Node-pg Relationships (Further Study)

Download exercise

In this exercise, you'll build a REST-ful backend API server for a simple company/invoice tracker.

Step 0: Setup

- Create a project folder, a Git repo, and a ***package.json***
- Install ***express*** and ***pg*** via NPM
- Add ***node_modules*** to ***.gitignore***

Step 1: Add a Database

- Create a database, ***biztime***
- Load the initial data from ***data.sql***
- Fix ***db.js*** so that it connects to the database and exports the client object.
- Familiarize yourself with the data model.

Step 2: Add Company Routes

Create ***routes/companies.js*** with a router in it.

All routes in this file should be found under *companies/*.

All routes here will respond with JSON responses. These responses will be in an object format where the value is the data from the database.

So, for example, the "get list of companies should return":

```
{companies: [{code, name}, ...]}
```

Assuming ***result*** is the result from your query, you could produce this with a line like:

```
return res.json({companies: result.rows})
```

These routes need to be given data in JSON format, not the standard "url-encoded form body" — so you'll need to make sure that your ***app.js*** includes the middleware to parse JSON.

Routes Needed

GET /companies : Returns list of companies, like `{companies: [{code, name}, ...]}`

GET /companies/[code] : Return obj of company: `{company: {code, name, description}}`
If the company given cannot be found, this should return a 404 status response.

POST /companies : Adds a company. Needs to be given JSON like: `{code, name, description}` Returns obj of new company: `{company: {code, name, description}}`

PUT /companies/[code] : Edit existing company. Should return 404 if company cannot be found.
Needs to be given JSON like: `{name, description}` Returns update company object: `{company: {code, name, description}}`

DELETE /companies/[code] : Deletes company. Should return 404 if company cannot be found.
Returns `{status: "deleted"}`

Step 3: Add Invoices

Add ***routes/invoices.js***. All routes in this file should be prefixed by ***/invoices***.

GET /invoices : Return info on invoices: like `{invoices: [{id, comp_code}, ...]}`

GET /invoices/[id] : Returns obj on given invoice.
If invoice cannot be found, returns 404.
Returns `{invoice: {id, amt, paid, add_date, paid_date, company: {code, name, description}}}`

POST /invoices : Adds an invoice. Needs to be passed in JSON body of: `{comp_code, amt}`
Returns: `{invoice: {id, comp_code, amt, paid, add_date, paid_date}}`

PUT /invoices/[id] : Updates an invoice. If invoice cannot be found, returns a 404.
Needs to be passed in a JSON body of `{amt}`
Returns: `{invoice: {id, comp_code, amt, paid, add_date, paid_date}}`

DELETE /invoices/[id] : Deletes an invoice.If invoice cannot be found, returns a 404.
Returns: `{status: "deleted"}` Also, one route from the previous part should be updated:

GET /companies/[code] : Return obj of company: `{company: {code, name, description, invoices: [id, ...]}}` If the company given cannot be found, this should return a 404 status response.

Further Study

Write some tests!

Make sure that your routes are tested, use ***jest –coverage*** to see how well you have tested your routes.

Slugify Company Names

It might be difficult for customers to make up a customer code themselves when making new companies (preferably, they should have no spaces or weird punctuation, and should be all lower-case).

Fortunately, there's an NPM library that can help out, ***slugify***. Read about this, and then change the ***POST /companies*** route so that they don't provide a code directly, but you make this by using ***slugify()*** on the given name.

Allow paying of invoices

Change the logic of this route:

PUT /invoices/[id] : Updates an invoice. If invoice cannot be found, returns a 404.
Needs to be passed in a JSON body of `{amt, paid}`
• If paying unpaid invoice: sets ***paid_date*** to today
• If un-paying: sets ***paid_date*** to null
• Else: keep current ***paid_date***
Returns: `{invoice: {id, comp_code, amt, paid, add_date, paid_date}}`

Add a Many-to-Many

A larger feature.

Add a table for "industries", where there is a ***code*** and an ***industry*** field (for example: "acct" and "Accounting").

Add a table that allows an industry to be connected to several companies and to have a company belong to several industries.

Add some sample data (by hand in ***psql*** is fine).

Change this route:

- when viewing details for a company, you can see the names of the industries for that company

Add routes for:

- adding an industry
- listing all industries, which should show the company code(s) for that industry
- associating an industry to a company

Solution

View our Solution