MongoDB Query and Projection Operator

The MongoDB query operator includes comparison, logical, element, evaluation, Geospatial, array, bitwise, and comment operators.

MongoDB Comparison Operators

$eq

The $eq specifies the equality condition. It matches documents where the value of a field equals the specified value.

Syntax:

```
{ <field> : { $eq: <value> } }
```

Example:

```
db.books.find ( { price: { $eq: 300 } } )
```

The above example queries the books collection to select all documents where the value of the price filed equals 300.

$gt

The $gt chooses a document where the value of the field is greater than the specified value.

Syntax:

```
{ field: { $gt: value } }
```

Example:

```
db.books.find ( { price: { $gt: 200 } } )
```

$gte

The $gte choose the documents where the field value is greater than or equal to a specified value.

Syntax:

```
{ field: { $gte: value } }
```

Example:

```
db.books.find ( { price: { $gte: 250 } } )
```

$in

The $in operator choose the documents where the value of a field equals any value in the specified array.

Syntax:

```
{ filed: { $in: [ <value1>, <value2>, .....] } }
```

Example:

```
db.books.find( { price: { $in: [100, 200] } } )
```

$lt

The $lt operator chooses the documents where the value of the field is less than the specified value.

Syntax:

```
{ field: { $lt: value } }
```

Example:

```
db.books.find ( { price: { $lt: 20 } } )
```

$lte

The $lte operator chooses the documents where the field value is less than or equal to a specified value.

Syntax:

```
{ field: { $lte: value } }
```

Example:

```
db.books.find ( { price: { $lte: 250 } } )
```

$ne

The $ne operator chooses the documents where the field value is not equal to the specified value.

Syntax:

```
{ <field>: { $ne: <value> } }
```

Example:

```
db.books.find ( { price: { $ne: 500 } } )
```

$nin

The $nin operator chooses the documents where the field value is not in the specified array or does not exist.

Syntax:

```
{ field : { $nin: [ <value1>, <value2>, .... ] } }
```

Example:

```
db.books.find ( { price: { $nin: [ 50, 150, 200 ] } } )
```

MongoDB Logical Operator

$and

The $and operator works as a logical AND operation on an array. The array should be of one or more expressions and chooses the documents that satisfy all the expressions in the array.

Syntax:

```
{ $and: [ { <exp1> }, { <exp2> }, ....]}
```

Example:

```
db.books.find ( { $and: [ { price: { $ne: 500 } }, { price: { $exists: true } } ] } )
```

$not

The $not operator works as a logical NOT on the specified expression and chooses the documents that are not related to the expression.

Syntax:

```
{ field: { $not: { <operator-expression> } } }
```

Example:

```
db.books.find ( { price: { $not: { $gt: 200 } } } )
```

$nor

The $nor operator works as logical NOR on an array of one or more query expression and chooses the documents that fail all the query expression in the array.

Syntax:

```
{ $nor: [ { <expression1> } , { <expresion2> } , ..... ] }
```

Example:

```
db.books.find ( { $nor: [ { price: 200 }, { sale: true } ] } )
```

$or

It works as a logical OR operation on an array of two or more expressions and chooses documents that meet the expectation at least one of the expressions.

Syntax:

```
{ $or: [ { <exp_1> }, { <exp_2> }, ... , { <exp_n> } ] }
```

Example:

```
db.books.find ( { $or: [ { quantity: { $lt: 200 } }, { price: 500 } ] } )
```

MongoDB Element Operator

$exists

The exists operator matches the documents that contain the field when Boolean is true. It also matches the document where the field value is null.

Syntax:

```
{ field: { $exists: <boolean> } }
```

Example:

```
db.books.find ( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
```

$type

The type operator chooses documents where the value of the field is an instance of the specified BSON type.

Syntax:

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

Example:

```
db.books.find ( { "bookid" : { $type : 2 } } );
```

MongoDB Evaluation Operator

$expr

The expr operator allows the use of aggregation expressions within the query language.

Syntax:

```
{ $expr: { <expression> } }
```

Example:

```
db.store.find( { $expr: {$gt: [ "$product" , "price" ] } } )
```

$jsonSchema

It matches the documents that satisfy the specified JSON Schema.

Syntax:

```
{ $jsonSchema: <JSON schema object> }
```

$mod

The mod operator selects the document where the value of a field is divided by a divisor has the specified remainder.

Syntax:

```
{ field: { $mod: [ divisor, remainder ] } }
```

Example:

```
db.books.find ( { qty: { $mod: [ 200, 0] } } )
```

$regex

It provides regular expression abilities for pattern matching strings in queries. The MongoDB uses regular expressions that are compatible with Perl.

Syntax:

```
{ <field>: /pattern/<options> }
```

Example:

```
db.books.find( { price: { $regex: /789$/ } } )
```

$text

The $text operator searches a text on the content of the field, indexed with a text index.

Syntax:

```
        {
 $text:
  {
    $search: <string>,
    $language: <string>,
    $caseSensitive: <boolean>,
```

```
    $diacriticSensitive: <boolean>
  }
}
```

Example:

```
db.books.find( { $text: { $search: "Othelo" } } )
```

$where

The "where" operator is used for passing either a string containing a JavaScript expression or a full JavaScript function to the query system.

Example:

```
        db.books.find( { $where: function() {
  return (hex_md5(this.name)== "9b53e667f30cd329dca1ec9e6a8")
} } );
```

MongoDB Geospatial Operator

$geoIntersects

It selects only those documents whose geospatial data intersects with the given GeoJSON object.

Syntax:

```
{
  <location field>: {
    $geoIntersects: {
      $geometry: {
        type: "<object type>" ,
        coordinates: [ <coordinates> ]
      }
    }
  }
}
```

Example:

```
        db.places.find(
{
  loc: {
    $geoIntersects: {
      $geometry: {
        type: "Triangle" ,
        coordinates: [
          [ [ 0, 0 ], [ 3, 6 ], [ 6, 1 ] ]
```

```
        ]
      }
    }
  }
}
```

$geoWithin

The geoWithin operator chooses the document with geospatial data that exists entirely within a specified shape.

Syntax:

```
        {
<location field>: {
  $geoWithin: {
    $geometry: {
      type: <"Triangle" or "Rectangle"> ,
      coordinates: [ <coordinates> ]
    }
  }
}
```

$near

The near operator defines a point for which a geospatial query returns the documents from close to far.

Syntax:

```
       {
<location field>: {
 $near: {
   $geometry: {
     type: "Point" ,
     coordinates: [ <longitude> , <latitude> ]
   },
   $maxDistance: <distance in meters>,
   $minDistance: <distance in meters>
 }
}
```

Example:

```
          db.places.find(
{
 location:
   { $near :
     {
```

```
        $geometry: { type: "Point",  coordinates: [ -73.9667, 40.78 ] },
        $minDistance: 1000,
        $maxDistance: 5000
      }
   }
}
```

$nearSphere

The nearsphere operator specifies a point for which the geospatial query returns the document from nearest to farthest.

Syntax:

```
          {
  $nearSphere: [ <x>, <y> ],
  $minDistance: <distance in radians>,
  $maxDistance: <distance in radians>
}
```

Example:

```
             db.legacyPlaces.find(
   { location : { $nearSphere : [ -73.9667, 40.78 ], $maxDistance: 0.10 } }
)
```

$all

It chooses the document where the value of a field is an array that contains all the specified elements.

Syntax:

```
{ <field>: { $all: [ <value1> , <value2> ... ] } }
```

Example:

```
db.books.find( { tags: { $all: [ "Java", "MongoDB", "RDBMS" ] } } )
```

$elemMatch

The operator relates documents that contain an array field with at least one element that matches with all the given query criteria.

Syntax:

```
{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }
```

Example:

```

```

```
db.books.find(
   { results: { $elemMatch: { $gte: 500, $lt: 400 } } }
)
```

$size

It selects any array with the number of the element specified by the argument.

Syntax:

```
db.collection.find( { field: { $size: 2 } } );
```

MongoDB Bitwise Operator

$bitsAllClear

It matches the documents where all the bit positions given by the query are clear infield.

Syntax:

```
{ <field>: { $bitsAllClear: <numeric bitmask> } }
```

Example:

```
db.inventory.find( { a: { $bitsAllClear: [ 1, 5 ] } } )
```

$bitsAllSet

The bitallset operator matches the documents where all the bit positions given by the query are set in the field.

Syntax:

```
{ <field>: { $bitsAllSet: <numeric bitmask> } }
```

Example:

```
db.inventory.find( { a: { $bitsAllClear: [ 1, 5 ] } } )
```

$bitsAnyClear

The bitAnyClear operator matches the document where any bit of positions given by the query is clear in the field.

Syntax:

```
{ <field>: { $bitsAnyClear: <numeric bitmask> } }
```

Example:

```
db.inventory.find( { a: { $bitsAnyClear: [ 5, 10 ] } } )
```

$bitsAnySet

It matches the document where any of the bit positions given by the query are set in the field.

Syntax:

```
{ <field>: { $bitsAnySet: <numeric bitmask> } }
```

Example:

```
db.inventory.find( { a: { $bitsAnySet: [ 1, 5 ] } } )
```

MongoDB comments operator

$comment

The $comment operator associates a comment to any expression taking a query predicate.

Syntax:

```
db.inventory.find( { <query>, $comment: <comment> } )
```

Example:

```
        db.inventory.find(
{
  x: { $mod: [ 1, 0 ] },
  $comment: "Find Odd values."
}
```

MongoDB Projection Operator

$

The $ operator limits the contents of an array from the query results to contain only the first element matching the query document.

Syntax:

```
db.books.find( { <array>: <value> ... },
    { "<array>.$": 1 } )
db.books.find( { <array.field>: <value> ...},
    { "<array>.$": 1 } )
```

$elemMatch

The content of the array field made limited using this operator from the query result to contain only the first element matching the element $elemMatch condition.

Syntax:

```
db.library.find( { bookcode: "63109" },
{ students: { $elemMatch: { roll: 102 } } } )
```

$meta

The meta operator returns the result for each matching document where the metadata associated with the query.

Syntax:

```
{ $meta: <metaDataKeyword> }
```

Example:

```
        db.books.find(
<query>,
{ score: { $meta: "textScore" } }
```

$slice

It controls the number of values in an array that a query returns.

Syntax:

```
db.books.find( { field: value }, { array: {$slice: count } } );
```

Example:

```
db.books.find( {}, { comments: { $slice: [ 200, 100 ] } } )
```

← Prev Next →