Task 8: Data Modeling in MongoDB with Schema Validation

Objective:

Understand and practice data modeling techniques in MongoDB, including schema design, using embedded vs. referenced documents, denormalization, and enforcing schema validation on inserts.

Prerequisites:

- Basic understanding of JavaScript and MongoDB.
- Node.js installation.
- MongoDB installed and running.
- Mongoose installed (`npm install mongoose`).
- A MongoDB collection with sample movie data.

Concepts:

1. **Schema Design with Mongoose:**
   - Define schemas using Mongoose to enforce structure and validation.

     Example:

     JavaScript:

```javascript
const mongoose = require('mongoose');

const movieSchema = new mongoose.Schema({
  title: { type: String, required: true },
  release_date: { type: Date, required: true },
  genre: { type: [String], required: true },
  director: { type: String, required: true },
  cast: { type: [String], required: true },
  production_companies: { type: [String], required: true },
  synopsis: { type: String, required: true }
});

const Movie = mongoose.model('Movie', movieSchema);

async function insertMovie(movieData) {
  const movie = new Movie(movieData);
  try {
    const result = await movie.save();
    console.log('Inserted Movie:', result);
  } catch (error) {
    console.error('Error inserting movie:', error.message);
  }
}

mongoose.connect('mongodb://localhost:27017/moviesDB', { useNewUrlParser:
true, useUnifiedTopology: true })
  .then(() => {
```

```
    console.log('Connected to MongoDB');

    // Sample movie data
    const sampleMovie = {
      title: "Example Movie",
      release_date: new Date("2024-05-01"),
      genre: ["Action", "Adventure"],
      director: "Jane Doe",
      cast: ["Actor A", "Actor B"],
      production_companies: ["Company A"],
      synopsis: "This is an example movie."
    };

    // Insert sample movie
    insertMovie(sampleMovie);
  })
  .catch(err => console.error('Could not connect to MongoDB...', err));
```

2. **Embedded vs. Referenced Documents:**
   - **Embedded Documents:**
     - Use embedded documents when data is frequently accessed together.

     Example:

     JavaScript:

```
const movieWithEmbeddedSchema = new mongoose.Schema({
  title: { type: String, required: true },
  release_date: { type: Date, required: true },
  genre: { type: [String], required: true },
  director: {
    name: { type: String, required: true },
    birth_year: { type: Number, required: true }
  },
  cast: [{
    name: { type: String, required: true },
    role: { type: String, required: true }
  }]
});

const MovieWithEmbedded = mongoose.model('MovieWithEmbedded',
movieWithEmbeddedSchema);
```

   - **Referenced Documents:**
     - Use referenced documents when data is accessed separately or when data duplication should be minimized.

     Example:

     JavaScript:

```
const directorSchema = new mongoose.Schema({
  name: { type: String, required: true },
  birth_year: { type: Number, required: true },
  movies: { type: [String], required: true }
});

const Director = mongoose.model('Director', directorSchema);

const movieWithReferencedSchema = new mongoose.Schema({
  title: { type: String, required: true },
  release_date: { type: Date, required: true },
  genre: { type: [String], required: true },
  director_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Director',
required: true }
});

const MovieWithReferenced = mongoose.model('MovieWithReferenced',
movieWithReferencedSchema);

async function insertMovieWithReferenced(movieData, directorData) {
  const director = new Director(directorData);
  try {
    const directorResult = await director.save();
    movieData.director_id = directorResult._id;
    const movie = new MovieWithReferenced(movieData);
    const movieResult = await movie.save();
    console.log('Inserted Movie with Referenced Documents:', movieResult);
  } catch (error) {
    console.error('Error inserting movie with referenced documents:',
error.message);
  }
}

// Sample data
const sampleDirector = {
  name: "Jane Doe",
  birth_year: 1970,
  movies: ["Example Movie"]
};

const sampleReferencedMovie = {
  title: "Example Movie",
  release_date: new Date("2024-05-01"),
  genre: ["Action", "Adventure"]
};

// Insert sample movie with referenced director
insertMovieWithReferenced(sampleReferencedMovie, sampleDirector);
```

3. **Denormalization:**
   - Duplicate data to optimize read performance.

Example:

JavaScript:

```javascript
const movieDenormalizedSchema = new mongoose.Schema({
  title: { type: String, required: true },
  release_date: { type: Date, required: true },
  genre: { type: [String], required: true },
  director: { type: String, required: true },
  cast: { type: [String], required: true },
  production_companies: { type: [String], required: true },
  synopsis: { type: String, required: true }
});

const MovieDenormalized = mongoose.model('MovieDenormalized',
movieDenormalizedSchema);

async function insertDenormalizedMovie(movieData) {
  const movie = new MovieDenormalized(movieData);
  try {
    const result = await movie.save();
    console.log('Inserted Denormalized Movie:', result);
  } catch (error) {
    console.error('Error inserting denormalized movie:', error.message);
  }
}

// Sample denormalized movie data
const sampleDenormalizedMovie = {
  title: "Example Movie",
  release_date: new Date("2024-05-01"),
  genre: ["Action", "Adventure"],
  director: "Jane Doe",
  cast: ["Actor A", "Actor B"],
  production_companies: ["Company A"],
  synopsis: "This is an example movie."
};

// Insert sample denormalized movie
insertDenormalizedMovie(sampleDenormalizedMovie);
```

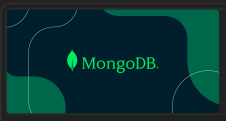
Instructions:

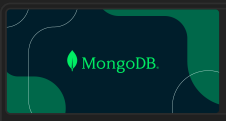
Perform the following tasks:

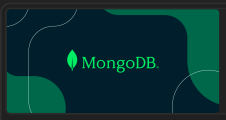
1. Design a schema for the movies collection that includes fields for title, release_date, genre, director, cast, production_companies, and synopsis.
2. Insert a sample movie document using the designed schema and Mongoose.
3. Use embedded documents to store director and cast information within the movie document.
4. Use referenced documents to store director information in a separate collection and reference it in the movie document.

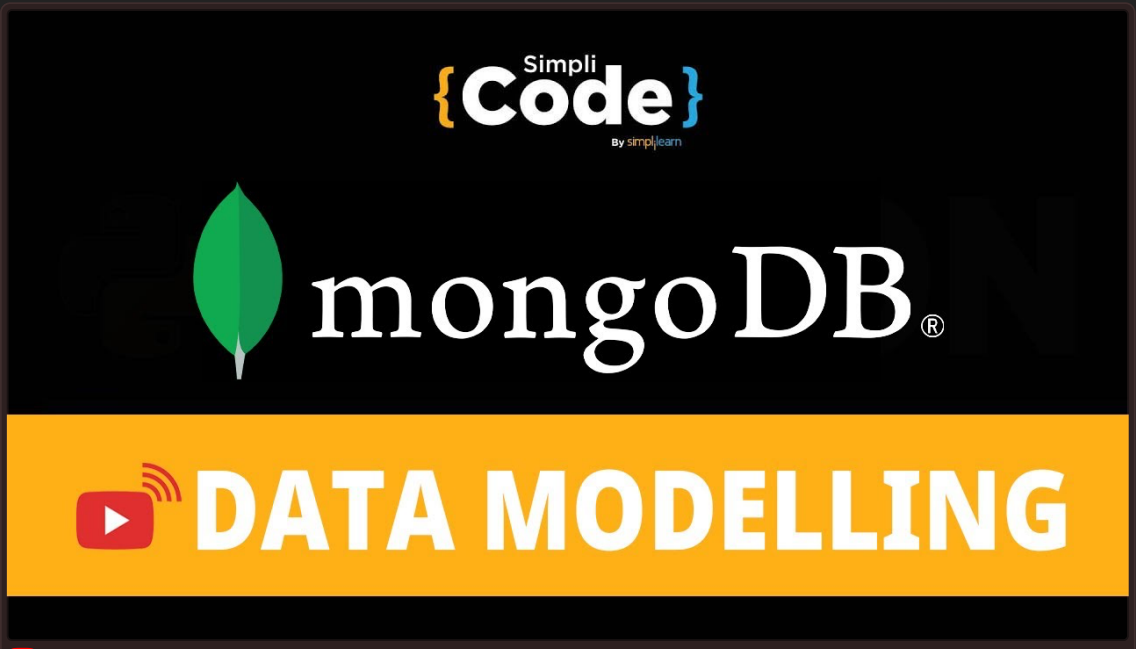
5. Apply denormalization by duplicating data for faster read performance.
6. Ensure that each document inserted follows the schema validation rules.

Resources:

- **Documentation:**
  - Mongoose Documentation
  - 

    Data Modeling
    Data modeling refers to the organization of data within a database and — the links b...
    https://docs.mongodb.com/manual/data-modeling/
  - 

    Model One-to-Many Relationships with Embedded Documents
    Create a data model that uses .leafygreen-ui-lh88kx{-webkit-align-items:center;-we...
    https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/
  - 

    Model One-to-Many Relationships with Document References
    This page describes a data model that uses .leafygreen-ui-lh88kx{-webkit-align-ite...
    https://docs.mongodb.com/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/
  - MongoDB Schema Design
- **Videos:**
  - MongoDB University Course: M320 Data Modeling
  - 

    SimpliCode 15:56 min 3,428 Views 64 Likes

- 

GitHub Instructions

1. **Fork the Repository:**
   - Go to the GitHub Classroom link provided.
   - Click on the "Fork" button in the top right corner of the repository page to create a copy of the repository under your own GitHub account.
2. **Open in Visual Studio Code:**
   - After forking the repository, clone it to your local machine.
   - Open Visual Studio Code (VSCode).
   - Open the terminal in VSCode by selecting Terminal > New Terminal from the top menu.

-github-username` with your actual GitHub username):

`git clone https://github.com/your-github-username/repository-name.git`

- Navigate into the cloned repository directory:

```
cd repository-name
```

2. **Complete the Task:**
   - In VSCode, open the `index.js` file in the root directory of your repository and write your solution.
   - Ensure the `package.json` file is present and contains all necessary dependencies. If you need to install additional packages, use:

```
npm install
```

3. **Run and Test Your Code:**
   - Run your code to ensure it works correctly. Use the following command:

```
node index.js
```

4. **Commit Your Changes:**
   - In the VSCode terminal, add your changes to git:

```
git add index.js package.json
```

   - Commit your changes with a meaningful message:

```
git commit -m "Completed task 8"
```

5. **Push Your Changes to Your Forked Repository:**
   - Push your changes to your forked repository:

```
git push origin main
```

6. **Create a Pull Request:**
   - Go to your forked repository on GitHub.
   - Click on the "Pull Requests" tab.
   - Click the "New Pull Request" button.
   - Ensure the base repository is the original template repository and the base branch is `main`.
   - Ensure the head repository is your forked repository and the compare branch is `main`.
   - Click "Create Pull Request".
   - Add a title and description for your pull request and submit it.

Summary of Commands

```
# Fork the repository on GitHub

# Clone the forked repository
git clone https://github.com/your-github-username/repository-name.git
cd repository-name

# Complete the task by editing index.js

# Run your code
node index.js

# commit, and push your changes
```

```
git add index.js package.json
git commit -m "Completed task 8"
git push origin main

# Create a pull request on GitHub
```