

# Topics

---

1. MongoDB Basic Operations
2. More on Searching for Documents
3. Replacing Documents
4. Updating Documents

# MongoDB

---

- 4 Basic Operations of Persistent Storage?
- CRUD
- CREATE, READ, UPDATE, DELETE
- SQL equivalent?
- INSERT, SELECT, UPDATE, DELETE
- MongoDB equivalent?

- Create
  - `db.collection.insertOne()`
  - `db.collection.insertMany()`
- Read
  - `db.collection.find()`
- Can include a query criteria, projection and a cursor modifier

# Projection

- db.MyCustomers.find({creditlimit:{\$lt:500000}}, {creditlimit:1, name:1});

The screenshot shows a MongoDB playground interface. On the left, the code pane displays the following JavaScript code:

```
1 // Connected to Another Connection with default database test
2 db.MyCustomers.find({creditlimit:{$lt:500000}}, {creditlimit:1, name:1});
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

The right pane shows the "Playground Result" containing the query results:

```
1 [
2   {
3     "_id": 141,
4     "name": "Euro+ Shopping Channel",
5     "creditlimit": 227600
6   },
7   {
8     "_id": 124,
9     "name": "Mini Gifts Distributors Ltd",
10    "creditlimit": 210500
11 },
12 {
13   "_id": 103,
14   "name": "Atelier graphique",
15   "creditlimit": 21000
16 },
17 {
18   "_id": 112,
19   "name": "Signal Gift Stores",
20   "creditlimit": 71800
21 }
```

# Cursor Modifier

- db.MyCustomers.find({creditlimit:{\$lt:500000}},
- {creditlimit:1, name:1}).limit(2);

The screenshot shows a MongoDB playground interface in a code editor. The code being run is:

```
1
2 db.MyCustomers.find({creditlimit:{$lt:500000}},
3   {creditlimit:1, name:1}).limit(2);
```

The output window displays the results of the query:

```
1 [
2   {
3     "_id": 141,
4     "name": "Euro+ Shopping Channel",
5     "creditlimit": 227600
6   },
7   {
8     "_id": 124,
9     "name": "Mini Gifts Distributors Ltd",
10    "creditlimit": 210500
11  }
12 ]
```

The interface includes standard code editor features like file navigation, search, and a terminal tab at the bottom.

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

- MongoDB Deletes
- db.collection.deleteOne()
- db.collection.deleteMany()
- There is a MongoDB method that provides the ability to perform bulk insert, update, and delete operations.
- db.collection.bulkWrite()

# bulkWrite

```
db.pizzas.bulkWrite( [
  { insertOne: { document: { _id: 3, type: "beef", size: "medium", price: 6
  { insertOne: { document: { _id: 4, type: "sausage", size: "large", price:
  { updateOne: {
    filter: { type: "cheese" },
    update: { $set: { price: 8 } }
  } },
  { deleteOne: { filter: { type: "pepperoni"} } },
  { replaceOne: {
    filter: { type: "vegan" },
    replacement: { type: "tofu", size: "small", price: 4 }
  } }
] )
catch( error ) {
  print( error )
```

# MongoDB Demonstration

- “\$or”/“\$and” Operator

- The “\$or” operator is used to check an array of possible criteria. The query returns the document if either condition is true.
- ```
db.raffle.find({"$or" : [{"ticket_no" : 725}, {"winner" : true}]})
```
- 
- ```
db.raffle.find({"$or" : [{"ticket_no" : {"$in" : [725, 542, 390]}}, {"winner" : true}]})
```
- See the following example of “\$and”:
- ```
db.users.find({"$and" : [{"x" : {"$gt" : 1}}, {"x" : {"$lt" : 4}}]})
```

# MongoDB Demonstration

- \$mod as a search criteria
- db.users.find({"id\_num" : {"\$mod" : [5, 1]}})
- This query returns documents if the key “id\_num” is 1, 6, 11, or etc.
- “\$mod” operator checks if the value of key “id\_num” divided by the first value have a remainder of the second value.

# ● Searching by ObjectId



```
File Edit Selection View Go Run ... ⏪ ⏩ ⏴ ⏵ Search ... t.mapleLeafs:6.json playground-3.mongodb.js /c/Users/Russe/playground-3.mongodb.js ... Playground Result ...  
Currently connected to RussellDatabase with default database te...  
1  
2 db.MyCustomers.insertOne(  
3 {  
4   name : "Toys R Us",  
5   city : "Toronto",  
6   country : "Canada",  
7   phone : null,  
8   email : "ToysrUS.com",  
9   creditlimit : 25000  
10 }  
11 );  
12 db.MyCustomers.find({_id :  
13 ObjectId("65fb1e7161435fcbe3119793")});  
14  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Playground output ... ^ x  
Ln 12, Col 1 Spaces: 4 UTF-8 CRLF () JavaScript
```

The screenshot shows a MongoDB playground interface within a code editor. On the left, there's a sidebar with various icons. The main area has tabs for 't.mapleLeafs:6.json' and 'playground-3.mongodb.js'. The playground result shows the execution of a MongoDB query to find a document with a specific ObjectId.

```
1 [  
2 {  
3   "_id": {  
4     "$oid": "65fb1e7161435fcbe3119793"  
5   },  
6   "name": "Toys R Us",  
7   "city": "Toronto",  
8   "country": "Canada",  
9   "phone": null,  
10  "email": "ToysrUS.com",  
11  "creditlimit": 25000  
12 }  
13 ]
```

- You can remove the document using ObjectId()

A screenshot of a MongoDB playground interface within a code editor. The interface includes a top bar with File, Edit, Selection, View, Go, Search, and window control buttons. Below the bar, there are tabs for 't.mapleLeafs:6.json' and 'playground-3.mongodb.js /c:/Users/Russe/playground-3.mongodbjs'. A sidebar on the left shows icons for file, search, and other tools. The main area displays a MongoDB query and its results.

Currently connected to RussellDatabase with default database t

```
1
2
3 db.MyCustomers.deleteOne({_id : ObjectId("65fb1e7161435fcbe3119793")});
4
5
```

Playground Result

```
1 {
2   "acknowledged": true,
3   "deletedCount": 1
4 }
```

Bottom navigation includes PROBLEMS, OUTPUT (selected), DEBUG CONSOLE, TERMINAL, PORTS, and Playground output. Status bar at the bottom shows Line 4, Col 2, Spaces: 2, JSON, and a file icon.

# Searching for Null

## ● Null

- Null means the value of a key is unknown.
- Assume the following documents:
- db.c.find()
- { "\_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
- { "\_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }
- { "\_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }
- To find documents with the NULL value for the "y" key:
  - db.c.find({"y" : null})
- { "\_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
- To find all documents that a specific key does not exist among their keys.
- db.c.find({"z" : null})
- { "\_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
- { "\_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }
- { "\_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }

- db.MyCustomers.find({phone: null});
  - {
  - "\_id": {
  - "\$oid": "65ff58c05981e81f2adec4c9"
  - },
  - "name": "Toys R Us",
  - "city": "Toronto",
  - "country": "Canada",
  - "phone": null,
  - "email": "ToysrUS.com",
  - "creditlimit": 25000
  - }

# db.MyCustomers.find({phone2: null});

- Returns documents where phone2 is set as null and documents without a phone2

The screenshot shows the MongoDB playground interface. On the left, there's a sidebar with various icons. The main area has a toolbar with File, Edit, Selection, View, Go, Run, and other options. A search bar is at the top right. Below it, there are tabs for 6.json and playground-3.mongodb.js, with playground-3.mongodb.js currently selected. The code editor shows the following command:

```
1
2
3 db.MyCustomers.find({phone2: null});
4
5
```

Below the code, a message says "Currently connected to RussellDatabase with default database". To the right, the "Playground Result" tab is open, showing the query results. The results are numbered from 2 to 18, with the first result being a document object:

```
2 {
  "_id": 141,
  "name": "Euro+ Shopping Channel",
  "city": "Madrid",
  "country": "Spain",
  "phone": "(91) 555 94 44",
  "email": "EuroShops.com",
  "creditlimit": 227600,
  "enroldate": {
    "$date": "2022-04-22T12:45:30Z"
  },
  "contact": {
    "first": "Diego",
    "last": "Frevre"
  }
},
{
  "_id": 103,
  "name": "Atelier graphique",
  "city": "Nantes",
  "country": "France",
  "phone": null
}
```

At the bottom of the results, there are buttons for "Edit Document" and "Delete Document".

# db.MyCustomers.find({phone2: {\$ne :null}});

The screenshot shows a MongoDB playground interface within a code editor window. The top bar includes File, Edit, Selection, View, Go, Run, and a search field. The left sidebar has icons for file operations like Open, Save, and Find.

The main area displays a code editor with a file named `playground-3.mongodb.js`. The code contains two MongoDB find queries:

```
1
2
3 db.MyCustomers.find({phone2: {$ne :null}});
4
5 db.MyCustomers.find({phone2 : {$not: {$eq:null}}});
6
7
```

The result pane on the right shows the output of the second query, which returns two documents. The first document is:

```
2 {
3   "_id": 112,
4   "name": "Signal Gift Stores",
5   "city": "Las Vegas",
6   "country": "USA",
7   "phone": "7025551838",
8   "email": "Signal.com",
9   "creditlimit": 71800,
10  "enroldate": {
11    "$date": "2013-11-05T14:10:30Z"
12  },
13  "contact": {
14    "first": "Jean",
15    "last": "King"
16  },
17  "phone2": "7024443333"
18 }
19 {
```

Below this, there is an option to "Edit Document". The second document is partially visible:

```
20  "_id": 124,
21  "name": "Mini Gifts Distributors Ltd",
22  "city": "San Rafael",
23  "country": "USA",
24  "phone": "4155551450",
25  "phone2": "4153334444",
26  "phone3": "4166662222",
```

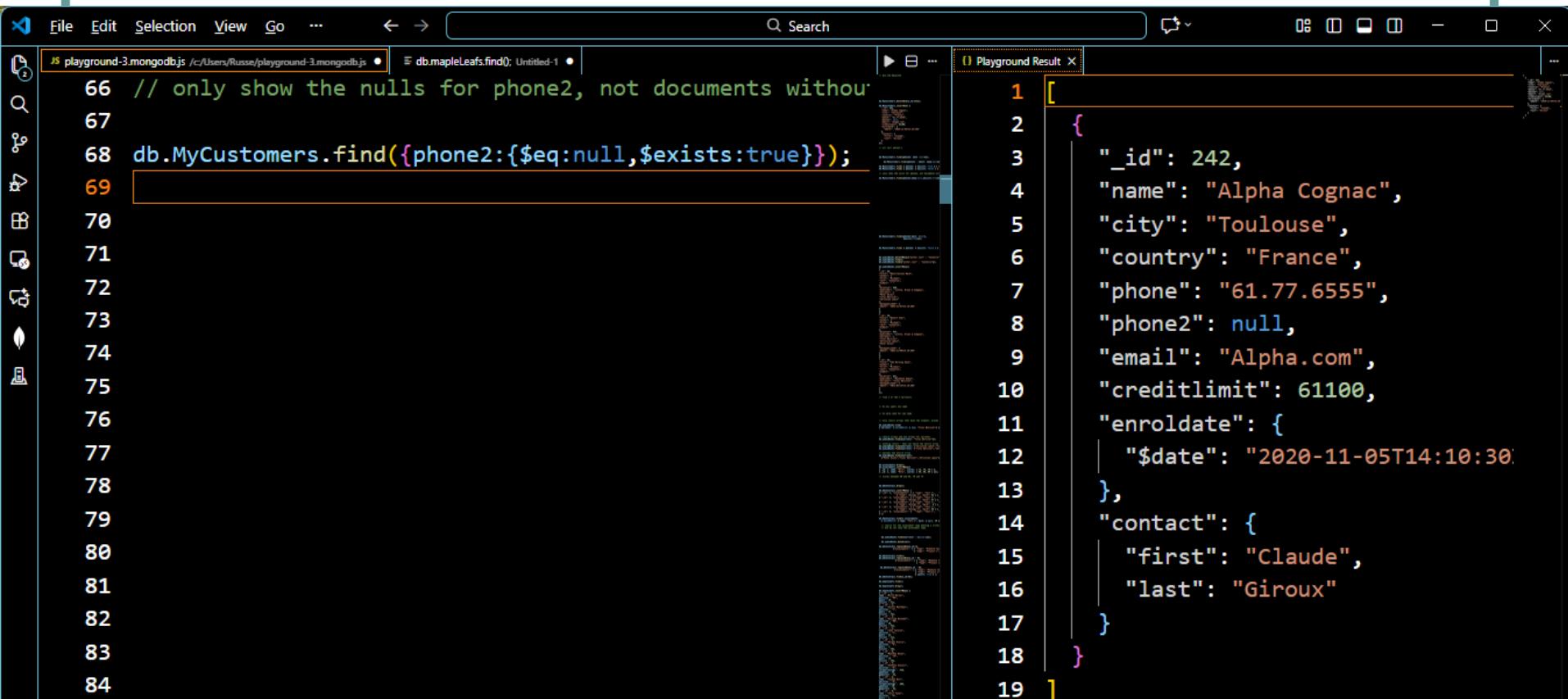
The bottom navigation bar includes PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS tabs. The OUTPUT tab is currently selected.

# MongoDB

- “\$exists” Operator
  - The \$exists operator matches documents that contain or do not contain a specified field, including documents where the field value is null.
  - Documents that contain the key phone2
- `db.MyCustomers.find( { phone2: { $exists: true } } )`
  - and documents that do not
- `db.MyCustomers.find( { phone2: { $exists: false } } );`

# Omit documents without a phone2

- db.MyCustomers.find({phone2:{\$eq:null, \$exists:true}});



A screenshot of a MongoDB playground interface. On the left, there is a code editor window titled "playground-3.mongodb.js" containing the following JavaScript code:

```
// only show the nulls for phone2, not documents without it
db.MyCustomers.find({phone2:{$eq:null,$exists:true}});
```

The code editor has line numbers from 66 to 84. To the right of the code editor is a results pane titled "Playground Result". It displays a single document as JSON:

```
[{"_id": 242, "name": "Alpha Cognac", "city": "Toulouse", "country": "France", "phone": "61.77.6555", "phone2": null, "email": "Alpha.com", "creditlimit": 61100, "enroldate": {"$date": "2020-11-05T14:10:30"}, "contact": {"first": "Claude", "last": "Giroux"}}
```

# Omit documents without a phone2

- db.MyCustomers.find({phone2:{\$in: [null], "\$exists":true}});

The screenshot shows a MongoDB playground interface with two tabs: "playground-3.mongodb.js" and "Playground Result".

**Script Tab (Left):**

```
61
62 db.MyCustomers.find({phone2:{$in: [null],
63 | | | | | | | | "$exists":true}});
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
```

**Results Tab (Right):**

```
1 [
2 {
3   "_id": 242,
4   "name": "Alpha Cognac",
5   "city": "Toulouse",
6   "country": "France",
7   "phone": "61.77.6555",
8   "phone2": null,
9   "email": "Alpha.com",
10  "creditlimit": 61100,
11  "enroldate": {
12    "$date": "2020-11-05T14:10:30Z"
13  },
14  "contact": {
15    "first": "Claude",
16    "last": "Giroux"
17  }
18 }
19 ]
```

# Searching Arrays

- “\$all” Operator

- db.food.insert({"\_id" : 1, "fruit" : ["apple", "banana", "peach"]})
- db.food.insert({"\_id" : 2, "fruit" : ["apple", "kumquat", "orange"]})
- db.food.insert({"\_id" : 3, "fruit" : ["cherry", "banana", "apple"]})
- Let's say we want or find all documents with both apple and banana elements.
- db.food.find({fruit : {\$all : ["apple", "banana"]}})
- {"\_id" : 1, "fruit" : ["apple", "banana", "peach"]}
- {"\_id" : 3, "fruit" : ["cherry", "banana", "apple"]}
- To check key/values pairs with the exact match does not return the above result. It looks for documents with only values apple and banana.
- db.food.find({"fruit" : ["apple", "banana"]})
- The following query does not return any documents:
- db.food.find({"fruit" : ["banana", "apple", "peach"]})
- Returns 0 documents

- db.audioBooks.find({narrator: {\$all :["Christine Lakin","Titus Welliver"]}});
- Vs
- db.audioBooks.find({narrator: {\$all :["Titus Welliver","Christine Lakin"]}});
- The order did not matter
- Vs
- db.audioBooks.find({narrator: {\$all :["Titus Well","Christine Lakin"]}});
- This does not work, it requires all names match.
- Vs
- db.audioBooks.find({narrator: {\$all :["Titus Welliver"]}});
- This matches all documents with array narrators, but it also returns a match to a narrator Titus Welliver where the narrator field does not hold an array of values – just one value.

# \$all used with arrays

The screenshot shows a MongoDB playground interface within a code editor. On the left, there's a sidebar with various icons. The main area has tabs for 'appleLeafs:6.json' and 'JS db.audioBooks.find({narrator: /c/Users/Russe/playground-3.mongodbjs}).review'. The 'JS' tab is active, displaying a MongoDB query:

```
1 db.audioBooks.find({narrator:  
2   {$all :["Titus Welliver" , "Christine Lakin"]}});  
3  
4
```

The results tab on the right shows the output of the query, which is a list of documents. One document is fully visible:

```
2 {  
11   "publisher": "Little, Brown & Company",  
12   "narrator": [  
13     "Peter Giles",  
14     "Titus Welliver",  
15     "Christine Lakin"  
16   ],  
17   "datepublished": {  
18     "$date": "2023-11-07T14:10:30Z"  
19   }  
20 },  
21 {  
22   "_id": 29,  
23   "title": "Desert Star",  
24   "author": {  
25     "first": "Michael",  
26     "last": "Connelly",  
27     "middle": " "  
28   },  
29   "duration": 577,  
30   "publisher": "Little, Brown & Company",  
31   "narrator": [  
32     "Titus Welliver",  
33     "Christine Lakin",  
34     "Peter Giles"  
35   ],  
36 }
```

At the bottom, there are tabs for 'PROBLEMS', 'OUTPUT' (which is selected), 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. There's also a 'Playground output' dropdown and some other UI elements.

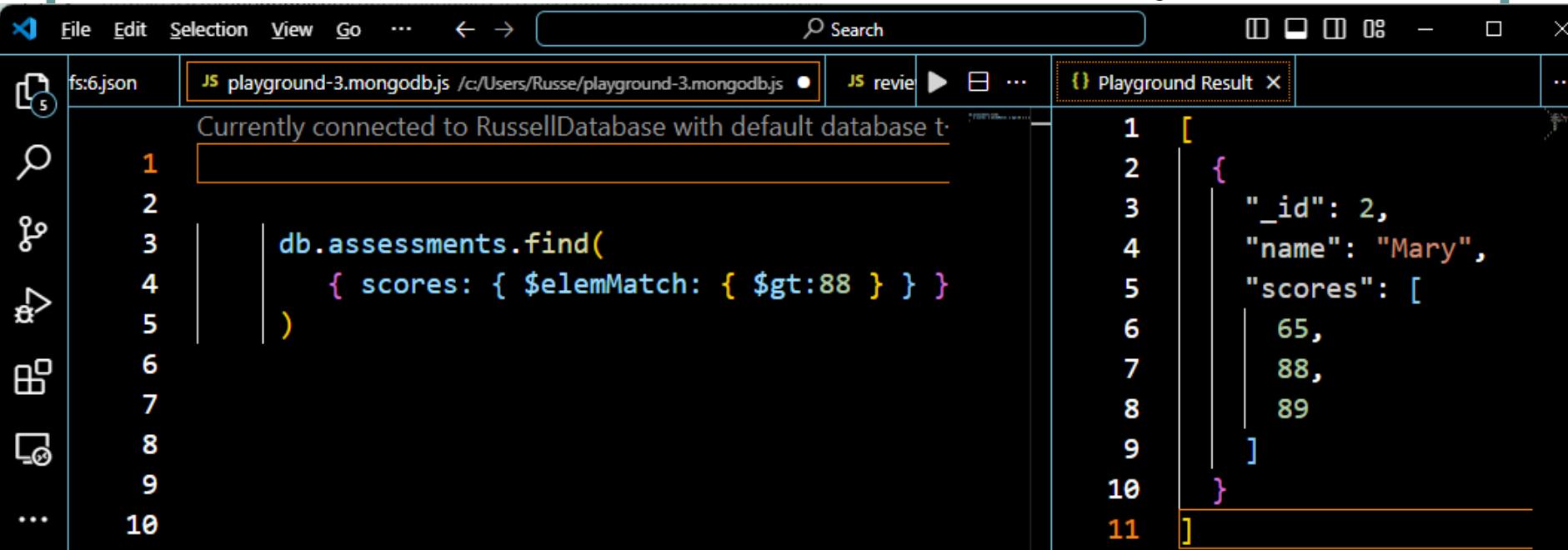
- To only include arrays that have a particular narrator name and exclude non arrays with that narrator name
  - Returning all the documents where Titus works with other narrators, so exclude the ones where he is alone
- 
- db.audioBooks.find(
  - { narrator: { \$elemMatch: { \$eq: "Titus Welliver"} } }
  - )

# Find without \$all or \$elemMatch

- This technique returns no matches because there are 3 elements in the narrator array and the query only includes 2 elements.
  - db.audioBooks.find({narrator: ["Christine Lakin", "Titus Welliver"]});
  - db.audioBooks.find({narrator: ["Titus Welliver", "Christine Lakin"]});
- This query returns a document because it matches an entire array.
  - db.audioBooks.find({narrator:
    - ["Peter Giles", "Titus Welliver", "Christine Lakin"]});

- db.assessments.insertMany([
  - { \_id: 1, name: "Bill", scores: [ 72, 75, 78 ] },
  - { \_id: 2, name: "Mary", scores: [ 65, 88, 89 ] }])
- db.assessments.find(
  - { scores: { \$elemMatch: { \$gte: 80, \$lt: 85 } } }
  - )
- Returns []
- db.assessments.find(
  - { scores: { \$elemMatch: { \$gte: 70, \$lt: 75 } } }
  - )
- Returns Bill document

- db.assessments.find()
- { scores: { \$elemMatch: { \$gt:88 } } }
- One element of Mary's score array matches, none of Bill's score array match



The screenshot shows a MongoDB playground interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, ..., Search, Playground Result.
- Left Sidebar:** Includes icons for file operations like Open, Save, Find, Replace, and a dropdown menu.
- Current File:** fs:6.json
- Code Area:** A code editor window containing the following JavaScript code:

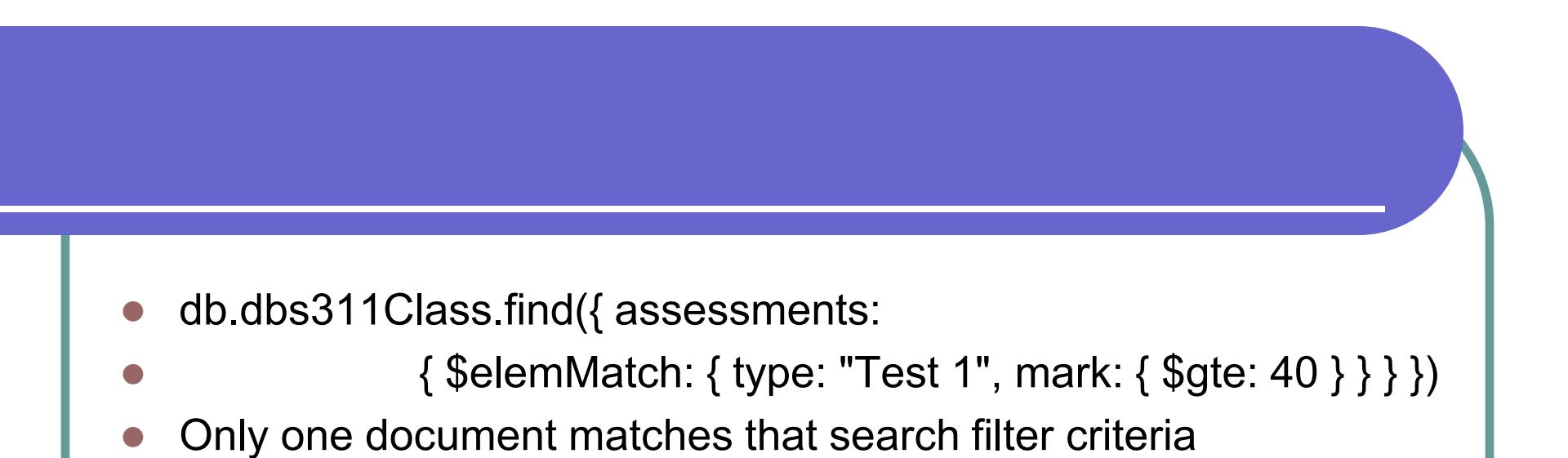
```
1
2
3     db.assessments.find(
4         { scores: { $elemMatch: { $gt:88 } } }
5     )
```
- Result Area:** Shows the query results for the document where Mary has a score greater than 88.

```
1 [ 2 {
3     "_id": 2,
4     "name": "Mary",
5     "scores": [
6         65,
7         88,
8         89
9     ]
10 }
11 ]
```

# \$elemMatch

- db.dbs311Class.insertMany( [
- { "\_id": 1, "assessments": [ { "type": "Test 1", "mark": 40 },  
•                                 { "type": "Assign 1", "mark": 45 } ] },
- { "\_id": 2, "assessments": [ { "type": "Test 1", "mark": 32 },  
•                                 { "type": "Assign 1", "mark": 37 } ] },
- { "\_id": 3, "assessments": [ { "type": "Test 1", "mark": 22 },  
•                                 { "type": "Assign 1", "mark": 8 } ] },
- { "\_id": 4, "assessments": [ { "type": "Test 1", "mark": 37 },  
•                                 { "type": "Assign 1", "mark": 38 } ] },
- { "\_id": 5, "assessments": { "type": "Test 1", "mark": 35 } }
- ]);

- db.dbs311Class.find({ assessments: { \$elemMatch: { type: "Test 1", mark: { \$gte: 40 } } } })
- Only one document matches that search filter criteria



A screenshot of a MongoDB playground interface in a code editor. The playground shows a query being run against a database named 'test'. The query is:

```

1
2
3 db.dbs311Class.find(
4   { assessments: { $elemMatch: { type: "Test 1", mark: { $gte: 40 } } } }
5 )

```

The results pane displays a single document:

```

1 [
2   {
3     "_id": 1,
4     "assessments": [
5       {
6         "type": "Test 1",
7         "mark": 40
8       },
9       {
10        "type": "Assign 1",
11        "mark": 45
12      }
13    ],
14    "studentname": "Bill"
15  }

```

The interface includes a navigation bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. Below the playground are tabs for afs:2.json, test.mapleLeafs:4.json, test.mapleLeafs:6.json, playground-3.mongodb.js, review.m, and Playground Result. The bottom of the screen shows standard VS Code status bars for Problems, Output, Debug Console, Terminal, Ports, and a JSON tab.

- // search for the assessment type meeting a criteria
- // but do not show the assessment type
- db.dbs311Class.find({ assessments:
  - { \$elemMatch: { type: "Test 1", mark: { \$gte: 40 } } },
  - {"assessments.mark":1});
- Or
- db.dbs311Class.find({ assessments:
  - { \$elemMatch: { type: "Test 1", mark: { \$gte: 40 } } },
  - {"assessments.type":0});

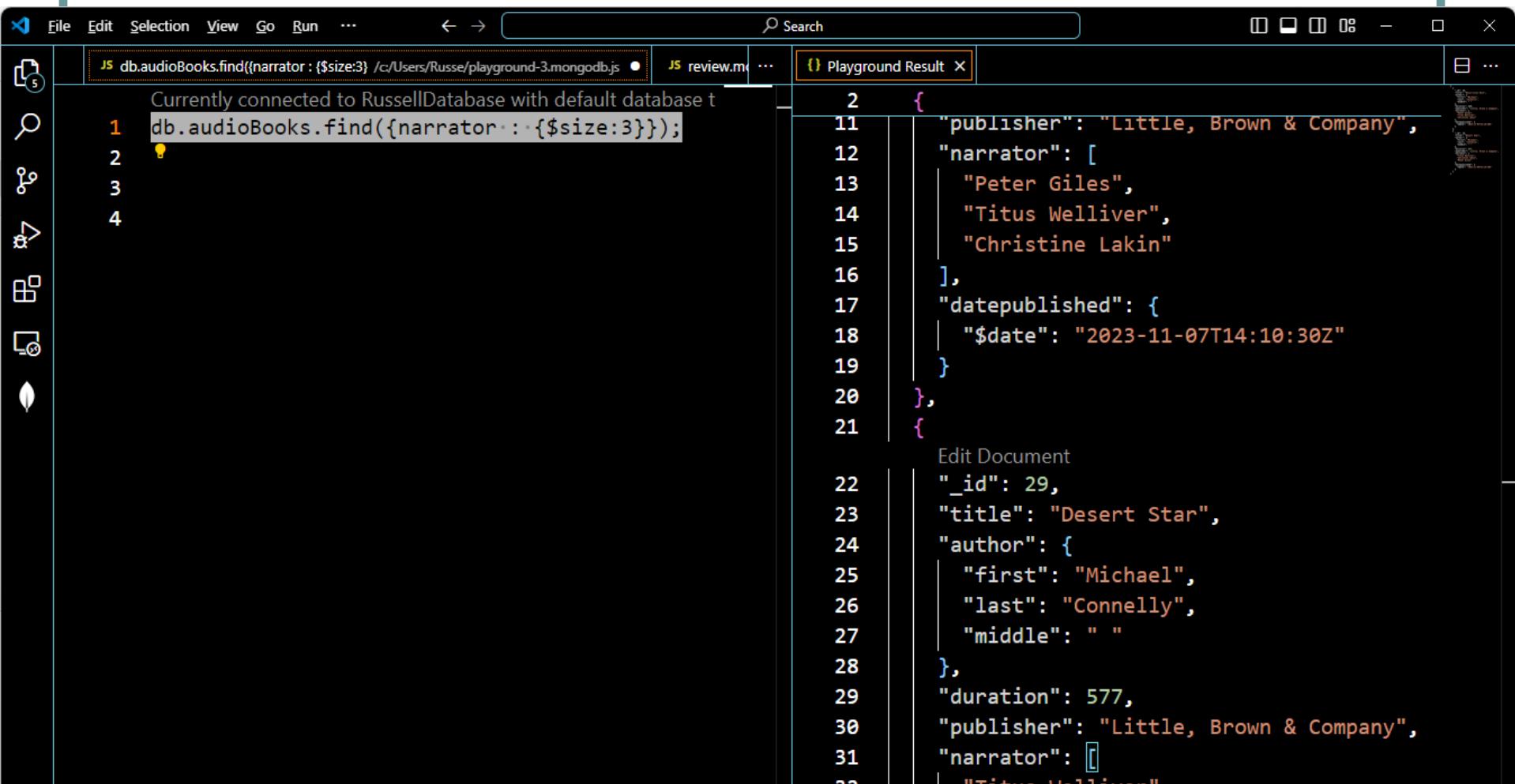
- Projection document {"assessments.type":0});
- [ {
- "\_id": 1,
- "assessments": [
- {
- "mark": 40
- },
- {
- "mark": 45
- }
- ]
- }]

# MongoDB Demonstration

---

- “\$size” Operator
- To query arrays for a given size, the “\$size” operator is used.
- `db.food.find({"fruit" : {"$size" : 3}})`
- You cannot combine the "\$size" operator with other \$ conditional operators.

## ● db.audioBooks.find({narrator : {\$size:3}});



The screenshot shows the MongoDB Compass interface with the following details:

- File**, **Edit**, **Selection**, **View**, **Go**, **Run**, **...** menu bar.
- Search** input field.
- Toolbars**: Includes icons for file operations, search, and database selection.
- Left Panel**: Shows a tree view of the database structure.
- Current Database**: RussellDatabase
- Current Collection**: audioBooks
- Current Document**: playgroundResult
- Code Area**: Shows the query `db.audioBooks.find({narrator : {$size:3}});`.
- Results Area**: Displays the results of the query, showing three documents with narrators Peter Giles, Titus Welliver, and Christine Lakin.

```
2   {
11     "publisher": "Little, Brown & Company",
12     "narrator": [
13       "Peter Giles",
14       "Titus Welliver",
15       "Christine Lakin"
16     ],
17     "datepublished": {
18       "$date": "2023-11-07T14:10:30Z"
19     }
20   },
21   {
22     "_id": 29,
23     "title": "Desert Star",
24     "author": {
25       "first": "Michael",
26       "last": "Connelly",
27       "middle": " "
28     },
29     "duration": 577,
30     "publisher": "Little, Brown & Company",
31     "narrator": [
32       "Titus Welliver"
33     ]
34   }
```

- You can also return the size in bytes of the collection
- db.audioBooks.dataSize();
- 1 3787

# MongoDB Demonstration

---

## ● Document Replacement

- To replace a document with a new one, the `replaceOne` function is used.
- `db.collection.replaceOne()`
- Replaces the first matching document in the collection that matches the filter, using the replacement document.
- Replaces at most a single document that match a specified filter even though multiple documents may match the specified filter.
- There is no `db.collection.replaceMany` option

# MongoDB Demonstration

---

## ● Document Replacement

- Assume the following user document: { "name" : "joe", }
- Let's replace this document with the new one
- db.people.replaceOne( { "name" : "joe"}, { "name" : "joe", "friends" : 32, "enemies" : 2} )
- If multiple documents have “name”: “joe” the first one found will be replaced
- If you want a specific document – not the first then:
- Use a unique id in your filter to replace a specific document in the collection

# replaceOne()

- db.dbs311Class.replaceOne({\_id : 1},  
• {"assessments": [ { "type": "Midterm Test", "mark": 40 },  
• { "type": "Project 1", "mark": 45 } ] }));
- Replaces one document
- db.dbs311Class.replaceOne({\_id : 9},  
• {"assessments": [ { "type": "Midterm Test", "mark": 40 },  
• { "type": "Project 1", "mark": 45 } ] }));
- "matchedCount": 0,
- "modifiedCount": 0,
- "upsertedCount": 0

# replaceOne()

- Upsert is a Boolean option that defaults to false,
- when it is true the unmatched document is added
- db.dbs311Class.replaceOne({\_id : 9},
  - {"assessments": [ { "type": "Midterm Test", "mark": 40 },
    - { "type": "Project 1", "mark": 45 } ] },
      - { upsert: true } );
  - "acknowledged": true,
  - "insertedId": 9,
  - "matchedCount": 0,
  - "modifiedCount": 0,
  - "upsertedCount": 1

# MongoDB Demonstration

## ● Update Documents

- The update function is used to update the value of a key value in a document.
- `update()` takes two parameters:
  - A query document
  - Locates document to update
  - Modifier document
  - Describes changes to make
- The update operation is atomic:
  - If there are two update requests coming to the server, the one reaches the server first will be executed and when it is done the second one will be applied

# MongoDB Demonstration

- db.members.update({\_id : 1}, {\$inc: {"points":10}});
- DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
- The warning means this will be removed in a future version of MongoDB and is replaced by ...
- Another method to alter a value in a key value pair
- “\$set”
- The \$set operator replaces the value of a field with the specified value.
- If the field does not exist, \$set will add a new field with the specified value

# MongoDB Demonstration

- “\$set” (Add a Field)
- db.members.updateOne(
  - {\_id : 2},
  - {\$set: {"location": "Toronto"} } );
  - {
    - "acknowledged": true,
    - "insertedId": null,
    - "matchedCount": 1,
    - "modifiedCount": 1,
    - "upsertedCount": 0
  - }

- A new collection
- db.mapleLeafs.insertMany( [
- { "\_id" : 1,
- name : "Nick Robertson",
- position : "LW",
- goals : 25,
- assists : 51},
- { "\_id" : 2,
- name : "Austin Matthews",
- position : "C", ...

# Updates

---

- Increase all the goals for maple leafs by 1
- db.runCommand(
  - {
  - update: "mapleLeafs",
  - updates: [
    - {
    - q: {},
    - u: { \$inc: { goals: 1 } },
    - multi: true}]
    - });
  - Don't use this method, use updateOne or updateMany

- Another method to add goals to a total for a single document
- db.mapleLeafs.update({name : "Austin Matthews"}, {\$inc: {goals:3}});
- DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.

- The best way to add to the goal total
- db.mapleLeafs.updateOne({name : "Austin Matthews"}, {\$inc: {goals:3}});
- How do you subtract from a total
- db.mapleLeafs.updateOne({name : "Austin Matthews"}, {\$inc: {goals: -3}});

- db.mapleLeafs.updateOne({\_id:2}, {\$set : {"location" : "Toronto"}})
  - db.mapleLeafs.updateMany({}, {\$set : {"location" : "Toronto"}});

The screenshot shows the MongoDB playground interface within a Visual Studio Code extension. The left sidebar contains icons for file operations, search, and other tools. The main area has tabs for 'JSON' (with 6 files) and 'JS'. A search bar at the top right contains the text 'Search'. Below the tabs, a message says 'Currently connected to RussellDatabase with default data...'. The code editor displays the following MongoDB query:

```
1 db.mapleLeafs.find();
2
3
4
5 db.mapleLeafs.updateMany({}, {
6   $set : {"location" : "Toronto"}
7 })
```

The result pane on the right shows the response from the database:

```
1 {
2   "acknowledged": true,
3   "insertedId": null,
4   "matchedCount": 8,
5   "modifiedCount": 8,
6   "upsertedCount": 0
7 }
```

At the bottom, the 'OUTPUT' tab is selected in the status bar, which also includes 'PROBLEMS' (16), 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The status bar also shows 'Playground output' and file navigation information.

# MongoDB Demonstration

- “\$set” (Modify a Field)

- The “\$set” operator sets the value of a field if the field exists.
- Let's say we want to change the value of “favorite book”:
- db.users.updateOne({"name" : "joe"},
  - {"\$set" : {"favorite book" : "Green Eggs and Ham"}})
- Using the “\$set” operator, we can change the value of “favorite book” to an array. The user has different favorite books:
  - db.users.updateOne({"name" : "joe"},
    - {"\$set" : {"favorite book" :
      - ["Cat's Cradle", "Foundation Trilogy", "Ender's Game"]}}

# MongoDB Demonstration

- “\$unset” Operator

- used to remove a key value pair from a document.
- Suppose the user does not have any favorite books and we want to remove the “favorite book” key.
- db.users.updateOne({"name" : "joe"},  
... {"\$unset" : {"favorite book" : 1}})
- The document now is
- db.users.findOne()  

```
{  "_id" : ObjectId("4b253b067525f35f94b60a31"),
  "name" : "joe",
  "age" : 30,
  "sex" : "male",
  "location" : "Wisconsin" }
```

# MongoDB Demonstration

- “\$set” (Embedded Documents)

```
● "_id": 8,  
●   "title": "In Pieces",  
●   "author": {  
●     "first": "Sally",  
●     "last": "Field"  
●   },  
●   "duration": 641,  
● db.audioBooks.updateOne({_id : 8}, {$set: {"author.first": "Toronto"}},  
● {  
●   "acknowledged": true,  
●   "insertedId": null,  
●   "matchedCount": 1,  
●   "modifiedCount": 1,  
●   "upsertedCount": 0  
● })
```

# Remaining Schedule

- Lab 9 has been posted

| Week      | Lecture                                 | Lab                          |
|-----------|-----------------------------------------|------------------------------|
| Nov 17 WK | Online Lecture MongoDB Update Documents | Visual Code Lab 8 Due online |
| Nov 24 WK | Online Lecture Aggregation              | Visual Code Lab 9 due online |
| Dec 1 WK  | Online Lecture Review for Final         | Assign2 due online           |
| Dec 8 WK  | In Class Final Test                     | Lab 10 due online            |