Übung 10

1 Datenbanken, Sequelize und Migrations

Erstellen Sie eine Datenbank namens `CoffeeShopDB`. Die Datenbank soll die Tabellen `Shops` und `Coffees` besitzen. Die `Shops`-Tabelle beinhaltet neben `id`, ebenfalls Spalten für `shopName`, `openAt` (beginn Öffnungszeit) und `closeAt` (Ladenschluss). Die `Coffees`-Tabelle besitzt Spalten für `coffeeName` und `price`. Zusätzlich wird in der `Coffees`-Tabelle über einen Fremdschlüsseleintrag in der `shopfk` Spalte ein `Coffee` mit einem `Shop` verknüpft. Diese Datenbank ist in Abbildung 1 visualisiert. Nutzen Sie für die Erstellung ausschließlich Sequelize, indem Sie die Modelle manuell erzeugen und mit Daten füllen.

Lösungsweg

Starten Sie ihren Datenbank-Server. Nutzen Sie dafür am besten XAMPP und starten Sie die Anwendung wie in den Vorlesungsfolien aufgezeigt. Sobald der Server läuft, können Sie den Datenbank-Manager unter der Adresse `http://localhost/phpmyadmin/` in Ihrem Browser erreichen. Erstellen Sie über diese Anzeige eine leere Datenbank namens `CoffeeShopDB`.

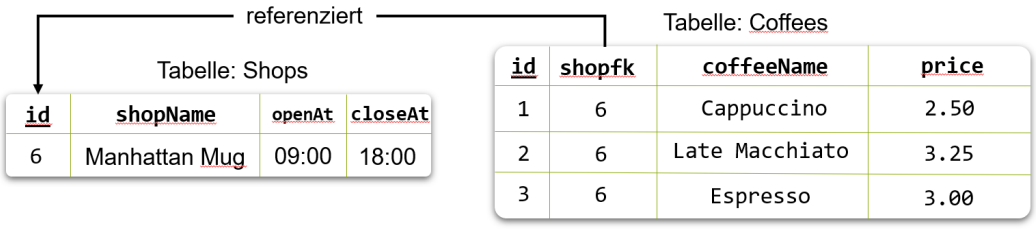


Abbildung 1: Inhalt von Datenbank CoffeeShopDB

Nachdem Sie mit *npm* die Module für `sequelize` und `mysql2` erfolgreich ihrem Projektordner hinzugefügt haben, können Sie Sequelize über folgenden Code initialisieren:

```
const Sequelize = require('sequelize');

// Option 1: Passing parameters separately
/*const sequelize = new Sequelize('CoffeeShopDB', 'root', '', {
  host: 'localhost',
  dialect: 'mysql'
});*/

// Option 2:
const sequelize = new Sequelize("mysql://root:@localhost:3306/CoffeeShopDB");
```

Um die Tabellen über Sequelize zu erzeugen, werden ES6 Klassen definiert, welche von `Sequelize.Model` erben. Durch den Aufruf der `init`-Methode werden die Modelle ausdefiniert. Effektiv werden dort die Spalten und ihr erwarteter Datentyp angegeben:

```
class Shop extends Sequelize.Model {}
Shop.init({
  shopName: Sequelize.STRING,
  openAt: Sequelize.TIME,
  closeAt: Sequelize.TIME
}, { sequelize, modelName: 'shop' });

class Coffee extends Sequelize.Model {}
Coffee.init({
  coffeeName: Sequelize.STRING,
  price: Sequelize.FLOAT
}, { sequelize, modelName: 'coffee' });
```

Zwischen `Shop` und `Coffee` soll eine Eins-zu-Viele Assoziation existieren. Gemäß der Vorlesung wird dies über die Methoden `hasMany` und `belongTo` erzeugt. Die fehlende Spalte für den Fremdschlüssel wird dabei automatisch mit konstruiert:

```
Shop.hasMany(Coffee);
Coffee.belongsTo(Shop);
```

Über die `sync`-Methode werden Modelle in die Datenbank überführt. Die Methode gibt dabei ein Promise zurück, durch welches mit der `then`-Methode eine weitere Veränderungen der Datenbank vorgenommen werden können. So kann über den Aufruf `Tabelle.create` ein Datensatz in die Tabelle eingepflegt werden. Voraussetzung ist allerdings, das die Tabelle bereits existiert. Wichtig zu beachten ist, dass die `create`-Methode asynchron ausgeführt wird und aufbauende Aufrufe durch `async/await` aufeinander warten sollten. Ist ein Dateneintrag erfolgreich erstellt, können über die Promise-Schreibweise Fremdschlüssel gesetzt werden. Dazu werden Hilfsmethoden (setShop) verwendet, die bei der Definition der Assoziation erstellt wurden:

```
sequelize.sync({ force: true }).then(() => {
  let dataTransfer = async () => {
```

```
    const coffeeHouse = await Shop.create({
        id : 6,
        shopName : "Manhattan Mug",
        openAt: "09:00",
        closeAt: "18:00"
    });

    const coffee1 = await Coffee.create({
        coffeeName: 'Cappuccino',
        price: 2.50
    }).then(coffee => {
        coffee.setShop(coffeeHouse);
    });

    const coffee2 = await Coffee.create({
        coffeeName: 'Latte Macchiato',
        price: 3.25
    }).then(coffee => {
        coffee.setShop(coffeeHouse);
    });

    const coffee3 = await Coffee.create({
        coffeeName: 'Espresso',
        price: 3.00
    }).then(coffee => {
        coffee.setShop(coffeeHouse);
    });

    sequelize.close();
  };
  dataTransfer();
});
```

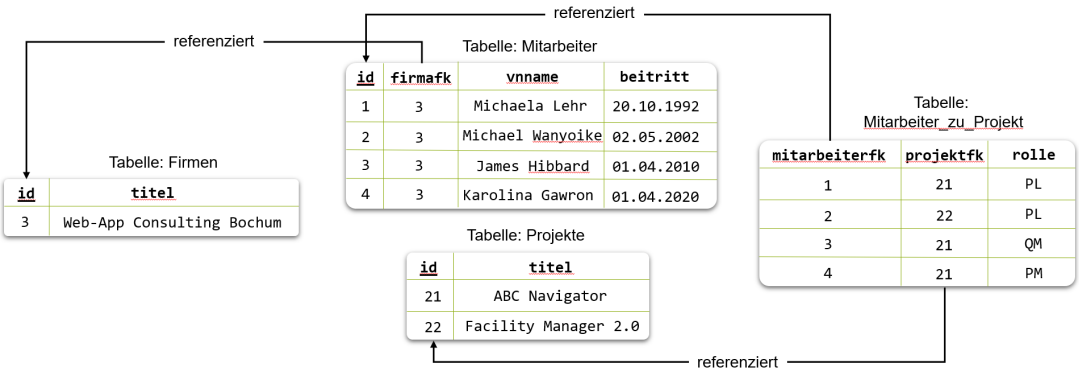
Nachdem der Quellcode über Node ausgeführt wurde, sollte die Datenbank (über `phpMyAdmin` einsehbar) sich verändert haben.

2 Aufgaben

Aufgabe 1

Ein fiktiver Konzern beauftragt Sie eine Datenbank über ihre Tochterunternehmen zu erstellen. Die Datenbank soll ConsultingDB heißen und Informationen über die Firmen, Mitarbeiter und Projekte aufnehmen. Die Tabellen sind über Relationen miteinander verknüpft. Der generelle Aufbau und Beispieldaten sind in Abbildung 2 zu sehen.

Implementieren Sie die Datenbank, indem Sie die aus der Vorlesung bekannten MySQL-Treiber, phpMyAdmin und sequelize.js verwenden. Erstellen Sie dafür zunächst in phpMyAdmin manuell eine leere Datenbank mit dem Namen ConsultingDB. Stellen Sie als nächstes eine Verbindung mittels sequelize zur ConsultingDB her, nutzen Sie dafür den MySQL-Treiber als Dialekt. Verwenden Sie die in der Vorlesung gezeigte sequelize Modell-Schreibweise um



id	firmafk	vnname	beitritt
1	3	Michaela Lehr	20.10.1992
2	3	Michael Wanyoike	02.05.2002
3	3	James Hibbard	01.04.2010
4	3	Karolina Gawron	01.04.2020

id	titel
3	Web-App Consulting Bochum

mitarbeiterfk	projektfk	rolle
1	21	PL
2	22	PL
3	21	QM
4	21	PM

id	titel
21	ABC Navigator
22	Facility Manager 2.0

Abbildung 2: Inhalt von Datenbank ConsultingDB

die Tabellen zu erzeugen und mit Inhalt zu füllen. Nutzen Sie für die Konstruktion der Relationen die Methoden `belongsTo`, `hasMany`, `belongsToMany` und `hasOne`.

Aufgabe 2

Überführen Sie die Tabellen aus der vorherigen Aufgabe nun durch sequelize-cli in einzelne Migrations-Dateien. Definieren Sie für die jeweiligen Dateien die `up` und `down`-Methode. Testen sie ihre Implementierung mit den `db:migrate` und `db:migrate:undo` befehlen. Da die dargestellten Daten aus Abbildung 2 nur als Test eingelesen wurden, soll der Datensatz über Seeds mit dem Befehl `db:seed:all` aufgespielt und wahlweise auch wieder mit `db:seed:undo` entfernt werden.