Database Engineering

Introduction to MongoDB

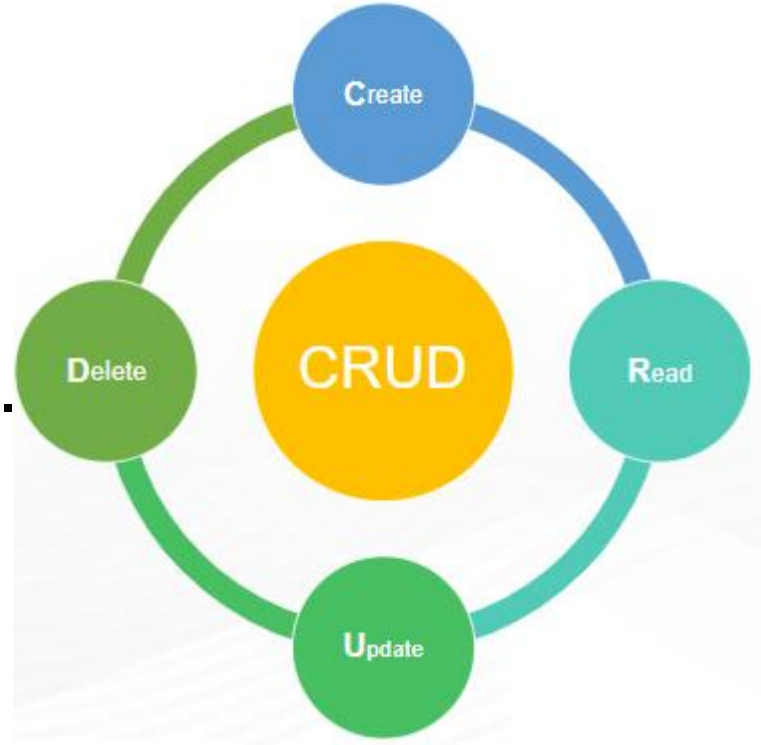
CRUD Operations

Create:
Adding new documents to the collection.

Read:
Retrieving documents from collection.

Update:
Updating documents in collection.

Delete:
Removing documents from collection.



Create Operation

Create Operation would add/insert a new document in a Collection

Makes use of insert () methods

If the desired collection does not exist where a document is required to be added, the insert method would create a new Collection

Insert operations

- Insert can be used to insert single or multiple documents.
- Each document stored in a collection required a unique _id field that act as primary key.
- Example: db.users.insert({“name”: “abc”})
- db.users.insert({“name”: “abc”, “email”: “abc@gmail.com”})
- Insert can also be used to insert multiple documents.
- db.users.insert([{“name”: “abc”, “email”: “abc@gmail.com”}, {“name”: “xyz”, “email”: “xyz@gmail.com”}, …….])

The _id Field

- The _id field is assigned to each document as a unique identifier.
- By default when inserting a new document into a collection, if you don't specify an _id field, then MongoDB will automatically generate a unique Object Id and assign it as the value for that document's _id field.

```
{
 _id: ObjectId("98232303df34948b4
 name: { first: "Ezio", last: "Au
 age: 33,
 major: [ "Italian", "Physics"]
}
```

Create Operation: .insertOne()

- The .insertOne() method inserts a document into a collection. It requires a single parameter, the document to be inserted.
- Adds a single document in the collection.
- Syntax:
  db.<collection_name>.insertOne({“field”: “value”})

```
> db.first_collection.insertOne( { "name" : "Harry" } )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("618916825eb2befec77b33c6")
}
```

Create Operation: .insertMany()

- The .insertMany() method inserts multiple documents into a collection.
- An array of multiple documents is passed through insertMany() method.
- Syntax: db.first_collection.insertMany([{“field”: “value”}])
- If the specified collection does not exist, it will create the collection and insert the given documents upon successful execution of the command.
- Both insertOne() and insertMany() would return newly inserted documents with _id field.

```
> db.first_collection.insertMany([
... {name: "Sarah Connors", gender: "F", age: 19,
... course_opted: "JavaScript", _id: "E1-002"},
... {name: "Ahfaz Salha", gender: "M", age: 18,
... course_opted: "HTML & CSS", _id: "E1-001"},
... {name: "Javed Ul-Haq", gender: "M", age: 19,
... course_opted: "React-Redux", _id: "E1-003"},
... {name: "Jessica Alba", gender: "F", age: 20,
... course_opted: "MySQL", _id: "E1-004"},
... {name: "Kevin Nash", gender: "M", age: 19,
... course_opted: "MongoDB", _id: "E1-005"},
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                "E1-002",
                "E1-001",
                "E1-003",
                "E1-004",
                "E1-005"
        ]
}
```

Update operations

- Modify created documents in the collection.
- These operations update a single collection at a time.
- Criteria or filters can be specified identifying the documents that are to be updated.
- MongoDB offers the following methods for updating documents:
- updateOne()
- updateMany()
- replaceOne()
- To update the entire collection with a field value, pass an empty filter {} in place of {<document>}

Syntax:

```
{ <document> },
{
<update operator>: {<field1>:  <value1>, . . .},
<update operator>: {<field2>: <value2>, . . .},
. . . }
```

Update operators

- To update a document, MongoDB has many in-built update operators. These operators are to be used for passing the updated form of the document to the update methods.

Operator	Description
Operator	Description
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp
$inc	Increments the value of the field by the specified amount
$min	Only updates the field if the specified value is less than the existing field value
$max	Only updates the field if the specified value is greater than the existing field value
$mul	Multiplies the value of the field by the specified amount
$rename	Renames a field
$set	Sets the value of a field in a document

updateOne() and updateMany()

- The .updateOne() method updates a single document that satisfies a given filter.
- db.users.updateOne({ name: “Bob"}, { $set: { course: “CSE"}, $currentDate: {lastModified:true } })
- The .updateMany() method updates all documents that satisfy a specific filter criteria.
- db.employees.updateMany({ salary: 70000 }, { $set: { salary: 85000 }})

replaceOne()

- The .replaceOne() method replaces a single document within the collection based on filter criteria.
- Will replace the first whole document which contains the specified field value.
- We cannot change the _id value using replaceOne()

```
db.course.replaceOne(
    {module: "MongoDB"},
    {module: "NoSQL MongoDB",
     time.days: 5,
     tags: "BD"
     description: "Basic database design"
    }
)
```

Delete Operation

- Removes the specified documents from the collection.
- These operations delete documents in a single collection at a time.
- The deleteOne() method removes a single document from a collection. It has a single required parameter which is a filter criteria to match a specific document to delete.
- db.courses.deleteOne({module: “java”})
- The .deleteMany() method removes all documents that match a given filter criteria. It takes in a single required parameter, the filter criteria to match multiple documents.
- db.courses.deleteMany({module: “java”})

Read/Query Operations

- Read through documents and retrieve the specified document(s) from a Collection by querying the collection for documents.
- MongoDB provides **find()** method to read and retrieve documents from collection.
- Syntax: **db.<collection_name>.find({field:**

```
> db.Rockers.find({'gender' : 'F'  })
{ "_id" : "E1-002", "name" : "Sarah Connors", "gender" : "F", "age" : 19, "course_opted" : "JavaScript" }
{ "_id" : "E1-004", "name" : "Jessica Alba", "gender" : "F", "age" : 20, "course_opted" : "MySQL" }
```

- Query filters could also be specified to retrieve multiple documents from a large Collection using a single or multiple filters.
- The db.collection.findOne() method also performs a read operation to return a single document.

- ***Select All Documents in a Collection:*** An empty query document ({}) selects all documents in the collection: **db.inventory.find({})**
- Not specifying a query document to the find() is equivalent to specifying an empty query document. **db.inventory.find()**
- ***Specify Equality Condition:***
- To specify equality condition, use the query document { <field>: <value> } to select all documents that contain the <field> with the specified <value>.
- **db.inventory.find({ type: "snacks" })**

Specify Conditions Using Query Operators

- A query to retrieve a specific document can be created using many operators provided by MongoDB.
- For example: using the operator **$in**, can retrieves all documents in the particular Collection where the field equals the specified value or either of the multiple specified values.
- By default, when only one value is specified and we do not indicate the operator, MongoDB uses the operator **$in**.
- For example:

  db.<collection_name>.find({field: {$in: “value”}}) is equivalent to **db.<collection_name>.find({field: “value”})**
- Syntax for querying documents having values equal to either of the specified multiple value:
- **db.<collection_name>.find({field: {$in: [“value1”, “value2”]}})**

SQL to MongoDB Mapping

Create and drop operation

SQL Schema Statements

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

MongoDB Schema Statements

Implicitly created on first `insert()` operation. The primary key `_id` is automatically added if `_id` field is not specified.

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
 } )
```

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

```
DROP TABLE users
```

```
db.users.drop()
```

Alter Operation

```
ALTER TABLE users
ADD join_date DATETIME
```

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `update()` operations can add fields to existing documents using the `$set` operator.

```
db.users.update(
    { },
    { $set: { join_date: new Date() } },
    { multi: true }
)
```

```
ALTER TABLE users
DROP COLUMN join_date
```

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `update()` operations can remove fields from documents using the `$unset` operator.

```
db.users.update(
    { },
    { $unset: { join_date: "" } },
    { multi: true }
)
```

Insert and update operation

SQL INSERT Statements

```
INSERT INTO users(user_id,
                  age,
                  status)
VALUES ("bcd001",
        45,
        "A")
```

MongoDB insert() Statements

```
db.users.insert(
   { user_id: "bcd001", age: 45, status: "A" }
)
```

SQL Update Statements

```
UPDATE users
SET status = "C"
WHERE age > 25
```

MongoDB update() Statements

```
db.users.update(
   { age: { $gt: 25 } },
   { $set: { status: "C" } },
   { multi: true }
)
```

```
UPDATE users
SET age = age + 3
WHERE status = "A"
```

```
db.users.update(
   { status: "A" } ,
   { $inc: { age: 3 } },
   { multi: true }
)
```

Select operation

SQL SELECT Statements	MongoDB find() Statements
`SELECT * FROM users`	`db.users.find()`
`SELECT * FROM users WHERE status = "A"`	`db.users.find( { status: "A" } )`
`SELECT * FROM users WHERE status != "A"`	`db.users.find( { status: { $ne: "A" } } )`

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.users.find( { status: "A" } ).sort( { user_id: 1 } )
```

```sql
SELECT COUNT(*)
FROM users
```

```
db.users.count()
```

or

```
db.users.find().count()
```

```sql
SELECT *
FROM users
LIMIT 1
```

```
db.users.findOne()
```

or

```
db.users.find().limit(1)
```

Drop: db.collection.drop()Removes a collection from the database. The method also removes any indexes associated with the dropped collection.

- It takes no arguments and will produce an error if called with any arguments. Returns: **true**, when successfully drops a collection and **false** when collection to drop does not exist.
- *To remove all documents from a collection, it may be more efficient to use the drop() method to drop the entire collection, including the indexes and then recreate the collection and rebuild the indexes.*

Delete:

- To delete multiple documents, use db.collection.deleteMany().
- To delete a single document, use db.collection.deleteOne().

$unset: The $unset operator deletes a particular field. The specified value in the $unset expression (i.e. "") does not impact the operation. If the field does not exist, then $unset has no effect.

1. What type of database is MongoDB?
A) Relational B) Document-Oriented C)Graph D)Key-Value

2. What is the primary function of the mongo shell?
A) Data backup B) Database management and querying
C) User authentication D) Server monitoring

3. Write down the query to delete the collection named “Employee” from your database in MongoDB.

4. Write a query to update the “email” of a student as “abc@gmail.com” whose department is “CSE” in the collection named “Student”.

5. Command to check existence of all collections in a particular database.

6. Which MongoDB command is used to display the database you are currently using?

7. How would you add a field with a default value to an existing MongoDB collection?

```
> db.first_collection.insertMany([
... {name: "Sarah Connors", gender: "F", age: 19,
... course_opted: "JavaScript", _id: "E1-002"},
... {name: "Ahfaz Salha", gender: "M", age: 18,
... course_opted: "HTML & CSS", _id: "E1-001"},
... {name: "Javed Ul-Haq", gender: "M", age: 19,
... course_opted: "React-Redux", _id: "E1-003"},
... {name: "Jessica Alba", gender: "F", age: 20,
... course_opted: "MySQL", _id: "E1-004"},
... {name: "Kevin Nash", gender: "M", age: 19,
... course_opted: "MongoDB", _id: "E1-005"},
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                "E1-002",
                "E1-001",
                "E1-003",
                "E1-004",
                "E1-005"
        ]
}
```

Logical Query Operators

- Logical operators return data based on expressions that evaluate to true or false.

Operator	Description
Sand	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
Snot	Inverts the effect of a query expression and returns documents that do not match the query expression.
Snor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
Sor	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

- **$and operator:** Returns all documents match the conditions of both clauses.
- Syntax: **db.<collection_name>.find({$and: [{field1: “value1”}, {field2: “value2”}]})**

Example: db.courses.find({$and: [{“course_name”: “Java”},{“marks”:60}]})

- **$or operator:** Returns all documents that match either of the given conditions.
- Syntax: **db.<collection_name>.find({$or: [{field1: “value1”}, {field2: “value2”}]})**

```
> db.users.find({"$or": [{"name": "mark"}, {"age": 30}]})

{ "_id" : ObjectId("6095719e8a42b5b1bca5175c"), "name" :
"paul", "age" : 30, "status" : true }
{ "_id" : ObjectId("609571a78a42b5b1bca5175d"), "name" :
"mark", "age" : 20, "status" : true }
```

- **$not operator:** Inverts the effect of a query expression and returns documents that do not match the query expression.
- Example: **db.course.find({“name”: {$not: {$eq: “Mark”}}})**
- **$nor operator:** Returns document which do not match with neither of the given condition.

```
> db.first_collection.find( { "$nor" : [ { "name" : "Harry" }, { "name" : "Sarah Connors" } ] })
{ "_id" : ObjectId("618915d25eb2befec77b33c4"), "name" : "Alex" }
{ "_id" : ObjectId("618916495eb2befec77b33c5"), "name" : "John", "email" : "abc@xyz.com" }
{ "_id" : "E1-001", "name" : "Ahfaz Salha", "gender" : "M", "age" : 18, "course_opted" : "HTML & CSS" }
{ "_id" : "E1-003", "name" : "Javed Ul-Haq", "gender" : "M", "age" : 19, "course_opted" : "React-Redux" }
{ "_id" : "E1-004", "name" : "Jessica Alba", "gender" : "F", "age" : 20, "course_opted" : "MySQL" }
{ "_id" : "E1-005", "name" : "Kevin Nash", "gender" : "M", "age" : 19, "course_opted" : "MongoDB" }
```

Comparison Query Operator

- Comparison operators return data based on value comparisons.

$eq: Specifies equality condition. The $eq operator matches documents where the value of a field equals the specified value.

- Syntax: { <field>: { $eq: <value> } }

Example: db.trips.find({tripduration:{$eq:800}})

$gt: selects those documents where the value of the specified field is greater than (i.e. >) the specified value.

- Syntax: { field: { $gt: value } }
- Example: db.trips.find({tripduration:{$gt:800}})

$lt: selects the documents where the value of the field is less than (i.e. <) the specified value.

- Syntax: { field: { $lt: value } }
- Example: db.trips.find({tripduration:{$lt:800}})

$gte: selects the documents where the value of the specified field is greater than or equal to (i.e. >=) a specified value (e.g. value.)

- Syntax: { field: { $gte: value } }
- Example: db.trips.find({"birth year":{$gte:1980}})

$lte: selects the documents where the value of the field is less than or equal to (i.e. <=) the specified value.

- Syntax: { field: { $lte: value } }
- Example: db.trips.find({tripduration:{$lte:800}})

$in: The $in operator selects the documents where the value of a field equals any value in the specified array.

- Syntax: { field: { $in: [<value1>, <value2>, ... <valueN>] } }
- Example: db.trips.find({ "start station name": { $in: ["Howard St & Centre St", "10 Ave & W 28 St"] } })

$ne: selects the documents where the value of the specified field is not equal to the specified value.

- Syntax: { field: { $ne: value } }
- Example: db.trips.find({tripduration:{$ne:800}})

$nin: selects the documents where: the specified field value is not in the specified array **or** the specified field does not exist.

- Syntax: { field: { $nin: [<value1>, <value2> ... <valueN>] } }
- Example: db.trips.find({ "start station name": { $nin: ["Howard St & Centre St", "10 Ave & W 28 St"] } })

Element Query Operators

- Element operators return data based on field existence or data types.
- **$exists:** The $exists operator matches documents that contain or do not contain a specified field, including documents where the field value is null.
- Syntax: **{ field: { $exists: <boolean> } }**
- When <boolean> is true, $exists matches the documents that contain the field, including documents where the field value is null.
- If <boolean> is false, the query returns only the documents that do not contain the field.
- Example: Find all the users where the phone field exists.

```
> db.users.find({"phone": {"$exists": true}})

{ "_id" : ObjectId("6095a5578a42b5b1bca51760"), "name" :
"rangel", "phone" : 9999 }
```

- MongoDB $exists does **not** correspond to SQL operator exists.

- **Exists and Not Equal To**

```
db.inventory.find( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
```

- This query will select all documents in the inventory collection where the qty field exists *and* its value does not equal 5 or 15.
- **Null Values:** $exists: true

```
db.spices.insertMany( [
   { saffron: 5, cinnamon: 5, mustard: null },
   { saffron: 3, cinnamon: null, mustard: 8 },
   { saffron: null, cinnamon: 3, mustard: 9 },
   { saffron: 1, cinnamon: 2, mustard: 3 },
   { saffron: 2, mustard: 5 },
   { saffron: 3, cinnamon: 2 },
   { saffron: 4 },
   { cinnamon: 2, mustard: 4 },
   { cinnamon: 2 },
   { mustard: 6 }
] )
```

Output:

```
{ saffron: 5, cinnamon: 5, mustard: null }
{ saffron: 3, cinnamon: null, mustard: 8 }
{ saffron: null, cinnamon: 3, mustard: 9 }
{ saffron: 1, cinnamon: 2, mustard: 3 }
{ saffron: 2, mustard: 5 }
{ saffron: 3, cinnamon: 2 }
{ saffron: 4 }
```

```
db.spices.find( { saffron: { $exists: true } } )
```

$exists: false

```
db.spices.insertMany( [
   { saffron: 5, cinnamon: 5, mustard: null },
   { saffron: 3, cinnamon: null, mustard: 8 },
   { saffron: null, cinnamon: 3, mustard: 9 },
   { saffron: 1, cinnamon: 2, mustard: 3 },
   { saffron: 2, mustard: 5 },
   { saffron: 3, cinnamon: 2 },
   { saffron: 4 },
   { cinnamon: 2, mustard: 4 },
   { cinnamon: 2 },
   { mustard: 6 }
] )
```

Query:

```
db.spices.find( { cinnamon: { $exists: false } } )
```

Output:

```
{ saffron: 2, mustard: 5 }
{ saffron: 4 }
{ mustard: 6 }
```

- **$type:** Selects documents if a field is of the specified type.
- $type selects documents where the *value* of the field is an instance of the specified BSON type(s).
- Querying by data type is useful when dealing with highly unstructured data where data types are not predictable.
- Syntax: **{ field: { $type: <BSON type> } }**
- BSON type can be specified either by number or name
- Exam field type i

```
> db.users.find({"phone": {"$type": "string"}})

{ "_id" : ObjectId("6095a6038a42b5b1bca51761"), "name" :
"smith", "phone" : "9999" }
```

- The $type expression can also accept an array of BSON types as:
- { field: { $type: [<BSON type1> , <BSON type2>, ...] } }
- The above query will match documents where the field value is any of the listed types.
- For documents where field is an array, $type returns documents in which at least one array element matches a type passed to $type.
- Queries for $type: "array" return documents where the field itself is an array.

```
db.grades.insertMany(
   [
      { "_id" : 1, name : "Alice King" , classAverage : 87.333333333333333 },
      { "_id" : 2, name : "Bob Jenkins", classAverage : "83.52" },
      { "_id" : 3, name : "Cathy Hart", classAverage: "94.06" },
      { "_id" : 4, name : "Drew Williams" , classAverage : NumberInt("93") }
   ]
)
```

```
db.grades.find( { "classAverage" : { $type : [ 2 , 1 ] } } );
db.grades.find( { "classAverage" : { $type : [ "string" , "double" ] } } );
```

queries return all documents where classAverage is the BSON type string or double *or* is an array containing an element of the specified types.

```
{ "_id" : 1, "name" : "Alice King", "classAverage" : 87.33333333333333 }
{ "_id" : 2, "name" : "Bob Jenkins", "classAverage" : "83.52" }
{ "_id" : 3, "name" : "Cathy Hart", "classAverage" : "94.06" }
```

MongoDB Data Types

MongoDB Data types	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Undefined	6
Object Id	7
Array	4
Binary data	5
Undefined	6
Object Id	7
Boolean	9
Date	10

MongoDB Data types	Number
Null	11
Regular Expression	12
JavaScript	13
symbol	14
JavaScript with scope	15
Integer	16 and 18
timestamp	10
Min key	255
Max key	127
Null	11
Regular Expression	12
JavaScript	13
symbol	14

Array Query Operators

- Array operators return data based on array conditions.

Operator	Description
$all	Matches arrays that contain all elements specified in the query
$elemMatch	Selects documents of element in the array field matches all specified $elemMatch conditions.
$size	Selects documents if the array field is a specified size.

$all operator

- The $all operator selects the documents where the value of a field is an array that contains all the specified elements.
- Syntax: **{ <field>: { $all: [<value1> , <value2> ...] } }**
- Behavior: Equivalent to $and Operation
- The $all is equivalent to an $and operation of the specified values; i.e. the following statement:
- { tags: { $all: [**"ssl"** , **"security"**] } } is equivalent to: { $and: [{ tags: **"ssl"** }, { tags: **"security"** }] }

```
> db.temp.insert([{"tags": ["a","b","c"]},{"tags": ["b","c", "d"]},{"tags":
["c","d","e"]}])
```

- Find all documents in which tag field array contains “b” and “c”

```
> db.temp.find({"tags": {"$all":["b","c"]}})

{ "_id" : ObjectId("6095aa688a42b5b1bca51767"), "tags" : [ "a", "b", "c" ] }
{ "_id" : ObjectId("6095aa688a42b5b1bca51768"), "tags" : [ "b", "c", "d" ] }
```

```
{
   _id: ObjectId("5234cc89687ea597eabee675"),
   code: "xyz",
   tags: [ "school", "book", "bag", "headphone", "appliance" ],
   qty: [
          { size: "S", num: 10, color: "blue" },
          { size: "M", num: 45, color: "blue" },
          { size: "L", num: 100, color: "green" }
        ]
}
```

```
{
   _id: ObjectId("5234ccb7687ea597eabee677"),
   code: "efg",
   tags: [ "school", "book" ],
   qty: [
          { size: "S", num: 10, color: "blue" },
          { size: "M", num: 100, color: "blue" },
          { size: "L", num: 100, color: "green" }
        ]
}
```

```
{
   _id: ObjectId("5234cc8a687ea597eabee676"),
   code: "abc",
   tags: [ "appliance", "school", "book" ],
   qty: [
          { size: "6", num: 100, color: "green" },
          { size: "6", num: 50, color: "blue" },
          { size: "8", num: 100, color: "brown" }
        ]
}
```

```
{
   _id: ObjectId("52350353b2eff1353b349de9"),
   code: "ijk",
   tags: [ "electronics", "school" ],
   qty: [
          { size: "M", num: 100, color: "green" }
        ]
}
```

Query:

```
db.inventory.find( { tags: { $all: [ "appliance", "school", "book" ] } } )
```

Output:

```
{
   _id: ObjectId("5234cc89687ea597eabee675"),
   code: "xyz",
   tags: [ "school", "book", "bag", "headphone", "appliance" ],
   qty: [
          { size: "S", num: 10, color: "blue" },
          { size: "M", num: 45, color: "blue" },
          { size: "L", num: 100, color: "green" }
        ]
}
```

```
{
    _id: ObjectId("5234cc8a687ea597eabee676"),
    code: "abc",
    tags: [ "appliance", "school", "book" ],
    qty: [
           { size: "6", num: 100, color: "green" },
           { size: "6", num: 50, color: "blue" },
           { size: "8", num: 100, color: "brown" }
         ]
}
```

$elemMatch Operator

- The $elemMatch operator matches documents that contain an array field with at least one element that matches all the specified query criteria.
- Syntax: **{ <field>: { $elemMatch: { <query1>,**

```
> db.results.insert({"name":"a", "scores": [50,45,60]})
WriteResult({ "nInserted" : 1 })
```

```
> db.results.insert({"name":"b", "scores": [80,88,89]})
WriteResult({ "nInserted" : 1 })
```

- Find all documents where score >=50 and <90

```
> db.results.find({"scores": {$elemMatch: {$gte: 50, $lt: 90}}})
{ "_id" : ObjectId("6095a76a8a42b5b1bca51762"), "name" : "a", "scores" : [ 50, 45,
60 ] }
{ "_id" : ObjectId("6095a77b8a42b5b1bca51763"), "name" : "b", "scores" : [ 80, 88,
89 ] }
```

- Documents in the *scores* collection:

{ _id: 1, results: [82, 85, 88] }, { _id: 2, results: [75, 88, 89] }

```
db.scores.find(
   { results: { $elemMatch: { $gte: 80, $lt: 85 } } }
)
```

- This query matches only those documents where the results array contains at least one element that is both greater than or equal to 80 and is less than 85.

```
{ "_id" : 1, "results" : [ 82, 85, 88 ] }
```

- The query returns the following document since the element 82 is both greater than or equal to 80 and is less than 85.

Array of Embedded Documents:

```
db.survey.insertMany( [
   { "_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 } ] },
   { "_id": 4, "results": [ { "product": "abc", "score": 7 },
                            { "product": "def", "score": 8 } ] }
] )
```

Query:

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

Array of Embedded Documents:

```
db.survey.insertMany( [
   { "_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 } ] },
   { "_id": 4, "results": [ { "product": "abc", "score": 7 },
                            { "product": "def", "score": 8 } ] }
] )
```

Query:

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

- The query matches only those documents where the results array contains at least one element with both product equal to "xyz" and score greater than or equal to 8.

Outp

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

Single Query Condition: If specifying a single query predicate in the $elemMatch expression, and not using the $not or $ne operators inside of $elemMatch, $e

```
db.survey.find(
   { results: { $elemMatch: { product: "xyz" } } }
)
```

- With $elemMatch:
- Without $elemMatch:

```
db.survey.find(
   { "results.product": "xyz" }
)
```

- However, if $elemMatch expression contains the $not or $ne operators then omitting the $elemMatch expression changes the documents returned.

- $elemMatch:

```
db.survey.find(
   { "results": { $elemMatch: { product: { $ne: "xyz" } } } }
)
```

```
db.survey.find(
   { "results.product": { $ne: "xyz" } }
)
```

```
{ "_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 } ] }
{ "_id" : 4, "results" : [ { "product" : "abc", "score" : 7 },
                           { "product" : "def", "score" : 8 } ] }
```

$size Operator

- The $size operator matches any array with the number of elements specified by the argument.
- Find documents where tag field has 3 element

```
> db.temp.find({"tags": {"$size": 3}})
{ "_id" : ObjectId("6095aa688a42b5b1bca51767"), "tags" : [ "a", "b", "c" ] }
{ "_id" : ObjectId("6095aa688a42b5b1bca51768"), "tags" : [ "b", "c", "d" ] }
{ "_id" : ObjectId("6095aa688a42b5b1bca51769"), "tags" : [ "c", "d", "e" ] }
```

- **Example:** db.collection.find({ field: { $size: 2 } });
- This query returns all documents in collection where field is an array with 2 elements.
- For instance, the above expression will return { field: [red, green] } and { field: [apple, lime] } but *not* { field: fruit } or { field: [orange, lemon, grapefruit] }.
- To match fields with only one element within an array use $size with a value of 1, as follows:
- db.collection.find({ field: { $size: 1 } });