Cheatsheets / **Learn MongoDB**

codecademy

MongoDB CRUD II

The `_id` Field

The `_id` field is assigned to each document as a unique identifier.
By default when inserting a new document into a collection, if you don't specify an `_id` field, then MongoDB will automatically generate a unique ObjectId and assign it as the value for that document's `_id` field.

Create Operation: `.insertOne()`

The `.insertOne()` method inserts a document into a collection. It requires a single parameter, the document to be inserted.
The following command inserts a new document into the `employees` collection:

```
db.employees.insertOne({ _id:
```

```
db.<collection>.insertOne(
   <document>,
   {
      writeConcern: <document>
   }
)
```

codecademy

Create Operation: `.insertMany()`

The .insertMany() method inserts multiple documents into a collection. It requires one argument, an array of documents to be inserted. If the specified collection does not exist, it will create the collection and insert the given documents upon successful execution of the command.
The following command inserts multiple new documents into the pets collection:

```
db.pets.insertMany([
  { name: "Migo", type: "Dog"
  { name: "Snowball", type: "(
])
```

```
db.<collection>.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   }
)
```

Update Operation: `.updateOne()`

The .updateOne() method updates a single document that satisfies a given filter. It requires two parameters, a filter document and an update document that specifies the exact modifications to make.
The following command finds and updates the first document with the name of "Snowball" in pets collection:

```
db.pets.updateOne(
  { name: "Snowball"},
  { $set: { type: "Bengal Cat'
)
```

```
db.<collection>.updateOne(
   <filter>,
   <update>,
   {
     upsert: <boolean>,
     writeConcern: <document>,
     collation: <document>,
     arrayFilters: [ <filterdocument1>,
... ],
     hint:  <document|string>
   }
)
```

codecademy

Update Array Fields

Using dot notation, (`.`), we can access fields in a document at a particular index or position of the array in order to update them.
Consider the following document from a collection called `superheros` :

```
{
  name: "Superman"
  powers: ["Flight", "Lasers",
}
```

We can use dot notation to update the second value in the `powers` array:

```
db.superheros.updateOne(
  { name: "Superman"},
  { $set:
    { "powers.1": "Laser Eyes'
  }
)
```

codecademy

The `$push` Operator

The `$push` operator can be used with the `.updateOne()` and `.updateMany()` methods to append a specified value to an array field.

The following command updates the document with an `_id` of `1` by adding a new value to the `scores` field:

```
db.students.updateOne(
   { _id: 1 },
   { $push: { scores: 89 }}
)
```

```
{ $push: { <field1>: <value1>, ... } }
```

The `upsert` Option

The `upsert` option is an optional parameter that combines the update and insert functionality. If its value is assigned to `true` and no matching document is found in the collection, it will insert it. If it is set to `false`, its default behavior, it will not insert a new document if no matching document is found.

```
db.employee.updateOne(
  { name:"Eric Wikstrom" },
  { $set: { department: "Softw
  { upsert: true }
)
```

codecademy

Update Operation: `.updateMany()`

The `.updateMany()` method updates all documents that satisfy a specific filter criteria. It requires two arguments: a filter document and an update document that specifies the modifications to apply.
The following command updates multiple documents from the `employees` collection:

```
db.employees.updateMany(
  { salary: 70000 },
  { $set: { salary: 85000 }}
)
```

```
db.<collection>.updateMany(
   <filter>,
   <update>,
   {
     upsert: <boolean>,
     writeConcern: <document>,
     collation: <document>,
     arrayFilters: [ <filterdocument1>,
... ],
     hint:  <document|string>
   }
)
```

Update Operation: `.findAndModify()`

The `.findAndModify()` method, modifies and returns a single document. By default, it returns the original document (not the modified version). The modified document can be returned by including the `new` option and assigning it to `true`.
If no matching document is found in the collection, a new document will be inserted if the `upsert` option is set to `true`.
The following example finds a document in the `hotels` collection and modifies it:

```
db.hotels.findAndModify({
    query: { "name" : "Radegas
    update: { $set: { "rating'
})
```

```
db.<collection>.findAndModify({
    query: <document>,
    sort: <document>,
    remove: <boolean>,
    update: <document or aggregation
pipeline>,
    new: <boolean>,
    fields: <document>,
    upsert: <boolean>,
    bypassDocumentValidation: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    arrayFilters: [ <filterdocument1>,
... ],
    let: <document>
})
```

codecademy

Delete Operation: `.deleteMany()`

The .deleteMany() method removes all documents that match a given filter criteria. It takes in a single required parameter, the filter criteria to match multiple documents.
The following example removes all documents with a status of sold from the televisions collection:

```
db.televisions.deleteMany({
  status: "sold"
})
```

```
db.<collection>.deleteMany(
   <filter>,
   {
      writeConcern: <document>,
      collation: <document>
   }
)
```

Delete Operation: `.deleteOne()`

The .deleteOne() method removes a single document from a collection. It has a single required parameter which is a filter criteria to match a specific document to delete.
The following example removes a single document with a title of "King Burger Extravaganza" from the recipes collection:

```
db.recipes.deleteOne({
  title: "King Burger Extravag
});
```

```
db.<collection>.deleteOne(<filter>,
<options>)
```

codecademy

Replace Operation: `.replaceOne()`

The `.replaceOne()` method replaces a single document within the collection based on filter criteria. It takes two parameters, a filter criteria to match a document in the collection, and the new document to replace it with.

The following example replaces a document in the `inventory` collection:

```
db.inventory.replaceOne(
  { name: "Chango Chile"},
  { name: "Chango Chili", sco
)
```

```
db.<collection>.replaceOne(
  <filter_document>,
  <replacement_document>
)
```

Print Share