

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 Fremdschlüsseintrag 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.

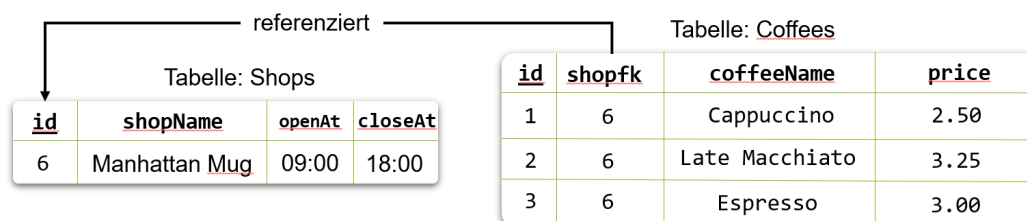


Abbildung 1: Inhalt von Datenbank CoffeeShopDB

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', '', {
5   host: 'localhost',
   dialect: 'mysql'
});*/

// Option 2:
10 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,
5  closeAt: Sequelize.TIME
}, { sequelize, modelName: 'shop' });

class Coffee extends Sequelize.Model {}
Coffee.init({
10  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 *belongsTo* 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, dass 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

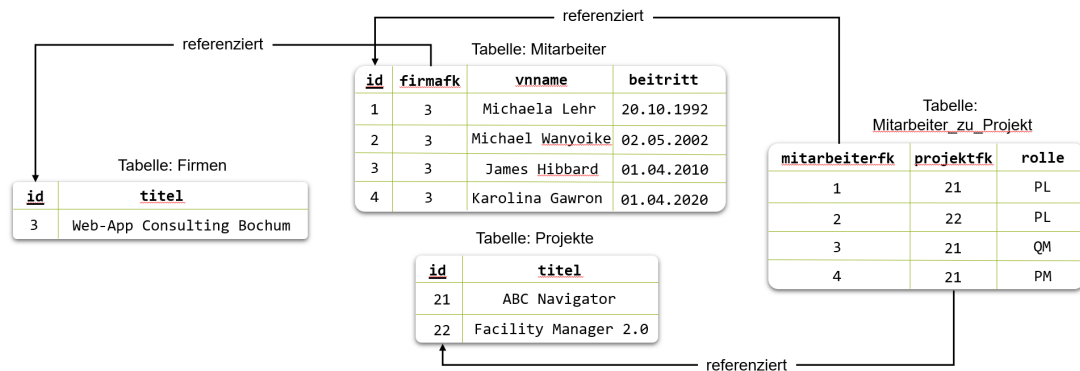


Abbildung 2: Inhalt von Datenbank ConsultingDB

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.

Lösungen zu den Aufgabes

Lösung 2

Listing 1

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

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

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

class Firma extends Sequelize.Model {}
Firma.init({
  titel: Sequelize.STRING
15 }, { sequelize, modelName: 'firma' });

class Mitarbeiter extends Sequelize.Model {}
Mitarbeiter.init({
  vnname: Sequelize.STRING,
  beitritt: Sequelize.DATE
20 }, { sequelize, modelName: 'mitarbeiter' });

class Projekt extends Sequelize.Model {}
Projekt.init({
  titel: Sequelize.STRING
25 }, { sequelize, modelName: 'projekt' });

Firma.hasMany(Mitarbeiter);
Mitarbeiter.belongsTo(Firma);

30 class MitarbeiterProjektRelation extends Sequelize.Model {}
MitarbeiterProjektRelation.init(
  { rolle : Sequelize.STRING },
  { sequelize, modelName: 'mitarbeiter_projekt' }
35 );

Mitarbeiter.belongsToMany(Pjekt, { through: MitarbeiterProjektRelation });
Projekt.belongsToMany(Mitarbeiter, { through: MitarbeiterProjektRelation });

40 sequelize.sync({ force: true }).then(() => {
  let sxnA = async () => {
    const firmaA = await Firma.create({
      id : 3,
      titel : "Web-App Consolting Bochum"
45   });

    const michaela = await Mitarbeiter.create({
      vnname: 'Michaela Lehr',
      beitritt: new Date(1992, 10, 20)
50   }).then(mitarbeiter => {
      mitarbeiter.setFirma(firmaA); //Setzen des Foreign Key muss auf Firma warten
    });
```

```

const michael = await Mitarbeiter.create({
55   vnname: 'Michael Wanyolke',
   beitrirt: new Date(2002, 05, 02)
}).then(mitarbeiter => {
   mitarbeiter.setFirma(firmaA); //Setzen des Foreign Key muss auf Firma warten
});

60
const james = await Mitarbeiter.create({
   vnname: 'James Hibbard',
   beitrirt: new Date(2010, 04, 01)
}).then(mitarbeiter => {
65   mitarbeiter.setFirma(firmaA); //Setzen des Foreign Key muss auf Firma warten
});

const karolina = await Mitarbeiter.create({
   vnname: 'Karolina Gawron',
70   beitrirt: new Date(2020, 04, 01)
}).then(mitarbeiter => {
   mitarbeiter.setFirma(firmaA); //Setzen des Foreign Key muss auf Firma warten
});

75
const projektABCNavi = await Projekt.create({
   id: 21,
   titel: 'ABC Navigator'
});

80
const projektFacility = await Projekt.create({
   id: 22,
   titel: 'Facility Manager 2.0'
});

85
const rolle1 = await MitarbeiterProjektRelation.create({
   rolle: 'PL',
   mitarbeiterId: 1,
   projektId: 21
});

90
const rolle2 = await MitarbeiterProjektRelation.create({
   rolle: 'PL',
   mitarbeiterId: 2,
   projektId: 22
95   });

const rolle3 = await MitarbeiterProjektRelation.create({
   rolle: 'QM',
   mitarbeiterId: 3,
100   projektId: 21
});

const rolle4 = await MitarbeiterProjektRelation.create({
   rolle: 'PM',
105   mitarbeiterId: 4,
   projektId: 21
});

sequelize.close();
110   }; sxnA();
});

/*
115 let synchro = (async () => {
   await Firma.sync({alter: true});

```

```
        await Mitarbeiter.sync({alter: true});
        await Projekt.sync({alter: true});
        await MitarbeiterProjektRelation.sync({alter: true});
120
        //await MitarbeiterProjektRelation.drop();
        //await Mitarbeiter.drop();
        //await Projekt.drop();
        //await Firma.drop();
125
        sequelize.close();
    });
    synchro();
    */
```
