

Chapter 3: Deeper Dive on the MongoDB Query Language

comparison query operators

extras

- <https://docs.mongodb.com/manual/reference/operator/query/>
- <https://docs.mongodb.com/manual/reference/operator/query-comparison/>
 - m001_comparisonoperators.zip

operators

- \$eq, \$ne : igual y no-igual
- \$gt, \$gte : mayor y mayor-igual
- \$lt, \$lte : menor y menor-igual
- \$in, \$nin : en array y no-en array

ejemplos

películas con **runtime** superior a 90 (minutos):

```
db.movieDetails.find({runtime: {$gt: 90}}, {_id: 0, title: 1, runtime: 1})
```

películas con **runtime** superior a 90 (minutos) e inferior a 120:

```
db.movieDetails.find({runtime: {$gt: 90, $lt: 120}}, {_id: 0, title: 1, runtime: 1})
```

películas con **runtime** mayor o igual a 180(minutos) y índice tomato igual a 100

```
db.movieDetails.find({runtime: {$gte: 180}, "tomato.meter": 100}, {_id: 0, title: 1, runtime: 1})
```

películas que **rated** diferente de «UNRATED» (incluso las que no tienen nada definido)

```
db.movieDetails.find({rated: {$ne: "UNRATED"}}, {_id: 0, title: 1, rated: 1})
```

películas que **rated** igual a los valores del array

```
db.movieDetails.find({rated: {$in: ["G", "PG"]}}, {_id: 0, title: 1, rated: 1})
```

Element Operators

extras

- <https://docs.mongodb.com/manual/reference/operator/query/type/>

- m001_element_operators.zip

operators

- `$exists` : verifica la existencia o no de un campo en un documento
- `$type` : podemos filtrar usando el tipo del campo (ver enlace anterior)
- `null` : pueden existir campos a NULL o no existir, ambos serán tratados igual

ejemplos

películas que tienen el campo **mpaaRating** (se podría hacer lo contrario cambiando a **false**)

```
db.moviesDetails.find({mpaaRating: {$exists: true}})
```

películas que no tienen el campo **mpaaRating** o que lo tienen = null

```
db.movieDetails.find({mpaaRating: null})
```

películas que tienen el campo **viewerRating** como un **int32**

```
db.movies.find({viewerRating: {$type: "int"}}).pretty()
```

logical operators

extras

- m001_logical_operators.zip

operators

- `$or`
- `$and`
- `$not`
- `$nor`

ejemplos

películas por dos campos (cualquier de ellos)

```
db.movieDetails.find({$or: [{"tomato.meter": {$gt: 95}}, {"metacritic": {$gt: 88}}]},
  {_id: 0, title: 1, "tomato.meter": 1, "metacritic": 1})
```

películas que cumplan los dos criterios a la vez

```
db.movieDetails.find({$and: [{"tomato.meter": {$gt: 95}}, {"metacritic": {$gt: 88}}]},
```

```
{_id: 0, title: 1, "tomato.meter": 1, "metacritic": 1})
```

de hecho, la instrucción anterior es equivalente a esta otra (por defecto se usa un AND en las búsquedas):

```
db.movieDetails.find({"tomato.meter": {$gt: 95},
  "metacritic": {$gt: 88}},
  {_id: 0, title: 1, "tomato.meter": 1, "metacritic": 1})
```

el uso del **\$and** tiene sentido cuando el campo es el mismo y ha de cumplir más de un criterio

```
db.movieDetails.find({$and: [{"metacritic": {$ne: null}},
  {"metacritic": {$exists: true}}]},
  {_id: 0, title: 1, "metacritic": 1})
```

array operators

extras

- m001_all_operator.zip
- m001_size_operator.zip
- m001_lemmatch_operator.zip

operators

- **\$all** : todos los valores indicados en el array han de estar en el campo para dar **true** y ser visualizados
- **\$size** : elementos de un campo del tipo array
- **\$elemMatch** : en el caso de campos del tipo **array** que contienen objetos, si realizamos una búsqueda por 2 criterios diferentes de alguna de las **key** de esos objetos y queremos que sean sobre el mismo objeto, debemos usar este operador. De lo contrario, find obviará que se tenga que buscar en el mismo objeto y cogerá cualquiera de las keys de objetos diferentes

ejemplos

películas que en el campo (Array) **genres** contengan «Comedy»,«Crime»,«Drama» (y otros, pero esos 3 obligatoriamente)

```
db.movieDetails.find({genres: {$all: ["Comedy", "Crime", "Drama"]}},
  {_id: 0, title: 1, genres: 1}).pretty()
```

Películas que tienen en el campo **countries** de tipo Array un solo elemento:

```
db.movieDetails.find({countries: {$size: 1}}).pretty()
```

trabajar con un documento en memoria:

```
mipelicula = db.movieDetails.findOne({title: "The Martian"}) // recupero el
documento en una variable
mipelicula // muestra el contenido de la variable
delete mipelicula._id // elimino el campo _id (para evitar duplicados al re-
insertarlo)
mipelicula.nuevoCampo = "mivalor" // creo un nuevo campo
```

```
db.movieDetails.insertOne(miPelicula); // inserto el documento (al no tener _id será uno nuevo respecto al anterior, con los cambios realizados
```

suponiendo que existe un campo llamado **boxOffice** que contiene un array de objetos tales:

```
boxOffice: [ { "country": "USA", "revenue": 228.4 },  
             { "country": "Australia", "revenue": 19.6 },  
             { "country": "UK", "revenue": 33.9 },  
             { "country": "Germany", "revenue": 16.2 },  
             { "country": "France", "revenue": 19.8 } ]
```

si hacemos una búsqueda de este tipo sin usar `$elemMatch` obtendremos un resultado erróneo. En este caso, buscamos que **country** sea «Germany» y que **revenue** sea mayor que 17 - cosa que no se cumple para el mismo objeto - pero si que se cumple para 2 keys de objetos diferentes (**country** igual a Germany existe y **revenue** mayor que 17 también - para todos aquellos que no sean «Germany», de hecho), así que nos devolverá el documento como si estuviese cumpliendo la condición cuando no es así:

```
db.movieDetails.find({"boxOffice.country": "Germany", "boxOffice.revenue": {$gt: 17}})
```

Si queremos buscar correctamente, se ha de usar `$elemMatch`. En este caso, este documento no será devuelto pq no satisface la condición impuesta:

```
db.movieDetails.find({boxOffice: {$elemMatch: {"country": "Germany", "revenue": {$gt: 17}}}})
```

regex operator

extras

operators

ejemplos

From: <https://miguelangel.torresegea.es/wiki/> - miguel angel torres egea

Permanent link: <https://miguelangel.torresegea.es/wiki/info: cursos: mongoduniversity: m001: cap3?rev=1544892075>

Last update: 15/12/2018 08:41

