MONGO DB

CLASS 4 & 5: DATATYPES, PROJECTIONS OPERATORS.

DATA TYPES:

In MongoDB, geospatial data can be represented using Geo JSON, a standardized format for encoding geographic features. Here's an explanation of the three data types you mentioned for geospatial data in MongoDB:

1. POINT:

Represents a single geographic location with a longitude and latitude.

Geo JSON Example:

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

Example Usage: Storing the location of a store, a sensor, or a user's current location.

2.LINE STRING:

Represents a sequence of connected coordinate pairs, defining a linear path.

Geo JSON Example:

```
{
 "type": "Line String",
"coordinates": [
[longitude1, latitude1], [longitude2, latitude2],
```

```
... // Additional coordinate pairs for the path
]
}
```

Example Usage: Storing the route of a delivery truck, the outline of a river, or a hiking trail.

3.POLYGON:

Represents a closed loop defined by an array of coordinate pairs, outlining an area on the map.

The first and last coordinate pairs should be the same to close the loop.

Geo JSON Example:

```
{
"type": "Polygon",
"coordinates": [
[ // Array of coordinate pairs for each vertex of the polygon longitude1, latitude1 ],
[ ... ],
[ longitude N, latitude N ] // Closing vertex (same as the first one)
]
}
```

Example Usage: Storing the boundary of a city, a national park, or a building footprint.

PROJECTION:

In MongoDB, projection refers to the process of specifying which fields should be included or excluded in the documents that are returned by a query. This is done to limit the amount of data that is retrieved.

Benefits of Using Projections:

Reduced Network Traffic: By selecting only the required fields, you minimize the data transferred between the database server and your application. This is especially beneficial for large documents or queries that return many documents.

Improved Performance: Retrieving a smaller subset of data leads to faster query execution times.

Enhanced Data Privacy: You can control what data is exposed in the results, potentially limiting sensitive information returned in certain queries.

Common Projection Options:

Include Fields (Positive Values): Set the value to 1 to include the specified field in the results.

Exclude Fields (Negative Values): Set the value to 0 to exclude the specific field from the results.

Include Subset of Fields in Embedded Documents: Use dot notation to target specific fields within embedded documents.

Exclude Specific Fields from Embedded Documents: Combine dot notation with 0 to exclude specific fields within embedded documents.

When performing a query in MongoDB, we can use projection to:

1. INCLUDE SPECIFIC FIELDS: It specify which fields want to be included in the result set.

2. EXCLUDE SPECIFIC FIELDS: It specify which fields want to be excluded from the result set.

The most common projection operators in MongoDB are:

1. Inclusion (1): This operator is used to include specific fields in the query results.

2. Exclusion (0): This operator is used to exclude specific fields from the query results.

3. Slice ($slice): This operator limits the number of array elements that are returned.

4. ElemMatch ($elemMatch): This operator projects only the first element from an array that matches the specified condition.

5. Meta ($meta): This operator can include metadata in the query results, such as text search scores.

1.Include fields (1):

To include age and name of candidates without _id we use

db.students.find({},{_id:0,name:1,age:1});

```
]
db> db.candidates.find({},{_id:0,name:1,age:1});
[
  { name: 'Alice Smith', age: 20 },
  { name: 'Bob Johnson', age: 22 },
  { name: 'Charlie Lee', age: 19 },
  { name: 'Emily Jones', age: 21 },
  { name: 'David Williams', age: 23 },
  { name: 'Fatima Brown', age: 18 },
  { name: 'Gabriel Miller', age: 24 },
  { name: 'Hannah Garcia', age: 20 },
  { name: 'Isaac Clark', age: 22 },
  { name: 'Jessica Moore', age: 19 },
  { name: 'Kevin Lewis', age: 21 },
  { name: 'Lily Robinson', age: 23 }
]
```

2.Exclude fields (0):

Here to retrieve data of candidates excluding _id and course details we use

db.candidates.find({},{_id:0,name:1});

```
db> db.candidates.find({},{_id:0,courses:0});
[
  {
    name: 'Alice Smith',
    age: 20,
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    name: 'Bob Johnson',
    age: 22,
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Charlie Lee',
    age: 19,
    gpa: 3.2,
    home_city: 'Chicago',
    blood_group: 'B+',
```

Here to retrieve including only name of candidates excluding _id we use

db.candidates.find({},{_id:0,name:1});

```
db> db.candidates.find({},{_id:0,name:1});
[
  { name: 'Alice Smith' },
  { name: 'Bob Johnson' },
  { name: 'Charlie Lee' },
  { name: 'Emily Jones' },
  { name: 'David Williams' },
  { name: 'Fatima Brown' },
  { name: 'Gabriel Miller' },
  { name: 'Hannah Garcia' },
  { name: 'Isaac Clark' },
  { name: 'Jessica Moore' },
  { name: 'Kevin Lewis' },
  { name: 'Lily Robinson' }
]
```

3.Elem operator($elemMatch):

In MongoDB, the $elemMatch operator is a valuable tool for filtering documents based on specific criteria within an array field. It allows you to target and match elements (documents within an array) that meet the specified conditions.

Applications of $elemMatch:

Filtering Arrays: Find documents where at least one element in an array field satisfies your filtering criteria.

Complex Data Relationships: Navigate and query data within nested arrays or embedded document structures.

Syntax:

```
{
 array_field: {$elemMatch: {matching_criteria }
}
```

Example:

here to find candidates who are enrolled in “Computer Science”with specific projection we use command **db.candidates.find({courses:{$elemMatch:{$eq:”ComputerScience”}}},{name:1,”courses.$”:1});**

```
db> db.candidates.find({courses:{$elemMatch:{$eq:"Computer Science"}}},{name:1,"courses.$":1})
;
[
  {
    _id: ObjectId('668330b9acc6041a15186d2a'),
    name: 'Bob Johnson',
    courses: [ 'Computer Science' ]
  },
  {
    _id: ObjectId('668330b9acc6041a15186d2f'),
    name: 'Gabriel Miller',
    courses: [ 'Computer Science' ]
  },
  {
    _id: ObjectId('668330b9acc6041a15186d33'),

    courses: [ 'Computer Science' ]
  }
]
```

To find count candidates who are enrolled in “Computer Science”with specific projection we use.

db.candidates.find({courses:{$elemMatch:{$eq:”ComputerScience”}}},{name:1,” courses.$”:1}).count()

```
db> db.candidates.find({courses:{$elemMatch:{$eq:"Computer Science"}}},{name:1,"courses.$":1})
.count();
3
```

Key Points:

$elemMatch ensures at least one element in the array meets the specified conditions.

You can combine multiple conditions within the **matching_criteria** document using logical operators like **$and** and **$or** for more complex filtering.

$elemMatch is particularly useful for querying nested data structures or arrays of embedded documents.

4.Slice operator($slice):

The $slice projection operator in MongoDB is a versatile tool used to limit the number of elements returned from an array field in your query results. It allows you to specify how many elements to include, starting from a particular position within the array.

Applications of $slice:

Pagination: Retrieve a specific page of results from a large array field, ideal for implementing features like pagination in your application.

Limiting Results: Control how many elements are returned from an array, reducing the amount of data transferred and potentially improving performance.

Focusing on Specific Elements: Target and return a particular subset of elements from within an array based on their position.

Syntax:

```
{
```

array_field: { $slice: [skip, limit] }

}

Example:

To retrieve all candidates with first two courses with name we use

db.candidates.find({},{name:1,courses:{$slice:2}});

```
db> db.candidates.find({},{name:1,courses:{$slice:2}});
[
  {
    _id: ObjectId('668330b9acc6041a15186d29'),
    name: 'Alice Smith',
    courses: [ 'English', 'Biology' ]
  },
  {
    _id: ObjectId('668330b9acc6041a15186d2a'),
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics' ]
  },
  {
    _id: ObjectId('668330b9acc6041a15186d2b'),
    name: 'Charlie Lee',
    courses: [ 'History', 'English' ]
  },
  {
    _id: ObjectId('668330b9acc6041a15186d2c'),
    name: 'Emily Jones',
    courses: [ 'Mathematics', 'Physics' ]
  },
  {
    _id: ObjectId('668330b9acc6041a15186d2d'),
    name: 'David Williams',
    courses: [ 'English', 'Literature' ]
```

One more without an _id to retrieve all candidates with first two courses we use

db.candidates.find({},_id:0,name:1,courses:{$slice:2}});

```
db> db.candidates.find({},{_id:0,name:1,courses:{$slice:2}});
[
  { name: 'Alice Smith', courses: [ 'English', 'Biology' ] },
  {
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics' ]
  },
  { name: 'Charlie Lee', courses: [ 'History', 'English' ] },
  { name: 'Emily Jones', courses: [ 'Mathematics', 'Physics' ] },
  { name: 'David Williams', courses: [ 'English', 'Literature' ] },
  { name: 'Fatima Brown', courses: [ 'Biology', 'Chemistry' ] },
  {
    name: 'Gabriel Miller',
    courses: [ 'Computer Science', 'Engineering' ]
  },
  {
    name: 'Hannah Garcia',
    courses: [ 'History', 'Political Science' ]
  },
  { name: 'Isaac Clark', courses: [ 'English', 'Creative Writing' ] },
```

To find count of candidates we use

db.candidates.find({},_id:0,name:1,courses:{$slice:2}});

```
db> db.candidates.find({},{_id:0,name:1,courses:{$slice:2}}).count();
12
```