Lecture 5 - CRUD operations in MongoDB

Instructor: Suresh Melvin Sigera
Email: suresh.sigera@cuny.csi.edu

Agenda

- Insert operation
- Read operation
- Comparison operators
- Projection
- Update operation
- Delete operation
- QA

MongoDB - data model

Database system structure: **Instance → databases → collections → documents**

- Database
- Collection
  - Collection of documents, usually of a similar structure
- Document
  - MongoDB document = one JSON object
  - Each document
    - belongs to exactly one collection
    - has a unique identifier `_id`

MongoDB - CRUD Operations

Overview

- `db.collection.insert()`
  - Inserts a new document into a collection
- `db.collection.update()`
  - Modifies an existing document / documents or inserts a new one
- `db.collection.remove()`
  - Deletes an existing document / documents
- `db.collection.find()`
  - Finds documents based on filtering conditions
  - Projection and / or sorting may be applied too

MongoDB - insert operation

Inserts are the basic method for adding data to MongoDB. To insert a single document, use the collection's insertOne method.

Document identifier (`_id` field)

- The provided value must be unique within the collection
- When missing, it is generated automatically (`ObjectId`)

```
db.<collection>.insertOne({ // document });
```

MongoDB - insert operation

The main focus when using the **`insertOne`** command is on the document. The document must be a well formed JSON expression. Document fields can contain arrays, or **even other documents**. Here is an example where we insert a movie title Stand by me into the CSC424 database, movies collection. Note that the unique identifier `ObjectId` is auto-generated:

```
> db.movies.insertOne({"title" : "Stand by Me"})
```

MongoDB - insert operation

If you need to insert multiple documents into a collection, you can use **`insertMany`**. This method enables you to pass an array of documents to the database. This is far more efficient because your code will not make a round a round trip to the database for each document inserted, but will insert them in bulk.

```
db.<collection>.insertMany({ // document });
```

MongoDB - insert operation

```
db.movies.insertMany([
                        {"title" : "Ghostbusters"},
                        {"title" : "E.T."},
                        {"title" : "Blade Runner"}
                        {"title" : "007"}
                        {"title" : "War"}
                        {"title" : "IP Man"}
]);
```

MongoDB - insert operation

Note:

In versions of MongoDB prior to 3.0, `insert()` was the primary method for inserting documents into MongoDB. MongoDB drivers introduced a new CRUD API at the same time as the MongoDB 3.0 server release. As of MongoDB 3.2 the mongo shell also supports this API, which includes `insertOne` and `insertMany` as well as several other methods. The goal of the current CRUD API is to make the semantics of all CRUD operations consistent and clear across the drivers and the shell. While methods such as `insert()` are **still supported for backward compatibility, they should not be used in applications going forward**. You should instead prefer `insertOne` and `insertMany` for creating documents.

MongoDB - read operations

To query data from MongoDB collection, you need to use MongoDB's **`find()`** method. To display the results in a formatted way, we can use **`pretty()`** method.

```
db.movies.find().pretty()
```

```
{ "_id" : ObjectId("572630ba11722fac4b6b4991"), "title" : "Stand by Me" }
{ "_id" : ObjectId("572630ba11722fac4b6b4992"), "title" : "Ghostbusters" }
{ "_id" : ObjectId("572630ba11722fac4b6b4993"), "title" : "E.T" }
{ "_id" : ObjectId("572630ba11722fac4b6b4994"), "title" : "Blade Runner" }
{ "_id" : ObjectId("572630ba11722fac4b6b4995"), "title" : "007" }
{ "_id" : ObjectId("572630ba11722fac4b6b4996"), "title" : "War" }
{ "_id" : ObjectId("572630ba11722fac4b6b4997"), "title" : "IP Man" }
```

MongoDB - read operations

One of the most common operations on any database is querying the database for information, which will then form the basis of reports and information that might appear on a web page. In MongoDB the primary command to perform a query is **`db.<collection>.find()`**. There is also a variation **`db.<collection>.findOne()`**, which only returns a single result.

```
db.movies.findOne({title:"War"})
```

```
{ "_id" : ObjectId("5c69fd2125754c82230aa3c4"), "title" : "War" }
```

MongoDB - read operations

It's important to note the return value from `find()` is not the actual result set itself, rather what the MongoDB documentation calls a cursor.

https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/#iterate-a-cursor-in-the-mongo-shell

This is an iteration, a resource pointer, which can then be used to extract results. The command shell will automatically iterate the cursor 20 times (default), so there is no need for extra logic. If there are more than 20 results, you will be prompted:

```
Type "it" for more.
```

MongoDB - read operations, importing data

MongoDB provides the mongoimport utility that lets you bulk load data directly into a collection of the database. It reads from a file and bulk loads the data into a collection. These methods are not suitable for production environment. The following three file formats are supported by mongoimport:

JSON: In this format you have JSON blocks per line, which represent a document.
CSV: This is a comma-separated file.
TSV: TSV files are same as CSV files; the only difference is it uses a tab as the separator.

```
mongoimport --db tv-shows --collection shows --file data.json --jsonArray
```

MongoDB - read operations

It's important to note the return value from `find()` is not the actual result set itself, rather what the MongoDB documentation calls a cursor.

https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/#iterate-a-cursor-in-the-mongo-shell

This is an iteration, a resource pointer, which can then be used to extract results. The command shell will automatically iterate the cursor 20 times (default), so there is no need for extra logic. If there are more than 20 results, you will be prompted:

```
Type "it" for more.
```

MongoDB - comparison operators

The following table summarizes the key aggregation pipeline operators (https://docs.mongodb.com/manual/meta/aggregation-quick-reference/#operator-expressions) that are currently available:

Arithmetic	This set currently consists of 15 basic operators, which let you perform arithmetic operations including $add, $subtract, $multiply, and so on.
Boolean	Used to formulate complex conditions. These include $and, $or, and $not.
Comparison	Comparison Standard comparison operators between values including $cmp (compare), $eq (equal), $ne (not equal), $gt (greater than), $gte (greater than or equal), $lt (less than), and $lte (less than or equal).

MongoDB - comparison operators

Conditional	Used to formulate conditions. This set includes $cond (ternary if/then/else), $ifNull (first non-null expression), and $switch (standard C-like switch/case).
Date	This set currently consists of 18 date operators which include: ● Converting between a date string and a BSON date object (for example, $dateFromString, $dateToString) ● Day, week, and month (for example, $dayOfMonth, $isoDayOfWeek) ● Hour, minute, and second (for example, $hour, $minute) Newly added in version 4.0 is $toDate, which converts a given value into a BSON date. It should also be noted that $add and $subtract can also handle date arithmetic.

MongoDB - comparison operators

Object	This set includes $mergeObjects and $objectToArray; both are self-explanatory.
String	These 18 operators perform string manipulation, and include $concat (concatenating strings), $substr (substring), and $trim (removing white space from beginnings and ends), among others.
Type	Type This group of 10 operators is used to perform data type conversion, and includes $convert (generic anything-to-anything conversion), $toInt (convert to integer), $toString (convert to string), and $toObjectId (converts a value to a BSON ObjectId object).

MongoDB - comparison operators

Array	These operators are needed when a document contains an array property. Notable operators in this set include: ● $in: Used to determine if a value is in an array or not ● $size: Gives you the number of elements in the array ● $slice: Returns a subset of the array

MongoDB - comparison operators

Download the JSON file https://raw.githubusercontent.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/master/Databases/MongoDB/catalog.books.json and import it via the MongoDB console using the following command.

```
mongoimport --db catalog --collection books --type json --drop --file catalog.books.json
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part21.txt

MongoDB - comparison operators

`$eq` compares two values and returns: true when the values are equivalent or else false when the values are not equivalent. Syntax: `{field: {$eq : value}}`

A number of books that has 300 pages.

```
db.books.find({
          pageCount:{$eq:300}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part21.txt

MongoDB - comparison operators

`$ne` selects the documents where the value of the field is not equal (i.e. !=) to the specified value.
Syntax: `{field: {$ne : value}}`

Books that don't have 300 pages.

```
db.books.find({
          pageCount:{$ne:300}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part22.txt

MongoDB - comparison operators

`$lte` selects the documents where the value of the field is less than or equal to the specified value.
Syntax: `{field: {$lte : value}}`

The books that have pages less than 300.

```
db.books.find({
          pageCount:{$lte:300}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part24.txt

MongoDB - comparison operators

`$lt` selects the documents where the value of the field is less than the specified value. Syntax:
`{field: {$lt : value}}`

The books that have pages less than 300.

```
db.books.find({
          pageCount:{$lt:300}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part27.txt

MongoDB - comparison operators

`$gte` selects the documents where the value of the field is greater than or equal to the specified value.
Syntax: `{field : {$gte : value}}`

The books that have pages greater than 300.

```
db.books.find({
          pageCount:{$gte:300}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part24.txt

MongoDB - comparison operators

`$gt` matches values that are greater than the values specified in the query. Syntax:
`{field :{$gt: value})`

The books that have pages greater than 300.

```
db.books.find({
          pageCount:{$gt:300}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part26.txt

MongoDB - comparison operators

`$in` select the documents where the value of a field equals any value in the specified array. Syntax:
`{field: { $in : [<value1>, <value2>,…. <valueN> ]})`

If the field holds an array, then the `$in` operator selects the documents whose field holds an array that contains at least one element that matches a value in the specified array.

```
db.books.find({
          pageCount:{$in: [300,400]}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part28.txt

MongoDB - comparison operators

`$nin` selects the documents where the field value is not in the specified array or the field does not exist. It is just opposite of `$in`. Syntax:

```
{field: {$nin: [<value1>,<value2>,….<valueN>]}}
```

```
db.books.find({
          pageCount:{$nin: [300,400]}
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part29.txt

MongoDB - logical operators

`$and` performs a logical AND operation on an array of two or more expressions (e.g.`<expression1>`, `<expression2>`, etc.) and selects the documents that satisfy all the expressions in the array.

syntax: `{ $and :[{<expression1>},{ <expression2>},...,{ <expressionN>}]}`

```
db.books.find({
     $and:[
          {"categories" : "Java"}, {"status" : "PUBLISH"}
     ]
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part30.txt

MongoDB - logical operators

The `$or` operator performs a logical OR operation on an array of two or more `<expressions>` and selects the documents that satisfy at least one of the `<expressions>`.

syntax: `{$or: [{<expression1>},{<expression2>}, . ., {<expressionN>}]}`

```
db.books.find({
     $or:[
          {"categories" : "Java"}, {"categories" : "Python"}
     ]
}).count()
```

MongoDB - logical operators

`$nor` performs a logical NOR operation on an array of one or more query expression and selects the documents that fail all the query expressions in the array.

Syntax: `{$nor:[{ <expression1>},{<expression2>}, ...,{<expression} ]}`

```
db.books.find({
     $nor:[
          {"categories" : "Java"}, {"categories" : "Python"}
     ]
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part30.txt

MongoDB - projection

The projection delineates which fields within a document are included in or excluded from the final output. In MongoDB, the projection takes the form of a JSON expression consisting of key:value pairs. The key is the name of the field, and the value is either 1 or 0. Fields to be included are assigned 1. Fields to be excluded are assigned 0. By default, all fields are included.

For this illustration, assume a collection books where each document has the fields _id, title, isbn, pageCount, publishedDate,thumbnailUrl, shortDescription, longDescription, status, authors, and categories. We start by searching for all documents that have Java in the category, and only want to see isbn and the title.

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part33.txt

MongoDB - projection

```
db.books.find({"categories":"Java"}, {"isbn":1, "title":1})
```

```
{ "_id" : 28, "title" : "Hibernate Search in Action", "isbn" : "1933988649" }
{ "_id" : 30, "title" : "jQuery in Action, Second Edition", "isbn" : "1935182323" }
{ "_id" : 24, "title" : "Java Persistence with Hibernate", "isbn" : "1932394885" }
{ "_id" : 10, "title" : "OSGi in Depth", "isbn" : "193518217X" }
{ "_id" : 22, "title" : "Hibernate in Action", "isbn" : "193239415X" }
{ "_id" : 21, "title" : "3D User Interfaces with Java 3D", "isbn" : "1884777902" }
{ "_id" : 2, "title" : "Android in Action, Second Edition", "isbn" : "1935182722" }
{ "_id" : 23, "title" : "Hibernate in Action (Chinese Edition)" }
{ "_id" : 9, "title" : "Griffon in Action", "isbn" : "1935182234" }
{ "_id" : 52, "title" : "Spring Dynamic Modules in Action", "isbn" : "1935182307" }
{ "_id" : 63, "title" : "POJOs in Action", "isbn" : "1932394583" }
{ "_id" : 66, "title" : "Seam in Action", "isbn" : "1933988401" }
{ "_id" : 69, "title" : "Struts 2 in Action", "isbn" : "193398807X" }
{ "_id" : 68, "title" : "Open Source SOA", "isbn" : "1933988541" }
{ "_id" : 73, "title" : "Spring Roo in Action", "isbn" : "193518296X" }
{ "_id" : 77, "title" : "Mule in Action", "isbn" : "1933988967" }
{ "_id" : 117, "title" : "Managing Components with Modeler", "isbn" : "1932394524k-e" }
```

MongoDB - projection in arrays

You can use the dot notation to project specific fields inside documents embedded in an array.

```
db.books.find({"authors":"Dave Crane"}, {"authors":1, "title":1})
```

```
{ "_id" : 59, "title" : "Ajax in Action", "authors" : [ "Dave Crane", "Eric Pascarello with Darren James" ] }
{ "_id" : 61, "title" : "Prototype and Scriptaculous in Action", "authors" : [ "Dave Crane", "Bear Bibeault with Tom Locke" ] }
{ "_id" : 60, "title" : "Ajax in Practice", "authors" : [ "Dave Crane", "Jord Sonneveld and Bear Bibeault with Ted Goddard", "Chris Gray", "Ram Venkataraman", "Joe Walker" ] }
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part34.txt

MongoDB - update operation

All updates require at least two arguments. The first specifies which documents to update, and the second defines how the selected documents should be modified. There are two styles of modification; in this section we're going to focus on targeted modifications, which are most representative of MongoDB's distinct features.

```
db.<collection>.updateOne({      // document},
                                 {// field update operators}
);
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part9.txt

MongoDB - update operation

To take an example, suppose that we decides to update the Chris's hobbies :

```
db.users.updateOne({"name": "Chris"}, {
     $set: {
          "hobbies": [
               {title: "Sports", "frequency": 2},
               {title: "Cooking", "frequency": 3},
               {title: "Hiking", "frequency": 3}
          ]
     }
})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part10.txt

MongoDB - update operation

```
{
        "_id" : ObjectId("5c6eec7f2e17055235f11400"),
        "name" : "Chris",
        "hobbies" : [
                {
                        "title" : "Sports",
                        "frequency" : 2
                },
                {
                        "title" : "Cooking",
                        "frequency" : 3
                },
                {
                        "title" : "Hiking",
                        "frequency" : 3
                }
        ]
}
```

MongoDB - update operation

`updateOne` updates only the first document found that matches the filter criteria. If there are more matching documents, they will remain unchanged. To modify all of the documents matching a filter, use `updateMany`. `updateMany` follows the same semantics as `updateOne` and takes the same parameters. The key difference is in the number of documents that might be changed.

```
db.<collection>.updateMany({// document},
                           {// field update operators}
);
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part11.txt

MongoDB - update operation

```
db.users.updateMany({"hobbies.title": "Sports"}, {
    $set: {
        "isSporty": true
    }
})

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

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part11.txt

MongoDB - update operation

`updateMany` provides a powerful tool for performing schema migrations or rolling out new features to certain users. Suppose, for example, we want to add the age, and the phone number to Chri's profile. We can use updateMany method. For example:

```
db.users.updateOne({"name": "Chris"}, {
    $set: {
        "age": 40,
        "phone": 2122221222
    }
})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part12.txt

MongoDB - incrementing

The `$inc` operator can be used to change the value for an existing key or to create a new key if it does not already exist. It is very useful for updating analytics, karma, votes, or anything else that has a changeable, numeric value.

```
db.users.updateOne({"name": "Manuel"}, {
    $inc: {
        "age": 1,
    }
})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part13.txt

MongoDB - decrementing

The `$inc` update operator accepts positive and negative values. A negative value effectively decrements the specified field.

```
db.users.updateOne({"name": "Manuel"}, {
    $inc: {
        "age": -1,
    }
})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part14.txt

MongoDB - incrementing and decrementing

"`$inc`" is similar to "`$set`", but it is designed for incrementing (and decrementing) numbers. "`$inc`" can be used only on values of type integer, long, or double. If it is used on any other type of value, it will fail. This includes types that many languages will automatically cast into numbers, like nulls, booleans, or strings of numeric characters:

Also, the value of the "`$inc`" key must be a number. You cannot increment by a string, array, or other non-numeric value. Doing so will give a “Modifier "`$inc`" allowed for numbers only” error message. To modify other types, use "`$set`" or one of the following array operations.

MongoDB - deleting a specified field

The `$unset` operator lets you delete a given field, as in this example:

```
db.users.updateMany({"isSporty": true}, {
    $unset: {
        "phone": "",
    }
})
```

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

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part15.txt

MongoDB - rename a specified field

The `$unset` operator lets you delete a given field, as in this example:

```
db.users.updateMany({}, {
    $rename: {
        "name": "username",
    }
})
```

{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part16.txt

MongoDB - upsert

An **upsert** is a special type of update. If no document is found that matches the update criteria, a new document will be created by combining the criteria and updated documents. If a matching document is found, it will be updated normally. Upserts can be handy because they can eliminate the need to “seed” your collection: you can often have the same code create and update documents.

```
db.users.updateOne({"name": "Suresh"}, {
    $set: {
        "age": 25,
            "hobbies":[
                  {"title": "Cooking"},
                  {"frequency":3}
            ],
            "isSporty": true
    }
})
```

MongoDB - upsert

```
db.users.updateOne({"name": "Suresh"}, {
    $set: {
        "age": 25,
            "hobbies":[
                  {
                        "title": "Cooking",
                        "frequency":3
                  }
            ],
            "isSporty": true
    }
},
{upsert:true})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part17.txt

MongoDB - upsert

```
{
        "acknowledged" : true,
        "matchedCount" : 0,
        "modifiedCount" : 0,
        "upsertedId" : ObjectId("5c6f23a3c736568b4071c404")
}
```

MongoDB - update operators

The following modifiers are available for use in update operations;

- **$currentDate** : Sets the value of a field to current date, either as a Date or a Timestamp
- **$inc** : Increments the value of the field by the specified amount
- **$max**: Only updates the field if the specified value is greater than the existing field value
- **$min**: Only updates the field if the specified value is less than the existing field value
- **$mul**: Multiplies the value of the field by the specified amount
- **$rename**: Renames a field
- **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
- **$set**: Sets the value of a field in a document
- **$unset**: Removes the specified field from a document

MongoDB - array update operators

- **$**: Acts as a placeholder to update the first element that matches the query condition in an update
- **$addToSet**: Adds elements to an array only if they do not already exist in the set
- **$pop**: Removes the first or last item of an array

MongoDB - updating matched array elements

If you want to match an entire array within a document, you can use the `$elemMatch` operator. This is particularly useful if you have multiple documents within your collection, some of which have some of the same information. This can make a default query incapable of finding the exact document you are looking for. This is because the standard query syntax doesn't restrict itself to a single document within an array.

```
db.users.find({$and:
     [
          {"hobbies.title": "Sports"},
          {"hobbies.frequency": {$gte: 3}}
     ]
}).count()
```

MongoDB - updating matched array elements

```
db.users.find({
     "hobbies":{
          $elemMatch:{
               "title": "Sports",
               "frequency": {$gte:3}
          }
     }
}).count()
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part18.txt

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part19.txt

MongoDB - updating matched array elements

```
db.users.updateMany({
                "hobbies":{
                          $elemMatch:{
                               "title": "Sports",
                               "frequency": {$gte:3}
                          }
                }
     },{ $set:{ "hobbies.$.highFrequency": true } })
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part20.txt

MongoDB - updating matched array elements

```
{
        "_id" : ObjectId("5c6eec7f2e17055235f113fd"),
        "hobbies" : [
                {
                        "title" : "Sports",
                        "frequency" : 3,
                        "highFrequency" : true
                },
                {
                        "title" : "Cooking",
                        "frequency" : 6
                }
        ],
        "isSporty" : true,
        "username" : "Max"
}
```

MongoDB - removing documents

Now that there's data in our database, let's delete it. The CRUD API provides `deleteOne` and `deleteMany` for this purpose. Both of these methods take a filter document as their first parameter.

The filter specifies a set of criteria to match against in removing documents. To delete the document that match the name "Chris", we use use deleteOne in the mongo shell as illustrated below.

```
db.movies.deleteOne({name: "Chris"})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part36.txt

MongoDB - removing documents

Now that there's data in our database, let's delete it. The CRUD API provides `deleteOne` and `deleteMany` for this purpose. Both of these methods take a filter document as their first parameter.

The filter specifies a set of criteria to match against in removing documents. In these cases, deleteOne will delete the first document found that matches the filter. To delete the document that match the name "Chris", we use use deleteOne in the mongo shell as illustrated below.

```
db.movies.deleteOne({name: "Chris"})
```

https://github.com/sureshmelvinsigera/CSC-424-Advanced-Database-Management-Systems/blob/master/week5/mongodb-part36.txt

MongoDB - removing documents

To delete more than one document matching a filter, use `deleteMany`.

```
db.users.deleteMany({age: {$gt: 20}})
```

```
{ "acknowledged" : true, "deletedCount" : 3 }
```

MongoDB - removing documents

In versions of MongoDB prior to 3.0, `remove()` was the primary method for deleted documents in MongoDB. MongoDB drivers introduced the `deleteOne` and `deleteMany` methods at the same time as the MongoDB 3.0 server release. The shell began supporting these methods in MongoDB 3.2.

While `remove()` still supported for backward compatibility, you should use `deleteOne` and `deleteMany` in your applications. The current CRUD API provides a cleaner set of semantics and, especially, for multidocument operations helps application developers avoid a couple of common pitfalls with the previous API.

MongoDB - removing documents

It is possible to use `deleteMany` to remove all documents in a collection. Removing documents is usually a fairly quick operation; however, if you want to clear an entire collection, it is faster to drop it.

```
db.users.deleteMany({})
```

```
db.users.drop()
```

QA