Sequeliz :

1 – معرفی و نصب

Npm I @sequelize/core

بعد درایور دیتا بیس که میخوایم ازش استفاده کنیمو نصب میکنیم .

*# One of the following:*  
$ npm install --save pg pg-hstore *# Postgres*  
$ npm install --save mysql2  
$ npm install --save mariadb  
$ npm install --save sqlite3  
$ npm install --save tedious *# Microsoft SQL Server*  
$ npm install --save oracledb *# Oracle Database*

روش های اتصال به دیتابیس با sequelize :

1 –

import { Sequelize } from "@sequelize/core";

const sequelize = new Sequelize("task", "root", "hd6730mrm", {

    dialect: "mysql",

    host: "localhost",

});

2 –

const sequelize = new Sequelize("mysql://root:hd6730mrm@localhost/task");

3-

const sequelize = new Sequelize({

    dialect: "mysql",

    host: "localhosst",

    database: "task",

    username: "root",

    password: "hd6730mrm",

    logging: false

});

دیفالتش logging: console.log عه ینی هرچی قراره لاگ گرفته بشه بفرست به تابع console.log که میتونیم به جاش یه تابع دیگه بدیم .

به روش زیر میتونیم چک کنیم ببینیم که به دیتا بیس وصل شدیم یا نه :

try {

    await sequelize.authenticate();

    console.log("Connected");

} catch (error) {

    console.log(error.message);

}

db.js :

import { Sequelize } from "@sequelize/core";

const sequelize = new Sequelize({

    dialect: "mysql",

    host: "localhost",

    database: "task",

    username: "root",

    password: "hd6730mrm",

    logging: false,

});

try {

    await sequelize.authenticate();

    console.log("Connected Successfullt");

} catch (error) {

    console.log(error.message);

}

export default sequelize;

روش های ایجاد مدل :

1 –

import { DataTypes, Model } from "@sequelize/core";

import sequelize from "./db.js";

class User extends Model {}

User.init(

    {

        firstname: DataTypes.STRING,

        lastname: DataTypes.STRING(100),

        age: DataTypes.INTEGER

    },

    {

        sequelize: sequelize

    }

);

await User.sync()

تو این حالت خودش میاد برامون یه id پرایمری کی و یک createdAt و updatedAt برامون میسازه .

await User.sync() تو این حالت اگه دیتا بیس وجود داشته باشه هیچ کاریش نداره حتی تغییرش هم نمیده. فقط اگه وجود نداشته باشه میسازدش .

روش 2 :

const Task = sequelize.define(“Task”, {

    title: DataTypes.STRING(100),

    completed: DataTypes.BOOLEAN,

})

await Task.sync()

2-3- تعریف مدل ها :

const Task = sequelize.define(

    “Task”,

    {

        title: DataTypes.STRING,

        completed: DataTypes.INTEGER,

    },

    {

        timestamps: false,

        noPrimaryKey: false,

    }

);

الان دیگه اون id و createdAt و updatedAt رو که دیفالت میساخت دیگه نمیسازه .

نکته :

// drop table if exists, then create a new one

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

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

نکته :

// alter table if any difference exists, but data will not deleted

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

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

نکته :

// sync all tables

// await sequelize.sync();

// await sequelize.sync({alter: true});

// await sequelize.sync({force: true});

نکته :

// drop a table

await User.drop()

// drop a table

await sequelize.drop()

ایجاد مدل با تنظیمات ویژگی های ستون ها :

const Book = sequelize.define(

    "book",

    {

        pages: DataTypes.SMALLINT,

        title: {

            type: DataTypes.STRING(100),

            defaultValue: "",

        },

        description: {

            type: DataTypes.TEXT,

            defaultValue: "No Description",

        },

        published: {

            type: DataTypes.DATEONLY,

            defaultValue: DataTypes.NOW,

            allowNull: false,

        },

        category: {

            type: DataTypes.STRING,

            allowNull: false,

        },

    },

    {

        timestamps: false,

    }

);

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

ساخت یک رکورد :

await Book.create({ category: "Programming" });

const Book = sequelize.define(

    "Book",

    {

        // you can use only one autoincremental field

        // primary key shoulb be autoincremental, otherwise you must provide in each record

        bookId: { type: DataTypes.BIGINT, primaryKey: true, autoIncrement: true },

        pages: DataTypes.SMALLINT,

        title: {

            type: DataTypes.STRING(200) + "charset latin1 collate latin1\_swedish\_ci",

            defaultValue: true,

        },

        description: {

            type: DataTypes.STRING(2000),

            defaultValue: "No Description",

        },

    },

    {

        charse: "utf8mb3",

        collate: "utf8mb3\_persian\_ci",

        engine: "MyISAM",

        initialAutoIncrement: 10,

        freezeTableName: true,

        tableName: "the\_books",

        underscored: true,

    }

);

انتخاب نام یک ستون ، تغییر نام deletedAt یا updadtedAt یا createdAt   
انتخاب اینکه از اینا deletedAt یا updadtedAt یا createdAt کدومشون باشه کدومشون نباشه :

const Book = sequelize.define(

    "Book",

    {

        title: {

            type: DataTypes.STRING(200) + "charset latin1 collate latin1\_swedish\_ci",

        },

        desc: {

            type: DataTypes.STRING(2000),

            defaultValue: "No Description",

            columnName: "description",

        },

    },

    {

        createdAt: "created\_at",

        updatedAt: false,

        deletedAt: true,

        paranoid: true

    }

);

وقتی deletedAt داشته باشیم وقتی یک رکورد رو حذف میکنیم ، در واقع اون رکورد حذف نمیشه و فقط یه تاریخ برای دیلیتد اتش ثبت میشه   
حالا وقتی هم میخوایم روش سلکت بزنیم ، دیگه اونهایی که تاریخ دیلیتد ات براشون ثبت شده رو به ما نشون نمیده مگه اینکه خودمون بیایم و برشون گردونیم .

نکته : اگه timestamp ، false باشه ، دیگه دیلیتد ات هم کار نمیکنه .

تعریف ایندکس ها در مدل :

import { DataTypes } from "@sequelize/core";

import sequelize from "./db.js";

const Book = sequelize.define(

    "Book",

    {

        title: DataTypes.STRING(200),

        completed: {

            type: DataTypes.BOOLEAN,

            defaultValue: false,

        },

        test1: {

            type: DataTypes.STRING,

            index: true,

        },

        test2: {

            type: DataTypes.STRING,

            index: "my\_index",

        },

        test3: {

            type: DataTypes.STRING,

            unique: true,

        },

        test4: {

            type: DataTypes.STRING,

            unique: "unique\_test4",

        },

        test5: {

            type: DataTypes.STRING,

            unique: "unique\_t",

        },

        test6: {

            type: DataTypes.STRING,

            unique: "unique\_t",

        },

    },

    {

        timestamps: false,

        indexes: [

            {

                name: "title\_idx",

                type: "UNIQUE",

                fields: ["title"],

            },

            {

                name: "completed\_idx",

                fields: ["completed"],

            },

        ],

    }

);

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

نکته : test5 و test6 باید ترکیبشون یونیک باشه ، چون اسم ایندکس یونیک هاشون یکسانه .

محدودیت گذاشتن روی فیلد ها :

const Test = sequelize.define(

    "Test",

    {

        filed1: {

            type: DataTypes.STRING,

            validate: {

                max: 100,

                min: 10,

                isEven(value) {

                    if (value % 2 != 0) {

                        throw new Error("Only even values are allowed");

                    }

                },

            },

        },

        filed2: DataTypes.INTEGER,

        filed3: DataTypes.INTEGER,

    },

    {

        timestamps: false,

        bothOrNone() {

            const isSetField2 = !(this.filed2 === null || this.filed2 === undefined);

            const isSetField3 = !(this.filed3 === null || this.filed3 === undefined);

            if (isSetField2 != isSetField3) {

                throw new Error("you can not set only one of 'field2' and 'field3' ");

            }

        },

    }

);

ولیدیشن های از پیش تعیین شده :

//   is: /^[a-z]+$/i,          // matches this RegExp

//   is: ["^[a-z]+$",'i'],     // same as above, but constructing the RegExp from a string

//   not: /^[a-z]+$/i,         // does not match this RegExp

//   not: ["^[a-z]+$",'i'],    // same as above, but constructing the RegExp from a string

//   isEmail: true,            // checks for email format (foo@bar.com)

//   isUrl: true,              // checks for url format (https://foo.com)

//   isIP: true,               // checks for IPv4 (129.89.23.1) or IPv6 format

//   isIPv4: true,             // checks for IPv4 (129.89.23.1)

//   isIPv6: true,             // checks for IPv6 format

//   isAlpha: true,            // will only allow letters

//   isAlphanumeric: true,     // will only allow alphanumeric characters, so "\_abc" will fail

//   isNumeric: true,          // will only allow numbers

//   isInt: true,              // checks for valid integers

//   isFloat: true,            // checks for valid floating point numbers

//   isDecimal: true,          // checks for any numbers

//   isLowercase: true,        // checks for lowercase

//   isUppercase: true,        // checks for uppercase

//   notNull: true,            // won't allow null

//   isNull: true,             // only allows null

//   notEmpty: true,           // don't allow empty strings

//   equals: 'specific value', // only allow a specific value

//   contains: 'foo',          // force specific substrings

//   notIn: [['foo', 'bar']],  // check the value is not one of these

//   isIn: [['foo', 'bar']],   // check the value is one of these

//   notContains: 'bar',       // don't allow specific substrings

//   len: [2,10],              // only allow values with length between 2 and 10

//   isUUID: 4,                // only allow uuids

//   isDate: true,             // only allow date strings

//   isAfter: "2011-11-05",    // only allow date strings after a specific date

//   isBefore: "2011-11-05",   // only allow date strings before a specific date

//   max: 23,                  // only allow values <= 23

//   min: 23,                  // only allow values >= 23

//   isCreditCard: true,       // check for valid credit card numbers

42 – 43 – CRUD

انواع روش های create :

1 – متد create

const user1 = await User.create({

    username: "user1",

    firstname: "first1",

    lastname: "last1",

});

user1.username = "username1";

user1.save();

بدون save هم اینسرت میکنه داخل دیتا بیس.  
ولی وقتی بخوابیم یکی از فیلد هارو تغییر بدیم ، باید از save استفاده کنیم .

متد create یک آبجکت رو هم برمیگردونه .

2 – متد build

const user2 = User.build({

    username: "user2",

});

user2.firstname = "user2";

user2.save();

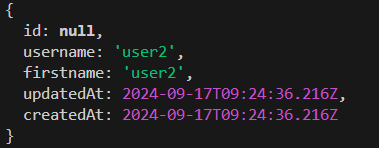
-برای درج تو دیتابیس حتما باید save کال بشه .

-آبجکت رو هم به ما برمیگردونه .

نکته : خوده متد save هم برا ما آبجکت برمیگردونه . پرامیسه .

Const user = await user2.save();

نکته : اگه خوده آبجکت رو چاپ کنیم خیلی خرت و پرت بهمون میده .   
میتونیم از toJSON استفاده کنیم تا فقط فیلد هاش رو نشون بده .



نکته :

console.log(user2 instanceof User);             // true

console.log(user2.toJSON() instanceof User);    // false

نکته : از پراپرتی dataValues هم میتونیم استفاده کنیم .که در واقع همون چیزی رو برمیگردونه که user2.toJSON() برمیگردونه .

user2.dataValues

متد reload : میره از دیتابیس میخونه و فیلد هارو جایگزین مقادیر فعلی میکنه . در واقع بیشتر برای زمانی استفاده میشه که ما ی سری تغییراتی میدیم و منصرف شدیم و میخوایم همون مقدار قبلی باشه .

const user2 = User.build({

    username: "user2",

});

user2.firstname = "user2";

await user2.save();

user2.lastname = "last2";

await user2.reload();

await user2.save();

الان تو کد بالا ، last2 اعمال نمیشه و هرچیزی که تو دیتابیس بوده همون مقدار باقی میمونه .

3 –

const user3 = User.create(

    {

        username: "user3",

        firstname: "first3",

        lastname: "last3",

    },

    {

        fields: ["firstname", "lastname"],

    }

);

فقط فیلد های ["firstname", "lastname"], مقدار دهی میشن.

3 – متد findOrCreate :

const result = await User.findOrCreate({

    where: { username: "user4" },

    defaults: {

        lastname: "last4"

    }

})

console.log(result[0].toJSON());    // the created or finded user

console.log(result[1]);          // true false اگه آبجکت جدیدی ایجاد بشه ترو میده اگه پیدا کنه فالس میده

اینجوری هم میشه نوشت :

const [user, created] = await User.findOrCreate({

    where: { username: "user4" },

    defaults: {

        lastname: "last4"

    }

})

console.log(user);    // the created or finded user

console.log(created);             // true false

4 – متد bulcCreate : اضافه کردن چند یوزر

const results = await User.bulkCreate([

    {

        username: “user5”,

        firstname: “first5”,

        lastname: “last5”,

    },

    {

        username: “user6”,

        firstname: “first6”,

        lastname: “last6”,

    },

]);

یه آرایه ای از یوزر هایی که اضافه شدن برمیگردونه .

خواندن داده ها از دیتابیس :

1- متد find