

mongoDB

Bộ môn: Kỹ Thuật Phần Mềm

Giáo viên: **Trần Thế Trung.**

Email: tranthetrong@iuh.edu.vn



3- MongoDB

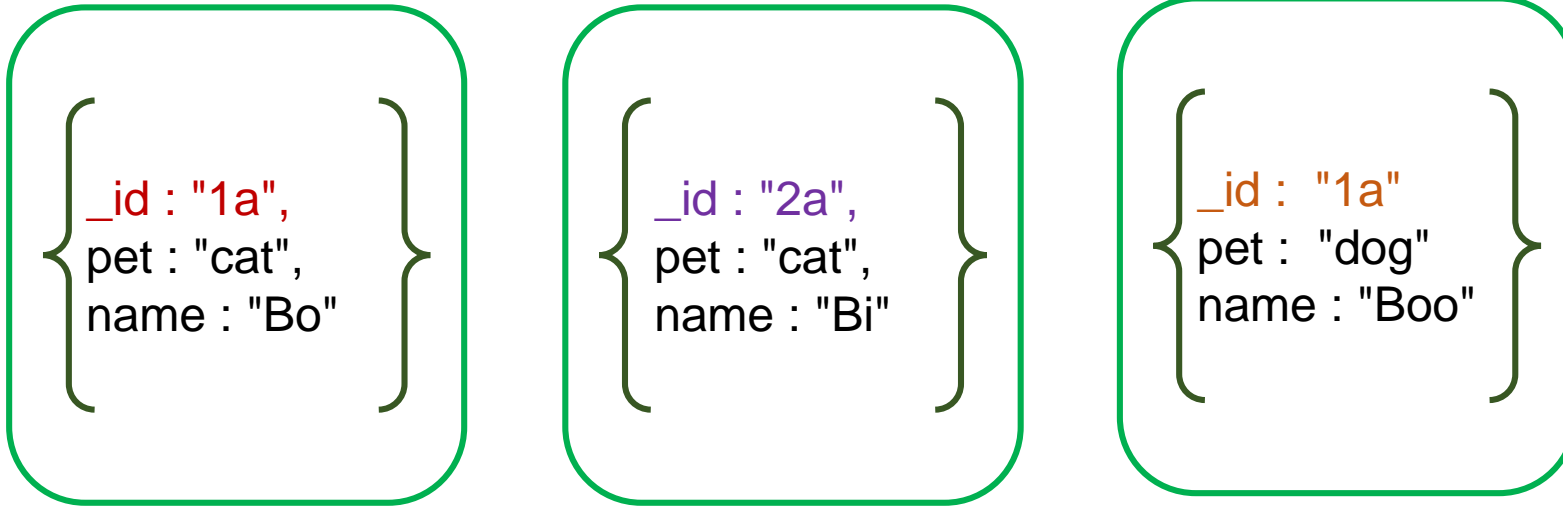
Creating and Manipulating Documents

1. **Insert** document.
2. **Update** document.
3. **Delete** document.



`_id`, `ObjectId()`

- Every document must have a **unique** `_id` value.



- **`ObjectId()`**: default value for `_id` field **unless otherwise specified**.

❖ Returns a new `ObjectId` value. The **12-byte** `ObjectId` value consists of:

- ✓ a **4-byte** timestamp value, representing the `ObjectId`'s creation, measured in seconds;
- ✓ a **5-byte** random value;
- ✓ a **3-byte** incrementing counter, initialized to a random value;

`_id: ObjectId("61d50295f6a4fa7b8149b5a0")`,

1- Insert Documents

Method	Description
<code>db.collection.insert()</code>	Inserts a single document or multiple documents into a collection. (Read more)
<code>db.collection.insertOne()</code>	Inserts a single document into a collection. (Read more)
<code>db.collection.insertMany()</code>	Inserts multiple documents into a collection. (Read more)

Insert Method

```
db.collection.insert(  
    <documents>,  
    {  
        writeConcern: <document>,  
        ordered: <boolean>  
    }  
)
```

//only UpdateMany command supported

Parameter	Type	Description
<documents>	document	<i>document or array of documents</i> to insert into the collection.
writeConcern	document	<u>Optional</u> . A document expressing the write concern. Omit to use the default write concern.
ordered	boolean	<u>Optional</u> . A boolean specifying whether the mongod instance should perform an ordered or unordered insert. Defaults to true.

Example Insert Document

- Insert a Document without specifying an `_id` field:

```
db.products.insert ( { item: "card", qty: 15 } )
```

mongod creates and adds the `_id` field and assigns it a unique `ObjectId()` value.

- Insert a document specifying an `_id` field:

```
db.products.insertOne( { _id: 10, item: "box", qty: 25 } )
```

- Insert a document specific write concern:

```
db.products.insertOne({ item: "envelopes", qty: 100, type: "self-sealing" }, { writeConcern: { wtimeout : 100} })
```

if the acknowledgement takes longer than the `wtimeout` limit, an exception is thrown. [Read more](#)

- Insert several document without specifying an `_id` field:

```
db.products.insertMany( [{ item: "envelope", qty: 100 }, { item: "stamps", qty: 200 } ])
```

- Insert several document specifying an `_id` field:

```
db.products.insertMany( [{ _id: 100, item: "large box", qty: 20 }, { _id: 101, item: "small box", qty: 30 }, { _id: 102, item: "medium box", qty: 30 } ])
```

Example Insert Document

- Unordered Inserts:

```
db.products.insert([ { _id: 105, item: "tape", qty: 20 }, { _id: 106, item: "bubble wrap", qty: 30 } ], { ordered: false } )
```

- Try to execute the commands below, then make your conclusion :

```
db.products.insert([ { id : 905, item: "tape" }, { id : 905, item: "glue" } ] )
```

```
db.products.insert([ { _id: 905, item: "tape", qty: 20 }, { _id: 905, item: "bubble wrap", qty: 30 }, { _id: 906, item: "bubble wrap", qty: 30 } ] )
```

```
db.products.insert([ { _id: 805, item: "tape", qty: 20 }, { _id: 806, item: "bubble wrap", qty: 30 }, { _id: 807, item: " medium box", qty: 30 } ], { ordered: false } )
```

```
db.products.insert([ { _id: 808, item: "tape", qty: 20 }, { _id: 809, item: "bubble wrap", qty: 30 } ] )
```

2- Update Documents

Method	Description
<code>db.collection.update()</code>	Modifies an existing document or documents in a collection. (Read more) <i>By default, the <code>update()</code> method updates a single document. Include the option <code>{multi : true}</code> to update all documents that match the query criteria. Hence we can use it as both ways.</i>
<code>db.collection.updateOne()</code>	Updates a single document within the collection based on the filter. (Read more)
<code>db.collection.updateMany()</code>	Updates all documents that match the specified filter for a collection. (Read more)

```
db.collection.update(  
  <query>,  
  <update>,  
  {  
    upsert: <boolean>,  
    arrayFilters: [ <filterdocument1>, ... ],  
    writeConcern: <document>,  
    ...  
  }  
)
```


Update Method

Parameter	Type	Description
<code>{query}</code>	document	The selection criteria for the update. The same query selectors as in the <code>find()</code> method are available.
<code>{update}</code>	document or pipeline	The modifications to apply. Can be one of the following:
		Update document Contains only update operator expressions.
		Aggregation pipeline Contains only the following aggregation stages: \$addField and its alias \$set \$project and its alias \$unset \$replaceRoot and its alias \$replaceWith.
<code>upsert</code>	boolean	<u>Optional</u> . When true, <code>updateOne()</code> either: <ul style="list-style-type: none">• Creates a new document if no documents match the filter.• Updates a single document that matches the filter.
<code>arrayFilters</code>	array	<u>Optional</u> . An array of filter documents that determine which array elements to modify for an update operation on an array field.

Update Method

Name	Description	Syntax and Example
<code>\$set</code>	Sets the value of a field in a document	<code>{ \$set: { <field1>: <value1>, ... } }</code>

- Set Top-Level Fields

```
db.products.update(  
  { _id: 100 },  
  { $set: {  
    qty: 500,  
    details: { model: "14Q3", make: "xyz" },  
    tags: [ "coats", "outerwear", "clothing" ]  
  }  
})
```

- Set fields in embedded documents:

```
db.products.update( { _id: 100 }, { $set: { "details.make": "zzz" } } )
```

- Set elements in arrays:

```
db.products.update( { _id: 100 }, { $set: { "tags.1": "rain gear", "ratings.0.rating": 2 } } )
```

Example Update Method

```
db.inventory.update (  
  { "item.name" : "ab"},  
  { $set: { qty : 111 } }  
)
```

```
db.inventory.update (  
  { "item.name": "Apple" },  
  { $set: { "size.uom": "cm", status: "P" },  
    $currentDate: { lastModified: true } }  
)
```

```
db.restaurants.update (  
  { name : "Pizza Rat's Pizzeria" },  
  { $set: { _id : 4, violations : 7, borough : "Manhattan" } },  
  { upsert: true }  
)
```

- The `$currentDate` operator sets the value of a field to the current date, either as a Date or a timestamp. The default type is Date. If lastModified field does not exist, \$currentDate will create the field.
- Since `upsert: true` the document is inserted based on the filter (if not founded) and update criteria. The operation returns: "upsertedCount: 1"

update Operator

The collection students with the following documents:

```
{ _id : 1, grades : [ 95, 92, 90 ] },  
{ _id : 2, grades : [ 98, 100, 102 ] },  
{ _id : 3, grades : [ 95, 110, 100 ] }
```

```
db.students.update(  
  { grades: { $gte: 100 } },  
  { $set: { "grades.$[dieukien]" : 100 } },  
  { arrayFilters: [  
    { dieukien: { $gte: 100 } }  
  ]  
}  
)
```

- Modify all elements that are greater than or equal to 100 in the grades array, use the filtered positional operator `$[<identifier>]` with the `arrayFilters` option.
- `$[<identifier>]`: identifies the array elements that match the `arrayFilters` conditions for an update operation.
- The positional `$[<identifier>]` operator acts as a placeholder for all elements in the array field that match the conditions specified in `arrayFilters`.
- The `<identifier>` must begin with a lowercase letter and contain only alphanumeric characters.

update Operator

Name	Description, syntax and example
\$inc	<p><code>{ \$inc: { <field1>: <amount1>, ... } }</code></p> <p>Increments the value of the field by the specified amount. If the field does not exist, \$inc creates the field and sets the field to the specified value.</p> <p>The \$inc operator accepts positive and negative values.</p> <pre>db.products.update({ sku: "abc123" }, { \$inc: { qty: -2, "metrics.orders": 1 } })</pre>
\$mul	<p><code>{ \$mul: { <field1>: <number1>, ... } }</code></p> <p>Multiplies the value of the field by the specified amount If the field does not exist in a document, \$mul creates the field and sets the value to zero of the same numeric type as the multiplier.</p> <pre>db.products.update({ sku: "abc123" }, { \$mul: { qty: 2 } })</pre> <p>Apply \$mul Operator to a Non-existing Field:</p> <pre>db.products.update({ _id: 104 }, { \$mul: { unit_price: 100 } }, { upsert: true })</pre>

Update Operator

Name	Description, syntax and example
\$unset	<p>{ \$unset: { <field1>: "", ... } }</p> <p>Removes the specified field from a document.</p> <pre>db.products.update({ _id: 104 }, { \$unset: { unit_price: "" } })</pre>
\$min	<p>Only updates the field if the specified value is less than the existing field value. If the field does not exist, the operator sets the field to the specified value.</p> <p>{ \$min: { <field1>: <value1>, ... } }</p> <p>Consider the following document in the collection scores: { _id: 1, highScore: 800, lowScore: 200 }</p> <pre>db.scores.update({ _id: 1 }, { \$min: { lowScore: 150 } })</pre> <p>The scores collection now contains the following modified document: { _id: 1, highScore: 800, lowScore: 150 }.</p> <p>Add "unit price" field (does not exist before) and set to specified value</p> <pre>db.products.update({ _id: 103 }, { \$min: { unit_price: 150 } })</pre>
\$max	<p>{ \$max: { <field1>: <value1>, ... } }</p> <p>Consider the following document in the collection scores: { _id: 1, highScore: 800, lowScore: 200 }</p> <pre>db.scores.update({ _id: 1 }, { \$max: { highScore: 950 } })</pre> <p>The scores collection now contains the following modified document: { _id: 1, highScore: 950, lowScore: 200 }</p>

Update Operator (for Array)

Name	Description	Syntax and Example
\$	The positional operator. Acts as a placeholder to update the first element that matches the query condition	<code>{ "<array>.\$" : value }.</code>

- Consider the following document in the collection:

```
{ _id : 4, grades : [ 88, 90, 92 ] },  
{ _id : 5, grades : [ 88, 90, 92 ] },  
{ _id : 6, grades : [ 85, 100, 90 ] }
```

To update the first element whose value is 80 to 82 in the in the grades array, use the positional \$ operator if you do not know the position of the element in the array:

```
db.students.update ( { _id: 1, grades: 88 }, { $set: { "grades.$" : 82 } } )
```

- Consider the following document in the students collection whose grades element value is an array of embedded documents:

```
{ _id: 5, subjects: [ { name: "MongoDB", grades: 90 }, { name: " MongoDB", grades: 76 }, { name: "Java", grades: 84 } ] }
```

Use the positional \$ operator to update the grades field of the first array element that matches the name equal to “MongoDB” condition:

```
db.students.update (  
  { _id: 7, "subjects.name": "MongoDB",  
    { $set: { "subjects.$.grades" : 79 } }  
)
```

Update Operator (for Array)

Update embedded documents using multiple field matches

```
{ _id: 8, grades: [
  { grade: 80, mean: 75, std: 8 },
  { grade: 85, mean: 90, std: 5 },
  { grade: 94, mean: 85, std: 8 } ]
}
```

The \$ operator can update the first array element that matches multiple query criteria specified with the \$elemMatch operator:

```
db.students.update(
  { _id: 9,
    grades: { $elemMatch: { grade: { $lte: 90 }, mean: { $gt: 80 } } } },
  { $set: { "grades.$.std" : 6 } }
)
```


Update Operator (for Array)

Name	Description	Syntax and Example
<code>\$[]</code>	The all positional operator. Acts as a placeholder to update all elements in an array for the documents that match the query condition (updateMany, update)	<code>{ <update operator>: { "<array>.\$[]" : value } }</code>

- Consider the following document in the collection:

```
{ _id : 10, grades : [ 88, 90, 92 ] },  
{ _id : 11, grades : [ 88, 90, 92 ] },  
{ _id : 12, grades : [ 85, 100, 90 ] }
```

To increment all elements in the grades array by 10 for all documents in the collection, use the all positional `$[]` operator:

```
db.students.updateMany(  
  {},  
  { $inc: { "grades.$[]" : 10 } }  
)
```

To increment all elements in the grades array by 10 for all documents except those with the value 100 in the grades array

```
db.students.update(  
  { grades : { $ne: 100 } },  
  { $inc: { "grades.$[]" : 10 },  
    { multi: true } }  
)
```

Update Operator (for Array)

- Consider the following document in the students2 collection whose grades element value is an array of embedded documents:

```
{ _id: 13, grades: [ { grade: 80, mean: 75, std: 8 },  
                    { grade: 85, mean: 90, std: 5 },  
                    { grade: 85, mean: 85, std: 8 } ]  
}
```

To update all documents in an array:

```
db.students.updateMany( { _id: 13 }, { $inc: { "grades.$.std" : -2 } })
```

- Consider the following document in the students3 collection:

```
{ _id: 14, 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 ] } ]  
}
```

To update all values that are greater than or equal to 8 in the nested grades.questions array:

```
db.students.updateMany(  
  { _id: 14 },  
  { $inc: { "grades.$.questions.$score": 2 } },  
  { arrayFilters: [ { score: { $gte: 8 } } ] }  
)
```

Update Operator (for Array)

Name	Description	Syntax and Example
<code>\$addToSet</code>	<ul style="list-style-type: none">- Adds elements to an array only <u>if they do not already exist</u> in the set.- <i>if you use \$addToSet on a field that is absent in the document to update, \$addToSet creates the array field with the specified value as its element.</i>	<code>{ \$addToSet: { <field1>: <value1>, ... } }</code>

- Consider a collection inventory with the following document:

```
{ _id: 1, item: "polarizing_filter", tags: [ "electronics", "camera" ] }
```

The following operation adds the element "accessories" to the tags array since "accessories" does not exist in the array:

```
db.inventory.updateOne(  
  { _id: 1 },  
  { $addToSet: { tags: "accessories" } } )
```

- Consider a collection inventory with the following document:

```
{ _id: 2, item: "polarizing_filter", tags: [ "camera" ] }
```

use the \$addToSet operator with the \$each modifier to add multiple values to the array field :

```
db.inventory.updateOne(  
  { _id: 2 },  
  { $addToSet: { tags: { $each: [ "camera", "electronics", "accessories" ] } } } )
```

Update Operator (for Array)

Name	Description	Syntax and Example
\$pop	- Removes the first or last item of an array	<pre>{ \$pop: { <field>: <-1 1>, ... } }</pre> <p>To specify a <field> in an embedded document or in an array, use dot notation.</p> <ul style="list-style-type: none">- Given the following document in a collection students:: <pre>{ _id: 1, scores: [8, 9, 10] }</pre>- To remove the first element (8) in the scores array: <pre>db.students.update({_id: 1},{ \$pop:{ scores: -1}})</pre>- To remove the first element (10) in the scores array: <pre>db.students.update({ _id: 1 }, { \$pop: { scores: 1 } })</pre>

Update Operator (for Array)

Name	Description	Syntax and Example
<code>\$pull</code>	Removes all array elements that match a specified query.	<pre>{ \$pull: { <field1>: <value/condition>, <field2>: <value/condition>, ... } }</pre> <p>To specify a <field> in an embedded document or in an array, use dot notation.</p>

- Given the following document in a collection students:

```
{ _id: 1, fruits: [ "apples", "pears", "oranges", "grapes", "bananas" ], vegetables: [ "carrots", "celery", "squash", "carrots" ] }  
{ _id: 2, fruits: [ "plums", "kiwis", "oranges", "bananas", "apples" ], vegetables: [ "broccoli", "zucchini", "carrots", "onions" ] }
```

To update all documents in the collection to remove "apples" and "oranges" from the array fruits and remove "carrots" from the array vegetables

```
db.stores.update( { }, { $pull: { fruits: { $in: [ "apples", "oranges" ] }, vegetables: "carrots" } }, { multi: true } )
```

After the operation:

```
{ _id : 1, fruits : [ "pears","grapes","bananas" ], vegetables : [ "celery", "squash" ] }  
{ _id : 2, fruits : [ "plums","kiwis", "bananas" ], vegetables : [ "broccoli","zucchini","onions" ] }
```

- Given the following document in the profiles collection

```
{ _id: 1, votes: [ 3, 5, 6, 7, 7, 8 ] }
```

```
db.profiles.update( { _id: 1 }, { $pull: { votes: { $gte: 6 } } } )
```

Update Operator (for Array)

Name	Description	Syntax and Example
\$push	- appends a specified value to an array.	<pre>{ \$push: { <field1>: <value1>, ... } }</pre> <ul style="list-style-type: none">- If the field is absent in the document to update, \$push adds the array field with the value as its element.- To append 89 to the scores array <code>db.students.update({_id: 1},{ \$push:{scores: 89} })</code>- To append Multiple Values to an Array <code>db.students.update({ name: "joe" }, { \$push: { scores: { \$each: [90, 92, 85] } } })</code>

Update Operator Modifiers (for Array)

Name	Description
<u>\$each</u>	Modifies the <u>\$push</u> and <u>\$addToSet</u> operators to append multiple items for array updates.
<u>\$position</u>	Modifies the <u>\$push</u> operator to specify the position in the array to add elements.
<u>\$slice</u>	Modifies the <u>\$push</u> operator to limit the size of updated arrays.
<u>\$sort</u>	Modifies the <u>\$push</u> operator to reorder documents stored in an array.

self-study

3- Delete Documents

Method	Description
db.collection.deleteOne()	Removes a single document from a collection. (Read more)
db.collection.deleteMany()	Removes all documents that match the <filter> from a collection. (Read more)

```
db.collection.deleteOne(  
    <filter>,           //Specify an empty document { } to delete the first document returned in the collection.  
    {  
        writeConcern: <document>,  
        ...  
    }  
)
```


Question?

