NIELIT Gorakhpur

Course name:A level **SUBJECT:DATABASE TECHNOLOGIES**

Topic:MongoDB **DATE:1/5/2020**

Data types in MongoDB

MongoDB supports many datatypes. Some of them are −

- **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.
- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This datatype is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This datatype is used to store the document's ID.
- **Binary data** − This datatype is used to store binary data.
- **Code** − This datatype is used to store JavaScript code into the document.
- **Regular expression** − This datatype is used to store regular expression.

Embedded documents

Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modeled via **Embedded** and **Referenced** approaches. Such relationships can be either 1:1, 1:N, N:1 or N:N.

Let us consider the case of storing addresses for users. So, one user can have multiple addresses making this a 1:N relationship.

Following is the sample document structure of **user** document −

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
```

```
   "name": "Anita kumari",
   "contact": "9882536455",
   "dob": "01-10-1995"
}
```

Following is the sample document structure of **address** document −

```
{
   "_id":ObjectId("52ffc4a5d85242602e000000"),
   "building": "24, Gulmohar apt",
   "pincode": 225631,
   "city": "Lucknow",
   "state": "Uttar pradesh"
}
```

Modeling Embedded Relationships

In the embedded approach, we will embed the address document inside the user document.

```
> db.users.insert({
    {
           "_id":ObjectId("52ffc33cd85242f436000001"),
           "contact": "9882536455",
           "dob": "01-10-1995",
           "name": "Anita kumari",
           "address": [
                  {
                         "building": "24, Gulmohar Apt",
                         "pincode": 225631,
                         "city": "Lucknow",
                         "state": "Uttar Pradesh"
                  },
                  {
                         "building": "149 A, Apsara Apt",
                         "pincode": 225678,
                         "city": "Gorakhpur",
                         "state": "Uttar pradesh"
                  }
           ]
    }
})
```

This approach maintains all the related data in a single document, which makes it easy to retrieve and maintain. The whole document can be retrieved in a single query such as −

```
>db.users.findOne({"name":"Anita Kumari"},{"address":1})
```

Note that in the above query, **db** and **users** are the database and collection respectively.

The drawback is that if the embedded document keeps on growing too much in size, it can impact the read/write performance.

Assignment

1.what are MongoDB data types?

2.What are embedded documents in MongoDB?