

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()`

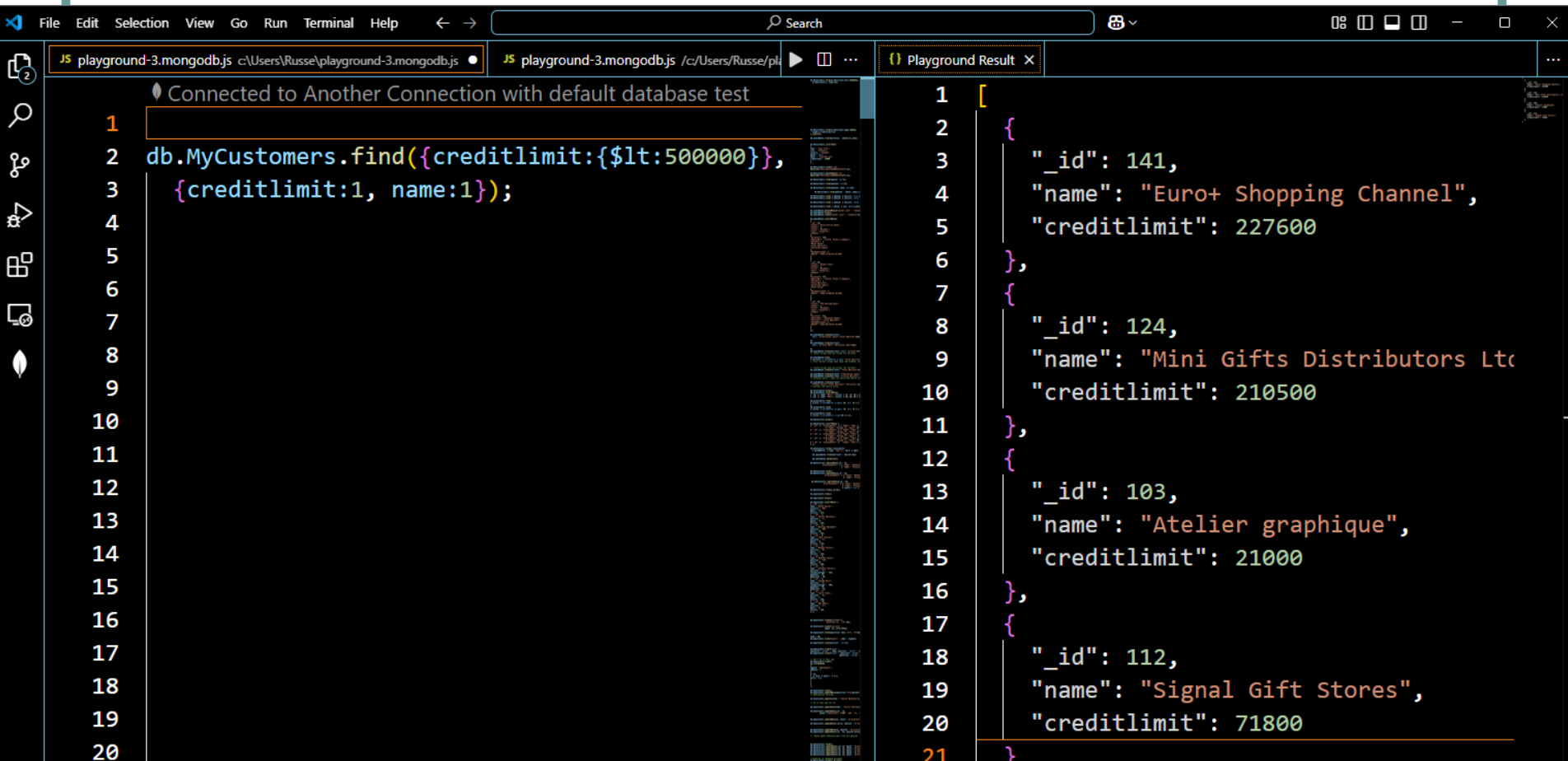
- Read

- `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 the MongoDB Playground interface. The left pane displays the MongoDB query: `db.MyCustomers.find({creditlimit:{$lt:500000}}, {creditlimit:1, name:1});`. The right pane, titled "Playground Result", shows the output of the query as a JSON array of documents. The documents are:

```
[
  {
    "_id": 141,
    "name": "Euro+ Shopping Channel",
    "creditlimit": 227600
  },
  {
    "_id": 124,
    "name": "Mini Gifts Distributors Ltd",
    "creditlimit": 210500
  },
  {
    "_id": 103,
    "name": "Atelier graphique",
    "creditlimit": 21000
  },
  {
    "_id": 112,
    "name": "Signal Gift Stores",
    "creditlimit": 71800
  }
]
```

Cursor Modifier

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

The screenshot shows the VS Code editor interface with a MongoDB Playground script. The script is connected to a database named 'test'. The code in the editor is:

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

The 'Playground Result' panel on the right shows the output of the query, which is a JSON array of two documents:

```
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 status bar at the bottom indicates the file is 'Ln 3, Col 37', using 'Spaces: 2', 'UTF-8' encoding, 'CRLF' line endings, and 'JavaScript' language.

- 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: 5 } },
    { 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}]})`

-

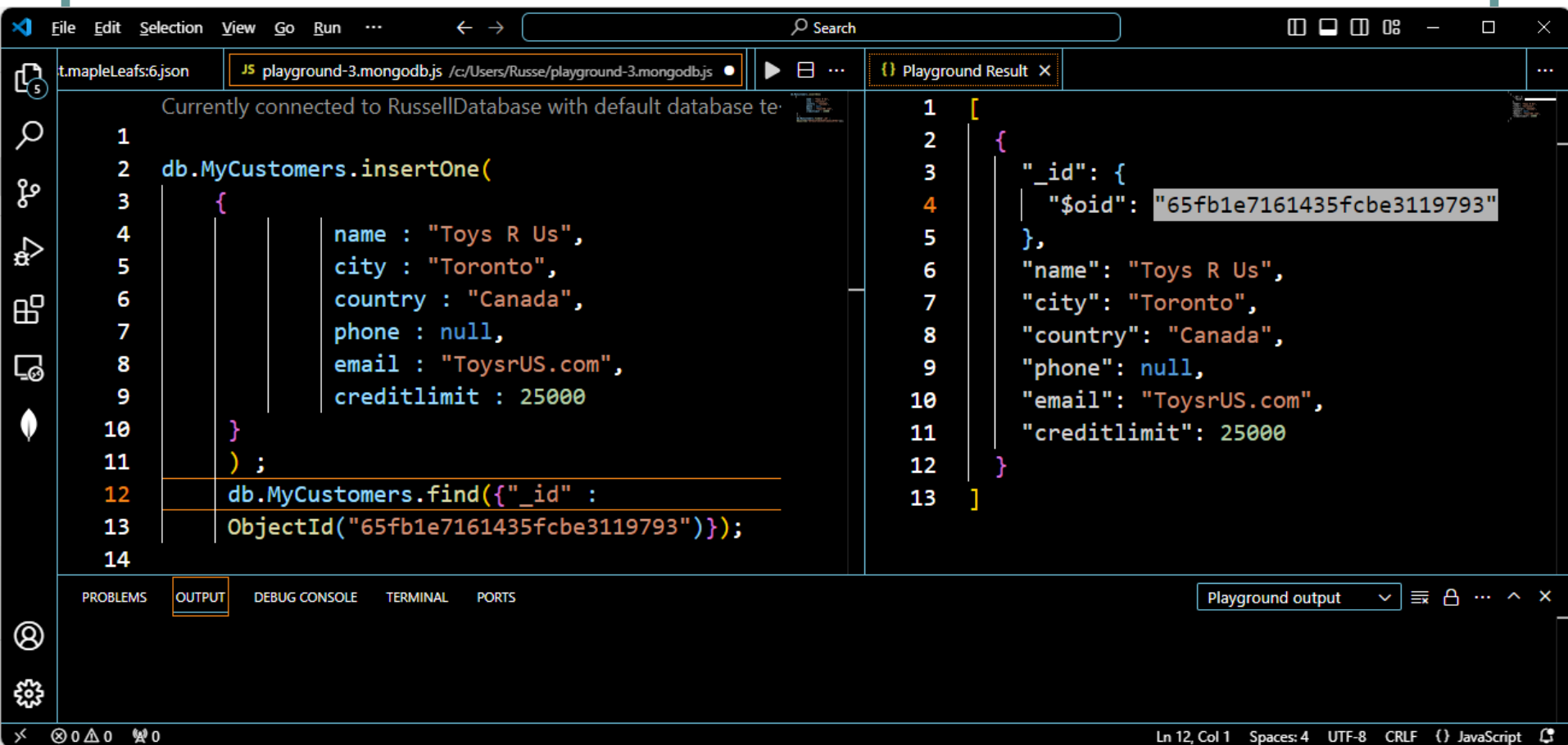
- See the following example of “\$and”:

- `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



The screenshot shows a code editor with a dark theme. The top bar includes a search bar and window controls. The editor has two tabs: 't.mapleLeafs:6.json' and 'JS playground-3.mongodb.js /c/Users/Russe/playground-3.mongodb.js'. The main editor area shows JavaScript code for connecting to a MongoDB database and inserting a document. The right sidebar shows the 'Playground Result' tab with the inserted document's JSON representation. The bottom status bar shows 'Ln 12, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', and 'JavaScript'.

```
File Edit Selection View Go Run ...  
t.mapleLeafs:6.json JS playground-3.mongodb.js /c/Users/Russe/playground-3.mongodb.js  
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  
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 ]  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
Ln 12, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript
```

- You can remove the document using `ObjectId()`

The screenshot shows the Visual Studio Code interface with a MongoDB playground script. The editor has a dark theme. The top bar shows the menu (File, Edit, Selection, View, Go, ...), a search bar, and window controls. The editor has three tabs: 't.mapleLeafs:6.json', 'JS playground-3.mongodb.js /c:/Users/Russe/playground-3.mongodb.js', and 'Playground Result'. The active tab is 'JS playground-3.mongodb.js', which contains a script to delete a document from a MongoDB database. The script is as follows:

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

The 'Playground Result' tab shows the output of the script, which is a JSON object:

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

The bottom status bar shows the 'OUTPUT' tab selected, with a dropdown menu showing 'Playground output'. The status bar also displays 'Ln 4, Col 2', 'Spaces: 2', and the file type 'JSON'.

Searching for Null

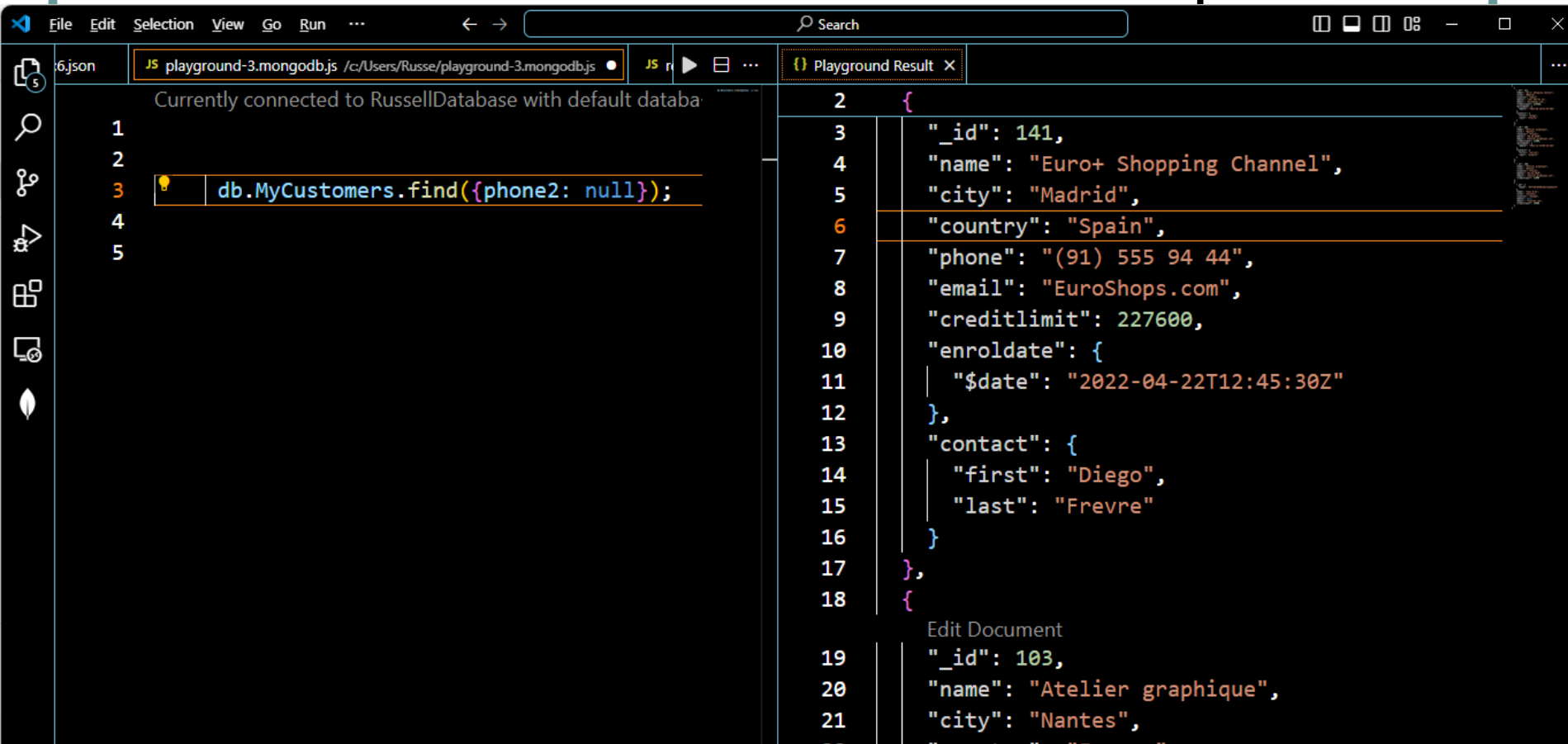
● Null

- 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": "65ff58c05981e81f2adee4c9"`
- `},`
- `"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 a MongoDB Playground interface. The left pane displays the code editor with the query `db.MyCustomers.find({phone2: null});` on line 3. The right pane shows the 'Playground Result' tab with two JSON documents. The first document is a customer with `_id: 141`, `name: "Euro+ Shopping Channel"`, `city: "Madrid"`, `country: "Spain"`, `phone: "(91) 555 94 44"`, `email: "EuroShops.com"`, `creditlimit: 227600`, and `enroldate: {"$date": "2022-04-22T12:45:30Z"}`. The second document is a customer with `_id: 103`, `name: "Atelier graphique"`, and `city: "Nantes"`.

```
File Edit Selection View Go Run ... Search
JS playground-3.mongodbj.js /c:/Users/Russe/playground-3.mongodbj.js JS r
Currently connected to RussellDatabase with default databa
1
2
3 db.MyCustomers.find({phone2: null});
4
5

2 {
3   "_id": 141,
4   "name": "Euro+ Shopping Channel",
5   "city": "Madrid",
6   "country": "Spain",
7   "phone": "(91) 555 94 44",
8   "email": "EuroShops.com",
9   "creditlimit": 227600,
10  "enroldate": {
11    "$date": "2022-04-22T12:45:30Z"
12  },
13  "contact": {
14    "first": "Diego",
15    "last": "Frevre"
16  }
17 },
18 {
19   "_id": 103,
20   "name": "Atelier graphique",
21   "city": "Nantes",
```

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

The screenshot shows a MongoDB Playground interface with a dark theme. The top bar includes a menu (File, Edit, Selection, View, Go, Run, ...), a search bar, and window management icons. Below the bar, there are tabs for 'mapleLeafs:6.json', 'JS playground-3.mongodbs.js /c:/Users/Russe/playground-3.mongodbs.js', 'JS review.mongodbs', and 'Playground Result'. The main editor area on the left contains a JavaScript query: `db.MyCustomers.find({phone2: {$ne :null}});` on line 3, and `db.MyCustomers.find({phone2 : {$not: {$eq:null}}});` on line 5, which is highlighted with a yellow box. A status message at the top of the editor says 'Currently connected to RussellDatabase with default database test. Click'. The right pane, titled 'Playground Result', displays two JSON documents. The first document (lines 2-19) has fields: `_id` (112), `name` ('Signal Gift Stores'), `city` ('Las Vegas'), `country` ('USA'), `phone` ('7025551838'), `email` ('Signal.com'), `creditlimit` (71800), and `enrolldate` (a date object for 2013-11-05T14:10:30Z). The second document (lines 20-26) has fields: `_id` (124), `name` ('Mini Gifts Distributors Ltd'), `city` ('San Rafael'), `country` ('USA'), `phone` ('4155551450'), `phone2` ('4153334444'), and `phone3` ('4166662222'). The bottom status bar shows 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'OUTPUT' tab is active, showing 'Playground output'.

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

```
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  "enrolldate": {  
11    "$date": "2013-11-05T14:10:30Z"  
12  },  
13  "contact": {  
14    "first": "Jean",  
15    "last": "King"  
16  },  
17  "phone2": "7024443333"  
18 },  
19 {  
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",
```


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

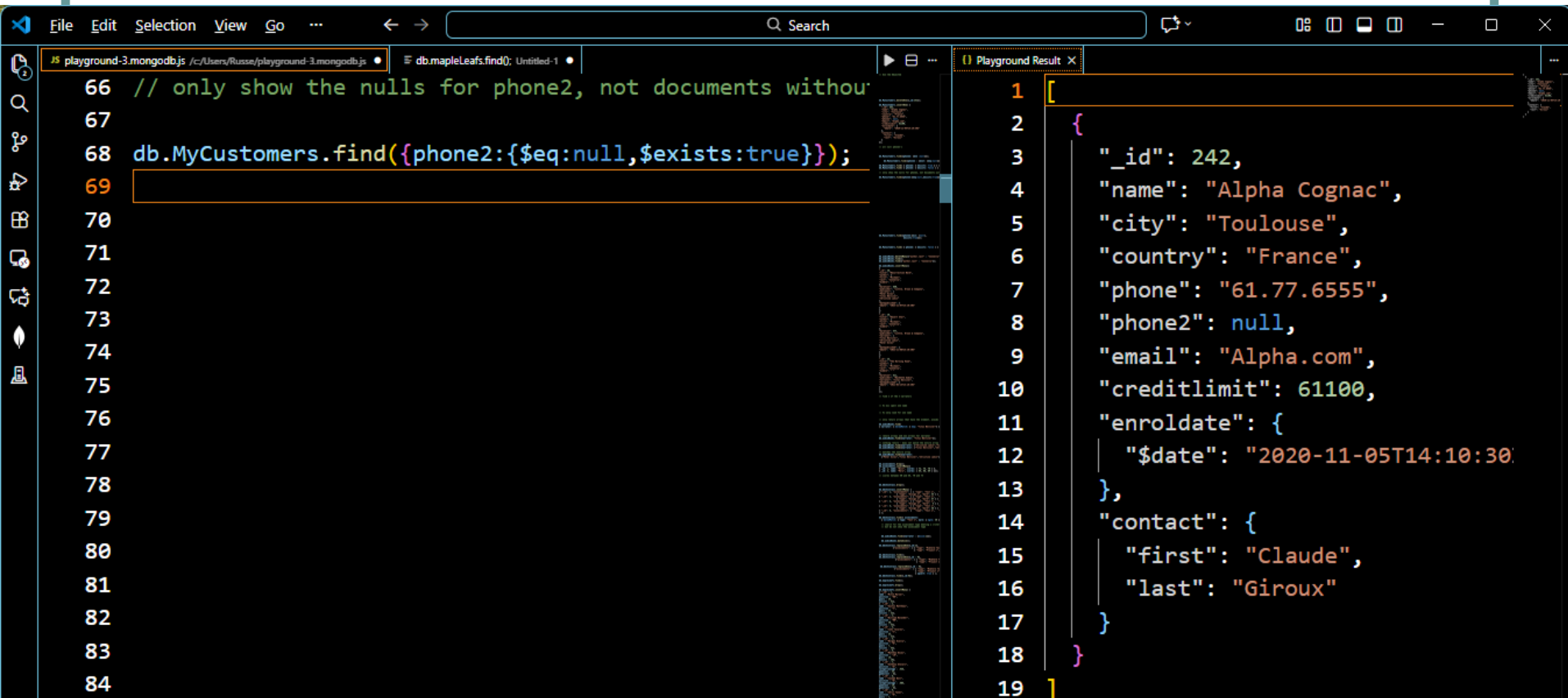
- `db.MyCustomers.find({ phone2: { $exists: true } })`

- and documents that do not

- `db.MyCustomers.find({ phone2: { $exists: false } });`

Omit documents without a phone2

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



The screenshot shows a MongoDB Playground interface. The left pane contains a JavaScript file named `playground-3.mongodb.js` with the following code:

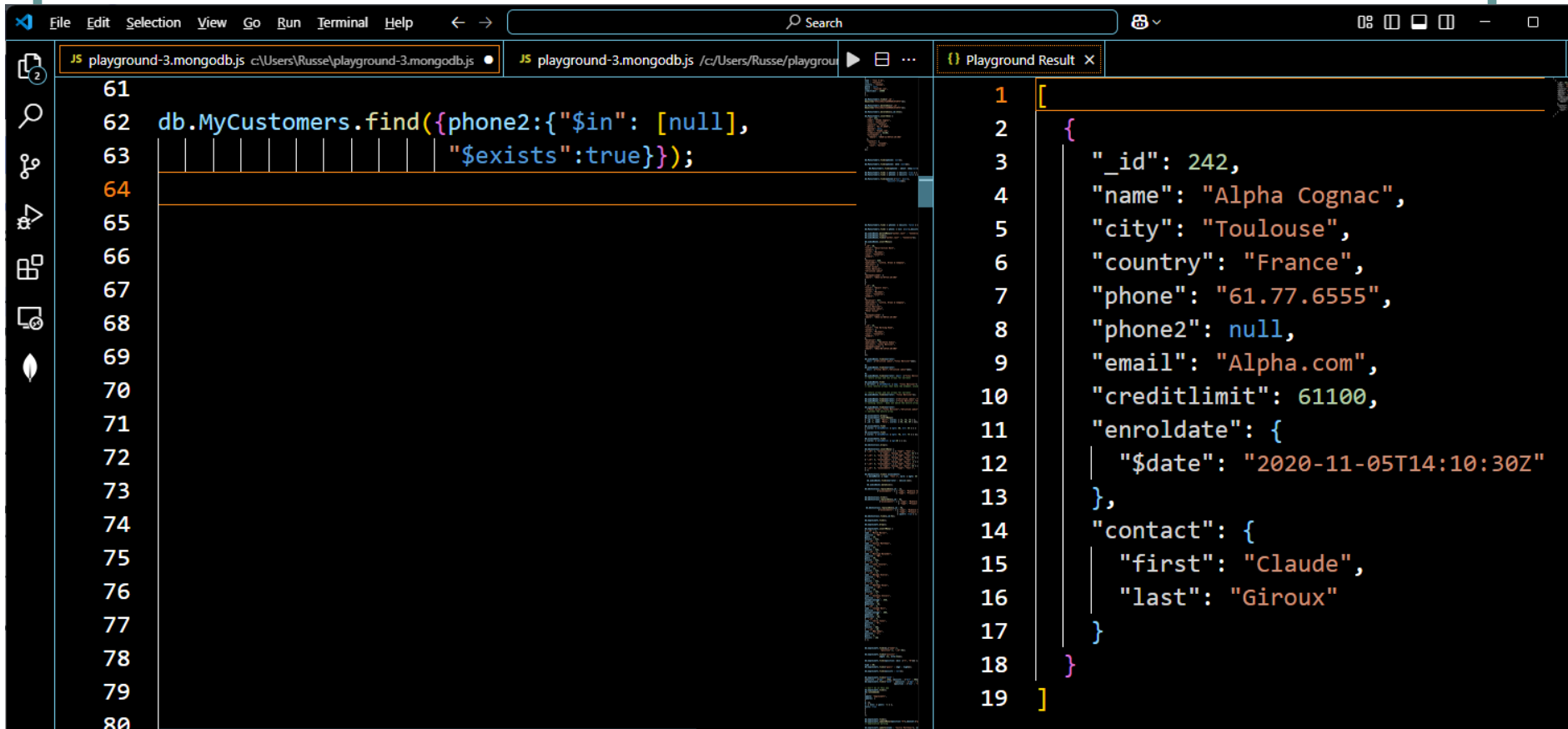
```
66 // only show the nulls for phone2, not documents without
67
68 db.MyCustomers.find({phone2:{$eq:null,$exists:true}});
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
```

The right pane, titled "Playground Result", displays the result of the query as a JSON array with one document:

```
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": "2020-11-05T14:10:30"
13    },
14    "contact": {
15      "first": "Claude",
16      "last": "Giroux"
17    }
18  }
19 ]
```

Omit documents without a phone2

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



The screenshot shows a code editor with a dark theme. The left pane contains a JavaScript query: `db.MyCustomers.find({phone2:{"$in": [null], "$exists":true}});` at line 62. The right pane shows the result of the query, which is a single document: `{ "_id": 242, "name": "Alpha Cognac", "city": "Toulouse", "country": "France", "phone": "61.77.6555", "phone2": null, "email": "Alpha.com", "creditlimit": 61100, "enroldate": { "$date": "2020-11-05T14:10:30Z" }, "contact": { "first": "Claude", "last": "Giroux" } }`. The editor has a menu bar at the top with options like File, Edit, Selection, View, Go, Run, Terminal, and Help. There is also a search bar and a playground result tab.

```
File Edit Selection View Go Run Terminal Help
JS playground-3.mongodb.js c:\Users\Russe\playground-3.mongodb.js
JS playground-3.mongodb.js /c:/Users/Russe/playground
Playground Result X
```

```
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
```

```
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 VS Code editor with a MongoDB playground. The left pane displays the query: `db.audioBooks.find({narrator: {$all :["Titus Welliver" ,"Christine Lakin"]}});`. The right pane shows the resulting JSON document. The bottom status bar indicates the output is from the 'Playground output'.

```
File Edit Selection View Go Run ... Search
```

Currently connected to RussellDatabase with default database test. Click

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

```
2 {
11   "publisher": "Little, Brown & Comp
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 & Comp
31   "narrator": [
32     "Titus Welliver",
33     "Christine Lakin",
34     "Peter Giles"
35   ],
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Playground output

- 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

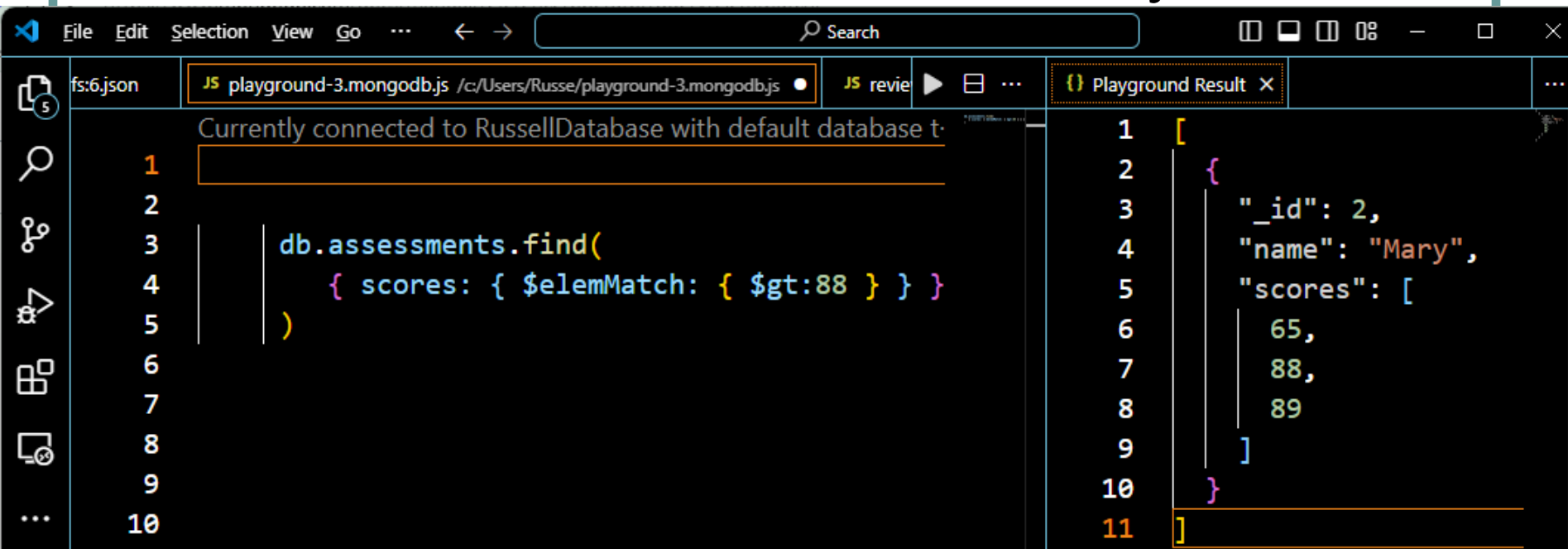
- This technique returns no matches because there are 3 elements in the narrator array and the query only includes 2 elements.
- `db.audioBooks.find({narrator: ["Christine Lakin","Titus Welliver"]});`
- `db.audioBooks.find({narrator: ["Titus Welliver","Christine Lakin"]});`
- This query returns a document 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. The top bar includes a file explorer with 'fs:6.json', a terminal with the command 'JS playground-3.mongodb.js /c:/Users/Russe/playground-3.mongodb.js', and a 'JS review' button. The main editor displays a MongoDB query: `db.assessments.find({ scores: { $elemMatch: { $gt:88 } } })`. The status bar indicates 'Currently connected to RussellDatabase with default database t...'. The right panel, titled 'Playground Result', shows the query result as a JSON array with one object: `[{ "_id": 2, "name": "Mary", "scores": [65, 88, 89] }]`. The line numbers 1 through 11 are visible on the left of the code editor.

```
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

The screenshot shows the Visual Studio Code editor with a MongoDB playground. The left pane displays a JavaScript query: `db.dbs311Class.find({ assessments: { $elemMatch: { type: "Test 1", mark: { $gte: 40 } } } })`. The right pane, titled "Playground Result", shows the JSON output of the query: `{ "_id": 1, "assessments": [{ "type": "Test 1", "mark": 40 }, { "type": "Assign 1", "mark": 45 }], "studentname": "Bill" }`. The bottom status bar indicates the current position is Line 15, Column 4, with 2 spaces, in JSON format.

```
File Edit Selection View Go Run Terminal Help
Currently connected to RussellDatabase with default database test. Click here to change connection.
1
2
3 db.dbs311Class.find(
4   { assessments: { $elemMatch: { type: "Test 1", mark: { $gte: 40 } } } }
5 )
6
7
8
9
10
11
12
13
14
```

```
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  }
16 ]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Playground output

Ln 15, Col 4 Spaces: 2 {} JSON

- // 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 a code editor with a dark theme. The top bar includes a menu (File, Edit, Selection, View, Go, Run, ...), a search bar, and window controls. The editor has two tabs: 'JS db.audioBooks.find({narrator : {\$size:3}} /c:/Users/Russe/playground-3.mongodbs.js' and 'JS review.m...'. The main editor area displays the query `db.audioBooks.find({narrator : {$size:3}});` on line 1. A status bar at the bottom indicates 'Currently connected to RussellDatabase with default database t'. To the right, a 'Playground Result' panel shows a JSON document with the following structure:

```
{
  "publisher": "Little, Brown & Company",
  "narrator": [
    "Peter Giles",
    "Titus Welliver",
    "Christine Lakin"
  ],
  "datepublished": {
    "$date": "2023-11-07T14:10:30Z"
  }
},
{
  "_id": 29,
  "title": "Desert Star",
  "author": {
    "first": "Michael",
    "last": "Connelly",
    "middle": " "
  },
  "duration": 577,
  "publisher": "Little, Brown & Company",
  "narrator": [
    "Titus Welliver"
  ]
}
```


- 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 a VS Code editor window with a JavaScript file named `db.mapleLeafs.find(); /c:/Users/Russe/playground-3.mongodb.js 9+`. The code in the editor is as follows:

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

Below the editor, the **OUTPUT** panel is active, displaying the results of the MongoDB operations:

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

The status bar at the bottom indicates the current position is **Ln 6, Col 38**, with **Spaces: 2**, **UTF-8** encoding, **CRLF** line endings, and the file is a **JavaScript** document.

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