

# **MongoDB - Rich Documents**

**CS477 – Server-side Programming**

**Maharishi International University**

**Department of Computer Science**

**M.S. Thao Huy Vu**

# 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 University of Management.

# Main Points

- MongoDB
- Insert
- Delete
- Search
- Update

Rest Application

CRUD

POST  
PUT  
Get All  
Get one  
Delete

client

frontend mob app  
web  
postman  
thunder

signup  
login

server  
nodejs

token

crud+token

response

mongodb

driver

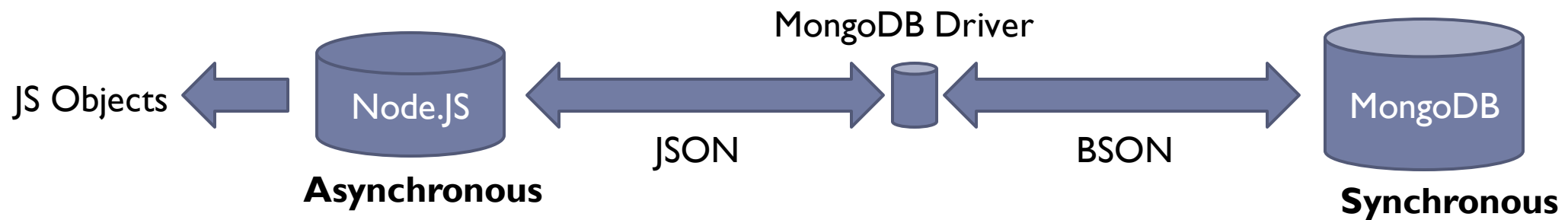


# MongoDB Driver

- ▶ A library written in JS to handle the communication, open sockets, handle errors and talk with MongoDB Server.

```
npm install mongodb
```

- ▶ Note that Mongo Shell is **Synchronous** while Node.js is **Asynchronous**.



# 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
Read	<code>find()</code>
Create	<code>insert()</code>
Update	<code>update()</code>
Delete	<code>delete()</code>

# Query Operators

Update Operator	Description
<b>\$in</b>	Matches any of the values specified in an array
<b>\$all</b>	Matches arrays that contain all elements specified in the query
<b>\$gt</b>	Matches values that are greater than a specified value
<b>\$lt</b>	Matches values that are lesser than a specified value
<b>\$elementMatch</b>	Projects the first element in an array that matches the specified condition.

# Connect to MongoDB using mongodb

---

```
const {MongoClient} = require('mongodb');
```

Let uri = mongodb+srv://<user>:<password>@cluster0.l9zyoim.mongodb.net

```
const client = new MongoClient(uri);
```

```
async function main(){  
  await client.connect();  
  const db = client.db('test');  
}
```

```
main().then(() => console.log('DB Connected'))  
  .catch(error => console.log(error));
```

# Insert

---

```
async function insert(student) {  
  try {  
    const s = await collection.insertOne(student);  
    return s;  
  } catch (error) {  
    console.log(error)  
    return null;  
  }  
}
```



# Delete

---

```
async function deleteOne(name){  
  try {  
    const s = await collection.deleteOne({name});  
    return s;  
  } catch (error) {  
    console.log(error)  
    return null;  
  }  
}
```

# Searching an Array

```
// { _id: 1, courses: [ "CS471", "CS571", "CS435" ] }  
// find all documents where courses value contains "CS571"  
db.col.find({ courses: "CS571" })
```

```
// find all documents where courses value contains "CS471" or "CS571"  
db.col.find({ courses: { $in: ["CS571", "CS471"] } })
```

```
// find all documents where courses value contains "CS471" and "CS571"  
db.col.find({ courses: { $all: [ "CS571" , "CS471" ] } })
```

# 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: { $in: [ 5, 10 ] } } )  
// scan results for value n times (accepts only value, no operators)  
// one value 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
```

# \$elemMatch

The **\$elemMatch** operator matches documents that contain an array field with at least one **element** that matches **ALL** the specified query criteria.

```
{ _id: 1, results: [ { product: "abc", score: 10 }, { product: "xyz", score: 5 } ] }  
{ _id: 2, results: [ { product: "abc", score: 8 }, { product: "xyz", score: 7 } ] }  
{ _id: 3, results: [ { product: "abc", score: 7 }, { product: "xyz", score: 8 } ] }
```

```
db.survey.find( { results: { $elemMatch: { product: "xyz", score: { $gte: 8 } } } } ) // _id: 3
```

# Searching an Object

```
{_id: 1, email: { work: "work@miu.edu", personal: "personal@gmail.com"} }
```

```
// nothing will be returned
```

```
db.col.find({ email: { work: "work@miu.edu" } })
```

```
db.col.find({ email: { personal: "personal@gmail.com" } })
```

```
db.col.find({ email: { personal: "personal@gmail.com", work: "work@miu.edu" } })
```

```
// will work
```

```
db.col.find({ email: { work: "work@miu.edu", personal: "personal@gmail.com" } })
```

```
// how to search for one key only?
```

```
db.col.find({ "email.work": "work@miu.edu" })
```

```
db.col.find({ "email.personal": "personal@gmail.com" })
```

# Searching projection

```
{ _id: 1, name: 'Michael', entry: 'Oct2021' }  
{ _id: 2, name: 'John', entry: 'Oct2021' }
```

```
db.test.find( {}, {name: 1, _id: 0} )  
// return [{name: 'Michael'}, {name: 'John'}]
```

# Update Methods

## **`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.

# Field Update Operators

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

```
// To target only one field use { $set: { field1: value1, ... } }  
// If the field does not exist, $set will add a new field with the  
// specified value  
{ "_id" : "1", "students": 250, "courses" : ["CS571", "CS472"] }  
db.col.updateOne({_id:"1"}, {$set: {"students": 500, "entry": "Aug"} })  
// Results  
{ "_id" : "1", "students": 500, "courses" : ["CS571", "CS472"] , "entry":"Aug" }
```



# Field Update Operators

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

# Field Update Operators

Update Operator	Description	Notes
<b>\$set</b>	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
<b>\$unset</b>	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
<b>\$inc</b>	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

# Examples - Field Update Operators

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

```
{ "_id" : "1", "students" : 250, "program": "MSD" }  
db.col.updateOne({_id:"1"}, { $unset : { "program":1 } })  
{ "_id" : "1", "students" : 250 }
```

# Array Update Operators

```
//Original Document { "_id" : 1, "a" : [1, 2, 3, 4] }
db.testCol.updateOne({_id:1}, { $set : { "a.2":5 } }) // Update item in Array by
INDEX
// output: { "_id" : 1, "a" : [1, 2, 5, 4] }
db.col.updateOne({_id:1}, { $push : { "a": 6 } }) // Add item to Array
// output: { "_id" : 1, "a" : [1, 2, 5, 4, 6] }
db.col.updateOne({_id:1}, { $pop : { "a": 1 } })
// output: { "_id" : 1, "a" : [1, 2, 5, 4] }
db.col.updateOne({_id:1}, { $pop : { "a": -1 } })
// output: { "_id" : 1, "a" : [2, 5, 4] }
db.col.updateOne({_id:1}, { $pull : { "a": 5 } }) // Remove items from Array
// output: { "_id" : 1, "a" : [2, 4] }
db.col.updateOne({_id:1}, { $addToSet : { "a": 5 } })
// output: { "_id" : 1, "a" : [2, 4, 5] }
db.col.updateOne({_id:1}, { $addToSet : { "a": 5 } })
// output: { "_id" : 1, "a" : [2, 4, 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, 80 ] }  
db.students.updateOne( { _id: 1, grades: 80 }, { $set: { "grades.$" : 82 } } )  
// { "_id" : 1, "grades" : [ 85, 82, 84, 80 ] }
```

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

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

# \$elemMatch vs. arrayFilters

```
{ _id: 5, grades: [ { total: 80, mean: 75, student: 8 },  
                    { total: 85, mean: 90, student: 5 },  
                    { total: 90, mean: 85, student: 3 } ] }
```

```
db.students.updateOne(  
  { _id: 5, grades: { $elemMatch: { total: { $lte: 90 }, mean: { $gt: 80 } } } },  
  { $set: { "grades.$.student" : 6 } }  
)
```

```
{ total: 85, mean: 90, student: 6 }  
Only updates one element that matches our condition
```

```
db.students.updateOne(  
  { _id: 5 },  
  { $set: { "grades.$[obj].student" : 6 } } ,  
  { arrayFilters: [{ "obj.total": { $lte: 90 } , "obj.mean": { $gt: 80 } }] }  
)
```

```
{ total: 85, mean: 90, student: 6 }, { total: 90, mean: 85, student: 6 }  
Always updates all elements in the array that match our condition
```

# Using arrayFilters

```
{ "_id" : 1,  
  "grades" : [  
    { type: "quiz", questions: [ 10, 8, 5 ] },  
    { type: "quiz", questions: [ 8, 9, 6 ] },  
    { type: "hw", questions: [ 5, 4, 3 ] },  
    { type: "exam", questions: [ 25, 10, 23, 0 ] },  
  ] }
```

```
db.students.updateMany(  
  {},  
  { $inc: { "grades.$[t].questions.$[score]": 2 } },  
  { arrayFilters: [ { "t.type": "quiz" } , { "score": { $gte: 8 } } ]  
}
```

```
{ "type" : "quiz", "questions" : [ 12, 10, 5 ] },  
{ "type" : "quiz", "questions" : [ 10, 11, 6 ] },
```

# \$elemMatch vs. arrayFilters

```
{grades:[10,20,20]}  
{grades:[10,20,20]}
```

```
db.test.updateMany({grades: { $elemMatch: { $gt: 10 , $lt: 30 } }},  
  {$set:{'grades.$': 15}})
```

```
{ "grades" : [ 10, 15, 20 ] }  
{ "grades" : [ 10, 15, 20 ] }  
db.test.updateMany({},  
  {$set:{'grades.$[c]': 30}},  
  { arrayFilters: [ { "c": { $gt: 10 , $lt: 30 } } ] })
```

```
{ "grades" : [ 10, 30, 30 ] }  
{ "grades" : [ 10, 30, 30 ] }
```



# Reference

- <https://www.mongodb.com/docs>
- Operators: <https://www.mongodb.com/docs/v4.4/reference/operator>  
[r](#)