

¿QUÉ ES UN ORM?

SQL

Para **hacer consultas** a una base de datos un programador **necesita escribir SQL**.

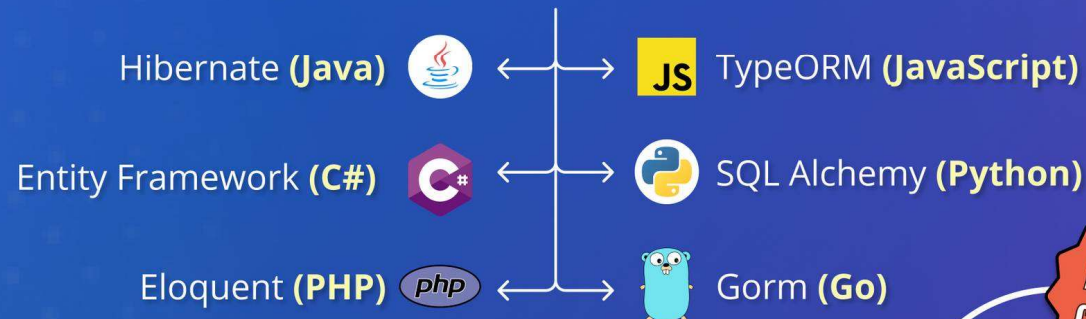


ORM

Un **ORM abstrae la base de datos** para que el programador haga consultas en el **lenguaje en el que está programando, sin necesitar SQL**.



LOS ORM MÁS POPULARES SON:



¡NUEVO
CURSO!



- Sequelize es un ORM de Node.js basado en promesas
- Postgres, MySQL, MariaDB, SQLite y Microsoft SQL Server.
- Cuenta con un sólido soporte de transacciones, relaciones, eager and lazy loading (Joins in DB), replicación de lectura y más.
- Sigue el versionamiento semántico y soporte Node 10.0 y posterior.
- Open source

Instalar y conectar el ORM Sequelize

- `npm install --save sequelize` para agregarlo a nuestro proyecto

dependencias instaladas

- `npm install mysql2` para que funcione hay usar el mysql2

requerir el objeto sequelize con un new

```
const { Sequelize } = require('sequelize');
```

objeto conexión en minúscula

```
// Opción 1: Pasar una URI de conexión
```

```
var sequelize = new
```

pasamos la cadena de conceccion

```
Sequelize('mysql://user:password@localhost:3306/databasename');
```

```
// Opción 2 pasar los parámetros de conexión separados
```

pasamos la cadena de conceccion separado tambien lo acepta

```
const sequelize = new Sequelize('database', 'username', 'password', {
```

```
host: 'localhost',
```

```
dialect: 'mysql',
```

con que motor de base de datos trabajamos

```
logging: false
```

para que en la consola no imprima en la consola todo lo que imprime

```
});
```

Probando la conexión

son promesas asíncronas

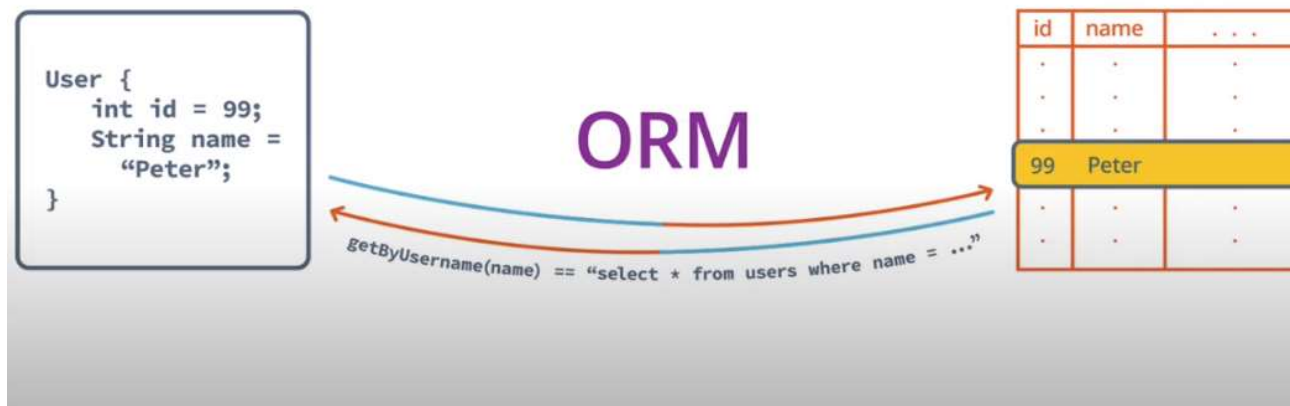
```
sequelize.authenticate() chequeo si se conecto correctamente  
  .then(() => {  
    console.log('Conectado')  
  })  
  .catch(err => {  
    console.log('No se conecto')  
  })  
  
// Cerrar la conexión para cerrar la conccccecion  
sequelize.close()  
.then()
```

Modelos

- Los modelos son la esencia de Sequelize.
- Un modelo es una abstracción que representa una tabla en su base de datos.
- Un modelo en **Sequelize** extiende de la clase **Model**

class User extends Model {}

orm ayuda a mapear la tabla de base de datos con mi aplicación



Definir un Modelo (Usuario.js)

`datatypes=` son lo tipos de tablas que vamos a usar

```
const { Sequelize, DataTypes, Model } = require('sequelize');
const sequelize = new Sequelize(URIMysql);
```

definimos la clase con el mismo modelo de la tabla que vamos a usar
en plural y extiende del objeto model

```
class Usuario extends Model {
```

para agregarle clases a model se usa la clase `init` (que son los que estan
entre las llaves en rojo)

```
  Usuario.init({
```

```
    // Atributos del modelo son definidos aquí
```

```
    nombre: {
```

```
      type: DataTypes.STRING,
```

```
      allowNull: false // allowNull es true por defecto
```

para que no se le asigne lugar null

```
    },
```

```
    apellido: DataTypes.STRING, //Solo si defino unicamente el tipo de datos
```

```
    fecha: { type: DataTypes.DATETIME, defaultValue: Sequelize.NOW }
```

```
  }, {
```

le pasamos el objeto con la conceccion y el objeto que vamos a usar como parametro

```
    // Otras opciones del modelo
```

```
    sequelize, // pasamos la instancia de la conexion
```

```
    modelName: 'Usuario' // El nombre del modelo
```

```
    //tableName: 'Usuarios'
```

ak le indico que la tabla que va a hacer mach con el modelo que
estamos definiendo es usuario

```
  });
```

```
//si no queremos la pluralización
{ define: { freezeTableName: true } }
```

Timestamps

```
class Foo extends Model {}  
Foo.init({ ATRIBUTOS }, {  
  sequelize,  
  // No olvidar activar timestamps  
  timestamps: true,  
  // Si no queremos el campo createdAt  
  createdAt: false, desactiva los timestamps lo pongo en false  
                     fecha de creacion del objeto  
  // Si queremos updatedAt pero con otro nombre  
  updatedAt: 'fechaActualizacion' actualizacion del objeto  
  //Soft delete  
  paranoid:true;  
  deletedAt: 'FechaBorrado' borrado de objeto logico pero no total  
});
```


Sincronización



Solo usar en el contexto de
desarrollo o
Despliegue de la aplicación

- `Usuario.sync()` Crea la tabla si no existe (y no hace nada si ya existe)
- `Usuario.sync({force: true})` Crea la tabla, borrándola primero si ya existía.
- `User.sync ({alter: true})` Verifica cuál es el estado actual de la tabla en la base de datos (qué columnas tiene, cuáles son sus tipos de datos, etc.), y luego realiza los cambios necesarios en la tabla para que coincida con el modelo.
- `sequelize.sync({ force: true });` Sincroniza automáticamente todos los modelos.

Tipos de Datos

- `const { DataTypes } = require("sequelize");`

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
```

Boolean

```
DataTypes.BOOLEAN          // TINYINT(1)
```

PostgreSQL and SQLite only.

Numbers

```
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
DataTypes.REAL(11)         // REAL(11)
DataTypes.REAL(11, 12)     // REAL(11,12)

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 & Zerofill integers - MySQL/MariaDB only

In MySQL and MariaDB, the data types `INTEGER`, `BIGINT`, `FLOAT` and `DOUBLE` can be set as unsigned or zerofill (or both), as follows:

```
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
```

PostgreSQL only.
PostgreSQL only.
PostgreSQL only.

Dates

```
DataTypes.DATE             // DATETIME for mysql / sqlite, TIMESTAMP WITH TIME ZONE
DataTypes.DATE(6)          // DATETIME(6) for mysql 5.6.4+. Fractional seconds supported
DataTypes.DATEONLY         // DATE without time
```

Opciones de columnas

un atributo no permite nulos

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

```
// Define un índice único para la columna.
```

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

```
/* Crear 2 objetos con el mismo valor lanzara un error. Si proveemos el mismo  
valor string para varias columnas se creará un índice compuesto */
```

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

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

```
// Define como clave primaria
```

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

```
// Autoincremento para columnas enteras
```

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

```
// Personaliza el nombre de la columna de la tabla
```

```
fieldWithUnderscores: { type: DataTypes.STRING, field: 'field_with_underscores'  
},
```

Personalizar métodos

```
class Usuario extends Model {  
  static metodoEstatico() { //define métodos estáticos  
    return 'foo';  
  }  
  getNombreCompleto() { //define método de instancia  
    return [this.nombre, this.apellido].join(' ');  
  }  
}  
Usuario.init({  
  nombre: DataTypes.STRING,  
  apellido: DataTypes.STRING  
}, { sequelize });  
  
console.log(Usuario.metodoEstatico()); // 'foo'  
const user1 = Usuario.build({ nombre: 'María', apellido: 'Gomez' }); //instancia modelo  
console.log(user1.getNombreCompleto()); // 'María Gomez',
```

Instanciar Modelos

esto esta
creado en
memoria y
no en base de
datos todavia

```
const mariaObj = Usuario.build({ nombre: "María" });  
console.log(mariaObj instanceof Usuario); // true  
console.log(mariaObj.nombre); // "María"
```

y me devuelve la promesa

```
await mariaObj.save();  
console.log('maría fue grabada en la base de datos!');
```

ak le digo q ue guarde el objeto que tengo en la memoria

esto hace un build y ademas hace un save (hace las dos cosas juntas de lo que hace el de arriba)

```
const mariaObj = await Usuario.create({ nombre: "María" });  
// maria ahora existe en la BD!  
console.log(mariaObj instanceof User); // true  
console.log(mariaObj.toJSON()); // Es muy fácil crear un json  
console.log(mariaObj.nombre); // "María"
```

Actualizar y Borrar instancias

```
const joseObj = await Usuario.create({ nombre: "Jose" });  
console.log(joseObj.nombre); // "Jose"  
joseObj.nombre = "Lucas";  
// El nombre todavía es "Jose" en la base de datos  
await joseObj.save();    para guardar los cambios con el save sino no actualiza lo de la bd  
// Ahora el nombre fue actualizado a "Lucas" en la BD!
```

```
const joseObj = await Usuario.create({ nombre: "Jose" });  
console.log(joseObj.nombre); // "Jose"  
await joseObj.destroy();    destroy borra el registro en la base de datos  
// Ahora este registro fue eliminado de la base de datos
```

```
const maria = await User.create({ nombre: "Maria Luz" });  
console.log(maria.nombre); // "Maria Luz"  
maria.nombre = "Alejandra"; //El nombre todavía es Maria Luz en la BD  
await maria.reload(); //console.log(maria.nombre); // "Maria Luz"
```

Consultas

// Busca todos los usuarios

```
const users = await User.findAll();  
console.log(users.every(user => user instanceof User)); // true  
console.log("Todos los usuarios:", JSON.stringify(users, null, 2));
```

users es un arreglo de instancia del modelo
retorna todos los registros de la tablas user
podemos usar los metodos de array
el 2 es la tabulación del objeto

```
Model.findAll({  
  attributes: ['foo', 'bar']  
}); //similar a realizar SELECT foo, bar FROM ...
```

```
Model.findAll({  
  attributes: [ 'foo',  
    [sequelize.fn('COUNT', sequelize.col('hats')), 'alias_hats'],  
    'bar' ]  
}); // Similar a SELECT foo, COUNT(hats) AS alias_hats, bar FROM
```

columna del tipo de agregada
nombre de la funcion, parametro y el alias que le vamos a poner a la columna

Aplicar cláusulas Where

```
Post.findAll({  
  // {....} condición para que filtre (ponemos la columna y la condicion)  
  where: {authorId: 2} // Operador de igualdad por defecto  
}); // SELECT * FROM posts WHERE authorId = 2
```

```
//Equivalente a  
// la clase op hay que importarla  
const { Op } = require("sequelize");  
Post.findAll({  
  // eq es igual que poner =  
  where: { authorId: { [Op.eq]: 2 }  
}
```

```
Post.findAll({  
  // ak podemos aplicar 2 condiciones con el operador and  
  where: { authorId: 12, status: 'active' } //Actua como operador and  
  // traeme todos los autores =al 12 y status este activo  
});  
// SELECT * FROM post WHERE authorId = 12 AND status = 'active';
```


Aplicar cláusulas Where (2)

esto cuando queremos usar el O

```
const { Op } = require("sequelize");
Post.findAll({
  where: {
    [Op.or]: [                ak recuperamos que el autor = 12 O autor =13
      { authorId: 12 },
      { authorId: 13 }
    ]
  }
}); // SELECT * FROM post WHERE
authorId = 12 OR authorId = 13;
```

```
[Op.and]: [{ a: 5 }, { b: 6 }],
[Op.or]: [{ a: 5 }, { b: 6 }],
someAttribute: {
  // Basics
  [Op.eq]: 3,
  [Op.ne]: 20,
  [Op.is]: null,
  [Op.not]: true,
  [Op.or]: [5, 6],

  // Using dialect specific column names
  [Op.col]: 'user.organization_id',

  // Number comparisons
  [Op.gt]: 6,
  [Op.gte]: 6,
  [Op.lt]: 10,
  [Op.lte]: 10,
  [Op.between]: [6, 10],
  [Op.notBetween]: [11, 15],

  // Other operators

  [Op.all]: sequelize.literal('SE

  [Op.in]: [1, 2],
  [Op.notIn]: [1, 2],

  [Op.like]: '%hat',
  [Op.notLike]: '%hat',
  [Op.startsWith]: 'hat',
  [Op.endsWith]: 'hat',
```

Operador IN

```
Post.findAll({
  where: {
    id: [1,2,3] // Mismo que usar `id: { [Op.in]: [1,2,3] }`
  }
});
// SELECT ... FROM "posts" AS "post" WHERE "post"."id" IN (1, 2, 3);
```

// Actualiza a todos los que no tienen apellido con el valor Doe

```
await User.update({ apellido: "Doe" }, {
  where: {
    apellido: null
  }
});
```

este metodo permite actualizar (primer dato es lo que yo quiero remplazar , segundo todo lo que queremos recuperar) nos sirve para remplazar varios registros

Ordenamiento

```
Subtask.findAll({  
  order: [  
    // Ordena por título en orden descendiente  
    ['title', 'DESC'], ak x titulo  
    // Ordena por el máximo de la edad  
    sequelize.fn('max', sequelize.col('age')), por el maximo de edad  
    // Ordena por el máximo de la edad en forma descendiente  
    [sequelize.fn('max', sequelize.col('age')), 'DESC'],  
    // Ordena por otra función (`col1`, 12, 'lalala') Descendiente  
    [sequelize.fn('otherfunction', sequelize.col('col1'), 12, 'lalala'),  
      'DESC'], cualquier otra función que traiga el motor de búsqueda  
    // Ordena por un campo de un modelo relacionado  
    [Task, 'createdAt', 'DESC'],
```

Agrupamiento, límites y paginación

```
// campos agrupados por 'nombre'  
Project.findAll({ group: 'nombre' });
```

```
// Recupera 10 instancias/filas  
Project.findAll({ limit: 10 });
```

```
// Skip 8 instancias/filas  
Project.findAll({ offset: 8 });
```

```
// Skip 5 instancias y recupera 5 instancias después del offset  
Project.findAll({ offset: 5, limit: 5 });
```

Otras funciones útiles

```
console.log(`Existen ${await Project.count()} proyectos`);  
const cantidad = await Project.count({  
  where: {  
    id: {  
      [Op.gt]: 25  
    }  
  }  
});  
console.log(`Hay ${cantidad} proyectos con un id mayor que 25`);
```

```
await User.max('age'); // 40  
await User.max('age', { where: { age: { [Op.lt]: 20 } } }); // 10  
await User.min('age'); // 5  
await User.min('age', { where: { age: { [Op.gt]: 5 } } }); // 10  
await User.sum('age'); // 55  
await User.sum('age', { where: { age: { [Op.gt]: 5 } } }); // 50
```

Buscadores – métodos finders

- FindAll, findByPk, findOne

solo me devuelve una sola instancia

```
const proyecto = await Project.findByPk(123);  
if (proyecto === null) {  
  console.log('No encontrado!'); }  
else { console.log(project instanceof Project); // true
```

```
const proyecto = await Project.findOne({  
  where: { titulo: 'Mi Titulo' } });  
if (proyecto === null) { console.log('No encontrado!'); }  
else { console.log(project instanceof Project); // true  
  console.log(proyecto.titulo); // 'Mi Titulo'  
}
```

Buscadores – métodos finders

FindOrCreate, findAndCountAll

```
const [user, created] = await User.findOrCreate({
  where: { username: 'jose' },
  defaults: { job: 'Experto en JavaScript' }
});
console.log(user.username); // 'jose'
console.log(user.job); // Podría ser o no 'Experto en JavaScript'
if (created) { console.log(user.job); } // 'Experto JavaScript'
```

```
const { count, rows } = await Project.findAndCountAll({
  where: { title: {
    [Op.like]: 'foo%'
  }
}, offset: 10, limit: 2 });
console.log(count);
console.log(rows);
```


Validaciones

```
sequelize.define('foo', {
  bar: {
    type: DataTypes.STRING,
    validate: {
      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 (http://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
      isNumeric: true,          // will only allow numeric
      isInt: true,              // checks for valid integer
      isFloat: true,            // checks for valid float
      isDecimal: true,          // checks for any number
      isLowercase: true,        // checks for lowercase
      isUppercase: true,        // checks for uppercase
      notNull: true,            // won't allow null
      allowNull: 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
    }
  }
});
```

Validadores personalizados

```
// Examples of custom validators:
isEven(value) {
  if (parseInt(value) % 2 !== 0) {
    throw new Error('Only even values are allowed!');
  }
}

isGreaterThanOtherField(value) {
  if (parseInt(value) <= parseInt(this.otherField)) {
    throw new Error('Bar must be greater than otherField.');
```


Validaciones

```
class User extends Model {}  
User.init({  
  edad: Sequelize.INTEGER,  
  nombre: {  
    type: DataTypes.STRING,  
    allowNull: true,  
    validate: {  
      customValidator(value) {  
        if (value === null && this.age < 8) {  
          throw new Error("Nombre no puede ser nulo a menos que edad  
            sea menor a 8");  
        }  
      }  
    })  
  }  
}, { sequelize });
```

Raw Queries

/ Results será un array vacío y metadata contendrá el número de filas afectadas*/*

```
const [results, metadata] = await sequelize.query("UPDATE  
users SET y = 42 WHERE x = 12");
```

/ Podemos pasar un modelo como parámetro. En este caso los datos retornados serán instancias de ese modelo.*/*

```
const projects = await sequelize.query('SELECT * FROM  
projects', {  
  model: Projects,  
  mapToModel: true // true si queremos campos mapeados  
});
```

Sequelize-cli

Se debe instalar como paquete global

```
npm install -g --save-dev sequelize-cli
```

Es una herramienta que permite crear la estructura clásica para una aplicación que usara sequelize.

```
sequelize init
```

Dispone de comandos para crear por la consola modelo, migraciones y seeds

```
sequelize model:create
```

```
sequelize db:migrate
```

```
sequelize db:seed
```


Sequelize-cli

se instala en forma global ya que es una herramienta y no una libreria

sequelize model:create --name Medico --attributes
dni:bigint,mail:string,nombre:string,especialidad:string,estado:
integer

```
"use strict";
const { Model } = require("sequelize");
module.exports = (sequelize, DataTypes) => {
  class Medico extends Model {
    static associate(models) {
      // define association here
    }
  }
  Medico.init(
    [
      dni: DataTypes.BIGINT,
      mail: DataTypes.STRING,
      nombre: DataTypes.STRING,
      especialidad: DataTypes.STRING,
      estado: DataTypes.INTEGER,
    ],
    {
      sequelize,
      modelName: "Medico",
    }
  );
  return Medico;
};
```

sequelize db:migrate

	#	Nombre	Tipo	Cotejamiento
<input type="checkbox"/>	1	id 	int(11)	
<input type="checkbox"/>	2	dni	bigint(20)	
<input type="checkbox"/>	3	mail	varchar(255)	utf8mb4_general_ci
<input type="checkbox"/>	4	nombre	varchar(255)	utf8mb4_general_ci
<input type="checkbox"/>	5	especialidad	varchar(255)	utf8mb4_general_ci
<input type="checkbox"/>	6	estado	int(11)	
<input type="checkbox"/>	7	createdAt	datetime	
<input type="checkbox"/>	8	updatedAt	datetime	

Resumen de Pasos para una app Express-Sequelize-Mysql

- 1.- crear carpeta del proyecto
- 2.- `cd proyecto`
- 3.- `npm init`
- 4.- `npm install express`
- 5.- `npm install --save sequelize`
- 6.- `npm install --save mysql2`
- 7.- `npm install --save-dev nodemon` (opcional)
- 8.- Actualizar package.json para hacer funcionar nodemon

```
"scripts": {  
  "start": "node index.js"  
},
```

Resumen de Pasos para una app Express-Sequelize-Mysql

9.- `npm install -g sequelize-cli` (Setea herramienta de Sequelize)
(Solo si no la tenemos)

10.- `sequelize init` (Crea estructura de carpetas neces.)

11.- Configuramos los datos de conexión para el archivo `config.js`
Que utilizara la herramienta sequelize-cli para correr migraciones.

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

Resumen de Pasos para una app Express-Sequelize-Mysql

12.- Crear la Base de Datos Vacía

13.- Generamos un index.js con la estructura normal

14.- `npm run start` (levantar el servidor)

15.- probar <http://localhost:3000>

16.- Generar los Modelos

`sequelize model: create --name Model --attributes col1:tipo1`

17.- Ejecutar las migraciones para crear las tablas

`sequelize db:migrate`

18.- Agregar las asociaciones en nuestros modelos

19.- Agregamos las rutas en nuestro index

Migrar desde la BD

Crear un Modelo

```
sequelize model:create --name Persona --attributes  
nombre:string,domicilio:string,telefono:string
```

Ejecutar migraciones

```
Sequelize db:migrate
```

Revertir la última migración

```
sequelize db:migrate:undo
```