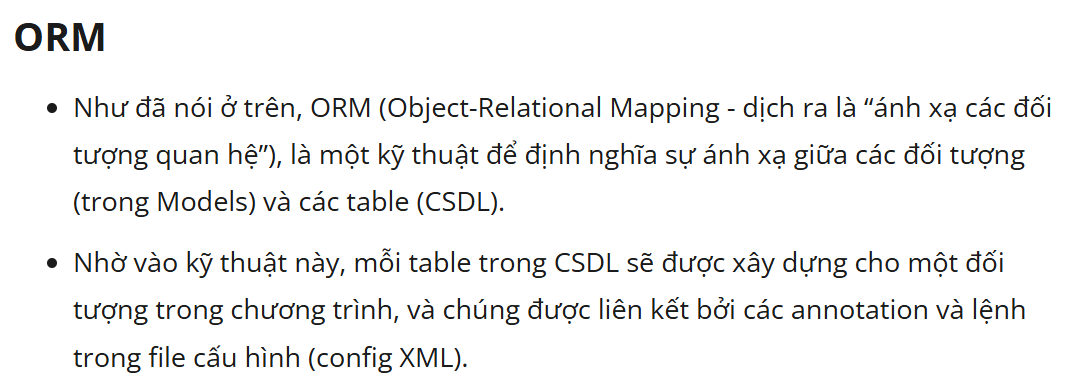
Sequelize

# 1, Giới thiệu qua

* Sequelize là ORM (Object Relational Mapping) dùng cho server Nodejs.
* Nó giống chức năng của Hibernate bên Java.



(trích)

* Đọc về Hibernate ở đây: <https://viblo.asia/p/nhat-ky-hoc-hibernate-tu-so-0-7ymJXGQa4kq>
* Hiểu ngắn gọn hơn, ORM hay ở đây là sequelize, là một cách thức để liên kết database với server nodejs.

# 2, Cài đặt

* Đảm bảo đã cài Nodejs.
* Cài với npm/yarn: $ npm install –save sequelize
* Cài thêm mysql2: $ npm install –save mysql2
* Sau đó nếu yêu cầu tải thêm cái gì thì npm install nốt cái đó.

**Các bước tiếp theo: bây giờ cần thiết lập và test kết nối với DB, sau đó định nghĩa các Models và mối quan hệ (association), đồng bộ hóa models và database (sync), sau đó thực hiện CRUD.**

# 3, Kết nối với Database

* Vì ở đây đang trong trường hợp có Database sẵn nên sẽ làm theo hướng có sẵn.

Các bước:

* Import thư viện
* Tạo instance Sequelize
* Thử kết nối. (tùy chọn)

**Phần kết nối này sẽ trả về một instance của đối tượng Sequelize (đại diện cho kết nối tới database)**

Phần kết nối với database sẽ được viết ở src/config/dbconnect.js.

|  |
| --- |
| /\* Import thư viện\*/  const { Sequelize } = require('sequelize');  /\*tạo instance Sequelize\*/  // Cách 1: Kết nối bằng URI  const sequelize1 = new Sequelize('mysql://username:password@localhost:3306/databasename');  // Cách 2: Truyền tham số riêng lẻ  const sequelize2 = new Sequelize('databasename', 'username', 'password', {  host: 'localhost', // Địa chỉ máy chủ MySQL  dialect: 'mysql', // Sử dụng MySQL làm cơ sở dữ liệu  logging: false, // Tắt log câu lệnh SQL (tuỳ chọn)  pool: {  max: 5, // Số kết nối tối đa trong pool  min: 0, // Số kết nối tối thiểu trong pool  acquire: 30000, // Thời gian tối đa để lấy một kết nối (ms)  idle: 10000 // Thời gian tối đa một kết nối có thể ở trạng thái nhàn rỗi (ms)  }  });  // Kiểm tra kết nối  (async () => {  try {  await sequelize2.authenticate();  console.log('Kết nối MySQL thành công!');  } catch (error) {  console.error('Kết nối MySQL thất bại:', error);  }  })(); |

Đây là cú pháp mẫu, còn code thật thì thêm data vào (từ file env) và xuất modules

|  |
| --- |
| //khai báo thư viện sequelize và dotenv  const { Sequelize } = require('sequelize');  require('dotenv').config();  //tạo instance sequelize  const sequelize = new Sequelize(process.env.DB\_NAME, process.env.DB\_USER, process.env.DB\_PASSWORD, {    host: process.env.DB\_HOST || 'localhost',    dialect: 'mysql',    logging: false,  });  // Kiểm tra thử kết nối  (async () => {    try {      await sequelize.authenticate();      console.log('Kết nối cơ sở dữ liệu thành công!');    } catch (error) {      console.error('Không thể kết nối cơ sở dữ liệu:', error);    }  })();  module.exports = sequelize; |

Với nội dung file /env như sau:

|  |
| --- |
| DB\_NAME=tutorium  DB\_USER=konan  DB\_PASSWORD=1234 |

# 4, Tạo Models

* Một “model” là một đối tượng (trong code) đại diện cho một bảng trong database.
* Một model là một class extend class [Model](https://sequelize.org/api/v6/class/src/model.js~model): là một abstract class xây dựng để đại diện cho một bảng trong cơ sở dữ liệu.
* Tạm thời biết như thế và không cần suy xét kỹ class Model có những gì.
* Bây giờ cần mô phỏng các bảng trong database bằng code js với sequelize.

Đầu tiên phải biết được quy tắc đặt tên các đối tượng.

## 1, đặt tên Model và Table

* Tên bảng: ở dạng số nhiều, dùng chữ thường, dùng snake\_case và tránh những từ khóa sql.
* Tên model: ở dạng số ít và PascalCase, không cần hậu tố "model"

## 2, xây dựng Model

* Có 2 cách để tạo ra một model: một là dùng hàm có sẵn của instance sequelize, 2 là tạo class để extend class tên là Model (như ở trên nói).
* 2 cách này đều cho ra kết quả như nhau nhưng xét về cấu trúc code thì cách 2 phù hợp hơn cho code nhiều Model => chốt cách 2 và không để ý tới cách 1 nữa.

Các bước để code ra một class model:

* Import thư viện và instance của sequelize đã có được sau bước 1.
* Định nghĩa class model (new class)
* Init model đó với những đặc điểm của bảng tương ứng.

**Kết quả của phần tạo model sẽ là một/những class model đại diện cho các bảng trong database (chưa có kết nối giữa các bảng).**

Code mẫu cho một model **(nhớ đặt tên cho chuẩn)**

|  |
| --- |
| /\* Import thư viện và instance sequellize đã tạo thành công ở bên trên\*/  const { Sequelize, DataTypes, Model } = require('sequelize');  const sequelize = require('../config/db’); // Import kết nối Sequelize từ file db đã code bên trên  /\* Định nghĩa class Model \*/  class ModelName extends Model {}  /\* Khởi tạo Model với các thuộc tính \*/  ModelName.init({  id: {  type: DataTypes.INTEGER,  autoIncrement: true,  primaryKey: true,  },  column1: {  type: DataTypes.STRING,  //dưới này là các thuộc tính  },  /\*các cột khác…\*/  }, {  // phần này là thuộc tính cấu hình của model  sequelize, // Instance của Sequelize  modelName: 'ModelName', // Tên Model (bảng trong database sẽ là 'ModelNames')  tableName: 'custom\_table\_name', // Nếu muốn đặt tên bảng tùy chỉnh  freezeTableName: 'true’, // Sequelize sẽ không tự động đổi tên bảng thành số nhiều  });  module.exports = ModelName; |

Vd cụ thể:

Với table User như sau:

|  |
| --- |
| CREATE TABLE `Users`(  `userid` BIGINT NOT NULL AUTO\_INCREMENT PRIMARY KEY,  `username` VARCHAR(255) NOT NULL UNIQUE,  `password` VARCHAR(255) NOT NULL,  `email` VARCHAR(255) NOT NULL UNIQUE,  `imgurl` TEXT NOT NULL,  `dateofbirth` DATE NOT NULL  ); |

Thì ta sẽ có file src/models/User.js như sau:

|  |
| --- |
| const { Sequelize, DataTypes, Model } = require('sequelize');  const sequelize = require('../config/database'); // Import kết nối Sequelize từ file cấu hình  class User extends Model {}  User.init({      userid: {          type: DataTypes.BIGINT,          allowNull: false,          autoIncrement: true,          primaryKey: true      },      username: {          type: DataTypes.STRING(255),          allowNull: false,          unique: true      },      password: {          type: DataTypes.STRING(255),          allowNull: false      },      email: {          type: DataTypes.STRING(255),          allowNull: false,          unique: true      },      imgurl: {          type: DataTypes.TEXT,          allowNull: false      },      dateofbirth: {          type: DataTypes.DATEONLY,          allowNull: false      }  }, {      sequelize, // Instance của Sequelize      modelName: 'User', // Tên Model      tableName: 'Users', // Đảm bảo tên bảng đúng với CSDL      timestamps: false //không có createdAt, updatedAt  });  module.exports = User; |

Bonus: các thuộc tính cấu hình Model đôi khi cần thiết và ảnh hưởng tới cách xây dựng bảng, vd như nếu như bảng có sẵn cột create\_at và update\_at thì phải tắt chức năng tự động sinh cột đó và chỉ định cột đó trong phần thuộc tính. (timestamps: false) *(Không khuyến khích dùng bh vì sẽ gây rối)*

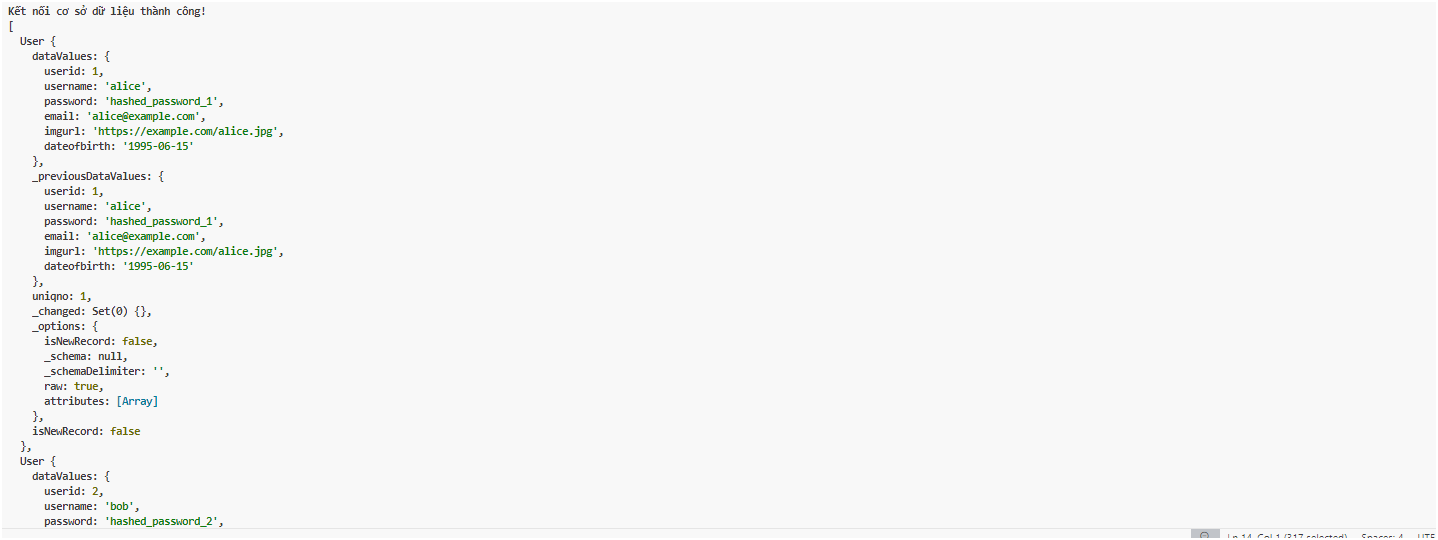
Các thuộc tính cấu hình hay dùng:

| **Thuộc tính** | **Ý nghĩa** |
| --- | --- |
| **Cấu hình cơ bản** | |
| sequelize | Instance của Sequelize (bắt buộc). |
| modelName | Tên model trong Sequelize (không phải tên bảng). |
| tableName | Tên bảng trong database (nếu không đặt, Sequelize sẽ tự động suy ra). |
| freezeTableName | Mặc định false. Nếu true, Sequelize sẽ không tự động chuyển tên bảng thành số nhiều. |
| timestamps | Mặc định true. Nếu false, Sequelize sẽ không tự động tạo createdAt, updatedAt. |
| **Cấu hình timestamps (ngày tháng tự động cập nhật)** | |
| createdAt | Định nghĩa tên cột createdAt, nếu bảng đã có sẵn cột này. |
| updatedAt | Định nghĩa tên cột updatedAt, nếu bảng đã có sẵn cột này. |
| deletedAt | Định nghĩa tên cột deletedAt khi sử dụng xóa mềm (Soft Delete). |
| paranoid | Mặc định false. Nếu true, khi gọi .destroy(), dữ liệu không bị xóa mà chỉ cập nhật deletedAt. |

Để test xem mình đã viết đúng hay chưa, thử chạy một lệnh đơn giản với model vừa tạo:

|  |
| --- |
| const User = require('../models/User'); // Import model vừa code bên trên  const getAllUsers = async (req, res) => {      try {  //thử lấy tất cả các User          const users = await User.findAll();          console.log(users)      } catch (err) {          console.error("Lỗi khi lấy dữ liệu:", err);          console.log("err")      }  };  getAllUsers(); |

Nếu tốt đẹp thì sẽ ra kết quả như sau:

Chú ý: vì đang code rất đơn giản nên lỗi sai hay đến từ việc sai chính tả.

Đây mới chỉ là một model riêng lẻ, chưa có kết nối gì với các model khác. Việc kết nối viết ở phần sau.

Đọc thêm: <https://sequelize.org/docs/v6/core-concepts/model-basics/>

Ở đây, một model được gọi là một DAO (Data Access Object).

## 3, Tương tác với data bằng model vừa code

Khi muốn tạo mới data, thông thường sẽ tạo một biến (new User() chẳng hạn) rồi thêm vào database (persist). Nguyên tắc làm việc của sequelize cũng vậy, nhưng hơi khác ở phần thực hiện một chút.

Cụ thể:

* Vì một đối tượng là một class kế thừa Model nên nó có các phương thức sau:
  + Class Methods: Được gọi trực tiếp từ model (Model.method()).

| **Phương thức** | **Chức năng** |
| --- | --- |
| init(attributes, options) | Khởi tạo model với các thuộc tính và tùy chọn. |
| sync(options) | Đồng bộ model với cơ sở dữ liệu (tạo bảng nếu chưa có). |
| findAll(options) | Tìm tất cả bản ghi theo điều kiện. |
| findOne(options) | Tìm một bản ghi đầu tiên theo điều kiện. |
| findByPk(id, options) | Tìm một bản ghi theo khóa chính. |
| create(values, options) | Tạo và lưu một bản ghi vào database. |
| bulkCreate(records, options) | Tạo nhiều bản ghi cùng lúc. |
| update(values, options) | Cập nhật bản ghi phù hợp với điều kiện. |
| destroy(options) | Xóa bản ghi phù hợp với điều kiện. |
| count(options) | Đếm số lượng bản ghi thỏa mãn điều kiện. |
| max(field, options) | Lấy giá trị lớn nhất của một cột. |
| min(field, options) | Lấy giá trị nhỏ nhất của một cột. |
| sum(field, options) | Tính tổng giá trị của một cột. |
| scope(name) | Sử dụng scope đã định nghĩa trước. |
| unscoped() | Bỏ scope hiện tại. |
| findOrCreate(options) | Tìm bản ghi, nếu không có thì tạo mới. |
| upsert(values, options) | Chèn hoặc cập nhật bản ghi (tùy vào việc có tồn tại hay không). |

* + Instance Methods: Chỉ gọi được từ một instance của model (instance.method()).

| **Phương thức** | **Chức năng** |
| --- | --- |
| save(options) | Lưu instance vào database (nếu chưa có thì tạo mới, nếu có thì cập nhật). |
| destroy(options) | Xóa instance khỏi database. |
| reload(options) | Tải lại instance từ database. |
| update(values, options) | Cập nhật dữ liệu của instance. |
| toJSON() | Chuyển instance thành object JSON. |
| getDataValue(column) | Lấy giá trị của một cột. |
| setDataValue(column, value) | Cập nhật giá trị của một cột. |
| increment(field, options) | Tăng giá trị của một cột. |
| decrement(field, options) | Giảm giá trị của một cột. |
| changed(field) | Kiểm tra xem cột có thay đổi không. |
| previous(field) | Lấy giá trị trước khi cập nhật. |

*Đây chỉ là bản giới thiệu qua về chức năng, cụ thể hàm ntn tự tìm hiểu sau tại đây:*

[*https://sequelize.org/api/v6/class/src/model.js~model*](https://sequelize.org/api/v6/class/src/model.js~model)

* Đây là những method để tương tác với dữ liệu, được chia làm 2 cấp: instance (phải gọi từ instance) và Class (gọi qua tên Model). Có thể suy luận như sau: với những thao tác chung lên tất cả dữ liệu thì sử dụng Class Method, với những thao tác riêng lên từng đối tượng đã xác định sẵn thì dùng Instance Method. Cụ thể hơn thì so sánh như sau:

| **Tiêu chí** | **Phương thức cấp lớp (Class Methods)** | **Phương thức cấp instance (Instance Methods)** |
| --- | --- | --- |
| **Phạm vi** | Tác động lên toàn bộ bảng trong database. | Tác động lên một bản ghi cụ thể. |
| **Cách gọi** | Gọi trực tiếp từ model (User.method()). | Gọi từ một instance của model (user.method()). |
| **Tính đồng bộ với DB** | Đồng bộ ngay với database. | Chỉ đồng bộ khi gọi save(). |
| **Khả năng sử dụng** | Tìm, tạo, cập nhật, xóa nhiều bản ghi cùng lúc. | Làm việc với một bản ghi cụ thể, cần gọi save() để lưu thay đổi. |
| **Hiệu suất** | Hiệu quả khi làm việc với nhiều bản ghi. | Phù hợp khi cần xử lý dữ liệu từng bản ghi. |

Các sample CRUD với các hàm của Model:

Create:

|  |
| --- |
| // Create - Tạo mới dữ liệu  await User.create({ firstname: 'John', lastname: 'Doe', email: 'john@example.com' });  await User.bulkCreate([{ firstname: 'Alice' }, { firstname: 'Bob' }]); |

Read:

|  |
| --- |
| // Read - Đọc dữ liệu await User.findAll(); // Lấy tất cả  await User.findOne({ where: { email: 'john@example.com' } }); // Lấy một bản ghi  await User.findByPk(1); // Lấy theo ID |

Update:

|  |
| --- |
| // Update - Cập nhật dữ liệu với class method  await User.update({ age: 31 }, { where: { email: 'john@example.com' } });  // Update - Cập nhật dữ liệu với instance method  const user = await User.findByPk(1);  user.age = 32;  await user.save(); |

Delete:

|  |
| --- |
| // Delete - Xóa dữ liệu với class method  await User.destroy({ where: { email: 'john@example.com' } });  // Delete – Xóa dữ liệu với instance method  const userToDelete = await User.findByPk(2);  await userToDelete.destroy(); |

Những câu lệnh tương tự có thể đưa chatGPT giải quyết hộ 1 phần.

Hoặc là đọc ở đây: <https://sequelize.org/docs/v6/core-concepts/model-querying-basics/>

<https://sequelize.org/docs/v6/core-concepts/model-querying-finders/>

những cách thực hiện cao cấp hơn thì đọc ở đây: <https://sequelize.org/docs/v6/category/core-concepts/>

Còn nếu muốn xây dựng cơ bản thì đọc tiếp.

# 5, Xây dựng Association

*Nếu có thắc mắc tại sao có thể CRUD rồi mà lại còn phải đi xây dựng mối quan hệ các bảng, thì đúng, nó mang nhiều ý nghĩa cho tối ưu backend hơn là render lên web và “chạy được”.*

* Từ mối quan hệ giữa các bảng, có thể coi có 3 mối quan hệ: 1-1, 1-n, n-n thông qua 1 bảng khác làm trung gian.
* Trong sequelize, lại có 4 mối quan hệ như sau:

|  |
| --- |
| A.hasOne(B, {  */\* options \*/* }); A.belongsTo(B, {  */\* options \*/* }); A.hasMany(B, {  */\* options \*/* }); A.belongsToMany(B, { through: 'C' */\* options \*/* }); |

Trong 4 mối quan hệ trên, A được gọi là source còn B gọi là target của mqh. Giải thích sâu hơn về 4 câu lệnh trên:

| **Quan hệ** | **Loại quan hệ** | **Vị trí khóa ngoại** | **Ý nghĩa** |
| --- | --- | --- | --- |
| A.hasOne(B) | One-To-One (1-1) | Trong B (target model) | Một bản ghi trong A có **một** bản ghi tương ứng trong B. |
| A.belongsTo(B) | One-To-One (1-1) | Trong A (source model) | Một bản ghi trong A thuộc về **một** bản ghi trong B. |
| A.hasMany(B) | One-To-Many (1-N) | Trong B (target model) | Một bản ghi trong A có **nhiều** bản ghi trong B. |
| A.belongsToMany(B, { through: 'C' }) | Many-To-Many (N-N) | Bảng trung gian C | Một bản ghi trong A có thể liên kết với nhiều bản ghi trong B, và ngược lại, thông qua bảng trung gian C. |

*Nghĩa là giờ phải tạo ra 3 mối quan hệ của table từ 4 mối quan hệ của sequelize*.

* Ngắn gọn lại:
  + **One-To-One:** Dùng hasOne và belongsTo kết hợp.
  + **One-To-Many:** Dùng hasMany và belongsTo kết hợp.
  + **Many-To-Many:** Dùng hai lần belongsToMany với bảng trung gian.

Nhớ kỹ 3 điều này sẽ khiến code dễ hơn.

Chú ý: sequelize sẽ tự động tạo các khóa ngoại và các bảng liên kết (nếu chưa có) và nếu có rồi mà không cấu hình thì nó cũng không biết, nên nếu đã tạo bảng liên kết ở bước xây DB rồi thì tới đây phải khai báo cho sequelize biết.

* Cụ thể hơn cho các mối quan hệ 1-1, 1-n và n-n:

## 1, 1-1

* Giả sử như có mối quan hệ A-B là 1-1, nghĩa là một bản ghi trong A sẽ có tương ứng 1 bản ghi trong B.
* Mối quan hệ từ A (source) sang B (target). => **khóa ngoại nằm ở B.**

Code mẫu sẽ như sau:

|  |
| --- |
| A.hasOne(B, { foreignKey: 'existing\_foreign\_key' }); // Khóa ngoại nằm trong bảng B  B.belongsTo(A, { foreignKey: 'existing\_foreign\_key' }); // Khóa ngoại nằm trong bảng B |