



LESSON

Querying Complex Data in MongoDB with MQL

Google slide deck available [here](#)

This work is licensed under the [Creative Commons
Attribution-NonCommercial-ShareAlike 3.0 Unported License](#)
(CC BY-NC-SA 3.0)

Overview



Learning Objectives

At the end of this lesson, learners will be able to:

- Return data from documents that contain embedded or nested subdocuments using MQL.
- Query for null or missing fields.
- Understand how to use the projection to query more precise data.
- Identify the constructs of an array and how to query when one is present in a document.

Suggested Uses

- A whole lecture or spread out across multiple lecture periods
- Handouts / asynchronous learning
- Supplemental reading material - read on your own / not part of formal teaching
- Complement to University courses [Introduction to MongoDB](#) and [MongoDB for SQL Pros](#)

This lesson is a part of the courses [Querying in Non-Relational Databases](#) and [Introduction to Modern Databases with MongoDB](#).

At a Glance



Length:
90 minutes



Level:
Intermediate



Prerequisites:
[The MongoDB Query Language \(MQL\)](#)

This work is licensed under the [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#)
(CC BY-NC-SA 3.0)

Share your feedback: We hope these curriculum materials will be a valuable resource for you and your learners. Let us know how the materials work for you, what we can improve on, and how MongoDB for Academia can support you via our brief [feedback form](#).

MongoDB for Academia: MongoDB for Academia offers resources for educators and students to support teaching and learning MongoDB. Check out our [educator resources](#) and join the Educator Community. Students can receive \$50 in Atlas credits and free certification through the [GitHub Student Developer Pack](#).

Last Update: March 2025

This lesson includes exercises



Follow along using these tools

Create a Database

- [MongoDB Atlas](#) (cloud)
- [MongoDB Community](#) (local install)

Connect to Your Database

- [MongoDB Shell](#) (open source)

For more instructions on how to use MongoDB Atlas with your students, see [Atlas for Educators](#)

These exercises were designed to work in the MongoDB Shell. If you prefer to use a GUI to interact with your data, please use [MongoDB Compass](#) or the [MongoDB Atlas UI](#).



Using MQL to Query Complex Data

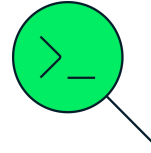
Embedded/Nested Documents

Arrays

Array of Nested Documents

Project fields to return

Query for null or missing fields



We are going to look at using MQL for more complex data in this lesson where the documents could have embedded or nested documents. This increases the related complexity of the queries and we'll work through how you can use MQL to return data from these types of documents.



Using MQL to Query Complex Data

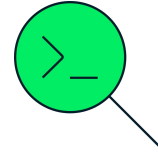
Embedded/Nested Documents

Arrays

Array of Nested Documents

Project fields to return

Query for null or missing fields



We will also investigate using MQL for more complex queries where the documents contain arrays and how we can query them.



Using MQL to Query Complex Data

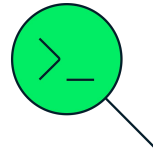
Embedded/Nested Documents

Arrays

Array of Nested Documents

Project fields to return

Query for null or missing fields



We will also explore using MQL for queries where the documents contain an array of nested documents and how we can query them.



Using MQL to Query Complex Data

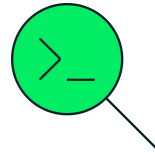
Embedded/Nested Documents

Arrays

Array of Nested Documents

Project fields to return

Query for null or missing fields



We will also explore using MQL for queries to use projection to return only the fields we want.

Projection is also useful as only the fields projected will be returned in the results to the client, this in cases of larger documents can significantly reduce the amount of data being sent back to the client. Ideally, we should design queries to return only the data from the document that they require.



Using MQL to Query Complex Data

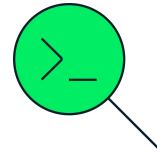
Embedded/Nested Documents

Arrays

Array of Nested Documents

Project fields to return

Query for null or missing fields



Finally, we will then explore using MQL for queries for null or for missing fields in documents.



Embedded/Nested Documents



Embedding/Nesting Documents

Embedded data exists in a single document

Can be viewed as “denormalized” model

Use dot notation to access data within the nested/embedded document

Provides better read performance, allows for all related data to be request and retrieved in one database operation.

Provides for the updates on related data to be done in a single *atomic* write.

Embedding related data within a single document is one of MongoDB’s really powerful features. A typical example might be having an address sub-document embedded into a person document which effectively merges all the information we want around the person and where they live into one document which can be retrieved with one query (no joins required).

It provides many advantages and we’ll look through some examples on how and why this mechanism is so powerful during this lesson.



Example: Embedded Data

```
{
  _id: <ObjectId>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: xyz@example.com
  },
  access: {
    level: 5
    group: "dev"
  }
}
```

● Embedded sub- document

● Embedded sub- document

Here's an example of embedding two documents within a 'user' document. The 'contact' and the 'access' documents bring related information about the user and store it alongside in the same document.



Embedding/Nesting Documents

Embedded data exists in a single document

Can be viewed as a “denormalized” model

Use dot notation to access data within the nested/embedded document

Provides better read performance, allows for all related data to be request and retrieved in one database operation.

Provides for the updates on related data to be done in a single *atomic* write.

Embedding data into a single document can be viewed as denormalizing the data and it's part of the ability of MongoDB to maintain parallel schemas in any collection. This is also referred to as documents being able to hold many shapes within a single MongoDB collection.



Embedding/Nesting Documents

Embedded data exists in a single document

Can be viewed as “denormalized” model

Uses dot notation to access data within the nested/embedded document

Provides better read performance, allows for all related data to be request and retrieved in one database operation.

Provides for the updates on related data to be done in a single *atomic* write.

We will look at the dot notation syntax within MongoDB which allows for easy access to data that is stored in a nested or in a embedded format within a document.



What is dot notation? How do I use it?

“<array>.<index>” is the syntax for array element access

“<embedded document>.<field>” is the syntax for accessing a field within an embedded document

Dot notation, or using the “.” character, is used to access the fields of an embedded document or the elements of an array.

Dot notation or using the “.” character allows us to access fields within an embedded document or within elements of an array.

If you are accessing an array then the syntax would be <array_name>.<index> to access the element at the index location in the array.

If you are using dot notation to access a field within an embedded document, then the syntax is <embedded_document>.<field> to access the specific field.



Embedding/Nesting Documents

Embedded data exists in a single document

Can be viewed as “denormalized” model

Use dot notation to access data within the nested/embedded document

Provides better read performance, allows for all related data to be request and retrieved in one database operation.

Provides for the updates on related data to be done in a single *atomic* write.

An important rationale behind embedding and/or nesting documents is that by placing all the related data in one document then it can be requested and retrieved using a single database operation.



Embedding Example

In this document we highlight an example using a tennis player who competed in both singles and doubles competitions.

We are able to use document embedding to keep all the data in a single document.

These details aren't exactly the same, so we will introduce the polymorphic pattern which helps us structure the document to allow similar but not exactly the same information to be embedded together in a single document.

Polymorphic Pattern

- Allows single query for all data on a sports person
- Adds additional code paths

```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d0b"),
  "sport": "tennis",
  "athlete_first_name": "Martina",
  "athlete_surname": "Navratilova",
  "athlete_full_name": "Martina Navratilova",
  "competition_earnings": {value:
NumberDecimal("216226089"), currency:"USD"},
  "number_of_tournaments": 390,
  "number_of_titles": 177,
  "event": [
    + {...}
  ],
  ...
}
```

In this document we highlight an example using a tennis player who competed in both singles and doubles competitions.

We are able to use document embedding to keep all the data in a single document.

Beyond embedding as the data isn't exactly the same, we'll introduce the polymorphic pattern which helps us structure the document to allow similar but not exactly the same information to be embedded together in a single document.

Patterns are transformations that you can apply to your schema. These transformations or schema "building blocks" can be used to assist in schema design. These schema design patterns encapsulate best practices in terms of how to represent data in MongoDB to support specific requirements. We'll cover a range of patterns and where they can assist your schema design in a later lesson.

This allows for a single call to retrieve all the data for a single sports person, however there is some additional code paths and logic required within your application to deal with processing this. This approach is recommended as it does allow for highly performant queries to be made on your database.

Let's look at the embedded document



```
{
  "_id" :
  ObjectId("5ad88534e3632e1a35a58d0b"),
  "sport": "tennis",
  "athlete_first_name": "Martina",
  "athlete_surname": "Navratilova",
  "athlete_full_name": "Martina
  Navratilova",
  "competition_earnings": {value:
  NumberDecimal("216226089"),
  currency:"USD"},
  "number_of_tournaments": 390,
  "number_of_titles": 177,
  "event": [
    + {...}
  ]
}
```

```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d0b"),
  "sport": "tennis",
  "athlete_first_name": "Martina",
  "athlete_surname": "Navratilova",
  "athlete_full_name": "Martina Navratilova",
  "competition_earnings": {value: NumberDecimal("216226089"),
  currency:"USD"},
  "number_of_tournaments": 390,
  "number_of_titles": 177,
  "event": [ {
    "type": "singles",
    "number_of_tournaments": 390,
    "number_of_titles": 167
  },
  {
    "type": "doubles",
    "number_of_tournaments": 233,
    "number_of_titles": 177,
    "partner_full_name": ["Renáta Tomanová",
    "Beatriz Fernández", "Olga Morozova", "Chris
    Evert"]
  } ]
}
```

We can see that the information is similar but that the “doubles” category required an additional field for the partner to track who they played with.

This is a simplified example and you would likely restructure this schema further but it serves to highlight embedding.



Querying Embedded Data: Exercise

Exercise: Embedding Documents



Let's insert a document with embedded data

```
>>> sportsCol = db.getCollection("sports")
>>> db.sportsCol.insertOne({"sport" : "tennis", "athlete_first_name" : "Martina",
"athlete_surname" : "Navratilova", "athlete_full_name" : "Martina Navratilova",
"competition_earnings" : { "value" : NumberDecimal("216226089"), "currency" : "USD" },
"number_of_tournaments" : 390, "number_of_titles" : 177, "event" : [ { "type" : "singles",
"number_of_tournaments" : 390, "number_of_titles" : 167 }, { "type" : "doubles",
"number_of_tournaments" : 233, "number_of_titles" : 177, "partner_full_name" : [ "Renáta
Tomanová", "Beatriz Fernández", "Olga Morozova", "Chris Evert" ] } ] })
...
{
  acknowledged : true,
  insertedId : ObjectId("5f3a594ae90262aae835efba")
}
```

Follow along using the [MongoDB Shell](#). If you prefer to use a GUI to interact with your data, please use [MongoDB Compass](#) or the [MongoDB Atlas UI](#).

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen.

```
sportsCol = db.getCollection("sports")

sportsCol.insertOne ({
  "sport": "tennis",
  "athlete_first_name": "Martina",
  "athlete_surname": "Navratilova",
  "athlete_full_name": "Martina Navratilova",
  "competition_earnings": {value: NumberDecimal("216226089"),
currency:"USD"},
  "number_of_tournaments": 390,
  "number_of_titles": 177,
  "event": [ {
    "type": "singles",
    "number_of_tournaments": 390,
    "number_of_titles": 167
  },
  {
```

```
    "type": "doubles",
    "number_of_tournaments": 233,
    "number_of_titles": 177,
    "partner_full_name": ["Renáta Tomanová", "Beatriz
Fernández", "Olga Morozova", "Chris Evert"]
  }
]
})
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/>



Exercise: Embedding Documents

Let's use find() and project on embedded data

```
>>> sportsCol.find({"athlete_full_name": "Martina  
Navratilova"}, {"event.type": 1})  
  
{ "_id": ObjectId("5f3a594ae90262aae835efba"), "event" : [ { "type" :  
"singles" }, { "type" : "doubles" } ] }
```

Now to use the MQL find() to query the data we've just added to the database. In this example, we have used find with a projection to return just some of the embedded data, specifically the event.type field from the event sub-document.

You can copy it from the slide or from the notes here.

```
sportsCol.find({"athlete_full_name": "Martina  
Navratilova"}, {"event.type": 1})
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Exercise: Embedding Documents

We will use `updateMany()` to update three documents. Using the same window, change **<a>** to the embedded field for event types.

Change **** to the value necessary to find the singles titles.

To shorten the response, we use `project` to limit it to the event document.

```
>>> sportsCol.find({"<a>": <b>}, {event: 1})

{ "_id" : ObjectId("5f3a594ae90262aae835efba"), "event" : [ { "type" :
"singles", "number_of_tournaments" : 390, "number_of_titles" : 167 }, {
"type" : "doubles", "number_of_tournaments" : 233, "number_of_titles" :
177, "partner_full_name" : [ "Renáta Tomanová", "Beatriz Fernández", "Olga
Morozova", "Chris Evert" ] } ] }
```

In this exercise and in the same window, you should replace **<A>** with embedded field for event types.

You should change **** to the value necessary to find the singles titles. The projection is used to limit the results to only the event sub-document.

The result should be similar (the ObjectIds will differ) to the results on the slide.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Embedding/Nesting Documents

Embedded data exists in a single document

Can be viewed as “denormalized” model

Use dot notation to access data within the nested/embedded document

Provides better read performance, allows for all related data to be request and retrieved in one database operation.

Provides for the updates on related data to be done in a single *atomic* write.

In addition to the advantage for retrieving (read) documents storing related data together allows for updates to be equally done with a single atomic write.

Quiz





Quiz

Which of the following are true for dot notation in MQL?

- A. Allows embedded documents to be accessed and updated
- B. Allows for aggregation expressions to be used in MQL
- C. Allows for array element access



Quiz

Which of the following are true for dot notation in MQL?

- ✓ A. Allows embedded documents to be accessed and updated
- ✗ B. Allows for aggregation expressions to be used in MQL
- ✓ C. Allows for array element access

This is correct. Dot notation supports accessing and updating your data.

CORRECT: Allows embedded documents to be accessed and updated - dot notation supports accessing and updating your data



Quiz

Which of the following are true for dot notation in MQL?

- ✓ A. Allows embedded documents to be accessed and updated
- ✗ B. Allows for aggregation expressions to be used in MQL
- ✓ C. Allows for array element access

This incorrect. While dot notation can be used in aggregations, it is not the reason why aggregation expression can be used in MQL.

INCORRECT: Allows for aggregation expressions to be used in MQL - This is incorrect. While dot notation can be used in aggregations, it is not the reason why aggregation expression can be used in MQL.



Quiz

Which of the following are true for dot notation in MQL?

- ✓ A. Allows embedded documents to be accessed and updated
- ✗ B. Allows for aggregation expressions to be used in MQL
- ✓ C. Allows for array element access

This is correct. Dot notation allows for array element access in a reasonable easy to understand and familiar format to developers.

CORRECT: Allows for array element access - This is correct. Dot notation allows for array element access in a reasonable easy to understand and familiar format to developers.

Arrays



Arrays

Query to match the entire array

Query to match for a specific element in the array

Query such that either a single array element meets these condition or any combination of array elements meets the conditions

Query for an element by the array index position

Query to match arrays of a given size

MQL can be used to query by matching for the entire array.

Let's move to inserting some data and then querying arrays to see how we can retrieve data from arrays using MQL.



Querying Arrays: Exercise

Let's insert some data with arrays!

```
>>> db.inventory.drop()
>>> db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [ 14, 21 ] },
  { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [ 14, 21 ] },
  { item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [ 14, 21 ] },
  { item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [ 22.85, 30 ] },
  { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [ 10, 15.25 ] }
]);
...
{
  acknowledged : true,
  insertedId : ObjectId('5f3b939f63a92a8719c01239')
}
```

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen.

```
db.inventory.drop()
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm:
[ 14, 21 ] },
  { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm:
[ 14, 21 ] },
  { item: "paper", qty: 100, tags: ["red", "blank", "plain"],
dim_cm: [ 14, 21 ] },
  { item: "planner", qty: 75, tags: ["blank", "red"], dim_cm:
[ 22.85, 30 ] },
  { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [ 10,
15.25 ] }
]);
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/>



Querying Arrays: Exercise

Find a complete exact array for the “tags” field.

Using the same window, change <A> to blank and to red in the query below to search the field tags for an array with those elements and in that specific order.

How many documents are returned?

```
>>> db.inventory.find( { tags: ["red", "blank"] } )  
  
{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook",  
  "qty" : 50, "tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }
```

Now to use the MQL find() to query the data we’ve just added to the database. You can copy it from the slide or from the notes here

```
db.inventory.find( { tags: ["red", "blank"] } )
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Arrays

Query to match the entire array

Query to match for a specific element in the array

Query such that either a single array element meets these condition or any combination of array elements meets the conditions

Query for an element by the array index position

Query to match arrays of a given size

In querying complex data and array data, sometimes we will want to search for a specific element in an array and we can do that using MQL.

Let's look at an example.



Querying Arrays to Match: Exercise

Let's match a specific element in the array

```
>>> db.inventory.find( { tags: "red" } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01235"), "item" : "journal", "qty" : 25,
  "tags" : [ "blank", "red" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" : 50,
  "tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01237"), "item" : "paper", "qty" : 100, "tags"
: [ "red", "blank", "plain" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01238"), "item" : "planner", "qty" : 75,
  "tags" : [ "blank", "red" ], "dim_cm" : [ 22.85, 30 ] }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here. In this example, let's query the array to see if it contains the element "red".

```
db.inventory.find( { tags: "red" } )
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Arrays

Query to match the entire array

Query to match for a specific element in the array

Query such that either a single array element meets these condition or any combination of array elements meets the conditions

Query for an element by the array index position

Query to match arrays of a given size

We can also use MQL to create more complex query where a single array element meets a condition or set of conditions which we can use operators to define.



Querying Arrays with Conditions: Exercise

Let's match on conditions for the array

```
>>> db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01235"), "item" : "journal", "qty" : 25,
  "tags" : [ "blank", "red" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01236"), "item" : "notebook", "qty" :
  50, "tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01237"), "item" : "paper", "qty" : 100,
  "tags" : [ "red", "blank", "plain" ], "dim_cm" : [ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" : "postcard", "qty" :
  45, "tags" : [ "blue" ], "dim_cm" : [ 10, 15.25 ] }
```

Now to use the MQL find() to query the data we've just added to the database using the conditionals. You can copy it from the slide or from the notes here.

```
db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )
```

In this query conditions, it is important to note that an element can match one or both of the conditions in the find. An element can be greater than 15, or an element can be less than 20, or an element can match both.

To only find array elements that match both, we can use \$elemMatch:

```
db.inventory.find( { dim_cm: { $elemMatch: { $gt: 15, $lt: 20 } } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" :
  "postcard", "qty" : 45, "tags" : [ "blue" ], "dim_cm" : [ 10,
  15.25 ] }
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>

Querying Arrays with Conditions: Exercise



Find results with a qty field where <50 and >20

Using the same window, change <A> to blank and to red in the query below to search the field tags for an array with those elements and in that specific order.

How many documents are returned?

```
>>> db.inventory.find( { qty: { $lt: <A>, <B>: 50 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01235"), "item" :
"journal", "qty" : 25, "tags" : [ "blank", "red" ], "dim_cm" :
[ 14, 21 ] }

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" :
"postcard", "qty" : 45, "tags" : [ "blue" ], "dim_cm" : [ 10,
15.25 ] }
```

Now again let's use the MQL find() to query the data we've just added to the database. Specifically, we want to only return the documents where the field 'qty' have a value greater than 20 and less than 50. You can copy it from the slide or from the notes here.

```
db.inventory.find( { qty: { $lt: 50, $gt: 20 } } )
```

2 documents are returned that match this criteria.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Arrays

Query to match the entire array

Query to match for a specific element in the array

Query such that either a single array element meets these condition or any combination of array elements meets the conditions

Query for an element by the array index position

Query to match arrays of a given size

Querying Arrays by Index Position: Exercise



Let's match by array index position

```
>>> db.inventory.find( { dim_cm.1: { $gt: 25 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01238"), "item" : "planner", "qty" :
75, "tags" : [ "blank", "red" ], "dim_cm" : [ 22.85, 30 ] }

>>> db.inventory.find( { dim_cm.0: { $lt: 14 } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" : "postcard", "qty"
: 45, "tags" : [ "blue" ], "dim_cm" : [ 10, 15.25 ] }
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Arrays

Query to match the entire array

Query to match for a specific element in the array

Query such that either a single array element meets these condition or any combination of array elements meets the conditions

Query for an element by the array index position

Query to match arrays of a given size



Querying Arrays by Size: Exercise

Let's match by the size/length of the array

```
>>> db.inventory.find( { "tags": { $size: 3 } } )  
  
{ "_id" : ObjectId("5f3b939f63a92a8719c01237"), "item" :  
  "paper", "qty" : 100, "tags" : [ "red", "blank", "plain" ],  
  "dim_cm" : [ 14, 21 ] }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here

```
db.inventory.find( { "tags": { $size: 3 } } )
```

In this result, we see that there was only one document with a “tags” array that had three elements.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>



Querying Arrays with One Element: Exercise

Find docs where the tags array has 1 element

Using the same window, change **<A>** to 1 in the query below to search the field tags for an array with one element. The results should be the same as shown below.

```
>>> db.inventory.find( { "tags": { $size: <A> } } )

{ "_id" : ObjectId("5f3b939f63a92a8719c01239"), "item" :
"postcard", "qty" : 45, "tags" : [ "blue" ], "dim_cm" : [
10, 15.25 ] }
```

Now again let's use the MQL find() to query the data we've just added to the database. Specifically, we want to only return the documents where the field 'tags' array has only one element.

```
db.inventory.find( { "tags": { $size: 1 } } )
```

Only one document is returned that match this criteria.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>

Quiz





Quiz

Which of the following are true for querying arrays in MQL?

- ☐ A. Allows for only the entire array to be matched
- ☐ B. Allows only single query conditions against the array
- ☐ C. Allows for querying by the array index position
- ☐ D. Allows of matching of arrays of a given size



Quiz

Which of the following are true for querying arrays in MQL?

- ☐ A. Allows for only the entire array to be matched
- ☐ B. Allows only single query conditions against an array
- ☒ C. Allows for querying by the array index position
- ☒ D. Allows of matching of arrays of a given size

INCORRECT: Allows for only the entire array to be matched - MQL allows for a query to match part of the array or arrays with specific elements present

INCORRECT: Allows only single query conditions against array - MQL allows for multiple query conditions in a query

CORRECT: Allows for querying by the array index position - This is true, MQL allows you to specify an array index position and will only query that array position of that field for all the documents

CORRECT: Allows of matching of arrays of a given size - The \$size operator can be used to find arrays of a specific length



Quiz

Which of the following are true for querying arrays in MQL?

- ☒ A. Allows for only the entire array to be matched
- ☒ B. Allows only single query conditions against the array
- ☒ C. Allows for querying by the array index position
- ☒ D. Allows of matching of arrays of a given size

This incorrect. MQL allows for a query to match part of the array or arrays with specific elements present.

INCORRECT: Allows for only the entire array to be matched - MQL allows for a query to match part of the array or arrays with specific elements present



Quiz

Which of the following are true for querying arrays in MQL?

- ☒ A. Allows for only entire array to be matched
- ☒ B. Allows only single query conditions against array
- ☒ C. Allows for querying by the array index position
- ☒ D. Allows of matching of arrays of a given size

This incorrect. MQL allows for multiple query conditions in a query.

INCORRECT: Allows only single query conditions against the array - This is incorrect. MQL allows for multiple query conditions in a query



Quiz

Which of the following are true for querying arrays in MQL?

- ☒ A. Allows for only entire array to be matched
- ☒ B. Allows only single query conditions against array
- ☒ C. Allows for querying by the array index position
- ☒ D. Allows of matching of arrays of a given size

This is correct. MQL allows you to specify an array index position and will only query that array position of that field for all the documents.

CORRECT: Allows for querying by the array index position - This is correct. MQL allows you to specify an array index position and will only query that array position of that field for all the documents



Quiz

Which of the following are true for querying arrays in MQL?

- ☒ A. Allows for only entire array to be matched
- ☒ B. Allows only single query conditions against array
- ☒ C. Allows for querying by the array index position
- ☒ D. Allows matching of arrays of a given size

This is correct. The \$size operator can be used to find arrays of a specific length.

CORRECT: Allows of matching of arrays of a given size - This is correct. The \$size operator can be used to find arrays of a specific length



Array of Nested Documents



Array of Nested Documents

In MongoDB we can have arrays which consist of elements that are in of themselves documents.

Field ordering within query criteria can be important:

Equality matching with field A: value X and field B: value Y will only return that exact sequence of field-value pairs.

It will not find an array with field B: value Y and then field A: value X.

In MongoDB, you can have arrays which themselves hold documents.

In these cases, the field ordering of the query criteria become important. Specifically, taking equality matching then where field A: value X with field B: value Y being the search criteria then only documents with this exact sequence of field-value pairs will be returned by the query.

That search criteria would not return documents matching the inverse criteria, where field B: value Y is followed by field A: value X for example. This is important when to be aware of when creating your query criteria to ensure all the data is returned.

We'll look at an example where we have a product document with a field, instock, where each document within the array represents the stock for that specific warehouse (of which there are multiple).

In designing queries for nested documents within arrays we need to consider the field ordering for certain matching such as equality to ensure we design our query to return all of what we expect it to return in terms of documents.



Querying an Array of Nested Documents: Exercise

Let's insert some nested documents in an array!

```
>>> db.inventory.drop()
>>> db.inventory.insertMany( [
  { item: "journal", instock: [ { warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 } ] },
  { item: "notebook", instock: [ { warehouse: "C", qty: 5 } ] },
  { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 } ] },
  { item: "planner", instock: [ { warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 } ] },
  { item: "postcard", instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
]);
...
{
  acknowledged : true,
  insertedId : ObjectId('5f3b939f63a92a8719c01239')
}
```

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen.

```
db.inventory.drop()
db.inventory.insertMany( [
  { item: "journal", instock: [ { warehouse: "A", qty: 5
}, { warehouse: "C", qty: 15 } ] },
  { item: "notebook", instock: [ { warehouse: "C", qty: 5
} ] },
  { item: "paper", instock: [ { warehouse: "A", qty: 60
}, { warehouse: "B", qty: 15 } ] },
  { item: "planner", instock: [ { warehouse: "A", qty: 40
}, { warehouse: "B", qty: 5 } ] },
  { item: "postcard", instock: [ { warehouse: "B", qty:
15 }, { warehouse: "C", qty: 35 } ] }
]);
```

See: <https://docs.mongodb.com/manual/tutorial/query-array-of-documents/>

Querying an Array of Nested Documents: Exercise



Let's query the nested documents

```
>>> db.inventory.find( { "instock": { warehouse: "A", qty: 5 } } )

{ "_id" : ObjectId("5f3bc03563a92a8719c0123a"), "item" : "journal", "instock" : [ {
  "warehouse" : "A", "qty" : 5 }, { "warehouse" : "C", "qty" : 15 } ] }

>>> db.inventory.find( { 'instock.qty': { $lte: 5 } } )

{ "_id" : ObjectId("5f3bc03563a92a8719c0123a"), "item" : "journal", "instock" : [ {
  "warehouse" : "A", "qty" : 5 }, { "warehouse" : "C", "qty" : 15 } ] }

{ "_id" : ObjectId("5f3bc03563a92a8719c0123b"), "item" : "notebook", "instock" : [ {
  "warehouse" : "C", "qty" : 5 } ] }

{ "_id" : ObjectId("5f3bc03563a92a8719c0123d"), "item" : "planner", "instock" : [ {
  "warehouse" : "A", "qty" : 40 }, { "warehouse" : "B", "qty" : 5 } ] }
```

Now to use the MQL find() to query the nested data we've just added to the database. You can copy it from the slide or from the notes here

```
db.inventory.find( { "instock": { warehouse: "A", qty: 5 }
} )
```

In this query we are only for an array within the array field 'instock' that has the field warehouse with a value of "A" and a field qty with a value of 5 (and in that ordering of fields) in a nested document.

```
db.inventory.find( { 'instock.qty': { $lte: 5 } } )
```

This query looks across all the 'qty' fields in each nested document within the 'instock' array for those with a quantity of less than or equal to 5.

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>



Querying an Array of Nested Documents: Exercise

Let's query with several conditionals

```
>>> db.inventory.find( { "instock": { $elemMatch: { qty: { $gt: 10, $lte: 20 } } } } )

{ "_id" : ObjectId("5f3bc03563a92a8719c0123a"), "item" : "journal", "instock" : [ {
  "warehouse" : "A", "qty" : 5 }, { "warehouse" : "C", "qty" : 15 } ] }

{ "_id" : ObjectId("5f3bc03563a92a8719c0123c"), "item" : "paper", "instock" : [ {
  "warehouse" : "A", "qty" : 60 }, { "warehouse" : "B", "qty" : 15 } ] }

{ "_id" : ObjectId("5f3bc03563a92a8719c0123e"), "item" : "postcard", "instock" : [ {
  "warehouse" : "B", "qty" : 15 }, { "warehouse" : "C", "qty" : 35 } ] }
```

Now to use the MQL find() to query the nested data we've just added to the database. You can copy it from the slide or from the notes here

```
db.inventory.find( { "instock": { warehouse: "A", qty: 5 }
} )
```

In this query we are only for an array within the array field 'instock' that has the field warehouse with a value of "A" and a field qty with a value of 5 (and in that ordering of fields) in a nested document.

```
db.inventory.find( { 'instock.qty': { $lte: 5 } } )
```

This query looks across all the 'qty' fields in each nested document within the 'instock' array for those with a quantity of less than or equal to 5.

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>

Querying an Array of Nested Documents: Exercise

Find docs with a complex conditional

Using the same window, change <A> to 5, to the less than or equal operator, and <C> to the field which represents which warehouse the inventory is stored in. The results should be the same as shown below.

```
>>> db.inventory.find( { "instock": { $elemMatch: { qty: { $gte:
<A>, <B>: 20 }, "<C>": "A" } } } )

{ "_id" : ObjectId("5f50c9962d4b45b7f11b6d89"), "item" :
"journal", "instock" : [ { "warehouse" : "A", "qty" : 5 }, {
"warehouse" : "C", "qty" : 15 } ] }
```

Now again let's use the MQL find() to query the data we've just added to the database. Specifically, we want to only return the documents where the quantity of the item in warehouse "a" is 5 or greater and less than or equal to 20.

```
db.inventory.find( { "instock": { $elemMatch: { qty: {
$gte: 5, $lte: 20 }, "warehouse": "A" } } } )
```

Only one document is returned that match this criteria.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.find/>

Quiz





Quiz

Which of the following are true for arrays with nested documents in MQL?

- ☐ A. Allows \$elemMatch to query on two conditions with both being required for the query in a single nested document
- ☐ B. Does not allow the use of the dot notation to access nested documents within arrays



Quiz

Which of the following are true for arrays with nested documents in MQL?

- ✓ A. Allows \$elemMatch to query on two conditions with both being required for the query in a single nested document
- ✗ B. Does not allow the use of the dot notation to access nested documents within arrays

CORRECT: Allows \$elemMatch to query on two conditions with both being required for the query in a single nested document - This is correct and the recommended approach

INCORRECT: Does not allow the use of the dot notation to access nested documents within arrays - Dot notation can be used to access fields in nested documents in arrays



Quiz

Which of the following are true for arrays with nested documents in MQL?

- ✓ A. Allows \$elemMatch to query on two conditions with both being required for the query in a single nested document
- ✗ B. Does not allow the use of the dot notation to access nested documents within arrays

This is correct. This is also the recommended approach when querying arrays with nested documents in MQL.

CORRECT: Allows \$elemMatch to query on two conditions with both being required for the query in a single nested document - This is correct and the recommended approach



Quiz

Which of the following are true for arrays with nested documents in MQL?

- ✓ A. Allows \$elemMatch to query on two conditions with both being required for the query in a single nested document
- ✗ B. Does not allow the use of the dot notation to access nested documents within arrays

This incorrect. Dot notation can be used to access fields in nested documents in arrays.

INCORRECT: Does not allow the use of the dot notation to access nested documents within arrays - This is incorrect. Dot notation can be used to access fields in nested documents in arrays.



Project Fields to Return



Project Fields to Return

A projection can help use restrict or explicitly specify which fields should be returned from the query.

Without a projection in a query, all documents matching the query are returned.

A projection can explicitly include several fields by setting the field to 1 in the projection document or explicitly exclude fields by setting the field to 0.

Dot notation can be used with projection on nested documents or arrays.



A projection can help use restrict or explicitly specify which fields should be returned from the query.

If the projection document is empty, then all fields for all of the documents matching the query are returned.

A projection can explicitly include or exclude fields (1/0).

Dot notation works in conjunction with projection, within the projection document you can use dot notation to return nested documents and array or indeed fields/position elements within these.



Projection: Exercise

Let's insert some nested documents in an array!

```
>>> db.inventory.drop()
>>> db.inventory.insertMany( [
  { item: "journal", status: "A", size: { h: 14, w: 21, uom: "cm" }, instock: [ {
warehouse: "A", qty: 5 } ] },
  { item: "notebook", status: "A", size: { h: 8.5, w: 11, uom: "in" }, instock: [ {
warehouse: "C", qty: 5 } ] },
  { item: "paper", status: "D", size: { h: 8.5, w: 11, uom: "in" }, instock: [ {
warehouse: "A", qty: 60 } ] },
  { item: "planner", status: "D", size: { h: 22.85, w: 30, uom: "cm" }, instock: [ {
warehouse: "A", qty: 40 } ] },
  { item: "postcard", status: "A", size: { h: 10, w: 15.25, uom: "cm" }, instock: [ {
warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
]);
...
```

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen.

```
db.inventory.insertMany( [
  { item: "journal", status: "A", size: { h: 14, w: 21,
uom: "cm" }, instock: [ { warehouse: "A", qty: 5 } ] },
  { item: "notebook", status: "A", size: { h: 8.5, w: 11,
uom: "in" }, instock: [ { warehouse: "C", qty: 5 } ] },
  { item: "paper", status: "D", size: { h: 8.5, w: 11,
uom: "in" }, instock: [ { warehouse: "A", qty: 60 } ] },
  { item: "planner", status: "D", size: { h: 22.85, w: 30,
uom: "cm" }, instock: [ { warehouse: "A", qty: 40 } ] },
  { item: "postcard", status: "A", size: { h: 10, w:
15.25, uom: "cm" }, instock: [ { warehouse: "B", qty: 15
}, { warehouse: "C", qty: 35 } ] }
]);
```

See: <https://docs.mongodb.com/manual/tutorial/project-fields-from-query-results/>

Projection: Exercise



Let's query using an implicit AND and OR

```
>>> db.inventory.find( { status: "A" }, { _id: 0, item: 1, "instock.qty": 1 } )  
  
{ "item" : "journal", "instock" : [ { "qty" : 5 } ] }  
  
{ "item" : "notebook", "instock" : [ { "qty" : 5 } ] }  
  
{ "item" : "postcard", "instock" : [ { "qty" : 15 }, { "qty" : 35 } ] }  
  
>>> db.inventory.find( { status: "A" }, { _id:0, item: 1, "size.uom": 1 } )  
  
{ "item" : "journal", "size" : { "uom" : "cm" } }  
  
{ "item" : "notebook", "size" : { "uom" : "in" } }  
  
{ "item" : "postcard", "size" : { "uom" : "cm" } }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here. The first query will return all the documents with a status of 'a' however with the projection we will only see the item and instock.qty fields in the results returned.

The second query also looks for documents with a status field of 'a' and only returns the item and 'size.uom' (unit of measurement) fields in the results.

```
db.inventory.find( { status: "A" }, { _id: 0, item: 1,  
"instock.qty": 1 } )
```

In this query we are only querying for documents with a status of "A" and only returning two fields the item and the instock.qty field. The zero for _id means we are explicitly requesting the document with the ObjectId field.

```
db.inventory.find( { status: "A" }, { _id:0, item: 1,  
"size.uom": 1 } )
```

In this query we are only querying for documents with a status of "A" and only returning two fields the item and the size.uom field. The zero for _id means we are explicitly requesting the document with the ObjectId field.

Quiz





Quiz

Which of the following are true for projections in MQL?

- ☐ A. Cannot have an empty projection document
- ☐ B. Cannot be used with dot notation
- ☐ C. Can be used for auditing
- ☐ D. Can set multiple fields for projection



Quiz

Which of the following are true for projections in MQL?

- ☐ A. Cannot have an empty projection document
- ☐ B. Cannot be used with dot notation
- ☐ C. Can be used for auditing
- ☒ D. Can set multiple fields for projection

INCORRECT: Cannot have an empty projection document - Incorrect, an empty document simply returns all the fields in the document

INCORRECT: Cannot be used with dot notation - Incorrect, dot notation is usable with projections

INCORRECT: Can be used for auditing - Incorrect, there are auditing features in MongoDB Enterprise but this is not an audit feature

CORRECT: Can set multiple fields for projection - Correct



Quiz

Which of the following are true for projections in MQL?

- ☒ A. Cannot have an empty projection document
- ☒ B. Cannot be used with dot notation
- ☒ C. Can be used for auditing
- ☒ D. Can set multiple fields for projection

This incorrect. An empty document simply returns all the fields in the document.

INCORRECT: Cannot have an empty projection document - Incorrect, an empty document simply returns all the fields in the document



Quiz

Which of the following are true for projections in MQL?

- ☒ A. Cannot have an empty projection document
- ☒ B. Cannot be used with dot notation
- ☒ C. Can be used for auditing
- ☒ D. Can set multiple fields for projection

This incorrect. Dot notation is usable with projections.

INCORRECT: Cannot be used with dot notation - Incorrect, dot notation is usable with projections



Quiz

Which of the following are true for projections in MQL?

- ☐ A. Cannot have an empty projection document
- ☐ B. Cannot be used with dot notation
- ☐ C. Can be used for auditing
- ☒ D. Can set multiple fields for projection

This incorrect. There are auditing features in MongoDB Enterprise but the querying of arrays in MQL is not an audit feature.

INCORRECT: Can be used for auditing - This is incorrect. There are auditing features in MongoDB Enterprise but the querying of arrays in MQL is not an audit feature



Quiz

Which of the following are true for projections in MQL?

- ☒ A. Cannot have an empty projection document
- ☒ B. Cannot be used with dot notation
- ☒ C. Can be used for auditing
- ☒ D. Can set multiple fields for projection

This is correct. It is possible to use multiple fields for a projection.

CORRECT: Can set multiple fields for projection - This is correct. It is possible to use multiple fields for a projection.



Query for Null or Missing Fields



Query for null or missing fields

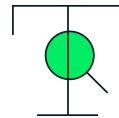
In MongoDB, different query operators treat the null field differently.

There are a number of approaches to querying:

Equality Filter

Existence Check

Type Check



MongoDB can query for null or missing fields, however you should be aware that different query operators treat the null field differently so care is needed when designing queries or schemas which use it.

Specifically, we'll look at how the equality filter, the existence check, and the type check deal with the null field.



Query for Null Fields: Exercise

Let's insert some data with null fields!

```
>>> db.inventory.drop()
>>> db.inventory.insertMany([
  { _id: 1, item: null },
  { _id: 2 }
])
...
{
  acknowledged : true,
  insertedIds" : [ 1, 2 ]
}
```

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen.

```
db.inventory.insertMany([
  { _id: 1, item: null },
  { _id: 2 }
])
```

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>



Query for Null Fields: Exercise

Let's query the null with the various approaches

```
>>> db.inventory.find( { item: null } )  
  
  { "_id" : 1, "item" : null }  
  
  { "_id" : 2 }  
  
>>> db.inventory.find( { item : { $exists: false } } )  
  
  { "_id" : 2 }  
  
>>> db.inventory.find( { item : { $type: 10 } } )  
  
  { "_id" : 1, "item" : null }
```

Now to use the MQL find() to query the data we've just added to the database. You can copy it from the slide or from the notes here

The first query will use the equality filter to check for null fields:

```
db.inventory.find( { item: null } )
```

In this query we are only checking the documents to see if either they contain the item field whose value is null or that do not contain the item field at all.

The second query will use the existence check by using the \$exists operator to find documents where the 'item' field does not exist. In the case of the query, we can see that where null is set, then it also exists so it won't be returned by this query.

The third query uses a BSON type check via the \$type operator where we look for 10 which is BSON Type for Null.

```
db.inventory.find( { item : { $exists: false } } )
```

In this query we are only checking to find documents that contain the field 'item' and which are also of the BSON Type NULL.

```
db.inventory.find( { item : { $type: 10 } } )
```

This query is only looking for documents where the item field's value is null, specifically where the value is equal to the BSON Type Null (type 10 in the BSON standard).

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>

Equality Filters on Null

```
>>> db.inventory.find( { item: null } )
```

```
{ item: null }
```

Equality filter matches

- When field value is null, or
- When documents do not contain the field

In this query, we will use the equality filter to match documents

```
db.inventory.find( { item: null } )
```

In this query we are only checking the documents to see if either they contain the item field whose value is null or that do not contain the item field at all.

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>



Existence Checks on Null

```
>>> db.inventory.find  
( { item : { $exists: false }  
  } )
```

```
{ item:  
{ $exists:  
false } }
```

Existence check
matches

- Only documents
which do or do not
contain the specified
field

In this query, we focus on using an existence check on null

```
db.inventory.find( { item : { $exists: false } } )
```

In this query we are only checking to find documents that **do not contain** the item field or if set to true which explicitly contained the field.

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>



Type checks on null

```
>>> db.inventory.find  
( { item: { $type: 10 } } )
```

```
{ item: {  
  $type: 10 }  
}
```

See: <https://docs.mongodb.com/manual/tutorial/query-for-null-fields/>

Type Check matches

- Only documents where the specified field matches the BSON Type, Null.
- This is type 10 in the BSON specification/standard.

```
db.inventory.find( { item : { $type: 10 } } )
```

This query is only looking for documents where the item field's value is null.

Specifically where the value is equal to the BSON Type Null (type 10 in the BSON standard).



Query for Null Fields: Exercise

Let's insert some more data to further explore null

```
>>> db.inventory.drop()
>>> db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [ 14, 21 ], colour: "red"},
  { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [ 14, 21 ], colour: "red" },
  { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [ 10, 15.25 ], colour: null }
]);
...
{
  acknowledged : true,
  insertedId : ObjectId(5f3b939f63a92a8719c01239)
}
```

You should cut and paste the following command directly from the slide or from these notes into the prompt (indicated by >>>). Once they have been inserted you will see the following output on the screen.

```
db.inventory.drop()
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"],
dim_cm: [ 14, 21 ], colour: "red"},
  { item: "notebook", qty: 50, tags: ["red", "blank"],
dim_cm: [ 14, 21 ], colour: "red" },
  { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [
10, 15.25 ], colour: null }
]);
```

See: <https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/>



Query for Null Fields: Exercise

Find docs with a complex conditional

Using the same window, change **<A>** to the not comparison operator and **** to the BSON Type Null. The results should be the same as shown below.

```
>>> db.inventory.find( { colour : { <A>: { $type: <B> } } } )

{ "_id" : ObjectId("5f50eb6f2d4b45b7f11b6d91"), "item" : "journal", "qty" :
25, "tags" : [ "blank", "red" ], "dim_cm" : [ 14, 21 ], "colour" : "red" }

{ "_id" : ObjectId("5f50eb6f2d4b45b7f11b6d92"), "item" : "notebook", "qty" :
50, "tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ], "colour" : "red" }
```

Now again let's use the MQL find() to query the data we've just added to the database. Specifically, we want to only return the documents which have non-null values in the field 'colour'.

```
db.inventory.find( { colour : { $not: { $type: 10 } } } )
```

Two documents are returned that match this criteria.

See: <https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/>

Quiz





Quiz

Which of the following are valid approaches to querying null fields in MongoDB documents?

- ☐ A. Equality filters
- ☐ B. Existence checks
- ☐ C. Type checks
- ☐ D. Object checks



Quiz

Which of the following are valid approaches to querying null fields in MongoDB documents?

- ✓ A. Equality filters
- ✓ B. Existence checks
- ✓ C. Type checks
- ✗ D. Object checks

CORRECT: Equality filters - Equality filters allow for checking if the field is null or for documents which do not have the field

CORRECT: Existence checks - Existence checks ensure that the field is either present or not depending on the boolean parameter passed to the check

CORRECT: Type checks - Checks to ensure the type of the field matches the specified BSON type.

INCORRECT: Object checks - There are no object checks in MongoDB, documents can be checked by schema validation, however there is no functionality for document or object checking outside of this within the database.



Quiz

Which of the following are valid approaches to querying null fields in MongoDB documents?

- ✓ A. Equality filters
- ✓ B. Existence checks
- ✓ C. Type checks
- ✗ D. Object checks

This is correct. Equality filters allow for checking if the field is null or for documents which do not have the field.

CORRECT: Equality filters - equality filters allow for checking if the field is null or for documents which do not have the field



Quiz

Which of the following are valid approaches to querying null fields in MongoDB documents?

- ✓ A. Equality filters
- ✓ B. Existence checks
- ✓ C. Type checks
- ✗ D. Object checks

This is correct. Existence checks ensure that the field is either present or not depending on the boolean parameter passed to the check.

CORRECT: Existence checks - This is correct. Existence checks ensure that the field is either present or not depending on the boolean parameter passed to the check



Quiz

Which of the following are valid approaches to querying null fields in MongoDB documents?

- ✓ A. Equality filters
- ✓ B. Existence checks
- ✓ C. Type checks
- ✗ D. Object checks

This is correct. It is a valid approach to use a type check to ensure the type of the field matches the specified BSON type.

CORRECT: Type checks - This is correct. It is a valid approach to use a type check to ensure the type of the field matches the specified BSON type.



Quiz

Which of the following are valid approaches to querying null fields in MongoDB documents?

- ✓ A. Equality filters
- ✓ B. Existence checks
- ✓ C. Type checks
- ✗ D. Object checks

This incorrect. There are no object checks in MongoDB, documents can be checked by schema validation, however there is no functionality for document or object checking outside of this within the database.

INCORRECT: Object checks - This is incorrect. There are no object checks in MongoDB, documents can be checked by schema validation however there is no functionality for document or object checking outside of this within the database.

Continue Learning!



[MongoDB University](#) has free self-paced courses and labs ranging from beginner to advanced levels.

GitHub Student Developer Pack



Sign up for the [MongoDB Student Pack](#) to receive \$50 in Atlas credits and free certification!

This concludes the material for this lesson. However, there are many more ways to learn about MongoDB and non-relational databases, and they are all free! Check out [MongoDB's University](#) page to find free courses that go into more depth about everything MongoDB and non-relational. For students and educators alike, MongoDB for Academia is here to offer support in many forms. Check out our [educator resources](#) and join the Educator Community. Students can receive \$50 in Atlas credits and free certification through the [GitHub Student Developer Pack](#).