CADT IDT

W7 PRACTICE

Sequelize - Part 1

At the end of his practice, you should be able to…

- Sequelize Basics
- CRUD Operations
- 1–1 (One-to-One) Relationships
- 1–* (One-to-Many) Relationships

How to start?

- ✓ Download **start code** from related MS Team assignment
- ✓ Run `npm install` on both front and back projects
- ✓ Run `npm run dev` on both front and back projects to run the client and the server

How to submit?

- ✓ Submit your **code** on MS Team assignment

Are you lost?

OFFICAL DOCUMENTATIONS
https://sequelize.org/docs/v6

TUTORIALS
https://www.digitalocean.com/community/tutorials/how-to-use-sequelize-with-node-js-and-mysql

VIDEOS
https://www.youtube.com/watch?v=YNyGD4rakmc
https://www.youtube.com/watch?v=3_9-JFXTVDE
https://www.youtube.com/watch?v=ZAk1YKzKkL4

*EXERCISE 1 – **Fix broken codes***

Your goal on the bellow questions is to diagnose common **Sequelize relationship mistakes**.

Q1 - Broken Code 1

```
User.hasOne(Profile);
await sequelize.sync();

const profile = await Profile.create({ bio: 'Test' });
const user = await profile.createUser({ username: 'joe' });
```

What is the problem ? Fix it

```
The association method createUser() is not available from Profile because
User.hasOne(Profile) creates the method createProfile() on User, not the
other way around.

Fix:
User.hasOne(Profile);
Profile.belongsTo(User);
await sequelize.sync();

const user = await User.create({ username: 'joe' });
const profile = await user.createProfile({ bio: 'Test' });
```

Q2 - Broken Code 2

```
Book.hasMany(Author);

await sequelize.sync();
const author = await Author.create({ name: 'Samnang' });
const book = await author.createBook({ title: 'Wrong Way' })
```

What is the problem ? Fix it

```
The relationship is reversed. Book.hasMany(Author) implies
book.createAuthor(), but the code uses author.createBook().

Fix:
Author.hasMany(Book);
Book.belongsTo(Author);
await sequelize.sync();

const author = await Author.create({ name: 'Samnang' });
const book = await author.createBook({ title: 'Correct Way' });
```

Q3 - Broken Code 3

```
User.hasOne(Profile);
Profile.belongsTo(User);
```

```
const user = await User.create({ username: 'Jon' });
const profile = await Profile.create({ bio: 'hello' });

await user.addProfile(profile);
```

What is the problem ? Fix it

```
The method addProfile() is for hasMany associations. Since it's a one-to-one,
use setProfile() instead.

Fix:
User.hasOne(Profile);
Profile.belongsTo(User);

const user = await User.create({ username: 'Jon' });
const profile = await Profile.create({ bio: 'hello' });

await user.setProfile(profile);
```

Q4 - Broken Code 4

```
Employee.hasOne(Manager);
Manager.hasOne(Employee);
```

What is the problem ? Fix it

```
Employee.hasOne(Manager) and Manager.hasOne(Employee) creates a confusing and
incorrect circular relationship. Use a self-reference instead.

Fix:
Employee.hasOne(Employee, { as: 'Manager', foreignKey: 'managerId' });
```

*EXERCISE 2 – **Author & Books***

We want to manage Author and Books

An author can write many books, but a book is written by one author.

🎯 **In this exercise, you will define Sequelize models, create sample data, and perform some queries.**

Q1 - Define the **models** and their **relationships**

```
Author:
      name: string
      birthYear: integer
Book:
      title: string
      publicationYear: integer
      pages: integer
```

Q2 - Create sample data

Create 3 authors:

- "Ronan The Best" (born 1990)
- "Kim Ang" (born 1995)
- "Hok Tim" (born 2015)

Each author should have at least 2 books. Use a mix of publication years and page count

Q3 - Queries

- Fetch all books by a given author.
- Create a new book for an existing author using .createBook().
- List all authors along with their books (include).

*EXERCISE 3 – **Attendance Tracker***

You're building an attendance system with these models:

- **Student**
- **Class**
- **AttendanceRecord** *(tracks each student's attendance per day)*

Q1 - Define **the 3 models** and their properties

Q2 - Define the **relationships** between the 3 tables (belongto, hasOne, hasMany)

Q3 - Write code to:

- Mark attendance for a student in a class on a given date
- Get attendance for a student on a specific date
- List attendance for all students in a class
- Get attendance summary for a student

Q4 - Develop a functional **REST API** for an attendance system involving:

POST	`/attendance?studentId=1&date=2025-06-17`	Mark attendance for a student in a class on a given date
GET	`/attendance?studentId=1&date=2025-06-17`	Get attendance for a student on a specific date
GET	`/classes/:id/attendance`	List attendance for all students in a class
GET	`/students/:id/attendance`	Get attendance summary for a student