e-PGPathshala
Subject : Computer Science
Paper: Data Analytics
Module No 33: CS/DA/33 –MongoDB-II
Quadrant 1 – e-text

1.1 Introduction

This chapter gives an overview of the data types available in MongoDB and working with Mongo's query.

1.2 Learning Objectives

- To Understand the datatypes in MongoDB
- To understand Mongo's query and update languages

1.3 MongoDB - Datatypes

MongoDB supports many datatypes. Some of them are:

- **String**: This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer**: This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean**: This type is used to store a boolean (true/ false) value.
- **Double**: This type is used to store floating point values.
- **Min/Max Keys**: This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays**: This type is used to store arrays or list or multiple values into one key.
- **Timestamp**: ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object**: This datatype is used for embedded documents.
- **Null**: This type is used to store a Null value.
- **Symbol**: This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date**: This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID**: This datatype is used to store the document's ID.
- **Binary data**: This datatype is used to store binary data.

- **Code**: This datatype is used to store JavaScript code into the document.
- **Regular expression**: This datatype is used to store regular expression.

1.4 MongoDB – Insert, Query,Update and Delete Document

(i) The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()**method.

Syntax >db.COLLECTION_NAME.insert(document)

Example

```
>db.mycol.insert({
_id: ObjectId(7df78ad8902c),
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
})
```

- ✓ Here **mycol** is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.
- ✓ In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.
- ✓ _id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows − _id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer).
- ✓ To insert multiple documents in a single query, you can pass an array of documents in insert() command.

Example:

```
>db.post.insert([
        {
        title: 'MongoDB Overview',
        description: 'MongoDB is no sql database',
        by: 'tutorials point',
        url: 'http://www.tutorialspoint.com',
        tags: ['mongodb', 'database', 'NoSQL'],
        likes: 100
        },
        {
        title: 'NoSQL Database',
        description: 'NoSQL database doesn't have tables',
        by: 'tutorials point',
        url: 'http://www.tutorialspoint.com',
        tags: ['mongodb', 'database', 'NoSQL'],
        likes: 20,
        comments: [
        {
        user:'user1',
        message: 'My first comment',
        dateCreated: new Date(2013,11,10,2,35),
        like: 0
        }
        ]
        }
        ])
```

To insert the document you can use **db.post.save(document)** also. If you don't specify **_id** in the document then **save()** method will work same as **insert()** method.

(ii) MongoDB - Query Document

The find() Method

To query data from MongoDB collection, you need to use MongoDB's **find()**method.

Syntax

The basic syntax of **find()** method is as follows: >db.COLLECTION_NAME.find()

find()method will display all the documents in a non-structured way.

The pretty() Method

To display the results in a formatted way, you can use **pretty()** method.

Syntax >db.mycol.find().pretty()

Example

```
>db.mycol.find().pretty()
{
"_id": ObjectId(7df78ad8902c),
"title": "MongoDB Overview",
"description": "MongoDB is no sql database",
"by": "tutorials point",
"url": "http://www.tutorialspoint.com",
"tags": ["mongodb", "database", "NoSQL"],
"likes": "100"
}
>
```

Apart from find() method, there is **findOne()** method, that returns only one document.

- **RDBMS Where Clause Equivalents in MongoDB**

To query the document on the basis of some condition, you can use following operations

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

AND in MongoDB

Syntax

In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition. Following is the basic syntax of **AND** –

```
>db.mycol.find({key1:value1, key2:value2}).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({"by":"tutorials point","title": "MongoDB Overview"}).pretty()
{
"_id": ObjectId(7df78ad8902c),
"title": "MongoDB Overview",
"description": "MongoDB is no sql database",
"by": "tutorials point",
"url": "http://www.tutorialspoint.com",
"tags": ["mongodb", "database", "NoSQL"],
"likes": "100"
}
>
```

For the above given example, equivalent where clause will be **' where by='tutorials point' AND title = 'MongoDB Overview' '**. You can pass any number of key, value pairs in find clause.

OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** –

```
>db.mycol.find(
{
$or: [
{key1: value1}, {key2:value2}
]
}
).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()
{
"_id": ObjectId(7df78ad8902c),
"title": "MongoDB Overview",
"description": "MongoDB is no sql database",
"by": "tutorials point",
"url": "http://www.tutorialspoint.com",
"tags": ["mongodb", "database", "NoSQL"],
"likes": "100" } >
```

Using AND and OR Together

Example

The following example will show the documents that have likes greater than 100 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},
{"title": "MongoDB Overview"}]}).pretty()
{
"_id": ObjectId(7df78ad8902c),
"title": "MongoDB Overview",
"description": "MongoDB is no sql database",
"by": "tutorials point",
"url": "http://www.tutorialspoint.com",
"tags": ["mongodb", "database", "NoSQL"],
"likes": "100" }
>
```

(iii) MongoDB - Update Document

MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows:

```
>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

Example

Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},
{$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

MongoDB Save() Method

The **save()** method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB **save()** method is −

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Example

Following example will replace the document with the _id '5983548781331adf45ec7'.

```
>db.mycol.save(
{
"_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic",
"by":"Tutorials Point"
}
)
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic",
"by":"Tutorials Point"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

(iv) MongoDB - Delete Document

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove () method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria**: (Optional) deletion criteria according to documents will be removed.
- **justOne**: (Optional) if set to true or 1, then remove only one document.

Syntax - Basic syntax of **remove()** method is as follows:

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Example

Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

```
>db.mycol.remove({'title':'MongoDB Overview'})
```

>db.mycol.find()

```
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

```
>db.mycol.remove()
>db.mycol.find()
```

1.5 MongoDB – Projection methods

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

(i) The find() Method

MongoDB's **find()** method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute **find()** method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

Syntax

The basic syntax of **find()** method with projection is as follows:

```
>db.COLLECTION_NAME.find({},{KEY:1})
```

Example

Consider the collection mycol has the following data

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the title of the document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

Please note _**id** field is always displayed while executing **find()** method, if you don't want this field, then you need to set it as 0.

(ii) The Limit() Method

To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of **limit()** method is as follows:

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Example

Consider the collection myycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

If you don't specify the number argument in **limit()** method then it will display all documents from the collection.

(iii) MongoDB Skip() Method

Apart from limit() method, there is one more method **skip()** which also accepts number type argument and is used to skip the number of documents.

Syntax

The basic syntax of **skip()** method is as follows:

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Example

Following example will display only the second document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

Please note, the default value in **skip()** method is 0.

1.6 MongoDB – Sorting and Indexing Records

(i) The sort() Method

To sort documents in MongoDB, you need to use **sort()** method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

The basic syntax of **sort()** method is as follows:

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Example

Consider the collection myycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Please note, if you don't specify the sorting preference, then **sort()** method will display the documents in ascending order.

(ii) Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and requires MongoDB to process a large volume of data.

Indexes are special data structures that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

The ensureIndex() Method

To create an index you need to use ensureIndex() method of MongoDB.

Syntax

The basic syntax of **ensureIndex()** method is as follows().

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

Here key is the name of the file on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

Example

```
>db.mycol.ensureIndex({"title":1})
>
```

In **ensureIndex()** method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.ensureIndex({"title":1,"description":-1})
>
```

ensureIndex() method also accepts list of options (which are optional). Following is the list:

Parameter	**Type**	**Description**
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**.
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**.

name	String	The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.
dropDups	Boolean	Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**.
sparse	Boolean	If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is **false**.
expireAfterSeconds	Integer	Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
	Index Version	The index version number. The default index version depends on the version of MongoDB running when creating the index.
weights	Document	The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.
default_language	String	For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **English**.
language_override	String	For a text index, specify the name of the field in the document that contains the language to override the default language. The default value is language.

1.7 MongoDB – Aggregation and Pipeline concepts

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(*) and with group by is an equivalent of mongodb aggregation.

(i)The aggregate() Method

For the aggregation in MongoDB, you should use **aggregate()** method.

Syntax

Basic syntax of **aggregate()** method is as follows:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example

In the collection you have the following data:

```
{
_id: ObjectId(7df78ad8902c)
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by_user: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
_id: ObjectId(7df78ad8902d)
title: 'NoSQL Overview',
description: 'No sql database is very fast',
by_user: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 10
},
{
_id: ObjectId(7df78ad8902e)
title: 'Neo4j Overview',
description: 'Neo4j is no sql database',
by_user: 'Neo4j',
url: 'http://www.neo4j.com',
tags: ['neo4j', 'database', 'NoSQL'],
likes: 750
},
```

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following **aggregate()** method:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
```

```
{
"result" : [
{
"_id" : "tutorials point",
"num_tutorial" : 2
},
{
"_id" : "Neo4j",
"num_tutorial" : 1
}
],
"ok" : 1
}
```

Sql equivalent query for the above use case will be **select by_user, count(*) from mycol group by by_user**.

In the above example, we have grouped documents by field **by_user** and on each occurrence of by_user previous value of sum is incremented. Following is a list of available aggregation expressions.

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min :

	all documents in the collection.	"$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

(iii) Pipeline Concept

In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework:

- **$project:** Used to select some specific fields from a collection. The $project stage has the following prototype form:

  {$project: {<specification(s)>}}

  The $project takes a document that can specify the inclusion of fields, the suppression of the _id field, the addition of new fields, and the resetting of the values of existing fields. Alternatively, you may specify the *exclusion* of fields.

  The $project specifications have the following forms:

Form	Description
<field>: <1 or true>	Specify the inclusion of a field.
_id: <0 or false>	Specify the suppression of the _id field.
<field>: <expression>	Add a new field or reset the value of an existing field.
<field>:<0 or false>	New in version 3.4. Specify the exclusion of a field. If you specify the exclusion of a field other than _id, you **cannot** employ any other $project specification forms.

- **$match:** This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.

  The $match stage has the following prototype form:

  {$match: {<query>}}

  $match takes a document that specifies the query conditions. The query syntax is identical to the read operation query syntax.

- **$group:** This does the actual aggregation as discussed above.

  The $group stage has the following prototype form:

  {$group: {_id: <expression>, <field1>” {<accumulator1>” <expression1>},... }}

  The _id field is *mandatory*; however, you can specify a _id value of null to calculate accumulated values for all the input documents as a whole.

  The remaining computed fields are *optional* and computed using the <accumulator> operators.

  The _id and the <accumulator> expressions can accept any valid expression.

- **$sort:** Sorts the documents.

  The $sort stage has the following prototype form:

  { $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }

  $sort takes a document that specifies the field(s) to sort by and the respective sort order. <sortorder> can have one of the following values:

  - 1 to specify ascending order.
  - -1 to specify descending order.

    - { $meta: "textScore" } to sort by the computed textScore metadata in descending order

- **$skip:** With this, it is possible to skip forward in the list of documents for a given amount of documents. The $skip stage has the following prototype form:

  {$skip: <positive integer>}

  $skip takes a positive integer that specifies the maximum number of documents to skip.

- **$limit:** This limits the amount of documents to look at, by the given number starting from the current positions.

  The $limit stage has the following prototype form:

  {$limit: <positive integer>}

  $limit takes a positive integer that specifies the maximum number of documents to pass along.

- **$unwind:** This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

  The $unwind stage has one of two syntaxes:

  ➢ The operand is a field path:

  {$unwind: <field path>}

  To specify a field path, prefix the field name with a dollar sign $ and enclose in quotes.

  ➢ The operand is a document: (*New in version 3.2.)*

```
{
  $unwind:
    {
      path: <field path>,
      includeArrayIndex: <string>,
      preserveNullAndEmptyArrays: <boolean>
    }
}
```

Field	Type	Description
path	string	Field path to an array field. To specify a field path, prefix the field name with a dollar sign $ and enclose in quotes.

Field	**Type**	**Description**
includeArrayIndex	string	Optional. The name of a new field to hold the array index of the element. The name cannot start with a dollar sign $.
preserveNullAndEmptyArrays	boolean	Optional. If true, if the path is null, missing, or an empty array, $unwind outputs the document. If false, $unwinddoes not output a document if the path is null, missing, or an empty array. The default value is false.

<u>SUMMARY</u>

- MongoDB supports many data types that are part of existing traditional DBs.
- Understanding MongoDB queries is as simple as SQL queries.