Introduction to the MongoDB $exists operator

The $exists is an element query operator that has the following syntax:

{ field: { $exists: <boolean_value> } }

Code language: CSS (css)

When the <boolean_value> is true, the $exists operator matches the documents that contain the field with any value including null.

If the <boolean_value> is false, the $exists operator matches the documents that don't contain the **field**.

The MongoDB $exists doesn't correspond to the EXISTS operator in SQL.

Notice that MongoDB 4.2 or later doesn't treat the $type: 0 as the synonym for $exists:false anymore.

MongoDB $exists operator examples

We'll use the following products collection:

db.products.insertMany([

{ "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate" : ISODate("2011-05-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 }, "color" : ["white", "black"], "storage" : [64, 128, 256] },

{ "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate" : ISODate("2011-09-01T00:00:00Z"), "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 }, "color" : ["white", "black", "purple"], "storage" : [128, 256, 512] },

{ "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate" : ISODate("2015-01-14T00:00:00Z"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 }, "color" : ["blue"], "storage" : [16, 64, 128] },

{ "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate" : ISODate("2020-05-14T00:00:00Z"), "spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 }, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256, 1024] },

{ "_id" : 5, "name" : "SmartPhone", "price" : 599, "releaseDate" : ISODate("2022-09-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 9.7, "cpu" : 1.66 }, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256] },

{ "_id" : 6, "name" : "xWidget", "spec" : { "ram" : 64, "screen" : 9.7, "cpu" : 3.66 }, "color" : ["black"], "storage" : [1024] },

{ "_id" : 7, "name" : "xReader","price": null, "spec" : { "ram" : 64, "screen" : 6.7, "cpu" : 3.66 }, "color" : ["black", "white"], "storage" : [128] }

```
])
```

Code language: JavaScript (javascript)

1) Using the MongoDB $exists operator example

The following example uses the $exists operator to select documents where the price field exists:

```
db.products.find(
   {
 price: {
 $exists: true
 }
},
   {
 name: 1,
 price: 1
 }
)
```

Code language: CSS (css)

It returned the following documents:

```
{ "_id" : 1, "name" : "xPhone", "price" : 799 }
{ "_id" : 2, "name" : "xTablet", "price" : 899 }
{ "_id" : 3, "name" : "SmartTablet", "price" : 899 }
{ "_id" : 4, "name" : "SmartPad", "price" : 699 }
{ "_id" : 5, "name" : "SmartPhone", "price" : 599 }
{ "_id" : 7, "name" : "xReader", "price" : null }
```

Code language: JSON / JSON with Comments (json)

In this example, the $exists operator matches the documents that have the price field including the non-null and null values.

The following query uses the $exists operator that select documents whose price field exists and has a value greater than 799:

```
db.products.find({
   price: {
      $exists: true,
      $gt: 699
   }
}, {
   name: 1,
   price: 1
});
```

Code language: CSS (css)

Output:

```
{ "_id" : 1, "name" : "xPhone", "price" : 799 }
{ "_id" : 2, "name" : "xTablet", "price" : 899 }
{ "_id" : 3, "name" : "SmartTablet", "price" : 899 }
```

Code language: JSON / JSON with Comments (json)

2) Using the MongoDB $exists operator to query documents that don’t have a specified field

The following example uses the $exists operator to select documents that **don’t** have the price field:

```
db.products.find({
   price: {
      $exists: false
   }
}, {
   name: 1,
   price: 1
});
```

Code language: CSS (css)

It returned one document that doesn’t have the price field:

```
{ "_id" : 6, "name" : "xWidget" }
```

Code language: JSON / JSON with Comments (json)

Summary

- Use the { field: {$exists: true} } to select documents that contain the field. It also includes the documents where the field contains null.
- Use the { field: {$exists: false} } to match documents where the field doesn’t exist.

Introduction to the MongoDB $type operator

Sometimes, you need to deal with highly unstructured data where **data types are unpredictable**. In this case, you need to use the $type operator.

The $type is an **element query operator** that allows you to select documents where the value of a field is an instance of a specified BSON type.

The $type operator has the following syntax:

```
{ field: { $type: <BSON type> } }
```

Code language: CSS (css)

The $type operator also accepts a list of BSON types like this:

```
{ field: { $type: [ <BSON type1> , <BSON type2>, ... ] } }
```

Code language: CSS (css)

In this syntax, the $type operator selects the documents where the type of the field matches any BSON type on the list.

MongoDB provides you with three ways to identify a BSON type: string, number, and alias. The following table lists the BSON types identified by these three forms:

Type	**Number**	**Alias**
Double	1	“double”
String	2	“string”
Object	3	“object”
Array	4	“array”
Binary data	5	“binData”

Type	**Number**	**Alias**
ObjectId	7	“objectId”
Boolean	8	“bool”
Date	9	“date”
Null	10	“null”
Regular Expression	11	“regex”
JavaScript	13	“javascript”
32-bit integer	16	“int”
Timestamp	17	“timestamp”
64-bit integer	18	“long”
Decimal128	19	“decimal”
Min key	-1	“minKey”
Max key	127	“maxKey”

The $type operator also supports the number alias that matches against the following BSON types:

- double
- 32-bit integer
- 64-bit integer
- decimal

MongoDB $type operator examples

We’ll use the following products collection:

db.products.insertMany([

{ "_id" : 1, "name" : "xPhone", "price" : "799", "releaseDate" : ISODate("2011-05-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 }, "color" : ["white", "black"], "storage" : [64, 128, 256] },

{ "_id" : 2, "name" : "xTablet", "price" : NumberInt(899), "releaseDate" : ISODate("2011-09-01T00:00:00Z"), "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 }, "color" : ["white", "black", "purple"], "storage" : [128, 256, 512] },

```
	{ "_id" : 3, "name" : "SmartTablet", "price" : NumberLong(899), "releaseDate" : ISODate("2015-01-14T00:00:00Z"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "blue" ], "storage" : [ 16, 64, 128 ] },

	{ "_id" : 4, "name" : "SmartPad", "price" : [599, 699, 799], "releaseDate" : ISODate("2020-05-14T00:00:00Z"), "spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 }, "color" : [ "white", "orange", "gold", "gray" ], "storage" : [ 128, 256, 1024 ] },

	{ "_id" : 5, "name" : "SmartPhone", "price" : ["599",699], "releaseDate" : ISODate("2022-09-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 9.7, "cpu" : 1.66 }, "color" : [ "white", "orange", "gold", "gray" ], "storage" : [ 128, 256 ] },

	{ "_id" : 6, "name" : "xWidget", "spec" : { "ram" : 64, "screen" : 9.7, "cpu" : 3.66 }, "color" : [ "black" ], "storage" : [ 1024 ] }

])
```

Code language: JavaScript (javascript)

This products collection contains the price field that has int, double, long values.

1) Using the $type operator example

The following example uses the $type operator to query documents from
the products collection where the price field is the string type or is an array containing an element that is a string type.

```
db.products.find({
   price: {
      $type: "string"
   }
}, {
   name: 1,
   price: 1
})
```

Code language: CSS (css)

It returned the following documents:

```
{ "_id" : 1, "name" : "xPhone", "price" : "799" }

{ "_id" : 5, "name" : "SmartPhone", "price" : [ "599", 699 ] }
```

Code language: JSON / JSON with Comments (json)

Since the string type corresponds to the number 2 (see the BSON types table above), you can use the number 2 in the query instead:

```
db.products.find({
   price: {
      $type: 2
   }
}, {
   name: 1,
   price: 1
})
```

Code language: CSS (css)

2) Using the $type operator with the number alias example

The following example uses the $type operator with the number alias to select documents where the value of the price field is the BSON type int, long, or double or is an array that contains a number:

```
db.products.find({
   price: {
      $type: "number"
   }
}, {
   name: 1,
   price: 1
})
```

Code language: CSS (css)

It returned the following documents:

```
{ "_id" : 2, "name" : "xTablet", "price" : 899 }
{ "_id" : 3, "name" : "SmartTablet", "price" : NumberLong(899) }
{ "_id" : 4, "name" : "SmartPad", "price" : [ 599, 699, 799 ] }
{ "_id" : 5, "name" : "SmartPhone", "price" : [ "599", 699 ] }
```

Code language: JSON / JSON with Comments (json)

3) Using the $type operator to query documents with array type example

The following query use the $type operator to select the documents in which the price field is an array:

```
db.products.find({
  price: {
    $type: "array"
  }
}, {
  name: 1,
  price: 1
})
```

Code language: CSS (css)

It returned the following documents:

```
{ "_id" : 4, "name" : "SmartPad", "price" : [ 599, 699, 799 ] }
{ "_id" : 5, "name" : "SmartPhone", "price" : [ "599", 699 ] }
```

Code language: JSON / JSON with Comments (json)

4) Using the $type operator to query documents with multiple types

The following query uses the $type operator to select documents where the price field is either number or string or an array that has an element is number or string:

```
db.products.find({
  price: {
    $type: ["number", "string"]
  }
}, {
  name: 1,
  price: 1
})
```

Code language: CSS (css)

It matched the following documents:

```json
{ "_id" : 1, "name" : "xPhone", "price" : "799" }

{ "_id" : 2, "name" : "xTablet", "price" : 899 }

{ "_id" : 3, "name" : "SmartTablet", "price" : NumberLong(899) }

{ "_id" : 4, "name" : "SmartPad", "price" : [ 599, 699, 799 ] }

{ "_id" : 5, "name" : "SmartPhone", "price" : [ "599", 699 ] }
```

Code language: JSON / JSON with Comments (json)

Notice that the result doesn't include the document with _id 6 because this document doesn't have the price field.

Summary

- Use the { field: { $type: <BSON type> } } to select the documents where the value of a field is an instance of a specified BSON type.
- Use the { field: { $type: [<BSON type1> , <BSON type2>, ...] } } to select documents where the value of the field matches against one of the BSON types on the list.