Let's Take new Data set :

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 4	Bit 5
cafe	campus	lobby

Bitwise Types:

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`.

In MongoDB, a "query" is a set of criteria used to retrieve documents from a collection. The criteria specify the conditions that the documents must meet to be selected. A "selector" is a part of the query that defines these conditions.

A query is composed of field-value pairs where the field is the name of a document field and the value specifies the criteria that the field must meet. MongoDB provides a rich set of query operators to define these criteria.

```
db.users.find({
  $and: [
    { age: { $gte: 18 } },
    { age: { $lt: 30 } },
    { permissions: { $bitsAnySet: 1 } }, // Matches if the first bit is set
    { name: { $regex: /^J/ } }
  ]
})
```

Geospatial query:

MongoDB supports geospatial queries, which allow you to perform location-based searches and calculations on geospatial data stored in your MongoDB collections. To use geospatial queries, you typically need to store location data in a specific format and create appropriate geospatial indexes.

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

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>
```

Data types and Operations :

MongoDB supports various data types and provides a wide range of operations to manipulate and query these data types. Here's a comprehensive guide on the data types and common operations in MongoDB:

- DataType
- Point
- Line String
- Polygon

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`.

Datatype :

MongoDB stores data in a format called BSON (Binary JSON). BSON supports a variety of data types that allow you to store different kinds of information in your documents. Here's a rundown of some common MongoDB data types:

- **String:** The most frequently used data type, for storing text data. BSON strings must be valid UTF-8 format.
- **Integer:** Stores whole numbers. Can be either 32-bit or 64-bit depending on your server configuration.
- **Double:** Stores floating-point numbers (numbers with decimals).
- **Boolean:** Stores true or false values.
- **Array:** Used to store lists of multiple values under a single key.
- **Object:** Represents embedded documents within a document.
- **Date:** Stores dates and times in Unix timestamp format (milliseconds since epoch). You can also create custom Date objects.
- **ObjectId:** A unique identifier automatically generated by MongoDB for each document.
- **Binary Data:** Stores binary data like images or files.
- **Null:** Represents missing or empty values.

Point:

MongoDB doesn't have a specific "point" data type itself. However, it excels at storing geospatial data using GeoJSON objects.

Here's how you can represent a point location in MongoDB:

GeoJSON Point: This is the recommended way to store a single location point. It's an embedded document with two fields:

- type: Set to "Point" to indicate the geometry type.
- coordinates: An array containing the longitude (first) and latitude (second) of the point.

Line String:

MongoDB utilizes GeoJSON objects for storing geospatial data, and within that context, it uses LineStrings to represent linear features.

A LineString in MongoDB is an ordered sequence of connected points forming a line. It's part of the GeoJSON specification for representing geometric shapes.

Here's how LineStrings work in MongoDB:

Structure: A LineString is an embedded document within a field in your document. It typically has two properties:

- type: Set to "LineString" to indicate the geometry type.
- coordinates: An array of longitude/latitude pairs representing the connected points along the line.

Polygon:

In MongoDB, polygons are represented using GeoJSON objects, specifically the Polygon type. A polygon defines a closed area with straight sides.

Here's how polygons work in MongoDB:

Structure: Similar to LineStrings, a Polygon is an embedded document within a field in your document. It has two key properties:

- type: Set to "Polygon" to indicate the geometry type.
- coordinates: An array where the first element is a closed LineString (outer boundary) defining the polygon's exterior. There can be optional additional elements representing interior holes (islands) within the polygon.