ADVANCE WEB APPLICATION DEVELOPMENT

WEEK 7

1.Explain Schema and Model in MONGODB in details

ANS;

A schema is a JSON object that defines the the structure and contents of your data. You can use Atlas App Services' BSON schemas, which extend the JSON Schema standard.

To define your application's data model and validate documents whenever they're created, changed, or deleted.

Data Modeling in MongoDB

In MongoDB, data has a flexible schema. It is totally different from SQL database where you had to determine and declare a table's schema before inserting data. MongoDB collections do not enforce document structure.

The main challenge in data modeling is balancing the need of the application, the performance characteristics of the database engine, and the data retrieval patterns.

Consider the following things while designing the schema in MongoDB

- Always design schema according to user requirements.
- Do join on write operations not on read operations.
- Objects which you want to use together, should be combined into one document. Otherwise they should be separated (make sure that there should not be need of joins).
- Optimize your schema for more frequent use cases.
- Do complex aggregation in the schema.
- You should duplicate the data but in a limit, because disc space is cheaper than compute time.

For example:

let us take an example of a client who needs a database design for his website. His website has the following requirements:

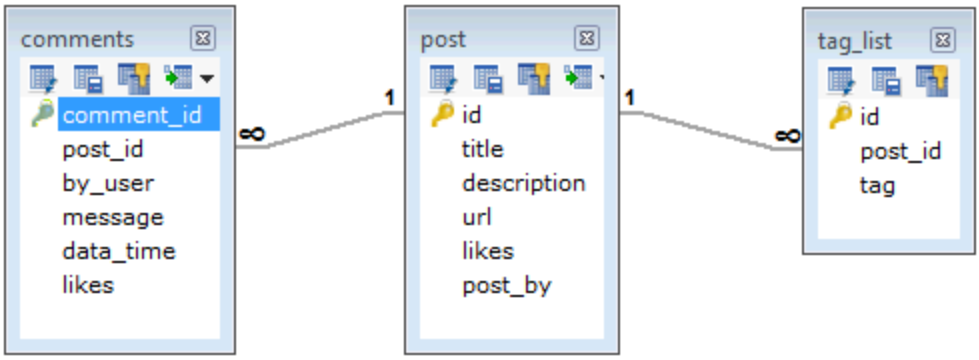
very post is distinct (contains unique title, description and url).

Every post can have one or more tags.

Every post has the name of its publisher and total number of likes.

Each post can have zero or more comments and the comments must contain user name, message, data-time and likes.

For the above requirement, a minimum of three tables are required in RDBMS.



But in MongoDB, schema design will have one collection post and has the following structure:

```
{
_id: POST_ID
title: TITLE_OF_POST,
description: POST_DESCRIPTION,
by: POST_BY,
url: URL_OF_POST,
tags: [TAG1, TAG2, TAG3],
likes: TOTAL_LIKES,
comments: [
{
user: 'COMMENT_BY',
message: TEXT,
datecreated: DATE_TIME,
like: LIKES
},
{
user: 'COMMENT_BY',
message: TEST,
dateCreated: DATE_TIME,
like: LIKES
}}}
```

2.Write a MongoDB Database to display all the documents in the collection library.

ANS;

CREATING A COLLECTION IN MONGODB

Creation of collection can be done using db.createCollection(name, options). But, typically you will not require building or constructing a collection of your own.

MongoDB does this creation task automatically as you start to insert some documents for making a database.

Here is a syntax that will tell you how to create your collection in MongoDB.

db.createcollection(collection name,option)

Here, db.createCollection() is the method used; "name" is a data type - string which is specifying the name of the collection to be formed.

"options" is an added document type which helps in specifying the size of memory along with indexing the collection in the database. This parameter is an optional one.

The following example shows the syntax of the createCollection() method with its options:

Example;

```
db.createCollection(<collection_name>, { capped: <boolean>,
        autoIndexId: <boolean>,
        size: <number>,
        max: <number>,
        storageEngine: <document>,
        validator: <document>,
        validationLevel: <string>,
        validationAction: <string>,
        indexOptionDefaults: <document>,
        viewOn: <string>,
        pipeline: <pipeline>,
        collation: <document>,
        writeConcern: <document>} )
```

3.How to define the Schema in mongoose?

In mongoose, a schema **represents the structure of a particular document, either completely or just a portion of the document**.

It's a way to express expected properties and values as well as constraints and indexes. A model defines a programming interface for interacting with the database (read, insert, update, etc).

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection.

```
import mongoose from 'mongoose';
const { Schema } = mongoose;


const blogSchema = new Schema({
```

```
  title:  String, // String is shorthand for {type: String}
  author: String,
  body:   String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs:  Number
  }
});
```

Each key in our code blogSchema defines a property in our documents which will be cast to its associated SchemaType. For example, we've defined a property title which will be cast to the String SchemaType and property date which will be cast to a Date SchemaType.

Keys may also be assigned nested objects containing further key/type definitions like the meta property above. This will happen whenever a key's value is a POJO that doesn't have a type property.

In these cases, Mongoose only creates actual schema paths for leaves in the tree. (like meta.votes and meta.favs above), and the branches do not have actual paths. A side-effect of this is that meta above cannot have its own validation. If validation is needed up the tree, a path needs to be created up the tree - see the Subdocuments section for more information on how to do this. Also read the Mixed subsection of the SchemaTypes guide for some gotchas.

The permitted SchemaTypes are:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Decimal128
- Map

Read more about SchemaTypes here.

Schemas not only define the structure of your document and casting of properties, they also define document instance methods, static Model methods, compound indexes, and document lifecycle hooks called middleware.