Relational Queries with Mongoose

Module 4 Tutorial: Relational Queries with Mongoose

In this tutorial lab, you will build relational queries between blog and comment objects using Mongoose.

Relational queries with Mongoose

You work at a news company and your task is to implement comments for a company blog. Each blog post (story) needs to have many comments. There are two ways to go about the blog implementation.

First, you can save/store all the comments right in the blog post document. This way you have a single collection. The downside is that if you need to access, index, or search individual comments, then having them inside of the blog post collection is not optimal.

The second approach is to create a new collection just for comments. In this collection, you can store comments which will be referenced to/from their blog posts. In the second approach (which is having two collections: posts and comments), you have two options: child refs and parent refs (refs - references).

Mongoose provides a mechanism to fetch related documents using `populate()` which greatly simplifies querying for both approaches. Without Mongoose, you still can implement parent-child relationship but you'll need to write more code yourself, such as making two queries instead of one.

So now that you understand the task. Let's get to implementing both approaches: child refs and parent refs starting with child refs first. Of course we will use Mongoose and its `populate()`.

Child Refs

In the child refs approach, you store references (Object IDs) to comments in a post document. This will allow you to use Mongoose's `populate()` to fetch comments in a post query. Instead of making two queries: find post and find comment, you'll run a single query on post using `populate()`.

Create a new file `fetch-blog-posts.js` in which you connect to a local MongoDB and configure Promise implementation to use default ES6 Promise (`mongoose.Promise = global.Promise`)

```
var mongoose = require('mongoose')
mongoose.Promise = global.Promise
mongoose.connect('mongodb://localhost:27017/edx-course-db',
  {useMongoClient: true})
```

Next, define Post and Comment schemas. Post has references (refs) to Comment using ObjectId type:

```
const Post = mongoose.model('Post',
  { name: String,
    url: String,
    text: String,
    comments: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Comment
  },
 )
const Comment = mongoose.model('Comment', {
  text: String
})
```

Now, create bunch of comments using `save()` and store the newly created comment document IDs in `ca`:

```
let ca = [{text: 'Cruel…..var { house, mouse} = No type optimization
  {text: 'I think you're undervaluing the benefit of 'let' and 'cons
  {text: '(p1,p2)=>{ … } ,i understand this ,thank you !'}
].map((comment) => {
  const c = new Comment(comment)
  c.save()
  return c._id
})
console.log(ca)
```

Next create the post document and use `ca` which is an array with comment IDs:

```
var post = new Post({
  name: 'Top 10 ES6 Features every Web Developer must know',
  url: 'https://webapplog.com/es6',
  text: 'This essay will give you a quick introduction to ES6. If yo
  comments: ca
})
```

The preparation is over. Save the post object.

```
post.save(function (err) {
  if (err) {
    console.log(err)
  } else {
    console.log('The post is saved: ', post.toJSON())
  }
```

In the callback of `post.save()`, invoke `findOne()` query with `populate()` to get not just the post document but all the child comment documents.

Obviously, the save and query parts will be in different places in a real project, but in our small example script, we are querying in the callback of `save`.

```
// Populate
Post
  .findOne({ name: /Top 10 ES6/i })
  .populate('comments')
  .exec(function (err, post) {
    if (err) return console.error(err)
    console.log(`The post is ${post}`)
    mongoose.disconnect()
  })
})
```

The result are the posts with comments. First you will see an array of comment IDs, then the actual post document as it is stored in the database and finally (where it says "The post...") the Mongoose query result. See for yourself:

```
$ node fetch-blog-posts.js
[ 59bfbbd96fcae76e2a1df389,
  59bfbbd96fcae76e2a1df38a,
  59bfbbda6fcae76e2a1df38b ]
The post is saved:  { __v: 0,
  name: 'Top 10 ES6 Features every Web Developer must know',
  url: 'https://webapplog.com/es6',
  text: 'This essay will give you a quick introduction to ES6. If you
don’t know what is ES6, it’s a new JavaScript implementation.',
  _id: 59bfbbda6fcae76e2a1df38c,
  comments:
   [ 59bfbbd96fcae76e2a1df389,
     59bfbbd96fcae76e2a1df38a,
     59bfbbda6fcae76e2a1df38b ] }
The post is { _id: 59bfbbda6fcae76e2a1df38c,
  name: 'Top 10 ES6 Features every Web Developer must know',
  url: 'https://webapplog.com/es6',
  text: 'This essay will give you a quick introduction to ES6. If you
don’t know what is ES6, it’s a new JavaScript implementation.',
  __v: 0,
  comments:
   [ { _id: 59bfbbd96fcae76e2a1df389,
       text: 'Cruel…..var { house, mouse} = No type optimization at
all',
       __v: 0 },
     { _id: 59bfbbd96fcae76e2a1df38a,
       text: 'I think you’re undervaluing the benefit of ‘let’ and
‘const’.',
       __v: 0 },
     { _id: 59bfbbda6fcae76e2a1df38b,
       text: '(p1,p2)=>{ … } ,i understand this ,thank you !',
       __v: 0 } ] }
```

Now let's flip the approach and fetch the comment with post info.

Parent Refs

Now let's implement the approach where children have parent refs, i.e., comments will have a reference to post. This way `populate()` on a comment query will bring post information. Again, you don't need to first fetch the comment and then execute a new

query to fetch the corresponding post. Instead with `populate()`, you run a single query and boom! The post data is populated for you.

Create a file `fetch-blog-posts-parent-refs.js` and import Mongoose:

```
const mongoose = require('mongoose')
mongoose.Promise = global.Promise
mongoose.connect('mongodb://localhost:27017/edx-course-db',
  {useMongoClient: true})
```

Define post and comment schemas. This time, do not define any comment refs in post, but define a post reference in the comment schema:

```
const Post = mongoose.model('Post',
  { name: String,
    url: String,
    text: String
  }
 )
const Comment = mongoose.model('Comment', {
  text: String,
  post: { type: mongoose.Schema.Types.ObjectId, ref: 'Post' }
})
```

Create a new post object without any comments and save it:

```
let post = new Post({
  name: 'Top 10 ES6 Features every Web Developer must know',
  url: 'https://webapplog.com/es6',
  text: 'This essay will give you a quick introduction to ES6. If yo
})
```

Next, save the post object and comments. Each comment will have the ID of the newly created post. When all comments are saved (`i==list.length`), then execute the query `queryCommentWithPost()`:

```
post.save((err) => {
  if (err) {
    console.log(err)
  } else {
    console.log('Post is saved: ', post.toJSON())
  }
  let i = 0
  let ca = [{text: 'Cruel…..var { house, mouse} = No type optimizati
    {text: 'I think you're undervaluing the benefit of 'let' and 'co
    {text: '(p1,p2)=>{ … } ,i understand this ,thank you !'}
  ].forEach((comment, index, list) => {
    comment.post = post._id
    const c = new Comment(comment)
    c.save((error, result)=>{
      if (error) return console.error(error)
      i++
      if (i==list.length) {
       queryCommentWithPost()
      }
    })
  })
})
```

The query uses `findOne()` on the Comment model and the `populate()` method to fetch related post:

```
const queryCommentWithPost = () => {
   // Populate
   Comment
   .findOne({ text: /Cruel/i })
   .populate('post')
   .exec(function (err, comment) {
     if (err) return console.error(err)
     console.log(`The comment is ${comment}`)
     mongoose.disconnect()
   })
}
```

The result will look like this. Notice in `The comment is...`, the full post information in the comment document and *not* an object ID (as would be in the actual database document):

```
node fetch-blog-posts-parent-refs.js
Post is saved:  { __v: 0,
  name: 'Top 10 ES6 Features every Web Developer must know',
  url: 'https://webapplog.com/es6',
  text: 'This essay will give you a quick introduction to ES6. If yo
  _id: 59bfc0d5d4bc5172a551b62e }
The comment is { _id: 59bfbf19326c7a7133340bea,
  text: 'Cruel…..var { house, mouse} = No type optimization at all',
  post:
   { _id: 59bfbf19326c7a7133340be9,
     name: 'Top 10 ES6 Features every Web Developer must know',
     url: 'https://webapplog.com/es6',
     text: 'This essay will give you a quick introduction to ES6. If
     __v: 0 },
  __v: 0 }
```

Wrap up

Mongoose provide a convenient way to fetch data of related objects. This is a useful feature when you need to get data in a single query.