

MongoDB CRUD

CS415 – Relational and Document-Based Databases

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.

Content

- Searching an Array, \$elemMatch
- Update
- Count, sort, limit, skip
- Delete

CRUD in MongoDB

There is no special SQL language to perform CRUD in MongoDB.

Many CRUD operations exist as methods/functions on objects in programming language API, NOT as separate language.

CRUD	MongoDB	SQL
Read	<code>find()</code>	<code>select</code>
Create	<code>insert()</code>	<code>insert</code>
Update	<code>update()</code>	<code>update</code>
Delete	<code>remove()</code>	<code>delete</code>

Searching an Array

```
// { _id: 1, courses: [ "CS472", "CS572", "CS435" ] }  
// find all documents where courses value contains "CS572"  
db.col.find({ courses: "CS572" })
```

```
// find all documents where courses value contains "CS472" or "CS572"  
db.col.find({ courses: { $in: ["CS572", "CS472"] } })
```

```
// find all documents where courses value contains "CS472" and "CS572"  
db.col.find({ courses: { $all: [ "CS572" , "CS472" ] } })
```

The **\$in** operator selects the documents where the value of a field is an array that contains at any order any (at least one) of the specified elements

The **\$all** operator selects the documents where the value of a field is an array that contains at any order all the specified elements

Search with \$elemMatch

```
{ _id: 1, results: [ 1, 2, 5, 10 ] }  
{ _id: 2, results: [ 5, 8, 9, 10 ] }  
{ _id: 3, results: [ 10, 11, 12 ] }
```


```
db.test.find( { results: { $all: [ 5,10 ] } } )  
// scan results for value x times (accepts only value, no operators)  
// scan if it has 5 AND scan again if it has 10 = returns _id: 1  
// All values must exist to return the document
```

```
db.test.find( { results: { $elemMatch: { $gt: 5, $lt: 10 } } } )  
// scan results array once: check if there is one value matches the condition  
// _id: 2
```

```
{field: {operator: value} }
```

Comparison Query Operators

Can be applied on **numeric** and **string** field values

- **\$eq** equal to 

```
{ field: { $eq: value } }
```



```
{ field: value }
```
- **\$gt** greater than
- **\$gte** greater than or equal to
- **\$lt** less than
- **\$lte** less than or equal to
- **\$ne** not equal to
- **\$in** matches any of the values specified in an array (implicit OR)
- **\$nin** matches none of the values specified in an array.
- **Comma** between operators works as (implicit AND)

Update Methods

db.collection.update()

Either updates or replaces a single document by default or updates all documents that match a specified filter.

db.collection.updateOne()

Updates at most a single document that match a specified filter even though multiple documents may match the specified filter.

db.collection.updateMany()

Update all documents that match a specified filter.

For more check the [documentation](#)

Field Update Operators

By default, `update()` will: update a **single document** and **replace** everything but the `_id`

```
// Original document
{ "_id" : "1", "students" : 250, "courses" : ["CS572", "CS472"] }
db.col.update({_id:"1"}, {"students":500})
// Results
{ "_id" : "1", "students" : 500 }
```

```
// To target only one field use { $set: { field1: value1, ... } }
// If the field does not exist, $set will add a new field with the
  specified value
{ "_id" : "1", "students": 250, "courses" : ["CS572", "CS472"] }
db.col.update({_id:"1"}, {$set: {"students": 500, "entry": "Oct"} })
// Results
{ "_id" : "1", "students": 500, "courses" : ["CS572", "CS472"] , "entry":"Oct" }
```

Field Update Operators

```
// Original document
{ "_id" : "1", "students" : 250, "courses" : ["CS572", "CS472"] }
db.col.update({_id:"2"}, {"students":500}, {upsert: true})
// Results
{ "_id" : "1", "students" : 250, "courses" : ["CS572", "CS472"] }
{ "_id" : "2", "students" : 500 }
```

```
// update all docs to have one more field (city: Fairfield)
{ "_id" : "1", "Dept" : "CompPro" }
{ "_id" : "2", "Dept" : "SustainableLiving" }
db.col.update({}, {$set: {"city":"Fairfield"}} , {multi: true})
// Results
{ "_id" : "1", "Dept" : "CompPro", "city":"Fairfield" }
{ "_id" : "2", "Dept" : "SustainableLiving", "city":"Fairfield" }
```

Examples - Field Update Operators

```
{ "_id" : "1", "students" : 250 }  
db.col.update({_id:"1"}, { $inc : { "students":1, "exams":1 } })  
{ "_id" : "1", "students" : 251, "exams":1 }
```

```
{ "_id" : "1", "students" : 250, "dept": "ComPro" }  
db.col.update({_id:"1"}, { $unset : { "dept":1 } })  
{ "_id" : "1", "students" : 250 }
```

Field Update Operators

Update Operator	Description	Notes
\$set	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
\$inc	Increments a field by a specified value, it accepts positive and negative values	If the field does not exist, \$inc creates the field and sets the field to the specified value
\$unset	Deletes a particular field, The specified value in the \$unset expression does not impact the operation.	If the field does not exist, then \$unset does nothing

Array Update Operators

```
//Original Document { "_id" : 1, "a" : [1, 2, 3, 4] }
db.testCol.update({_id:1}, { $set : { "a.2":5 } })
// output: { "_id" : 1, "a" : [1, 2, 5, 4] }
db.col.update({_id:1}, { $push : { "a": 6 } })
// output: { "_id" : 1, "a" : [1, 2, 5, 4, 6] }
db.col.update({_id:1}, { $pop : { "a": 1 } })
// output: { "_id" : 1, "a" : [1, 2, 5, 4] }
db.col.update({_id:1}, { $pop : { "a": -1 } })
// output: { "_id" : 1, "a" : [2, 5, 4] }
db.col.update({_id:1}, { $pushAll : { "a": [7, 8, 9] } })
// output: { "_id" : 1, "a" : [2, 5, 4, 7, 8, 9] }
db.col.update({_id:1}, { $pull : { "a": 5 } })
// output: { "_id" : 1, "a" : [2, 4, 7, 8, 9] }
db.col.update({_id:1}, { $pullAll : { "a": [2, 4, 8] } })
// output: { "_id" : 1, "a" : [7, 9] }
db.col.update({_id:1}, { $addToSet : { "a": 5 } })
// output: { "_id" : 1, "a" : [7, 9, 5] }
db.col.update({_id:1}, { $addToSet : { "a": 5 } })
// output: { "_id" : 1, "a" : [7, 9, 5] }
```

\$ (The Array POSITION that matched)

The positional **\$** operator identifies an **element** in an array to **update** without explicitly specifying the position of the element in the array.

```
{ "_id" : 1, "grades" : [ 85, 80, 84 ] }  
db.students.updateOne( { _id: 1, grades: 80 }, { $set: { "grades.$" : 82 } } )  
// { "_id" : 1, "grades" : [ 85, 82, 84 ] }
```

```
{ _id: 4, grades: [ { total: 80, mean: 75, std: 8 },  
                    { total: 85, mean: 90, std: 5 },  
                    { total: 86, mean: 85, std: 8 } ] }  
db.students.updateOne( { _id: 4, "grades.total": 85 },  
                      { $set: { "grades.$.std" : 6 } } )  
// { "total" : 85, "mean" : 90, "std" : 6 }
```

You must include the array field as part of the query.

count()

We can use `count()` method exactly like `find()` to get the count of all the documents that match a certain criteria.

```
// returns number of all documents in the collection  
db.collection.count()
```

```
// returns number of students who received A  
db.collection.count({ "grade": "A" })
```

sort() limit() skip()

Similar to SQL language, MongoDB provides certain methods on the collection object, they work as instructions sent to DB to affect the retrieval of data, all these methods will return a cursor back (chain):

SQL	MongoDB Method
Order by	sort()
Limit	limit()
Skip	skip()

Note: These will set instructions to DB server to process the information before its being sent to client.
No processing will ever happen at the client side.

Logical Query Operators

Joins query clauses with a logical operation:

- `$or` returns all documents that match the conditions of **either** clause.
- `$and` returns all documents that match the conditions of **both** clauses.
- `$not` returns documents that **do not match** the query expression.
- `$nor` returns all documents that **fail to match both** clauses.

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

```
// return all documents where either the quantity field value is  
// less than 20 or the price field value equals 10:  
db.col.find( { $or: [ { quantity: { $lt: 20 } }, { price: 10 } ] } )
```

Delete documents `db.collection.remove()`

```
// delete all documents - One by One
```

```
db.col.remove({})
```

```
// delete all students whose names start with N-Z
```

```
db.col.remove({"student": {$gt: "M"}})
```

```
// drop the collection - Faster than remove()
```

```
db.col.drop()
```

Notes

- When we want to delete large number of documents, it's faster to use `drop()` but we will need to create the collection again and create all indexes as `drop()` will take the indexes away (while `remove()` will keep them)
- Multi-docs remove are not atomic isolated transactions to other R/Ws and it will yield in between.
- Each single document is atomic, no other R/Ws will see a half removed document.