## Model

2 cách tạo model :

* sequelize.define(modelName, attributes, options)
* class extends Model

Cách 1 :

const { Sequelize, DataTypes } = require('sequelize');

const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', {

firstName: {

type: DataTypes.STRING,

allowNull: false

},

lastName: {

type: DataTypes.STRING

}

}, {// options});

console.log(User === sequelize.models.User); // true

Cách 2 :

class User extends Model {}

User.init({

// Model attributes are defined here

}, {

// Other model options go here

sequelize, // We need to pass the connection instance

modelName: 'User' // We need to choose the model name

});

console.log(User === sequelize.models.User); // true

## Table

* nếu không đặt 🡪 sử dụng tên model

Set table name :

sequelize.define('User', {

// ... (attributes)

}, {

tableName: 'Employees'

});

Force table name & model name giống nhau

sequelize.define('User', {

// ... (attributes)

}, {

freezeTableName: true

});

globally configure

const sequelize = new Sequelize('sqlite::memory:', {

define: {

freezeTableName: true

}

});

## Sync

Đồng bộ database schema với ORM

2 trường hợp : DB đã có table & chưa có table

3 cách xử lý :

Table chưa exist 🡪 tạo mới

await User.sync();

Table đã exist 🡪 drop & create

await User.sync({ force: true });

Table đã exist 🡪 check & alter ~ field & attr cần thiết

await User.sync({ alter: true });

## Safety check

// This will run .sync() only if database name ends with '\_test'

sequelize.sync({ force: true, match: /\_test$/ });

## Drop

Single model : await User.drop();

All model : await sequelize.drop();

## Timestamps

ORM sẽ tự động tạo [createdAt & updatedAt] ở mọi model (default)

sequelize.define('User', {

// ... (attributes)

}, {

timestamps: false,

// don't forget to enable timestamps!

timestamps: true,

// I don't want createdAt

createdAt: false,

// I want updatedAt to actually be called updateTimestamp

updatedAt: 'updateTimestamp'

});

## Default value

sequelize.define('User', {

name: {

type: DataTypes.STRING,

defaultValue: "John Doe",

// hoặc

defaultValue: Sequelize.NOW

}

});

Now Datetime : Sequelize.NOW

Import Datatypes

const { DataTypes } = require("sequelize"); // Import the built-in data types

Strings

DataTypes.STRING // VARCHAR(255)

DataTypes.STRING(1234) // VARCHAR(1234)

DataTypes.STRING.BINARY // VARCHAR BINARY

DataTypes.TEXT // TEXT

DataTypes.TEXT('tiny') // TINYTEXT

DataTypes.CITEXT // CITEXT PostgreSQL and SQLite only.

Boolean

DataTypes.BOOLEAN // TINYINT(1)

Number

DataTypes.INTEGER // INTEGER

DataTypes.BIGINT // BIGINT

DataTypes.BIGINT(11) // BIGINT(11)

DataTypes.FLOAT // FLOAT

DataTypes.FLOAT(11) // FLOAT(11)

DataTypes.FLOAT(11, 10) // FLOAT(11,10)

DataTypes.REAL // REAL PostgreSQL only.

DataTypes.REAL(11) // REAL(11) PostgreSQL only.

DataTypes.REAL(11, 12) // REAL(11,12) PostgreSQL only.

DataTypes.DOUBLE // DOUBLE

DataTypes.DOUBLE(11) // DOUBLE(11)

DataTypes.DOUBLE(11, 10) // DOUBLE(11,10)

DataTypes.DECIMAL // DECIMAL

DataTypes.DECIMAL(10, 2) // DECIMAL(10,2)

Unsigned & Zerofil – MySQL / MariaDB

DataTypes.INTEGER.UNSIGNED

DataTypes.INTEGER.ZEROFILL

DataTypes.INTEGER.UNSIGNED.ZEROFILL

// You can also specify the size i.e. INTEGER(10) instead of simply INTEGER

// Same for BIGINT, FLOAT and DOUBLE

Dates

DataTypes.DATE // DATETIME for mysql / sqlite, TIMESTAMP WITH TIME ZONE for postgres

DataTypes.DATE(6) // DATETIME(6) for mysql 5.6.4+. Fractional seconds support with up to 6 digits of precision

DataTypes.DATEONLY // DATE without time

UUIDs

{

type: DataTypes.UUID,

defaultValue: Sequelize.UUIDV4 // Or Sequelize.UUIDV1

}

BLOBs

DataTypes.BLOB // BLOB (bytea for PostgreSQL)

DataTypes.BLOB('tiny') // TINYBLOB (bytea for PostgreSQL)

DataTypes.BLOB('medium') // MEDIUMBLOB (bytea for PostgreSQL)

DataTypes.BLOB('long') // LONGBLOB (bytea for PostgreSQL)

Enum

DataTypes.ENUM('foo', 'bar') // An ENUM with allowed values 'foo' and 'bar'

JSON

DataTypes.JSON

DataTypes.JSONB // for PostgreSQL

## Options

Gồm :

* allowNull(boolean)
* defaultValue(any)
* unique(boolean, string)
* primaryKey(boolean)
* autoIncrement(boolean)
* field(string) : custom field name
* reference : liên kết gồm (model, key, deferrable)
* commentMe

const { Model, DataTypes, Deferrable } = require("sequelize");

class Foo extends Model {}

Foo.init({

flag: { type: DataTypes.BOOLEAN, allowNull: false, defaultValue: true },

myDate: { type: DataTypes.DATE, defaultValue: DataTypes.NOW },

title: { type: DataTypes.STRING, allowNull: false },

Creating two objects with the same value will throw an error.

The unique property can be either a boolean, or a string.

If you provide the same string for multiple columns, they will form a composite unique key.

uniqueOne: { type: DataTypes.STRING, unique: 'compositeIndex' },

uniqueTwo: { type: DataTypes.INTEGER, unique: 'compositeIndex' },

// The unique property is simply a shorthand to create a unique constraint.

someUnique: { type: DataTypes.STRING, unique: true },

// Go on reading for further information about primary keys

identifier: { type: DataTypes.STRING, primaryKey: true },

incrementMe: { type: DataTypes.INTEGER, autoIncrement: true },

// You can specify a custom column name via the 'field' attribute:

fieldWithUnderscores: { type: DataTypes.STRING, field: 'field\_with\_underscores' },

// It is possible to create foreign keys:

bar\_id: {

type: DataTypes.INTEGER,

references: {

model: Bar,

key: 'id',

// With PostgreSQL, it is optionally possible to declare when to check the foreign key constraint, passing the Deferrable type.

deferrable: Deferrable.INITIALLY\_IMMEDIATE

}

},

Options:

* `Deferrable.INITIALLY\_IMMEDIATE` - Immediately check the foreign key constraints
* `Deferrable.INITIALLY\_DEFERRED` - Defer all foreign key constraint check to the end of a transaction
* `Deferrable.NOT` - Don't defer the checks at all (default) - This won't allow you to dynamically change the rule in a transaction

// Comments can only be added to columns in MySQL, MariaDB, PostgreSQL and MSSQL

commentMe: {

type: DataTypes.INTEGER,

comment: 'This is a column name that has a comment'

}

}, {

sequelize,

modelName: 'foo',

// Using `unique: true` in an attribute above is exactly the same as creating the index in the model's options:

indexes: [{ unique: true, fields: ['someUnique'] }]

});

# APIs

Create model (build instance but not save)

const jane = User.build({ name: "Jane" });

console.log(jane instanceof User); // true

Save model

await jane.save();

Or build & save same time

const jane = await User.create({ name: "Jane" });

// Jane exists in the database now!

console.log(jane instanceof User); // true

Save only … field

const jane = await User.create({ name: "Jane" });

jane.name = "Jane II";

jane.favoriteColor = "blue";

await jane.save({ fields: ['name'] });

Update

const jane = await User.create({ name: "Jane" });

console.log(jane.name); // "Jane"

jane.name = "Ada";

// the name is still "Jane" in the database

await jane.save();

// Now the name was updated to "Ada" in the database!

\*nếu không thay đổi gì mà .save() thì tự động sẽ không exec bên DB

\*nếu chỉ change 1 vài field 🡪 tự động chỉ update field đó

Delete

await jane.destroy();

Reloading

const jane = await User.create({ name: "Jane" });

console.log(jane.name); // "Jane"

jane.name = "Ada";

// the name is still "Jane" in the database

await jane.reload();

console.log(jane.name); // "Jane"

Logging

const jane = await User.create({ name: "Jane" });

// console.log(jane); // Don't do this

console.log(jane.toJSON()); // This is good!

console.log(JSON.stringify(jane, null, 4)); // This is also good!

Default value

const User = sequelize.define("user", {

name: DataTypes.TEXT,

favoriteColor: {

type: DataTypes.TEXT,

defaultValue: 'green'

}

});

const jane = User.build({ name: "Jane" });

console.log(jane.favoriteColor); // "green"

## Increment & decrement

Tránh concurrency issues

const jane = await User.create({ name: "Jane", age: 100 });

const incrementResult = await jane.increment('age', { by: 2 });

// or

await jane.increment('age'); // by 1

await jane.increment({

'age': 2,

'cash': 100

});

// or

await jane.increment(['age', 'cash'], { by: 2 });

Chú ý :

In PostgreSQL, `incrementResult` will be the updated user, unless the option

`{ returning: false }` was set (and then it will be undefined).

In other dialects, `incrementResult` will be undefined. If you need the updated instance, you will have to call `user.reload()`.