MONGO DB : REPORT 5

SELECTORS->

$gt

Definition :

> `$gt` selects those documents where the value of the specified field is greater than (i.e. `>`) the specified value.
>
> For most data types, comparison operators only perform comparisons on fields where the BSON type matches the query value's type. MongoDB supports limited cross-BSON comparison through Type Bracketing.

```
test> db.stu.find({age:{$gt:20}}).count();
310
test>
```

AND OPERATOR->

This operator is used to perform logical AND operation on the array of one or more expressions and select or retrieve only those documents that match all the given expression. This operator is used in methods like find(), update(), etc,.

SYNTAX:

{ $and: [{ <expression1> }, { <expression2> } , ... , { <expressionN> }] }


```
{
        "std_name" : "Jack",
        "sex" : "Male",
        "class" : "VI",
        "age" : 11,
        "grd_point" : 33
},
{
        "std_name" : "Jenny",
        "sex" : "Female",
        "class" : "VI",
        "age" : 13,
        "grd_point" : 30
},
{
        "std_name" : "Thomas",
        "sex" : "Male",
        "class" : "V",
        "age" : 11,
        "grd_point" : 35.1257
},
```

```
        {
           "std_name" : "Lassy",
               "sex" : "Female",
               "class" : "X",
               "age" : 17,
               "grd_point" : 36.2514
        },
        {
               "std_name" : "Mia",
               "sex" : "Female",
               "class" : "X",
               "age" : 19,
               "grd_point" : 35.5201
        }
       {
               "std_name" : "Mike,
               "sex" : "Male",
               "class" : "V",
               "age" : 16,
               "grd_point" : 35.5201
        }
    ]
```


Matching two conditions values using $and operator:

In this example, we're only getting documents from students who meet these conditions:

1. "Class" of the student is VI.
2. "Sex" of the student is female.

```
>db.student.find({$and: [{"sex" : "Female"}, {"class" : "VI"}]}).pretty();
```

OUTPUT:

```
>db.student.find({$and: [{"sex" : "Female"}, {"class" : "VI"}]}).pretty();
{
        "_id" : ObjectId("5462sf85e4f31564d5243643df4g5"),
        "std_name" : "Jenny",
        "sex" : "Female",
        "class" : "VI",
        "age" : 13,
        "grd_point" : 30
}
```

OR OPERATOR->

There are two ways to do an OR query in MongoDB. “$in” can be used to query for a variety of values for a single key. “$or” is more general; it can be used to query for any of the given value across multiple keys.

```
test> db.stu.find({
...     $or:[
...         {is_hostel_resident:true},
...         {gpa:{$lt:3.0}}
...     ]
... }).count();
261
```

DOWNLOADING NEW DATASET:

- New Students Permission Data set
- LINK: https://drive.google.com/file/d/1SM6UZS5GHAeZXGP62u7nNKGM8T_EZfcg/view?usp=sharing

Explanation: Collection name: students_permission

- **name:** Student's name (string)
- **age:** Student's age (number)
- **permissions:** Bitmask representing user permissions (number)

BITWISE VALUE->

- In our example its a 32 bit each bit representing different things
- Bitwise value 7 means all access 7 -> 111

BIT 3	BIT 2	BIT 1
cafe	campus	lobby

BITWISE TYPES->

Bitwise

Name	Description
`$bitsAllClear`	Matches numeric or binary values in which a set of bit positions *all* have a value of `0`.
`$bitsAllSet`	Matches numeric or binary values in which a set of bit positions *all* have a value of `1`.
`$bitsAnyClear`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `0`.
`$bitsAnySet`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `1`.

QUERY->

```
test> db.student.find({
...       permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
...   });
[
  {
    _id: ObjectId('665f361536cc83d394ce92cd'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('665f361536cc83d394ce92ce'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('665f361536cc83d394ce92cf'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
```

GEOSPATIAL->

Geospatial Data

In MongoDB, you can store geospatial data as GeoJSON objects or as legacy coordinate pairs.

GeoJSON Objects

To calculate geometry over an Earth-like sphere, store your location data as GeoJSON objects.

To specify GeoJSON data, use an embedded document with:

- a field named `type` that specifies the GeoJSON object type, and
- a field named `coordinates` that specifies the object's coordinates.

- Create collections called locations
- Uploading this below Dataset:

```
[
    {
        "_id": 1,
        "name": "Coffee Shop A",
        "location": {
            "type": "Point",
            "coordinates": [
                -73.985,
                40.748
            ]
        }
    },
    {
        "_id": 2,
        "name": "Restaurant B",
        "location": {
            "type": "Point",
            "coordinates": [
                -74.009,
                40.712
            ]
        }
    },
    {
        "_id": 3,
        "name": "Library C",
        "location": {
            "type": "Point",
            "coordinates": [
                -77.036,
                38.907
            ]
        }
    },
    {
        "_id": 4,
        "name": "Museum D",
        "location": {
            "type": "Point",
            "coordinates": [
                -80.843,
                34.26
            ]
        }
    },
    {
        "_id": 5,
        "name": "Park E",
        "location": {
            "type": "Point",
            "coordinates": [
                -74.006,
                40.705
            ]
        }
    }
]
```

GEOSPATIAL QUERY->

<field>: { type: <GeoJSON type> , coordinates: <coordinates> }

Example:

```
test> db.location.find({
...       location:{
...         $geoWithin:{
...           $centerSphere:[[-74.005,40.712],0.00621376]
...       }
...      }
... });
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
test>
```

DATA TYPES & OPERATIONS->

Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The `2dsphere` and `2d` indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.