mongoDB

Bộ môn: Kỹ Thuật Phần Mềm

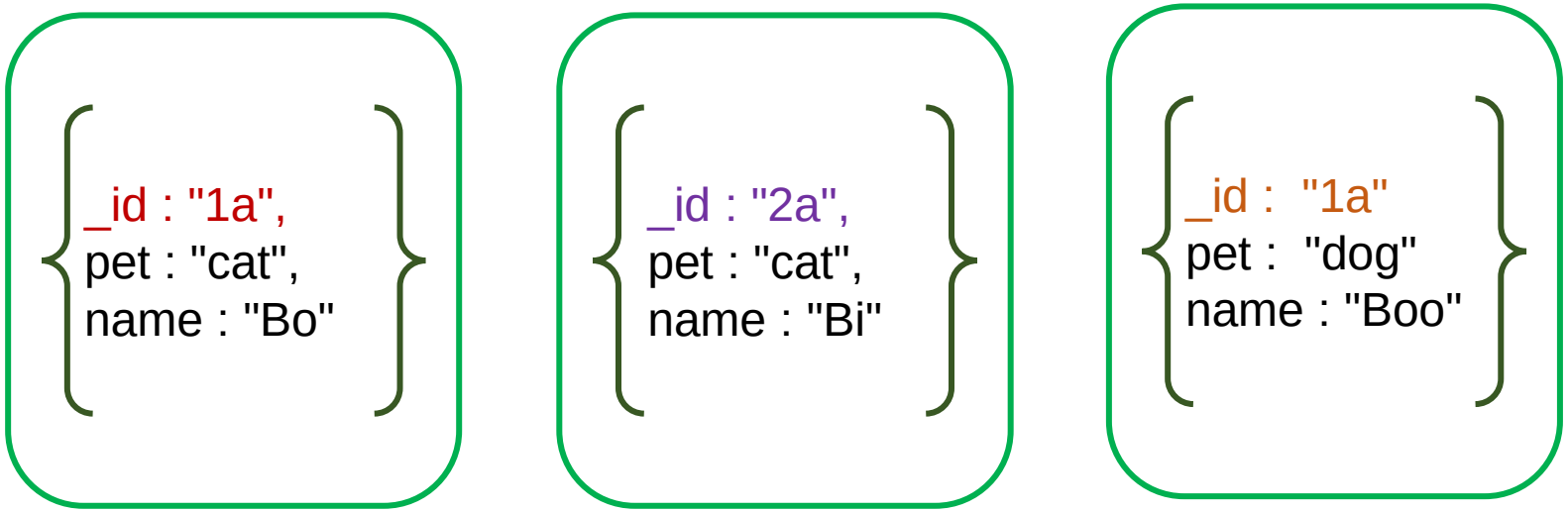
Giáo viên: **Trần Thế Trung.**
Email: tranthetrung@iuh.edu.vn

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 specifield.
  - ❖ *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"),

Document Validation

- **A collection** does not require its documents to have the same schema, do not need to have the same set of fields and the data type for a field can differ across documents within a collection.
- If the database doesn’t enforce rules about the data, development teams need to implement this logic in their applications. However, use of multiple development languages makes it hard to add a validation layer across multiple applications.
- **Document validation** solves the above problem and maintains the benefits of a dynamic schema..

Document Validation

- **Document Validation**
  - Provides significant flexibility to customize which parts of the documents are and are not validated for any collection. For any key it might be appropriate to check:
    - That a key exists
    - If a key does exist, is it of the correct type
    - That the value is in a particular format (regular expressions can be used to check if the contents of the string matches a particular pattern)
    - That the value falls within a given range

Document Validation

- **When and How to Add Document Validation**
  - **For existing data**, to allow the application to continue to operate as we introduce validation into our collections. Therefore, we want to allow updates and simply log failed validations so we can take corrective measures separately if necessary, or take no action.
  - **For new data**, to ensure the data is valid and therefore return an error if the validation fails.
  - **For any collection**, developers or the DBA can choose to specify validation rules for each collection as well as indicating whether failed validations result in a hard error or just a warning

Document Validation

- **Example**: the following snippet adds validations to the contacts collection that validates:
  - The year of **birth** is no **later than 1994**
  - The document contains a **phone number and/or an email address**
  - When present, the **phone number and email address are strings**

```
db.runCommand({
   collMod: "contacts",
   validator: {
      $and: [
        {yearOfBirth: {$lte: 1994}},
        {$or: [
                  {"contact.phone": { $type: "string"}},
                  {"email": { $type: "string"}}
               ]}]
    }})
```

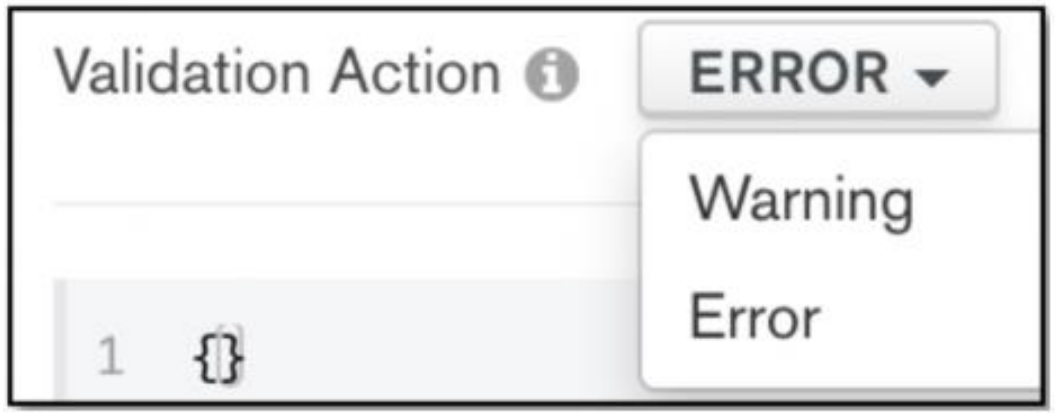
Document Validation

- **Add Document Validation Rules Using MongoDB Compass**
  - 1. Select the collection with the data you would like to add a rule for.
  - 2. Select the **Validation** tab on the collection Compass dashboard.

  - 3. Select **Add Rule** button in the center of the dashboard.

Document Validation

- **Add Document Validation Rules Using MongoDB Compass**
  - 4. Select the **Validation Action** as an **Error or a Warning** from the drop-down menu in the top-left of the dashboard.
  - 5. Select the **Validation Level** as **Off, Moderate, or Strict** from the drop-down menu next to the Validation Action dropdown menu



Document Validation

- 6. Enter the rule you want to apply to the cluster in the text box in the center of the dashboard using JSON syntax.

```
{
   $or: [
      { phone: { $type: "string" } },
      { email: { $regex: /@mongodb\.com$/ } },
      { status: { $in: [ "Unknown", "Incomplete" ] } }
   ]
}
```

- Review the validation results at the bottom beneath the text box.

Document Validation

- **Validation Actions and Levels**
  - **The validation action:** determines whether to warn but accept invalid documents, or error and reject invalid documents.
  - **The validation level:** determines how strictly MongoDB applies validation rules to existing documents.
    - **Strict validation** applies your rules to all document inserts and updates.
    - **Moderate validation** only applies your rules to new documents and existing valid documents. Existing invalid documents are not affected.

1- Insert Documents

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

Insert Method

```
db.collection.insert(
        <documents>,
        {
                writeConcern : <document>,
                ordered: <boolean>                    //only InsertMany command supported
        }
)
```

Parameter	Type	Description
<documents>	document	*document or array of documents* to insert into the collection.
writeConcern	document	Optional. A document expressing the write concern. Omit to use the default write concern.
ordered	boolean	Optional. 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 concert:

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

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

Update Method

Parameter	Type	Description	
{filter}	document	The selection criteria for the update. The same query selectors as in the find() method are available.	
{update}	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: $addFields and its alias $set $project and its alias $unset $replaceRoot and its alias $replaceWith.
upsert	boolean	Optional. When true, updateOne() either: • Creates a new document if no documents match the filter. • Updates a single document that matches the filter.	
arrayFilters	array	Optional. 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
$set	Sets the value of a field in a document	{ $set : { <field1>: <value1>, ... } }

- 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"  } } )
```

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 Pizzaria" },
        { $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	{ $inc : { <field1>: *<amount1>*, ... } } 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. The $inc operator accepts positive and negative values. db.products.update({ sku: *"abc123"* }, { $inc : { qty : *-2*, "metrics.orders": *1* } })
$mul	{ $mul : { <field1>: *<number1>*, ... } } 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. db.products.update({ sku: *"abc123"* }, { $mul : {qty : *2* } }) Apply $mul Operator to a Non-existing Field: db.products.update({ _id : *104* }, { $mul : { unit_price : *100* } }, {upsert : true})

Update Operator

Name	Description, syntax and example
$unset	{ $unset : { <field1>: "", ... } } Removes the specified field from a document. db.products.update({ _id : *104* }, { $unset : { unit_price : "" } })
	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.
$min	{ $min : { <field1>: *<value1>*, ... } } Consider the following document in the collection scores : { _id : *1*, highScore: *800*, lowScore : *200* } db.scores.update({ _id : *1* }, { $min : { lowScore : *150* } }) The scores collection now contains the following modified document: { _id : *1*, highScore: *800*, lowScore : *150* }. Add "unit price" field (does not exist before) and set to specified value db.products.update({ _id : *103* }, { $min : { unit_price : *150* } })
$max	{ $max : { <field1>: *<value1>*, ... } } Consider the following document in the collection scores : { _id : 1, highScore: 800, lowScore : 200 } db.scores.update({ _id : *1* }, { $max : { hightScore : *950* } }) The scores collection now contains the following modified document: { _id : 1, highScore: 950, lowScore : 200 }

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	{ "<array>.$" : *value* }.

- 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 : 4, grades : 88 }, { $set : { "grades.$" : 82 } } )
```

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

```
{ _id : 7, 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 : 8,
               grades : { $elemMatch: { grade : { $lte : 90 }, mean: { $gt : 80 } } } },
          { $set : { "grades.$.std" : 6 } }
)
```

Update Operator (for Array)

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

- 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:tor:

```
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 students 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 students 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
$addToSet	- Adds elements to an array only if they do not already exist in the set. - *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.*	{ $addToSet : { <field1>: *<value1>*, ... } }

- 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	{ $pop : { <field>: <-1 | 1>, ... } } To specify a <field> in an embedded document or in an array, use dot notation. 1: to remove the last element. -1: to remove the first element.

- Given the following document in a collection students:

```
{ _id : 15, scores : [ 8, 9, 10 ] }
```

To remove the first element (8) in the scores array:

```
db.students.update( { _id : 15},{ $pop : { scores : -1} } )
```

To remove the last element (10) in the scores array:

```
db.students.update( { _id : 15 }, { $pop : { scores : 1 } } )
```

Update Operator (for Array)

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

- 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.	{ $push : { <field1>: *<value1>*, ... } } 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

```
db.students.update( { _id : 1 }, { $push : { scores : 89 } } )
```

- To append Multiple Values to an Array

```
db.students.update( { name : "joe" },{ $push : { scores : { $each: [ 90, 92, 85 ] } } } )
```

Update Operator Modifiers (for Array)

Name	Description
$each	Modifies the $push and $addToSet operators to append multiple items for array updates.
$position	Modifies the $push operator to specify the position in the array to add elements.
$slice	Modifies the $push operator to limit the size of updated arrays.
$sort	Modifies the $push 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?