More SQL

When Postgres meets Node

Contents

Approach

So...

We just officially introduced you to Mongoose, the MongoDB ODM. Given your familiarity with that concept, we will just jump into the Postgres equivalent--an Object Relational Mapper (ORM)--to see how that works.

Postgres ORM Options

Two often used options

- Bookshelf -- what we will use for our examples
- Sequelize -- an alternative you can explore for yourself

Installing

Packages you'll need

```
$ npm install knex --save

$ npm install bookshelf --save

$ npm install pg
```

Connecting

models/db/bookshelf.js

```
var connectionString = process.env.PG_CONNECTION_STRING ||
'postgres://localhost:5432/rando';

var knex = require('knex')({
 client: 'pg',
 connection: connectionString,
 searchPath: 'knex,public'
});

var Bookshelf = require('bookshelf')(knex);

module.exports = Bookshelf;
```

Models

What are models?

Objects that correspond to one record (or row) in a database table.

Example

models/talent.js (Basic)

```
// where talent is the actual name of my database table
var Talent = Bookshelf.Model.extend({
    tableName: 'talent'
});
```

models/talent.js (with Timestamps)

```
// It's pretty standard to have created_at and updated_at
// fields in a database.
// hasTimestamps being set to true means bookshelf will
// automatically update those fields for us when appropriate
var Talent = Bookshelf.Model.extend({
    tableName: 'talent',
    hasTimestamps: true
});
```

models/talent.js (with Relationships)

```
// Talent has a many-to-many relationship with skills
var Talent = Bookshelf.Model.extend({
    tableName: 'talent',
    hasTimestamps: true,
    skills: function() {
        return this.belongsToMany(Skill, 'talent_skills');
    }
});
```

Collections

What are collections?

Groups of Models that we can perform operations on.

Example

models/collections/talents.js (Basic)

```
var Talents = Bookshelf.Collection.extend({
    model: Talent
});
```

Creating

Create a new talent

```
// Forge is a convenience Bookshelf method that allows us to not use
// the 'new' keyword. We can put our data inside.
Talent.forge({
    talent_legacy_id: 9,
    first_name: 'Lisa',
    last_name: 'Leslie',
    city: 'Gardena',
    state: 'CA'
    })
    .save()
    .then(function(talent) { /* do stuff */ });
```

Reading

Read all talents

```
// Fetching our collection returns all our talent records
Talents.forge()
    .fetch()
    .then(function(collection) { /* do stuff */ });
```

Read all talents with their skills

```
// Fetch all talent records with their related skills
Talents.forge()
    .fetch({withRelated: ['skills']})
    .then(function(collection) { /* do stuff */ });
```

Updating

Update a talent

```
// forge includes the talent data we want to find.
// fetch with require as true will throw an error if we find nothing
Talent.forge({
    first_name: 'Lisa',
    last_name: 'Leslie'})
    .fetch({require: true})
    .then(function(talent) {
        talent.save({city: 'Los Angeles'})
        .then(function() { /* do stuff */ }
        .catch(function(err) {/* do stuff */}
        })
    .catch(function(err) { /* do stuff */ });
```

Deleting

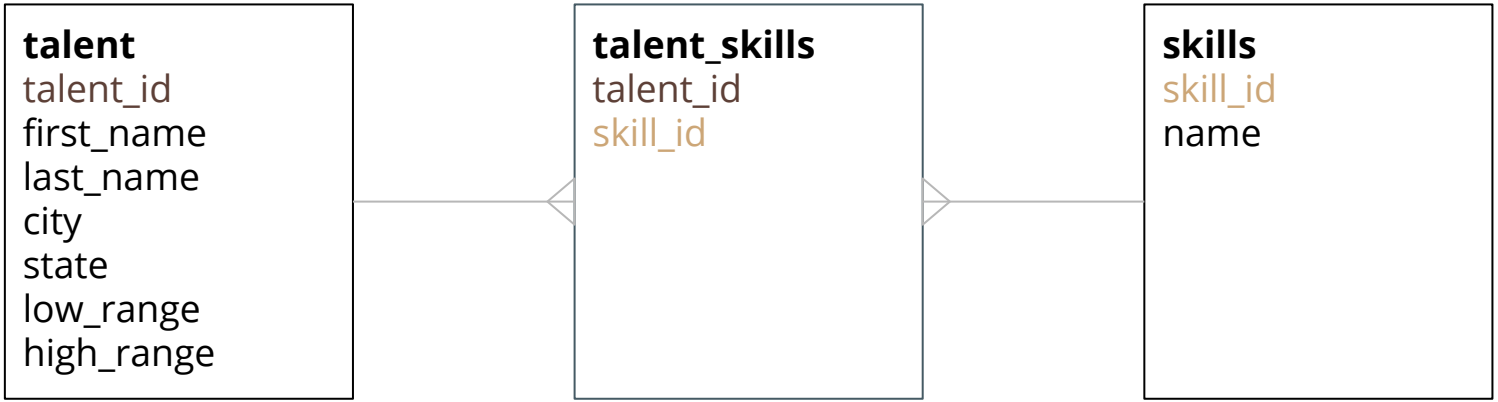
Delete a talent

```
// meet destroy
Talent.forge({
    first_name: 'Lisa',
    last_name: 'Leslie'})
    .fetch({require: true})
    .then(function(talent) {
        talent.destroy()
        .then(function() { /* do stuff */ }
        .catch(function(err) {/* do stuff */}
        })
    .catch(function(err) { /* do stuff */ });
```

For completeness…

Keys

talent_id is a *primary key* of the talent table and a ***foreign key*** of the talent_skills table.
skill_id is a *primary key* of the skills table and a ***foreign key*** of the talent_skills table.



Example

SQL Foreign Key Constraint

```
-- talent_id is a foreign key
-- skill_id is a foreign key
CREATE TABLE talent_skills (
  talent_id int REFERENCES talent,
  skill_id int REFERENCES skills,
  PRIMARY KEY(talent_id, skill_id),
  created_at timestamp,
  updated_at timestamp
);
```

Resources

Helpful Links

- Bookshelf ORM Docs
- Knex Docs
- Building a simple API with Express and Bookshelf.js

You Might Also Like

- Node and Postgres using only the Postgres driver

Group Challenge

Instructions

Using today's lecture and via exploration of the Resources, perform the following tasks.

1. Fork this repo
2. Start postgres
3. Navigate to utils directory in your terminal
4. Type `psql -f rando_create_tables_data.sql rando`
5. Press enter to create some tables in your rando database
6. Note the code in routes/talent.js and complete any TODOs
7. Add routes and logic for GET, POST, PUT, and DELETE for skills
8. Use your favorite REST client to test
9. Update the talent POST route to add skills (provided via the request body) to talent