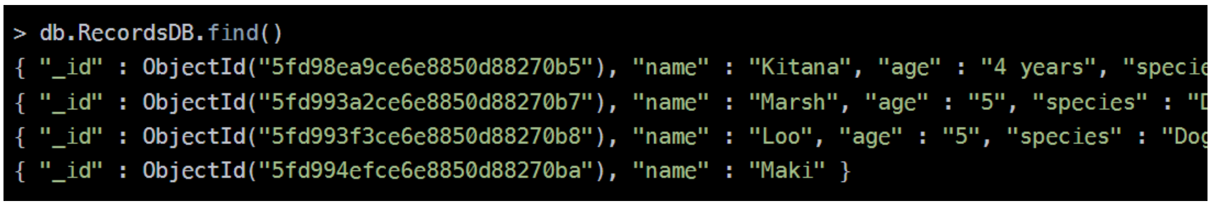
```
> db.RecordsDB.find()
{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "specie
{ "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "D
{ "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" : "Dog
{ "_id" : ObjectId("5fd994efce6e8850d88270ba"), "name" : "Kevin", "age" : "5", "species" : "D
```

replaceOne()

The replaceOne() method is used to replace a single document in the specified collection. replaceOne() replaces the entire document, meaning fields in the old document not contained in the new will be lost.



```
db.RecordsDB.replaceOne({name: "Kevin"}, {name: "Maki"})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```



```
> db.RecordsDB.find()
{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "specie
{ "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "D
{ "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" : "Dog
{ "_id" : ObjectId("5fd994efce6e8850d88270ba"), "name" : "Maki" }
```

Delete Operations

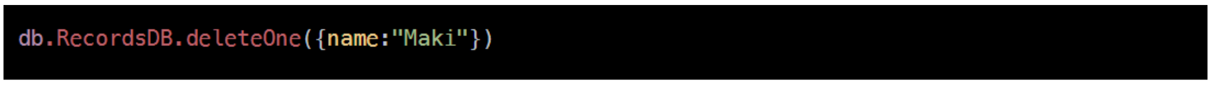
Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.

MongoDB has two different methods of deleting records from a collection:

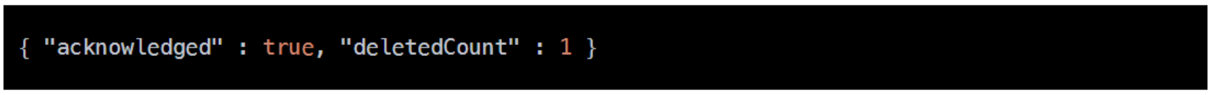
- db.collection.deleteOne()
- db.collection.deleteMany()

deleteOne()

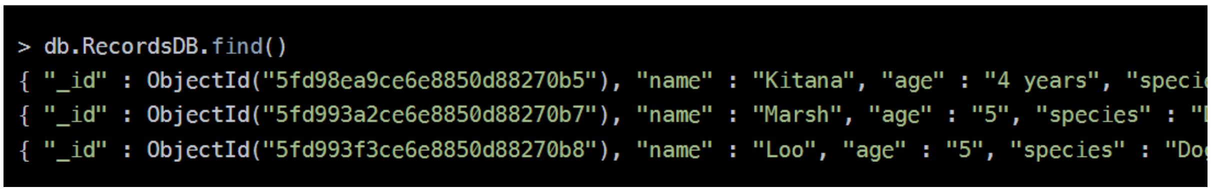
deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.



```
db.RecordsDB.deleteOne({name:"Maki"})
```



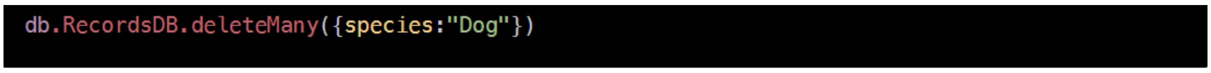
```
{ "acknowledged" : true, "deletedCount" : 1 }
```



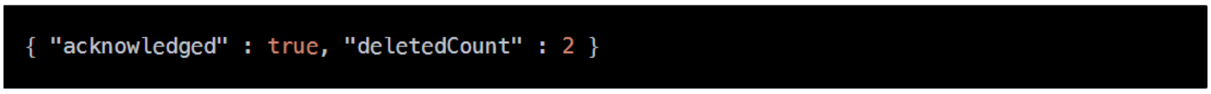
```
> db.RecordsDB.find()
{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "speci
{ "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "
{ "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" : "Do
```

deleteMany()

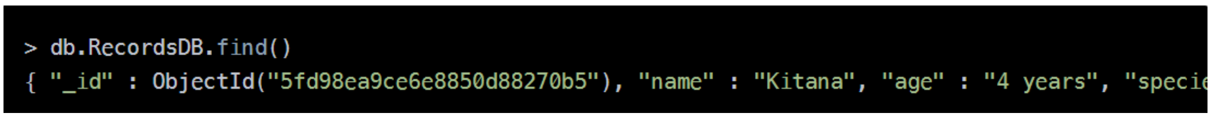
deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in deleteOne().



```
db.RecordsDB.deleteMany({species:"Dog"})
```



```
{ "acknowledged" : true, "deletedCount" : 2 }
```



```
> db.RecordsDB.find()
{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "speci
```