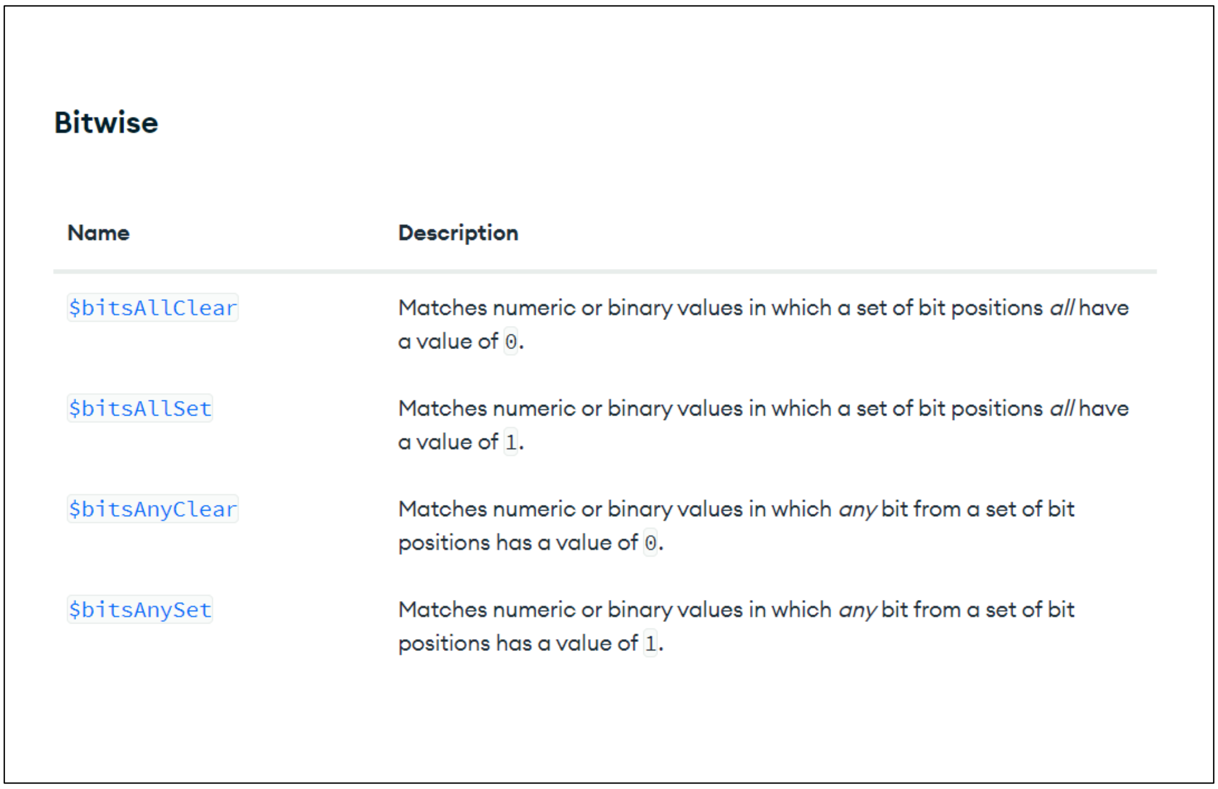
Bitwise Types



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`.

Bitwise value:

- In our example it’s 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

Query

A **MongoDB query** is a fundamental aspect of MongoDB that allows users to fetch data from the database.

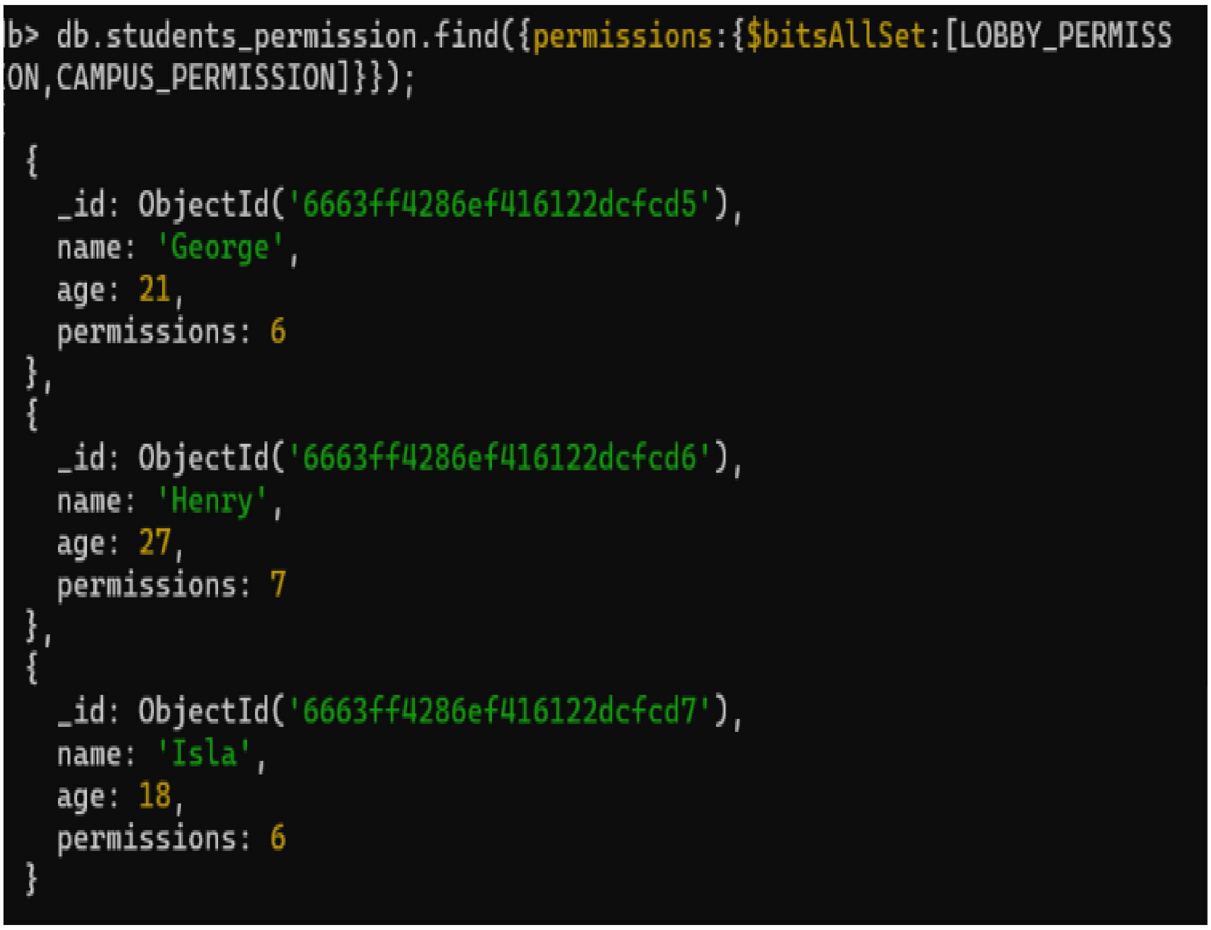
MongoDB provides various query operators for complex queries:

- Comparison Operators ($eq, $gt, $lt, etc.)
- Logical Operators ($and, $or, $not, $nor)
- Element Operators ($exists, $type)
- Array Operators ($in, $all, $elemMatch)

```
const LOBBY_PERMISSION=1;
const CAMPUS-PERMISSION=2;
```

To find the students permission with the both lobby and campus we use the permission like,

```
db.students_permission.find({permissions:{$bitsAllSet:[LOBBY_PER MISSION, CAMPUS-PERMISSION]}});
```

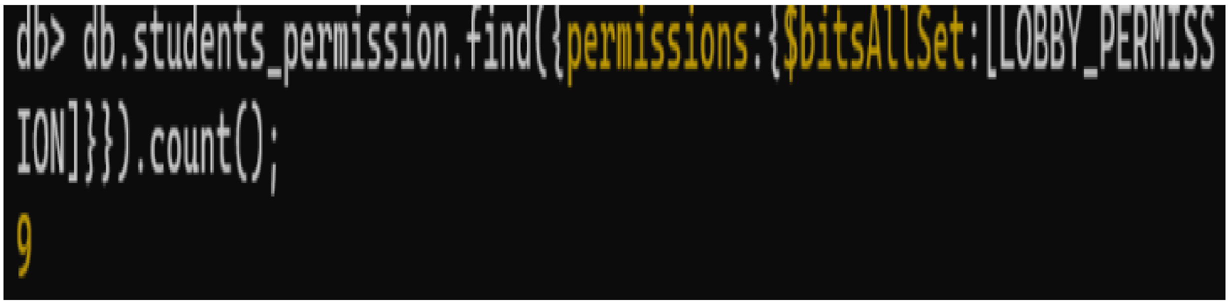


```
b> db.students_permission.find({permissions:{$bitsAllSet:[LOBBY_PERMISS
ON,CAMPUS_PERMISSION]}});

 {
   _id: ObjectId('6663ff4286ef416122dcfcd5'),
   name: 'George',
   age: 21,
   permissions: 6
 },
 {
   _id: ObjectId('6663ff4286ef416122dcfcd6'),
   name: 'Henry',
   age: 27,
   permissions: 7
 },
 {
   _id: ObjectId('6663ff4286ef416122dcfcd7'),
   name: 'Isla',
   age: 18,
   permissions: 6
 }
```

To find the total number students or total count of students permissions with the both lobby and campus we use the permission like,

db.students_permission.find({permissions:{$bitsAllSet:[LOBBY_PER MISSION, CAMPUS-PERMISSION]}}).count();



```
db> db.students_permission.find({permissions:{$bitsAllSet:[LOBBY_PERMISS
ION]}}).count();
9
```

The above example shows the total count of students.

Geospatial Query:

MongoDB supports geospatial queries for location-based data.

To perform geospatial queries you can create a 2dsphere index on the desired field, such as "location".

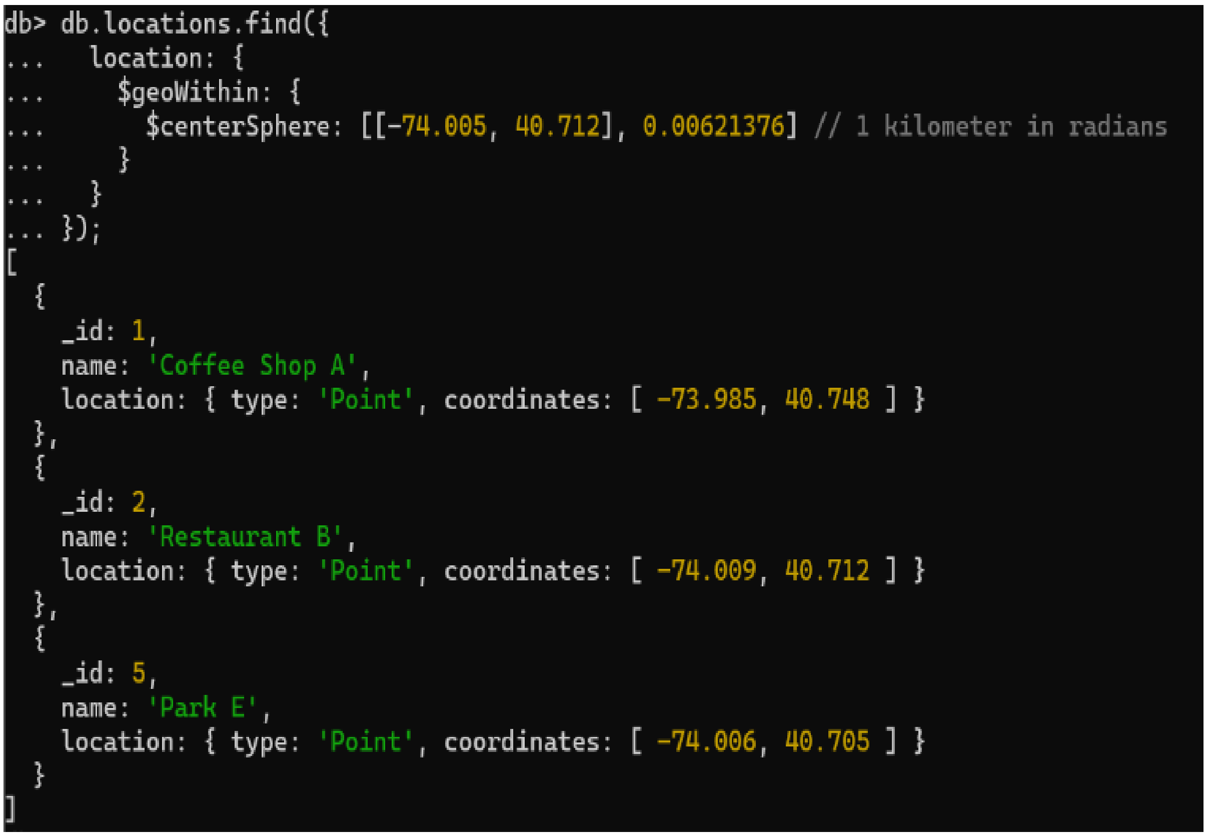
Below the example :

- We querying a collection named locations using the find() method.
- Then query is based on a geospatial filter – "$geoWithin".

- The center of the search area is specified as longitude -74.005 and latitude 40.712. The radius is 0.0062137.
- The query will return documents where the location falls within a circular area centered at the specified coordinates with the given radius.

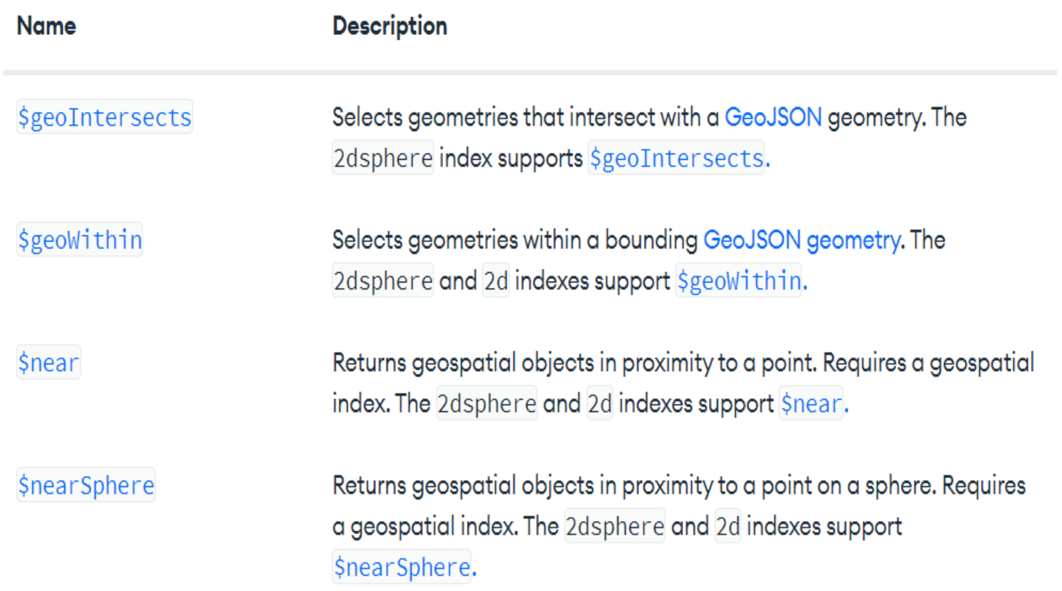
```
db.locations.find({
location: {
  SgeoWithin: {
     $centerSphere: [[-74.005, 40.712], 0.00621376]
}
}
});
```

Output:



```
db> db.locations.find({
...   location: {
...     $geoWithin: {
...       $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in radians
...     }
...   }
... });
[
  {
    _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 ] }
  }
]
```

Datatypes and 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`.