**Let’s take a new data set**

- New Students Permission dataset link

## Explanation: Collection name: students_permission

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

## Bit wise:

bitwise query operators allow you to perform bitwise operations on numeric fields and match documents based on the result.

- 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 |

**Bit wise type:**

The bitwise operators can be used with different types of numeric fields, primarily integers. When performing bitwise operations, it's important to ensure that the field values and bitmasks are integers.

## Bitwise Operators in MongoDB

The bitwise operators in MongoDB include:

1. `$bitsAllClear`: Matches documents where all the specified bit positions are clear .
2. `$bitsAllSet`: Matches documents where all the specified bit positions are set .
3. `$bitsAnyClear`: Matches documents where any of the specified bit positions are clear .

4. `$bitsAnySet`: Matches documents where any of the specified bit positions are set .

**Query:**

MongoDB queries using bitwise operators, you need to understand how these operators work and how to structure your queries properly.

Ex: const  LOBBY_PERMISSION=1;

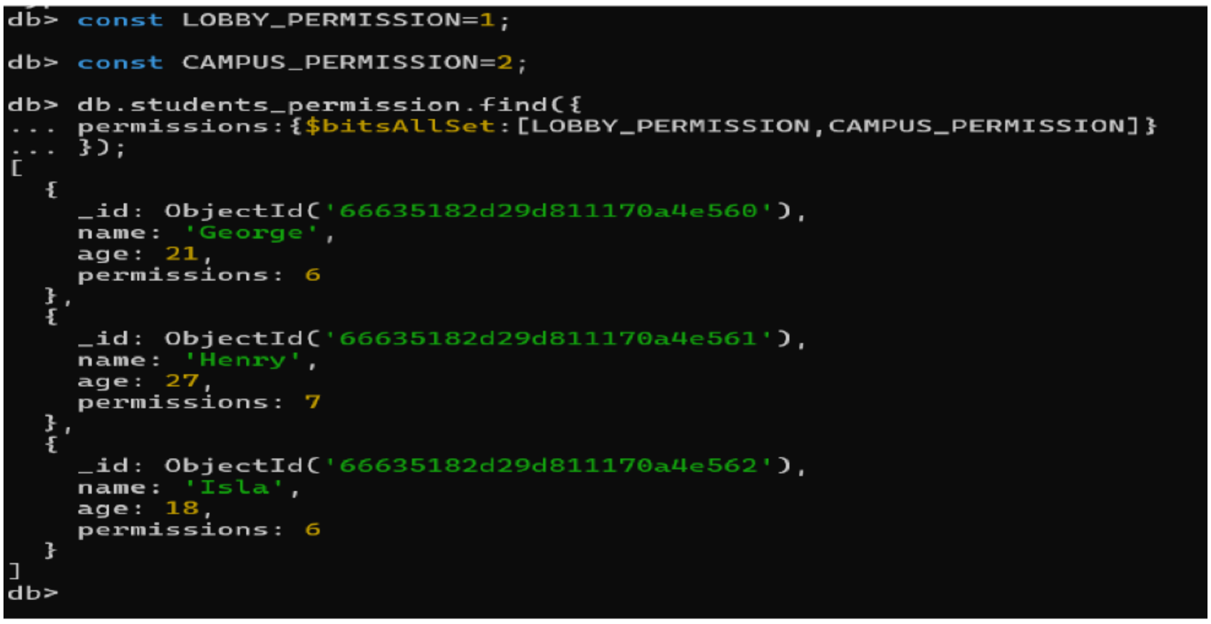
const CAMPUS_PERMISSION =2;

db .students_ permission. find({permissions:{

$bitsAllSet:

[ LOBBY_PERMISSION,CAMPUS_PERMISSION]}});

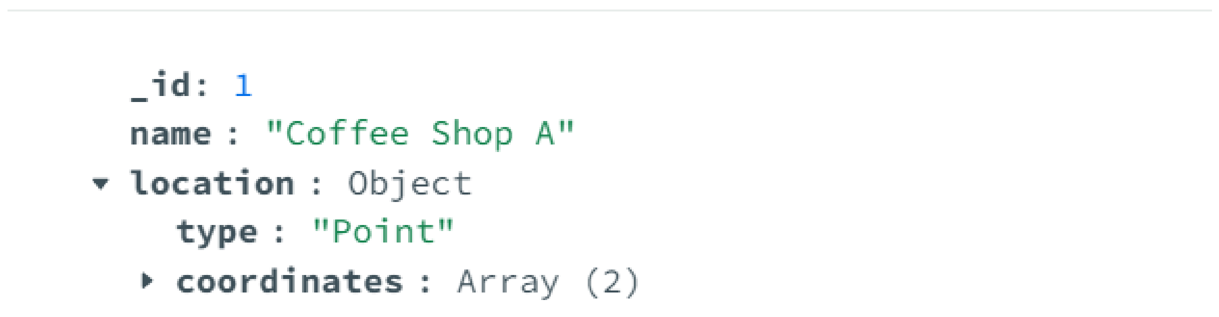
Output:



```
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>
```

**Geosptial:**

- Official Documentation link
- Create collection called “locations”
- Upload the dataset using json link

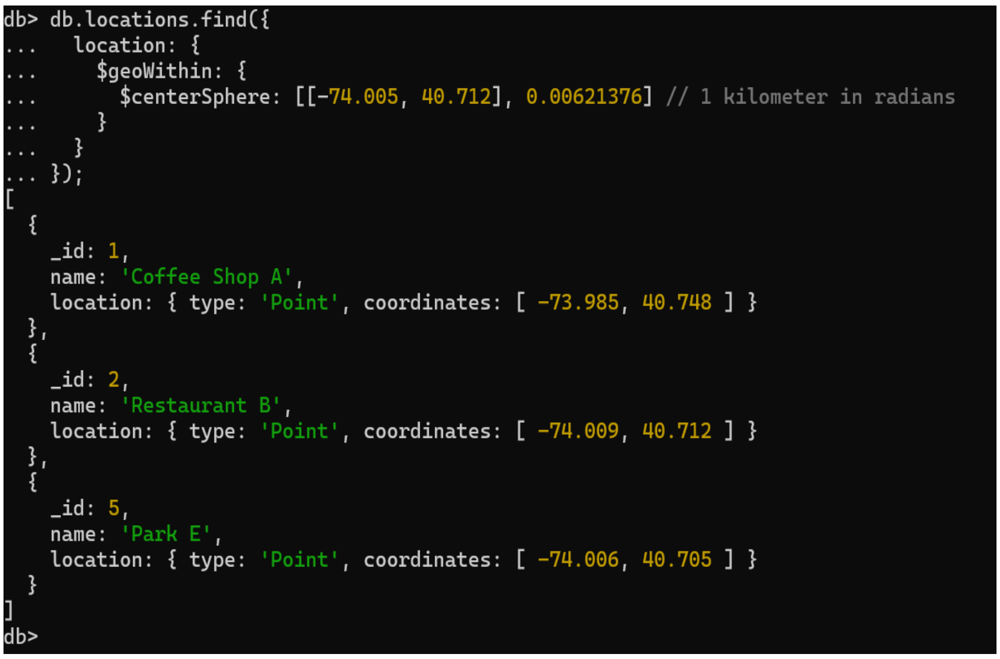


```
_id: 1
name : "Coffee Shop A"
▾ location : Object
    type : "Point"
  ▸ coordinates : Array (2)
```

**Geospatial Query:**
Geospatial queries in MongoDB leverage geospatial indexes to efficiently execute spatial queries. MongoDB supports two types of geospatial indexes: `2d` indexes for planar (Euclidean) data and `2dsphere` indexes for spherical data.

```
db.  location. find({

location:{

$geoWithin:{

$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 ] }
  }
]
db>
```

MongoDB query it is uesd  to find data within a certain radius using the $geoWithin operator and $centerSphere. This particular query specifies a radius in radians.

## Data types and operation

**Data type**

Point

Line string

polygon

### *1. Point*

A `Point` represents a single geographic coordinate.

- **Structure**:

```
json
Copy code
{
  "type": "Point",
  "coordinates": [longitude, latitude]
}
```

### 2.LineString

A LineString represents a series of connected geographic coordinates, forming a line.

**Structure**:

```
json
Copy code
{
  "type": "LineString",
  "coordinates": [
    [longitude1, latitude1],
    [longitude2, latitude2],
    ...
  ]
}
```

### 3. Polygon

A Polygon represents an area enclosed by a series of linear rings. A polygon must have at least one ring, and the first ring is the outer boundary, while any subsequent rings are holes.

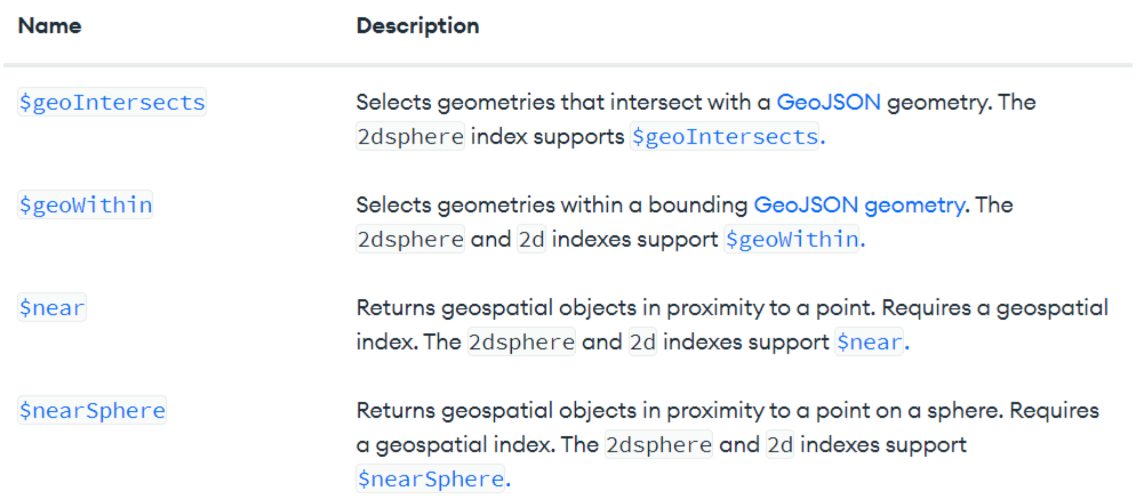
- **Structure**:

  json

  Copy code

  ```
  {
    "type": "Polygon",
    "coordinates": [
      [ [longitude1, latitude1], [longitude2, latitude2], ..., [longitude1, latitude1] ],
      [ [longitudeA, latitudeA], [longitudeB, latitudeB], ..., [longitudeA, latitudeA] ]  //
  optional inner ring (hole)
    ]
  }
  ```

## Data types 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`. |