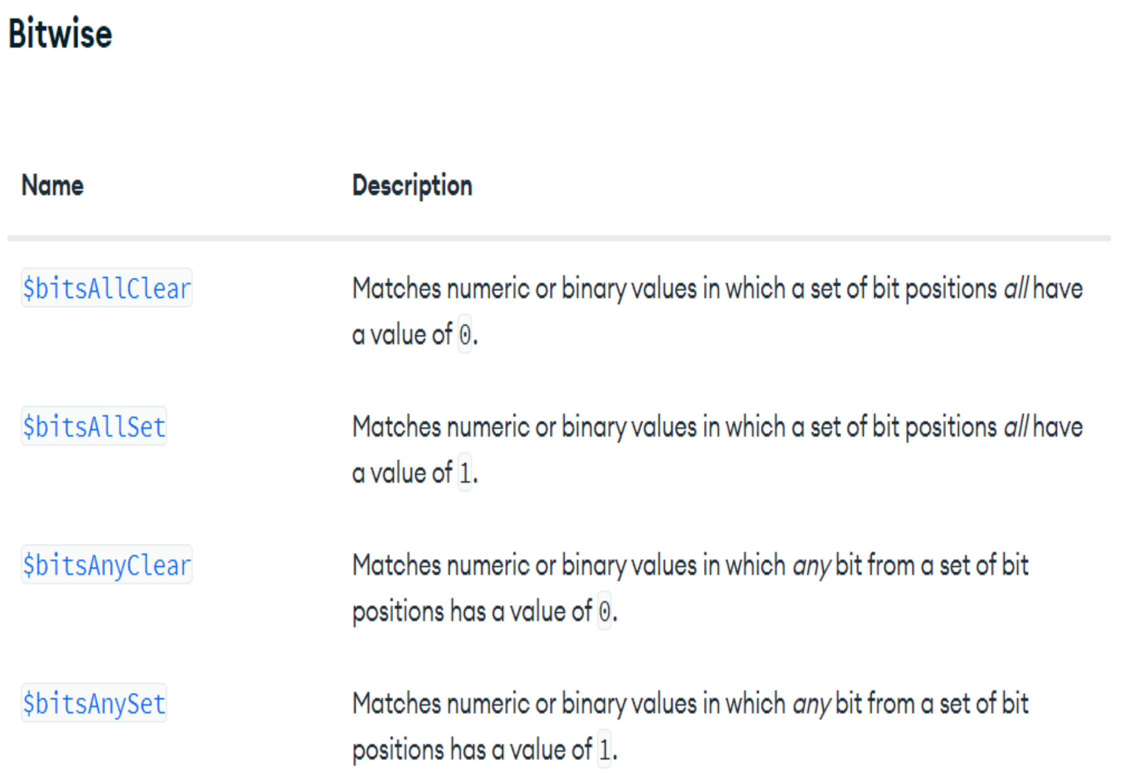
BITWISE VALUE:

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

Bit 3	Bit 4	Bit 5
cafe	campus	lobby

BITWISE TYPES:

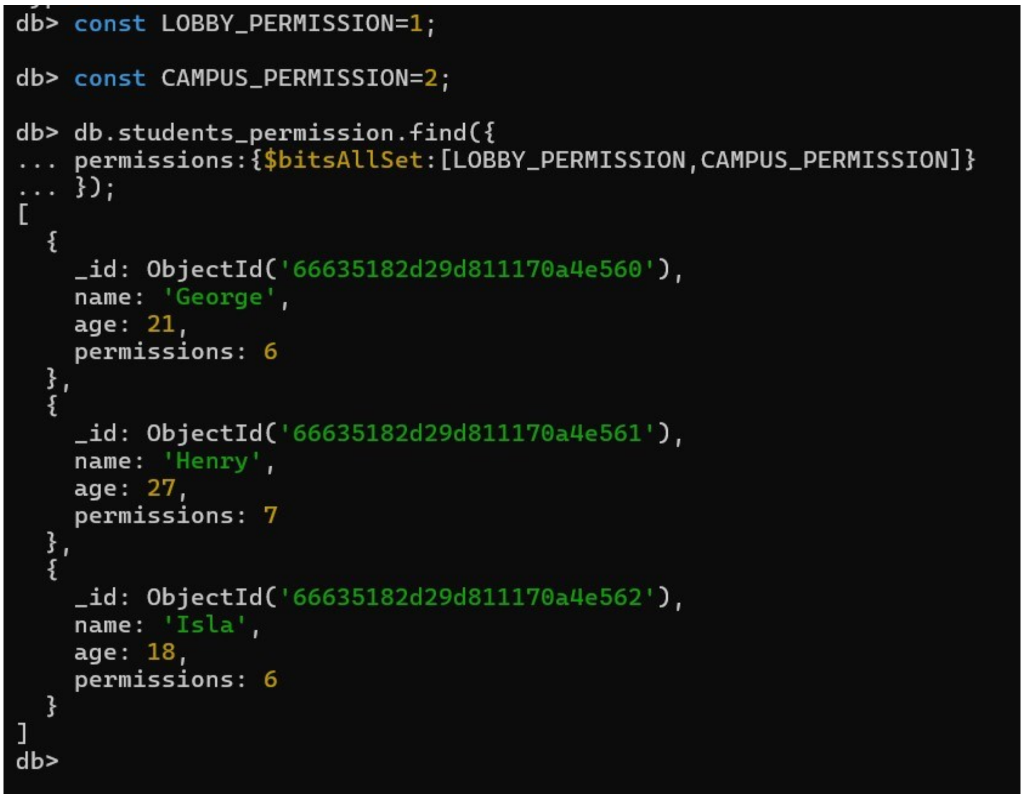


Bitwise

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:

In MongoDB, a query is a command used to retrieve documents from a collection that match certain criteria. Queries are written in MongoDB's query language and can include conditions, projections, and other modifiers to specify exactly which documents to retrieve. MongoDB queries can be complex, allowing for precise filtering, sorting, and aggregation of data.



```
db> const LOBBY_PERMISSION=1;

db> const CAMPUS_PERMISSION=2;

db> db.students_permission.find({
... permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
... });
[
  {
    _id: ObjectId('66635182d29d811170a4e560'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('66635182d29d811170a4e561'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('66635182d29d811170a4e562'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
db>
```

1.Constant Defination:

```
const LOBBY_PERMISSION = 1;
const CAMPUS_PERMISSION = 2;
```

Two constants are defined: LOBBY_PERMISSION with a value of 1 and CAMPUS_PERMISSION with a value of 2. These constants are used to represent specific permissions.

2.Query Execution:

```
 db.students_permission.find({
permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
 });
```

This line runs a query on the `students_permission` collection in the database. The query uses the `$bitsAllSet` operator to find documents where both the `LOBBY_PERMISSION` and `CAMPUS_PERMISSION` bits are set in the `permissions` field.

Explanation of $bitsAllSet Operator:

- The $bitsAllSet operator checks if all of the specified bit positions in a binary representation of a number are set to 1.

- In this example, LOBBY_PERMISSION (1) and CAMPUS_PERMISSION (2) correspond to bit positions 0 and 1, respectively.

- The permissions field value is checked to ensure both of these bits are set.

Sample Output:

```
[
 {
  _id: ObjectId('66635182d29d811170a4e560'),
  name: 'George',
  age: 21,
  permissions: 6
 },
 {
  _id: ObjectId('66635182d29d811170a4e561'),
  name: 'Henry',
  age: 27,
  permissions: 7
 },
 {
  _id: ObjectId('66635182d29d811170a4e562'),
  name: 'Isla',
  age: 18,
  permissions: 6
 }
]
```

The result of the query is an array of documents (JSON objects). Each document represents a student with the following fields:

- `_id`: A unique identifier for the document.
- `name`: The student's name.
- `age`: The student's age.

- `permissions`: A numeric value representing the permissions assigned to the student.

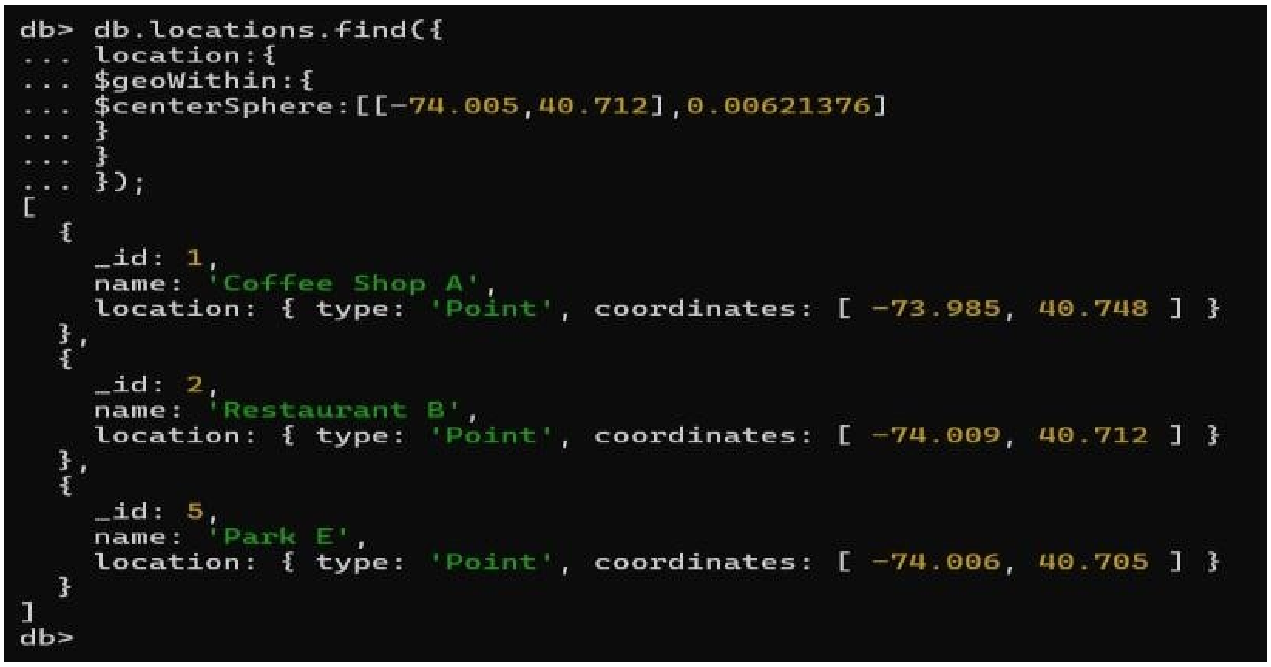
GEOSPATIAL:

In MongoDB, geospatial refers to the capability to store and query data based on its geographic location. MongoDB supports various geospatial queries, including finding points within a specified distance of a location, finding objects within a specified polygon, and finding the nearest objects to a location. Geospatial indexes can be created to efficiently perform these types of queries on geospatial data.

```
_id : 1
name : “Coffee Shop A”
location : Object
    type : “Point”
    coordinates : Array(2)
```

GEOSPATIAL QUERY:

In MongoDB, a geospatial query is used to retrieve documents based on their geographical location. These queries utilize special operators like $geoNear, $geoWithin, and $near to find



```
db> db.locations.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 ] }
  }
]
db>
```

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

This query is used to find documents in the locations collection that are within a specified radius of a given point. Let's break down each part of the query:

db.locations.find(...):

This part of the command indicates that we are performing a find query on the locations collection.

location: { $geoWithin: { $centerSphere: [[-74.005, 40.712], 0.00621376]}:

The query is looking for documents where the location field contains geospatial data that is within a certain radius of a center point.

$geoWithin: This operator selects documents with geospatial data within a specified geometry.

$centerSphere: This operator specifies a circle for a geospatial query. The circle is defined by a center point and a radius measured in radians.

- **`[[-74.005, 40.712], 0.00621376]`**: This array specifies the center of the circle and its radius.
- **`[-74.005, 40.712]`**: This array specifies the coordinates of the center point (longitude, latitude).
- **`0.00621376`**: This value specifies the radius of the circle in radians. The radius is calculated based on the Earth's radius in the specified unit (usually miles or kilometers). In this case, it represents a distance of approximately 25 miles (0.00621376 radians).