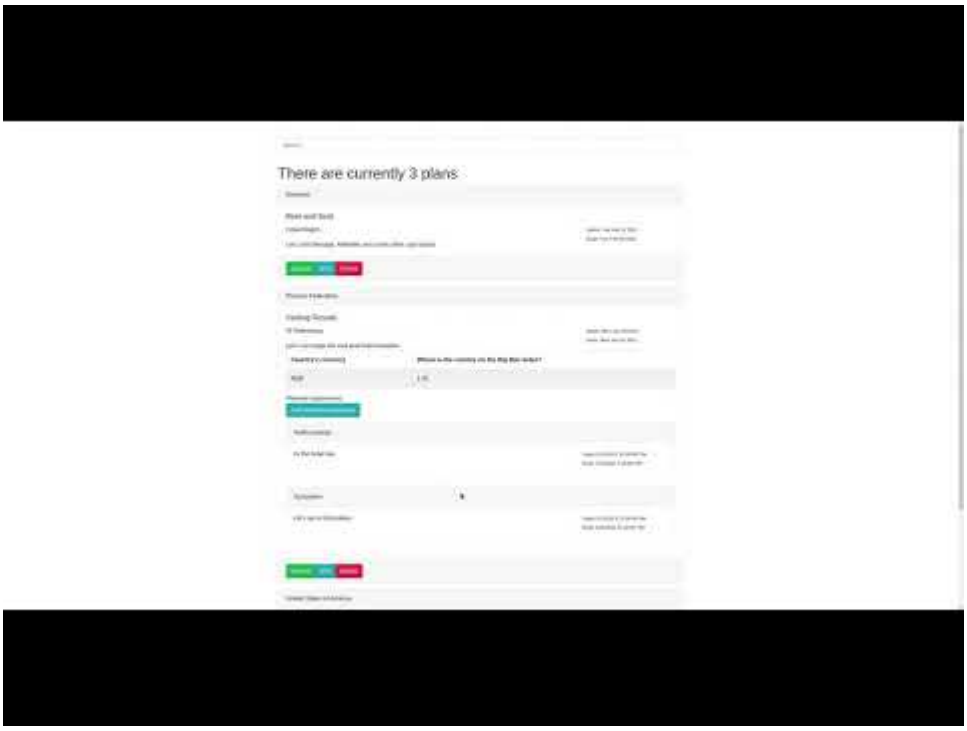
Database Design - Assignment 3

Video demonstration of application:



Task 1:

How many times have you spent in front of your computer browsing travel destinations when the christmas holiday is around the corner? All you can remember is the week you spent in the canary islands 14 years ago before the kids arrived and the trip to Skara sommarland **every** single year. Perhaps you know little of the world and its destinations that are well suited for families, maybe all you ever heard about is the Canary Islands, southern Spain and Thailand. Perhaps you want to see something other than all inclusive resorts and other swedish people that have travelled 12 hours to spend a month by the pool in Phuket.

Imagine you're 20-something years old, just finished school. Before work, you might want to do something fun. But what? Perhaps another day trip to Copenhagen? But, after all these years in school, why not try the world? But where? You haven't had the time to work full time yet, all you have is your savings from your moonlighting. All you have to rely on for budgeting are people on Tripadvisor, telling you it's impossible to leave your country without $100 per day as a budget. If only there was a place where you can do quick searches for fun destinations and sort by budget...

1. Idea

- The user should be able to perform CRUD functionalities:
- The user should be able to view travel plans created by the user
- The user should be able to make filtered search queries and the application will display travel plans that match the query.
- The user should also be able to browse a discovery section without having to search for anything,
- The user should also be able to create its own travel plans using relevant and necessary information.

Task 2

Our database handles three major entities — the travel plan (Plan), the Country in which this Plan is supposed to be executed, and the experiences you plan for during your holiday (Planned experience). As can be seen, there are additional entities in the model, but the aforementioned three are the ones that are most essential to the database.

Plan

The plan is the most important piece of this application.

Big Mac Index

Whereas a Country itself, its name and country code, may not change for decades, its economy and relative currency strength is everchanging. This means we want to easily update and change the data in the Big Mac Index table without anomalies. The same reason applies to why we decided to decompose Big Mac Index into its own entity, and in that case we would also like to avoid redundancy.

Experience

A Planned Experience is a weak entity, for the reason that it cannot exist without a Plan. Likewise, when a Plan is deleted, its Planned Experiences should be deleted as well. It should also be possible to create a Plan without any Planned Experience.

Task 3

There will be both the SQL query and a smaller overview of each relation. Primary keys are underlined and foreign keys will have (FK) as suffix. If nothing else is stated, you can safely suppose that each attribute is of typ VARCHAR. Unique attributes will have the suffix (UNIQUE). We can also safely suppose that each attribute has the constraints 'NOT NULL'.

Plan

```
CREATE TABLE  IF NOT EXISTS plan (
    id SERIAL NOT NULL PRIMARY KEY UNIQUE,
    country VARCHAR(3) NOT NULL references country(country_code),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    title VARCHAR(200) NOT NULL,
    description VARCHAR(9999) NOT NULL,
    location VARCHAR(200) NOT NULL
);
```

Experience

```
CREATE TABLE IF NOT EXISTS experience (
    id SERIAL NOT NULL PRIMARY KEY,
    start_datetime timestamp NOT NULL,
    end_datetime timestamp NOT NULL,
    title VARCHAR(200) NOT NULL,
    description VARCHAR(9999) NOT NULL,
    plan INTEGER NOT NULL REFERENCES plan(id) ON DELETE CASCADE
);
```

Country

```
CREATE TABLE IF NOT EXISTS country (
    country_code VARCHAR(10) NOT NULL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);
```

Big Mac

```
CREATE TABLE IF NOT EXISTS bigmac (
    country_code VARCHAR(10) NOT NULL PRIMARY KEY UNIQUE,
    exc_rate FLOAT NOT NULL,
    currency VARCHAR(10) NOT NULL
    );
```

plan(id:SERIAL, country(FK), start_date:date, end_date:date, title, description, location)

experience(id:SERIAL, start_datetime:timestamp, end_datetime:timestamp, title, description, plan(FK):INTEGER)

country(country_code, name(UNIQUE))

bigmac(country_code(UNIQUE), exc_rate:FLOAT, currency)

Task 4

Plan

The queries are baked into the code related to the application as a whole. So, we will first show the query in isolation, but also in its context if deemed necessary.

We want to get all available information about one or more plans. To achieve this, we first create a new view:

```
CREATE VIEW detailed_plan AS
SELECT (id, country, start_date, end_date, title, description, location,
country.country_code, exc_rate, currency, name) FROM (plan  AS z FULL OUTER
JOIN bigmac ON (z.country = bigmac.country_code)) AS x INNER JOIN country
ON (x.country = country.country_code);
```

And here is a sample query which we could use in our application:

```
SELECT * FROM detailed_plan;
/* Or */
SELECT * FROM detailed_plan WHERE (country = 'SWE' OR name LIKE '%Swe%');
```

In the context of our application:

```
const allPlans = req.query.country
  ? await pool.query(
      "SELECT * FROM detailed_plan WHERE (country = $1 OR name LIKE $2)",
      [req.query.country.toUpperCase(), "%" + req.query.country + "%"]
    )
  : await pool.query("SELECT * FROM detailed_plan");
```

Here, `req.query.country` is a search the user is doing for a country. Maybe the have input the country's abbreviated country code, or its full name, which is why we are looking at both columns in the first query.

Based on the design of our API, we do not return a Plan's related Experience's in the query above. But here is how we do just that in another part of the application:

```
SELECT * FROM experience WHERE (plan = 32); /* Or whatever ID you're
looking for*/
```

We also want to be able to add a new Plan. This is done like so:

```
INSERT INTO plan (location, country, start_date, end_date, title,
description)
VALUES ('My location', 'SWE', '2020-02-21', '2020-02-28', 'My amazing
vacation', 'We'll go bathing!');
```

If we want to do stuff on our vacation, we can also create planned experiences belonging to our plan.

```
INSERT INTO experience(start_datetime, end_datetime, title, description,
plan)
VALUES ('2020-02-21 09:29:32', '2020-02-21 12:29:32', 'Go swimming', 'We
need to swim', (SELECT id FROM plan WHERE id = 12));
```

Here we insert a planned experience with some data. The fifth value is a foreign key, which means that we fetch this value from the table `plan`. So we can keep track on which plan ID this experience is associated with.

To be able to find just one specific plan, of which we already know the ID, we do;

```
SELECT * FROM plan WHERE id = 12 /* Or whatever ID it is*/
```

And simply provide the ID.

We can also update one plan if we'd like.

```
UPDATE plan SET title = 'My new updated title' WHERE id = 12;
```

The same principle applies to updating an experience:

```
UPDATE experience SET title = 'My new updated experience title' WHERE id =
23;
```

Finally, if you change your mind about going for vacation, you can simply delete your travel plan.

```
DELETE FROM plan WHERE id = 12;
```

Experience

As we browse our application, we might want to take a look at all planned Experiences belonging to Plans within one specific country. This is done by:

```
SELECT * FROM
(experience INNER JOIN plan ON (experience.plan = plan.id)) AS z
WHERE z.country = 'SWE';
```

Here we join Experiences with their corresponding Plans, and selecting the tuples where the country is Sweden.

Country

Last but no least, we must have a country to tie our travel plans to. This will return each country's name, and ISO-compliant country code.

```
SELECT * FROM country
```

Task 5

Implementation

This is a PERN application. It consists of the technologies PostreSQL, Express, ReactJS and NodeJS. The source code is divided into two parts, one that handles the RESTful API and one that handles the frontend. The backend is written in NodeJS while the Frontend is written in Javascript, with ReactJS.

The application assumes that PostgreSQL is installed and configured with these credentials:

```
    host='localhost',
    database='postgres',
    user='postgres',
    password='pw'
```

The script which connects, populates the database with its relations and with dummy data is written in python and will be supplied.

To run the application, you need a Postgres database running with the user "postgres" and password "pw" and simply:

1. Clone the repo
2. cd into repo
3. Run `npm install`
4. `cd python`
5. `python3 py.main`
6. `cd ../client`
7. `npm install`
8. `npm start`
9. `cd ..`
10. `npm start`
11. Navigate to localhost:3000
12. Plan your next amazing vacation.
13. Enjoy