Creating Complex Queries

Dan Geabunea
SENIOR SOFTWARE DEVELOPER

@romaniancoder www.romaniancoder.com

Overview

- Multiple conditions using $and / $or
- Query nested documents
- Handle null or missing fields
- Creating data type queries
- Implementing free text search
- Demo: Creating complex queries in MongoDB

Querying Documents Using Multiple Conditions

Logical Query Operators

$and

Query documents that satisfy multiple expressions

$or

Query documents that satisfy any expression

$and Syntax

Find

```
find(query, projection): cursor
```

Expression

```
{ field : { $operator : value} }
```

$and Query

```
{ $and : [ {expr1}, {expr2}, {expr3} ]}
```

$or Syntax

Find

```
find(query, projection): cursor
```

Expression

```
{ field : { $operator : value} }
```

$or Query

```
{ $or : [ {expr1}, {expr2}, {expr3} ]}
```

$and

Performs logical AND operation on a list of expressions and selects the documents that match each expression

```
db.aircraft.find({ $and: [ {capacity : 124}, {range : {$gt : 6000} } ] })
```

Example

All the aircraft that meet all the conditions

- Capacity is 124
- Range is greater than 6000

```
// normal syntax
db.aircraft.find({ $and: [ { range: {$lt : 6000 }} , { range: {$gt : 3000 }} ] })

// short syntax, only when using same field in all expressions
db.aircraft.find({range : { $lt:6000, $gt:3000 } })
```

When Using the Same Field

You can use the full $and syntax

You can opt for the shorter syntax if the same field is present in all expressions

$or

Performs logical OR operation on a list of expressions and selects the documents that match at least one expression

```
db.aircraft.find({ $or: [ {capacity : {$gt: 200}} , {range : {$gt : 6000}} ] })
```

Example

All the aircraft that meet at least one of the conditions

- Capacity greater than 200
- Range is greater than 6000

The order of the expressions matters; both operators use short circuit evaluation

The Power of Combining Logical Operators

```
db.aircraft.find({ $and: [
        { minRunwayLength: {$lte: 3000} },
        { $or: [ {capacity: {$gt: 200}} , {range: {$gt: 6500}} ] },
    ]
}).pretty()
```

Querying Nested Documents

Example of Nested Document

```
{
    "name" : "Gunter Hoff",
    "skills" : ["engineering"],
    "address" : {
        "city" : "Berlin",
        "country" : "Germany"
    }
}
```

```
db.crew.find({ “address.city” : “Berlin” }).pretty()
```

Syntax

```
{ “field.nestedField” : {operator} }
```

Syntax

```
{ “field.nested1.nested2” : {operator} }
```

Don’t forget the quotation marks around your field name

Querying Null Fields, Missing Fields and Field Types

Unstructured Data

Some fields may be null

A field may be present in some documents, but absent in others

A field with the same name can have multiple types

Example

```
{
    "name" : "Francois Picard",
    "address" : {
        "city" : "Paris",
        "country" : "France"
    }
}
```

```
{ "name" : "Andrei Luca", "address" : "Bucharest, Romania, 110022" }
```

```
{ "name" : "Anna Smith", "address" : null }
```

```
{ "name" : "Gunter Hoff" }
```

```
db.crew.find( {address : null} ).pretty()
```

Querying for Null Fields

Query will return documents where value is null, or ‘address’ does not exist

- { "name" : "Anna Smith", "address" : null }
- { "name" : "Gunter Hoff" }

$exists

Can be used to query documents where a field exists or not, regardless of its value

```
// Query 1
db.crew.find( {address : { $exists: false } } ).pretty()

// Query 2
db.crew.find( {address : { $exists: true } } ).pretty()
```

Querying if Field Exists

Query 1 will return documents where ‘address’ field does not exist

- { "name" : "Gunter Hoff" }

Query 2 will return documents where ‘address’ exists, even if value is null

$type

Can be used to query documents where the value of a field is of a specified BSON type

BSON Types

Type	Number	Alias
Double	1	“double”
String	2	“string”
Object	3	“object”
Array	4	“array”
Boolean	8	“bool”
Date	9	“date”
64-bit integer	18	“long”

https://docs.mongodb.com/manual/reference/bson-types/

There is also the “number” alias which can match against all numeric types

```
// Case 1

db.crew.find({address: {$type: 3}})

db.crew.find({address: {$type: “object”}})


// Case 2

db.crew.find({address: {$type: 2}})

db.crew.find({address: {$type: “string”}})


// Case 3

db.crew.find({address: {$type: 10}})

db.crew.find({address: {$type: “null”}})
```

```
// Address is object
{
   “name” : “Francois Picard”,
   “address” : {
      “city” : “Paris”,
      “country” : “France”
   }
}

// Address is string
{
   “name” : “Andrei Luca”,
   “address” : “Bucharest, Romania, 110022”
}

// Address is null
{
   “name” : “Anna Smith”,
   “address” : null
}
```

Free Text Search

Implementing Free Text Search

Text index

$text

$meta

```
{
    "name" : "Andrei Luca", "skills" : ["technical", "management"]
}
{
    "name" : "Anna Smith", "skills" : [“sales","management"]
}
```

```
db.crew.createIndex({ name: "text", skills: "text“ })
```

Text Index

They support fast text searches on string and arrays of string fields

You can not perform free text searches without a text index

```
{
    "name" : "Andrei Luca", "skills" : ["technical", "management"]
}
{
    "name" : "Anna Smith", "skills" : [“sales","management"]
}


db.crew.find({ $text : { $search : “technical Anna” } } )   // 2 matches
```

Perform a Simple Text Search

You can use the $text query operator in conjunction with $search

- In our example, the query will find all documents that have ‘technical’ or ‘Anna’ values present in the text indexed fields

```
{
    "name" : "Andrei Luca", "skills" : ["technical", "management"]
}
{
    "name" : "Anna Smith", "skills" : [“sales","management"]
}
```

```
db.crew.find({ $text : { $search : “management Anna” } } )  // 2 matches
```

Sorting by Relevance

You can aggregate results by score using the $meta projection operator

```
{
    "name" : "Andrei Luca", "skills" : ["technical", "management"]
}
{
    "name" : "Anna Smith", "skills" : [“sales","management"]
}
```

```
db.crew.find({$text: {$search: “management Anna”}}, {score: {$meta: “textScore”}})
```

Sorting by Relevance

You can aggregate results by score using the $meta projection operator

- Andrei Luca will have a score of 1
- Anna Smith will have a score of 1.75

```
{
    "name" : "Andrei Luca", "skills" : ["technical", "management"]
}
{
    "name" : "Anna Smith", "skills" : [“sales","management"]
}


db.crew.find({$text: {$search: “management Anna”}}, {score: {$meta: “textScore”}})
    .sort({score: {$meta: “textScore”}})
```

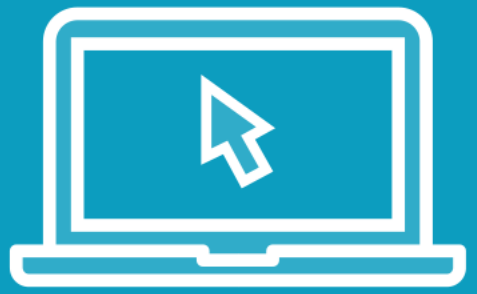
Sorting by Relevance

You can aggregate results by score using the $meta projection operator

- Andrei Luca will have a score of 1
- Anna Smith will have a score of 1.75

Bring relevant results on top

Demo



Creating complex queries in MongoDB

- Query nested documents
- Multiple query expressions
- Tackle edge cases
- Implement text search

Summary

You can create extremely powerful queries by using the logical operators

Querying nested documents is a breeze

Handle unstructured data gracefully

Implement free text search

You are on your way becoming a MongoDB query ninja

Up Next
Working with Arrays