

JOB SHEET

NODE JS

Sequelize Bagian I

Disusun Oleh:
TIM MGMP RPL
SMK TELKOM MALANG

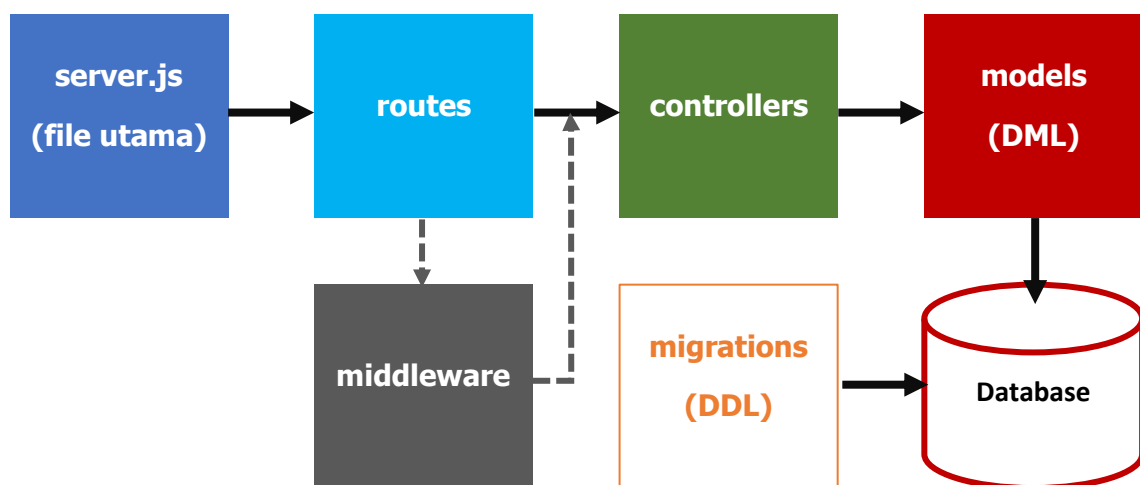
Restful API Node.js dan MySQL menggunakan Sequelize

Pada modul sebelumnya kita telah mempelajari tentang REST API pada Node JS, membuat proses *request* dan *response* pada REST API. Pada modul ini kita akan belajar bagaimana membuat REST API dengan melibatkan *database* MySQL sebagai *data resource*. Terdapat standarisasi dalam pembuatan REST API terutama dalam pemilihan *request method*. Adapun standar dalam pemilihan *request method* adalah sebagai berikut.

1. **GET**, digunakan untuk proses **READ DATA** atau mendapatkan data dari database.
2. **POST**, digunakan untuk proses **CREATE DATA** atau menambahkan data baru ke database.
3. **PUT**, digunakan untuk proses **UPDATE DATA** atau mengubah data pada database.
4. **DELETE**, digunakan untuk proses **DELETE DATA** atau menghapus data pada database.

A. Struktur Proyek

Struktur proyek merupakan salah satu hal yang penting dalam pengembangan sebuah proyek aplikasi. Struktur proyek mengatur pengelompokan file berdasarkan fungsinya agar proyek aplikasi yang dibuat dapat di maintenance dengan baik jika ada perubahan atau perbaikan. Setiap pengembangan aplikasi, tim pengembang mempunyai standar masing-masing untuk menentukan struktur proyek yang akan digunakan. Dalam modul ini kita akan membuat stuktur proyek aplikasi secara umum dan mudah untuk dipelajari. Berikut struktur proyek yang akan digunakan dalam praktikum ini.



Keterangan:

- **Server.js** merupakan file utama yang dieksekusi saat proyek aplikasi akan dijalankan.
- **Routes** merupakan kumpulan jalur endpoint yang disediakan di dalam proyek.
- **Controllers** merupakan kumpulan logic atau proses mengolah request yang diterima dan memberikan response.
- **Models** merupakan kumpulan proses manipulasi data ke database meliputi *create*, *read*, *update*, dan *delete*.
- **Migration** merupakan kumpulan proses pembuatan struktur database meliputi struktur table dan relasinya.
- **Middleware** merupakan kumpulan proses yang menjembatani antara route dan controller. Proses *middleware* ini bersifat opsional (bisa didefinisikan atau tidak)

B. Sequelize

Sequelize merupakan sebuah library yang digunakan untuk melakukan interaksi dengan database. Interaksi yang dapat dilakukan mulai dari pembuatan struktur dan manipulasi data pada database. Sequelize berbasis ORM (Object Relational Mapping) dan berbasis promise, sehingga proses kueri ke database berorientasi pada object. Sequelize sudah mendukung untuk MySQL, PostgreSQL, MariaDB, SQLite, dan MSSQL.

Pemanfaatan library sequelize pada project yang kita kembangkan diawali dengan mengeksekusi perintah berikut pada terminal.

```
> npx sequelize-cli init
```

Ketika perintah di atas dijalankan, maka pada folder project kita akan tercipta folder berikut.

- Folder **config**, berisi file konfigurasi untuk menunjukkan database mana yang akan terhubung pada project tersebut.
- Folder **models**, berisi file model yang digunakan untuk manipulasi data pada database meliputi perintah CRUD.
- Folder **migrations**, berisi file migration yang digunakan untuk membuat struktur database meliputi pembuatan atau penghapusan tabel, relasi antar tabel, dan pengubahan struktur database lainnya.
- Folder **seeders**, berisi file seeder yang digunakan untuk pembenihan data pada database (inisiasi data awal).

Pada modul praktikum ini, silakan membuat folder project baru dengan nama **"school_library"**. Setelah itu lakukan inisiasi package.json pada folder project ini dengan perintah berikut.

```
D:\Nodejs\school_library>npm init --y
```

Setelah itu buka file package.json yang tercipta hasil proses di atas. Pada bagian "scripts" tambahkan key "start" dengan value "nodemon server.js".

```
"scripts": {  
  "start": "nodemon server.js",  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```




Kemudian instalasi library yang dibutuhkan pada project tersebut meliputi library **express, mysql2, sequelize, cors, md5**.

```
D:\Nodejs\school_library>npm install express cors mysql2 sequelize md5
```

Setelah library terinstall pada project tersebut, selanjutnya adalah inisiasi sequelize pada project ini dengan cara menjalankan perintah berikut.

```
D:\Nodejs\school_library>npx sequelize-cli init
```

Setelah dijalankan akan tercipta folder seperti berikut.

 config	5/10/2022 10:51 AM	File folder
 migrations	5/10/2022 10:51 AM	File folder
 models	5/10/2022 10:51 AM	File folder
 node_modules	5/10/2022 9:36 AM	File folder
 seeders	5/10/2022 10:51 AM	File folder

1. Sequelize Configuration

Pada folder config, terdapat file **config.json** yang berisikan code seperti berikut ini.

```
{
  "development": {
    "username": "root",
    "password": null,
    "database": "database_development",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

Pada Sequelize CLI secara default menggunakan DBMS MySQL, jika ingin mengganti dengan DBMS yang lain, dapat mengubah isi dari pilihan "dialect". Selain itu, file ini mengatur database yang terhubung pada project yang dikembangkan dengan mengganti konfigurasi pada *host*, *username*, *password* untuk masuk ke DBMS dan konfigurasi *database* untuk menentukan nama database yang terhubung.

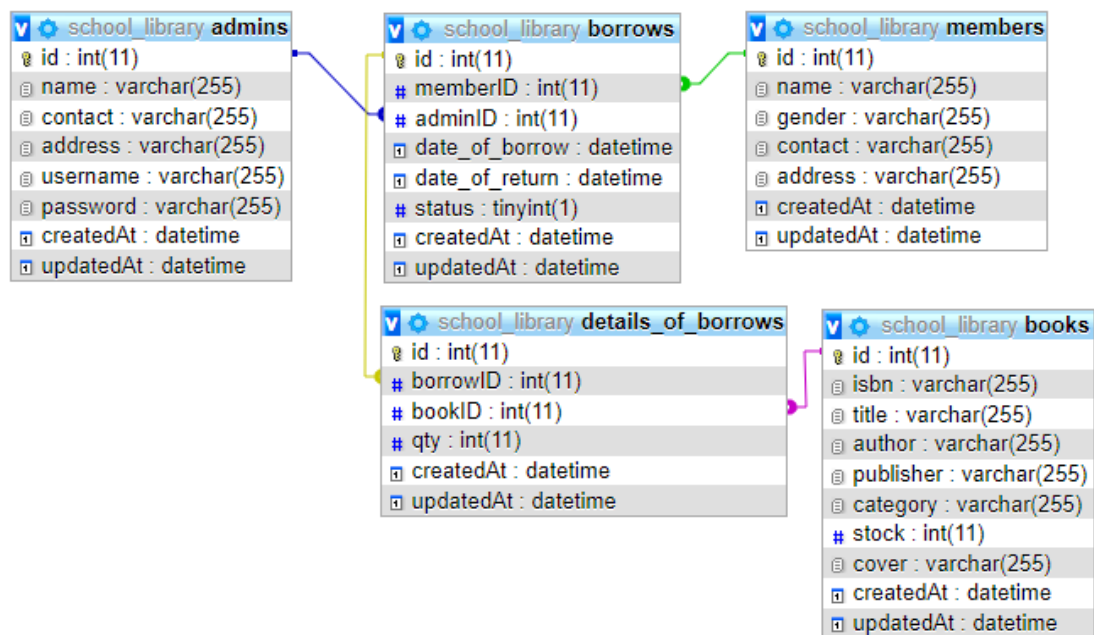
Dalam kasus ini silakan membuat database baru dengan nama "**school_library**" melalui phpMyAdmin. Setelah terbuat database tersebut silakan untuk mengubah value dari key "database" menjadi "school_library".

```
"development": {
  "username": "root",
  "password": null,
  "database": "school_library",
  "host": "127.0.0.1",
  "dialect": "mysql"
},
```

2. Sequelize Migration and Model

Migration merupakan proses untuk mengatur struktur database seperti table dan relasi, sedangkan Model merupakan proses untuk mengatur data yang tersimpan pada database.

Dalam kasus ini kita akan membuat struktur tabel seperti berikut ini.



Struktur tabel yang dibuat terlebih dahulu adalah tabel yang bersifat parent (tidak ada relasi ke tabel yang lain)

- Pembuatan file migration dan model tabel "book"

Eksekusi perintah berikut ini pada terminal

```
> npx sequelize-cli model:generate --name book
--attributes
isbn:string,title:string,author:string,publisher:string,category:string,stock:integer,cover:string
```

Jika berhasil dieksekusi maka akan tercipta file migration dan model untuk tabel "book".

```
New model was created at D:\Nodejs\school_library\models\book.js .
New migration was created at D:\Nodejs\school_library\migrations\20220510043132-create-book.js .
```

Buka file migration untuk tabel “**book**” pada **folder migrations**. Dalam file tersebut terdapat dua function yang telah tergenerate yaitu **function up()** dan **function down()**.

```
async up(queryInterface, Sequelize) {
  await queryInterface.createTable('books', {
```

Pada **function up()** didefinisikan sebagai pembuatan tabel “books” yang didalamnya telah degenerate struktur kolom untuk table tersebut. Pada penamaan table terdapat tambahan huruf “s” di akhir nama tabel yang menunjukkan suatu **kemajemukan** dalam aturan Bahasa Inggris. Hal tersebut karena suatu tabel akan berisi banyak baris data yang tersimpan di dalamnya. Selain itu di dalam function up() terdapat kolom “**id**” sebagai *primary key* yang secara default tergenerate saat pembuatan file migration. Oleh karena itu saat pembuatan file migration dan model, kita tidak perlu mendenisikan kolom untuk *primary key*-nya.

Function up() atau *create table* akan dijalankan saat kita mengeksekusi file migration dengan perintah

```
> npx sequelize-cli db:migrate
```

Jika perintah di atas dijalankan maka akan memberikan output seperti berikut.

```
D:\Nodejs\school_library>npx sequelize-cli db:migrate

Sequelize CLI [Node: 17.2.0, CLI: 6.4.1, ORM: 6.19.0]

Loaded configuration file "config\config.json".
Using environment "development".
== 20220510043132-create-book: migrating =====
== 20220510043132-create-book: migrated (0.100s)
```

Dari output di atas menunjukkan bahwa file migration untuk tabel “book” telah dieksekusi dan di database akan tercipta tabel “books” dengan struktur yang telah didefinisikan di file migrate.

#	Name	Type
1	id 	int(11)
2	isbn	varchar(255)
3	title	varchar(255)
4	author	varchar(255)
5	publisher	varchar(255)
6	category	varchar(255)
7	stock	int(11)
8	cover	varchar(255)
9	createdAt	datetime
10	updatedAt	datetime

Selain function `up()` di file migration, terdapat function `down()` yang digunakan untuk menghapus struktur yang telah dibuat (dalam konteks ini function `down` digunakan untuk menghapus tabel).

```
async down(queryInterface, Sequelize) {  
  await queryInterface.dropTable('books');  
}
```

Function `down()` akan dijalankan ketika kita melakukan **undo** terhadap proses migration. Proses undo migration dilakukan dengan menjalankan perintah berikut

```
> npx sequelize-cli db:migrate:undo
```

Perintah di atas akan melakukan undo terhadap proses migration yang terakhir dilakukan. Jika ingin melakukan proses undo terhadap semua proses migration yang telah dilakukan, dapat menggunakan perintah

```
> npx sequelize-cli db:migrate:undo:all
```



```
D:\Nodejs\school_library>npx sequelize-cli db:migrate:undo:all

Sequelize CLI [Node: 17.2.0, CLI: 6.4.1, ORM: 6.19.0]

Loaded configuration file "config\config.json".
Using environment "development".
== 20220511232729-create-admin: reverting =====
== 20220511232729-create-admin: reverted (0.138s)
```

- Pembuatan file migration dan model tabel "member"

```
> npx sequelize-cli model:generate --name
member --attributes
name:string,gender:string,contact:string,addresses:string
```

Jika berhasil dieksekusi maka akan tercipta file migration dan model untuk tabel "member".

```
New model was created at D:\Nodejs\school_library\models\member.js .
New migration was created at D:\Nodejs\school_library\migrations\20220511232645-create-member.js .
```

- Pembuatan file migration dan model tabel "admin"

```
> npx sequelize-cli model:generate --name
admin --attributes
name:string,contact:string,address:string,user
name:string,password:string
```

Jika berhasil dieksekusi maka akan tercipta file migration dan model untuk tabel "admin".

```
New model was created at D:\Nodejs\school_library\models\admin.js .
New migration was created at D:\Nodejs\school_library\migrations\20220511232729-create-admin.js .
```

- Pembuatan file migration dan model tabel "borrow"

```
> npx sequelize-cli model:generate --name
borrow --attributes
memberID:integer,adminID:integer,date_of_borrow:date,date_of_return:date,status:boolean
```

Jika berhasil dieksekusi maka akan tercipta file migration dan model untuk tabel "borrow".

Sequelize CLI [Node: 17.2.0, CLI: 6.4.1, ORM: 6.19.0]

New model was created at D:\Nodejs\school_library\models\borrow.js .

New migration was created at D:\Nodejs\school_library\migrations\20220512001555-create-borrow.js .

Karena pada tabel "borrow" terdapat relasi dengan tabel lain (tabel member dan admin), kita akan memodifikasi file migration dari tabel "borrow" untuk menghubungkan *relationship*-nya.

```
memberID: {
  type: Sequelize.INTEGER,
  allowNull: false,
  references: {
    model: "members",
    key: "id",
  },
},
adminID: {
  type: Sequelize.INTEGER,
  allowNull: false,
  references: {
    model: "admins",
    key: "id"
  }
},
```

Pada bagian attribute "memberID" kita menambahkan key "allowNull" bernilai false yang berarti kolom memberID tidak boleh dikosongkan datanya. Selain itu terdapat tambahan key "references" yang digunakan untuk membuat relasi ke tabel "members" dengan *column reference*-nya adalah "id".

Pada bagian attribute "adminID" kita menambahkan key "allowNull" bernilai false yang berarti kolom adminID tidak boleh dikosongkan datanya. Selain itu

terdapat tambahan key "references" yang digunakan untuk membuat relasi ke tabel "admins" dengan *column reference*-nya adalah "id".

- Pembuatan file migration dan model tabel "details_of_borrow"

```
> npx sequelize-cli model:generate --name  
details_of_borrow --attributes  
borrowID:integer,bookID:integer,qty:integer
```

Jika berhasil dieksekusi maka akan tercipta file migration dan model untuk tabel "details_of_borrow".

```
Sequelize CLI [Node: 17.2.0, CLI: 6.4.1, ORM: 6.19.0]  
  
New model was created at D:\Nodejs\school_library\models\details_of_borrow.js .  
New migration was created at D:\Nodejs\school_library\migrations\20220512002415-create-details-of-borrow.js .
```

Karena pada tabel "details_of_borrow" terdapat relasi dengan tabel lain (tabel borrows dan books), kita akan memodifikasi file migration dari tabel "details_of_borrow" untuk menghubungkan *relationship*-nya.

```
borrowID: {  
  type: Sequelize.INTEGER,  
  allowNull: false,  
  references: {  
    model: "borrows",  
    key: "id"  
  }  
},  
bookID: {  
  type: Sequelize.INTEGER,  
  allowNull: false,  
  references: {  
    model: "books",  
    key: "id"  
  }  
},
```

Pada bagian attribute "borrowID" kita menambahkan key "allowNull" bernilai false yang berarti kolom borrowID tidak boleh dikosongkan datanya. Selain itu terdapat tambahan key "references" yang digunakan untuk membuat relasi ke tabel "borrows" dengan *column reference*-nya adalah "id".

Pada bagian attribute "bookID" kita menambahkan key "allowNull" bernilai false yang berarti kolom bookID tidak boleh dikosongkan datanya. Selain itu terdapat tambahan key "references" yang digunakan untuk membuat relasi ke tabel "books" dengan *column reference*-nya adalah "id".

Setelah semua file migration kita siapkan, selanjutnya kita akan mengeksekusi semua file migration dengan perintah

```
> npx sequelize-cli db:migrate
```

Kemudian lihatlah pada database untuk memastikan struktur tabel dan relasi telah diciptakan.

3. Sequelize Seeder

Seeder merupakan sebuah proses untuk memberikan sample data pada struktur database yang telah dibuat sebelumnya. Data sample digunakan sebagai data testing yang akan dimanage dalam pengembangan project. Dalam praktikum ini kita akan menambahkan sampling data pada tabel "members".

```
> npx sequelize-cli seed:generate --name sample-members
```

Jika berhasil dieksekusi maka akan tercipta file seeder untuk tabel "members". Kemudian buka file seeder member pada folder **seeders** dan lakukan modifikasi seperti berikut ini.

```
'use strict';

module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.bulkInsert("members", [
      {
        name: `Soekarno`, gender: `Male`,
        contact: `021-223311`, address: `Tokyo, Japan`,
        createdAt: new Date(), updatedAt: new Date()
      },
      {
        name: `Soeharto`, gender: `Male`,
        contact: `0331-474747`, address: `Beijing, China`,
        createdAt: new Date(), updatedAt: new Date()
      },
      {
        name: `Megawati`, gender: `Female`,
        contact: `091-23981`, address: `Bangkok, Thailand`,
        createdAt: new Date(), updatedAt: new Date()
      },
    ], {}),
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.bulkDelete('members', null, {});
  }
};
```

Pada file seeder, terdapat dua fungsi utama yaitu function up() dan function down(). Function up() digunakan untuk menambahkan data pada sebuah tabel

sesuai dengan struktur dari tabel tersebut. Function `up()` akan dijalankan saat kita mengeksekusi perintah berikut.

```
> npx sequelize-cli db:seed:all
```

Setelah perintah di atas dijalankan, silakan untuk mengecek isi tabel "members".

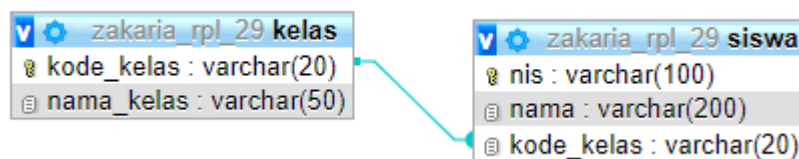
Function `down()` digunakan untuk menghapus semua data pada sebuah tabel.

Function `down()` akan dieksekusi saat kita menjalankan perintah berikut.

```
> npx sequelize-cli db:seed:undo:all
```

4. Sequelize Relationship

Pada library Sequelize disediakan sebuah fitur untuk menghubungkan antar dua tabel melalui file model yang disebut *association*. Sebelum lebih lanjut membahas tentang *association*, kita akan membahas tentang status tabel dalam sebuah hubungan. Perhatikan contoh hubungan dua tabel berikut.



Berdasarkan tabel di atas, ada relasi antara tabel siswa dan tabel kelas yang dihubungkan antara kolom "kode_kelas" di tabel kelas dan kolom "kode_kelas" di tabel siswa. Dari sisi tabel "kelas" maka akan tercipta relasi "1 data kelas mempunyai banyak data siswa" (*Each class **has many** students*). Sedangkan dari sisi tabel "siswa" akan tercipta relasi "1 data siswa hanya dimiliki oleh 1 data kelas" (*Each student **belongs to** one class*). Selain itu dari relasi di atas, tabel "kelas" berlaku sebagai tabel **parent** karena kolom "kode_kelas" sebagai **primary key** di tabel tersebut, sedangkan tabel "siswa" berlaku sebagai tabel **child** karena kolom "kode_kelas" berlaku sebagai **foreign key** di tabel tersebut.

Dalam Library Sequelize terdapat beberapa tipe *association* yang disediakan yaitu sebagai berikut.

- `hasOne`, relasi ini digunakan untuk hubungan bertipe "one to one" dari tabel *parent* ke tabel *child*.
- `belongsTo`, relasi ini digunakan untuk hubungan bertipe "one to one" dari tabel *child* ke tabel *parent*.

- `hasMany`, relasi ini digunakan untuk hubungan bertipe “*one to many*” dari tabel *parent* ke tabel *child*.
- `belongsToMany`, relasi ini digunakan untuk hubungan bertipe “*one to many*” dari tabel *child* ke tabel *parent*.

Dalam project ini terdapat beberapa relasi antar tabel yang telah kita buat

- Relasi tabel “`admins`” dan tabel “`borrows`” dengan key “`id`” dari tabel “`admins`” dan key “`adminID`” dari tabel “`borrows`”. Dari sisi tabel “`admins`”, relasi yang terjadi adalah “*each admin **has many** borrowed books*”. Oleh karena itu di file model “`admin`” kita akan menambahkan code untuk membuat relasi tersebut.

```
static associate(models) {
  // define association here
  this.hasMany(models.borrow, {
    foreignKey: `adminID`, as: "borrowed"
  })
}
```

Sedangkan relasi dari sisi tabel “`borrows`”, relasi yang terjadi adalah “*each borrowed book **belongs to** one admin*”. Oleh karena itu di file model “`borrow`” kita akan menambahkan code untuk mengimplementasi relasi tersebut.

```
static associate(models) {
  // define association here
  this.belongsTo(models.admin)
}
```

- Relasi tabel “`members`” dan tabel “`borrows`” dengan key “`id`” dari tabel “`members`” dan key “`memberID`” dari tabel “`borrows`”. Dari sisi tabel “`members`”, relasi yang terjadi adalah “*each member **has many** borrowed books*”. Oleh karena itu di file model “`member`” kita akan menambahkan code untuk membuat relasi tersebut.

```
static associate(models) {
  // define association here
  this.hasMany(models.borrow, {
    foreignKey: `memberID`, as: "borrow"
  })
}
```

Sedangkan relasi dari sisi tabel “`borrows`”, relasi yang terjadi adalah “*each borrowed book **belongs to** one member*”. Oleh karena itu di file model

“borrow” kita akan menambahkan code untuk mengimplementasi relasi tersebut.

```
static associate(models) {
  // define association here
  this.belongsTo(models.admin)
  this.belongsTo(models.member)
}
```

- Relasi tabel “borrows” dan tabel “details_of_borrows” dengan key “id” dari tabel “borrows” dan key “borrowID” dari tabel “details_of_borrows”. Dari sisi tabel “borrows”, relasi yang terjadi adalah “*each borrowed book **has many** details of borrowed books*”. Oleh karena itu di file model “borrow” kita akan menambahkan code untuk membuat relasi tersebut.

```
static associate(models) {
  // define association here
  this.belongsTo(models.admin)
  this.belongsTo(models.member)
  this.hasMany(models.details_of_borrow, {
    foreignKey: `borrowID`, as: `details_of_borrow`
  })
}
```

Sedangkan relasi dari sisi tabel “details_of_borrows”, relasi yang terjadi adalah “*each detail borrowed book **belongs to** one data of borrowed book*”. Oleh karena itu di file model “details_of_borrow” kita akan menambahkan code untuk mengimplementasi relasi tersebut.

```
static associate(models) {
  // define association here
  this.belongsTo(models.borrow)
}
```

- Relasi tabel “books” dan tabel “details_of_borrows” dengan key “id” dari tabel “books” dan key “bookID” dari tabel “details_of_borrows”. Dari sisi tabel “books”, relasi yang terjadi adalah “*each book **has many** details of borrowed books*”. Oleh karena itu di file model “book” kita akan menambahkan code untuk membuat relasi tersebut.


```
static associate(models) {  
  // define association here  
  this.hasMany(models.details_of_borrow, {  
    foreignKey: `bookID`, as: `details_of_borrow`  
  })  
}
```

Sedangkan relasi dari sisi tabel "details_of_borrows", relasi yang terjadi adalah "each detail of borrowed book **belongs to** one book". Oleh karena itu di file model "details_of_borrow" kita akan menambahkan code untuk mengimplementasi relasi tersebut.

```
static associate(models) {  
  // define association here  
  this.belongsTo(models.borrow)  
  this.belongsTo(models.book)  
}
```