$type

On this page

Definition

`$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.

A `$type` expression for a single BSON type has the following syntax:

Changed in version 3.2.

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

You can specify either the number or alias for the BSON type

The `$type` expression can also accept an array of BSON types and has the following syntax:

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

The above query will match documents where the `field` valu
in the array can be either numeric or string aliases.

See Querying by Multiple Data Type for an example.

Available Types describes the BSON types and their corresponding numeric and string aliases.

SEE ALSO:

- `$isNumber` - checks if the argument is a number. *New in MongoDB 4.4*
- `$type (Aggregation)` - returns the BSON type of the argument.

Behavior

`$type` returns documents where the BSON type of the `field` matches the BSON type passed to `$type`.

Arrays

For documents where `field` is an array, `$type` returns documents in which at least one array element matches a type passed to `$type`.

Querying for the Array BSON Type

With MongoDB 3.6 and later, querying for `$type: "array"` returns documents where the field itself is an array. Prior to MongoDB 3.6, `$type: "array"` returned documents where the field is an array containing at least one element of type `array`. For example, given the following documents:

```
{ "data" : [ "values", [ "values" ] ] }
{ "data" : [ "values" ] }
```

With MongoDB 3.6 and later, the query `find( {"data" : { $type : "array" } } )` returns both documents. Prior to MongoDB 3.6, the query returns only the first document.

Available Types

Starting in MongoDB 3.2, `$type` operator accepts string aliases f
corresponding to the BSON types. Previous versions only accepte
type. [1]

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Deprecated.
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated.
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated.
JavaScript code with scope	15	“javascriptWithScope”	Deprecated in MongoDB 4.4.
32-bit integer	16	“int”	

Type	Number	Alias	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Decimal128	19	“decimal”	New in version 3.4.
Min key	-1	“minKey”	
Max key	127	“maxKey”	

`$type` supports the `number` alias, which will match against the following BSON types:

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

For examples, see Examples.

[1] Starting in MongoDB 4.2, users can no longer use the query filter `$type: 0` as a synonym for `$exists:false`. To query for null or missing fields, see Query for Null or Missing Fields.

SEE ALSO:

`$isNumber` *New in MongoDB 4.4*

MinKey and MaxKey

MinKey and MaxKey are used in comparison operations and exist primarily for internal use. For all possible BSON element values, `MinKey` will always be the smallest value while `MaxKey` will always be the greatest value.

Querying for `minKey` or `maxKey` with `$type` will only return fields values.

Suppose that the `data` collection has two documents with `MinKey` and `MaxKey`:

```
{ "_id" : 1, x : { "$minKey" : 1 } }
{ "_id" : 2, y : { "$maxKey" : 1 } }
```

The following query will return the document with `_id: 1`:

```
db.data.find( { x: { $type: "minKey" } } )
```

The following query will return the document with `_id: 2`:

```
db.data.find( { y: { $type: "maxKey" } } )
```

Examples

Querying by Data Type

The `addressBook` contains addresses and zipcodes, where `zipCode` has `string`, `int`, `double`, and `long` values:

```
db.addressBook.insertMany(
   [
      { "_id" : 1, address : "2030 Martian Way", zipCode : "90698345" },
      { "_id" : 2, address: "156 Lunar Place", zipCode : 43339374 },
      { "_id" : 3, address : "2324 Pluto Place", zipCode: NumberLong(3921412) },
      { "_id" : 4, address : "55 Saturn Ring" , zipCode : NumberInt(88602117) },
      { "_id" : 5, address : "104 Venus Drive", zipCode : ["834847278", "1893289032"]}
   ]
)
```

The following queries return all documents where `zipCode` is the an element of the specified type:

```
db.addressBook.find( { "zipCode" : { $type : 2 } } );
db.addressBook.find( { "zipCode" : { $type : "string" } } );
```

These queries return:

```
{ "_id" : 1, "address" : "2030 Martian Way", "zipCode" : "90698345" }
{ "_id" : 5, "address" : "104 Venus Drive", "zipCode" : [ "834847278", "1893289032" ] }
```

The following queries return all documents where `zipCode` is the BSON type `double` *or* is an array containing an element of the specified type:

```
db.addressBook.find( { "zipCode" : { $type : 1 } } )
db.addressBook.find( { "zipCode" : { $type : "double" } } )
```

These queries return:

```
{ "_id" : 2, "address" : "156 Lunar Place", "zipCode" : 43339374 }
```

The following query uses the `number` alias to return documents where `zipCode` is the BSON type `double`, `int`, or `long` *or* is an array containing an element of the specified types:

```
db.addressBook.find( { "zipCode" : { $type : "number" } } )
```

These queries return:

```
{ "_id" : 2, "address" : "156 Lunar Place", "zipCode" : 43339374 }
{ "_id" : 3, "address" : "2324 Pluto Place", "zipCode" : NumberLong(3921412) }
{ "_id" : 4, "address" : "55 Saturn Ring", "zipCode" : 88602117 }
```

Querying by Multiple Data Type

The `grades` collection contains names and averages, where `classAverage` has `string`, `int`, and `double` values:

```
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") }
   ]
)
```

The following queries return all documents where `classAverage` is the BSON type `string` or `double` *or* is an array containing an element of the specified types. The first query uses numeric aliases while the second query uses string aliases.

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

These queries return the following documents:

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

Querying by MinKey and MaxKey

The `restaurants` collection uses `minKey` for any grade that is a failing grade:

```
{
   "_id": 1,
   "address": {
      "building": "230",
      "coord": [ -73.996089, 40.675018 ],
      "street": "Huntington St",
      "zipcode": "11231"
   },
   "borough": "Brooklyn",
   "cuisine": "Bakery",
   "grades": [
      { "date": new Date(1393804800000), "grade": "C", "score": 15 },
      { "date": new Date(1378857600000), "grade": "C", "score": 16 },
      { "date": new Date(1358985600000), "grade": MinKey(), "score": 30 },
      { "date": new Date(1322006400000), "grade": "C", "score": 15 }
   ],
   "name": "Dirty Dan's Donuts",
   "restaurant_id": "30075445"
}
```

And `maxKey` for any grade that is the highest passing grade:

```
{
   "_id": 2,
   "address": {
      "building": "1166",
      "coord": [ -73.955184, 40.738589 ],
      "street": "Manhattan Ave",
      "zipcode": "11222"
   },
   "borough": "Brooklyn",
   "cuisine": "Bakery",
   "grades": [
      { "date": new Date(1393804800000), "grade": MaxKey(), "score": 2 },
      { "date": new Date(1378857600000), "grade": "B", "score": 6 },
      { "date": new Date(1358985600000), "grade": MaxKey(), "score": 3 },
      { "date": new Date(1322006400000), "grade": "B", "score": 5 }
   ],
   "name": "Dainty Daisey's Donuts",
   "restaurant_id": "30075449"
}
```

The following query returns any restaurant whose `grades.grade` field contains `minKey` *or* is an array containing an element of the specified type:

```
db.restaurants.find(
   { "grades.grade" : { $type : "minKey" } }
)
```

This returns

```
{
   "_id" : 1,
   "address" : {
      "building" : "230",
      "coord" : [ -73.996089, 40.675018 ],
      "street" : "Huntington St",
      "zipcode" : "11231"
   },
   "borough" : "Brooklyn",
   "cuisine" : "Bakery",
   "grades" : [
      { "date" : ISODate("2014-03-03T00:00:00Z"), "grade" : "C", "score" : 15 },
      { "date" : ISODate("2013-09-11T00:00:00Z"), "grade" : "C", "score" : 16 },
      { "date" : ISODate("2013-01-24T00:00:00Z"), "grade" : { "$minKey" : 1 }, "score" :
      { "date" : ISODate("2011-11-23T00:00:00Z"), "grade" : "C", "score" : 15 }
   ],
   "name" : "Dirty Dan's Donuts",
   "restaurant_id" : "30075445"
}
```

The following query returns any restaurant whose `grades.grade` field contains `maxKey` *or* is an array containing an element of the specified type:

```
db.restaurants.find(
   { "grades.grade" : { $type : "maxKey" } }
)
```

This returns

```
{
   "_id" : 2,
   "address" : {
      "building" : "1166",
      "coord" : [ -73.955184, 40.738589 ],
      "street" : "Manhattan Ave",
      "zipcode" : "11222"
   },
   "borough" : "Brooklyn",
   "cuisine" : "Bakery",
   "grades" : [
      { "date" : ISODate("2014-03-03T00:00:00Z"), "grade" : { "$maxKey" : 1 }, "score" :
      { "date" : ISODate("2013-09-11T00:00:00Z"), "grade" : "B", "score" : 6 },
      { "date" : ISODate("2013-01-24T00:00:00Z"), "grade" : { "$maxKey" : 1 }, "score" :
      { "date" : ISODate("2011-11-23T00:00:00Z"), "grade" : "B", "score" : 5 }
   ],
   "name" : "Dainty Daisey's Donuts",
   "restaurant_id" : "30075449"
}
```

Querying by Array Type

A collection named `SensorReading` contains the following documents:

```
{
   "_id": 1,
   "readings": [
      25,
      23,
      [ "Warn: High Temp!", 55 ],
      [ "ERROR: SYSTEM SHUTDOWN!", 66 ]
   ]
},
{
   "_id": 2,
   "readings": [
      25,
      25,
      24,
      23
   ]
},
{
   "_id": 3,
   "readings": [
      22,
      24,
      []
   ]
},
{
   "_id": 4,
   "readings": []
},
{
   "_id": 5,
   "readings": 24
}
```

The following query returns any document in which the `readings` field is an array, empty or non-empty.

```
db.SensorReading.find( { "readings" : { $type: "arr
```


The above query returns the following documents:

```
{
   "_id": 1,
   "readings": [
      25,
      23,
      [ "Warn: High Temp!", 55 ],
      [ "ERROR: SYSTEM SHUTDOWN!", 66 ]
   ]
},
{
   "_id": 2,
   "readings": [
      25,
      25,
      24,
      23
   ]
},
{
   "_id": 3,
   "readings": [
      22,
      24,
      []
   ]
},
{
   "_id": 4,
   "readings": []
}
```

In the documents with `_id : 1`, `_id : 2`, `_id : 3`, and `_id`

Additional Information

- Query for Null or Missing Fields
- `db.collection.find()`
- BSON Types.