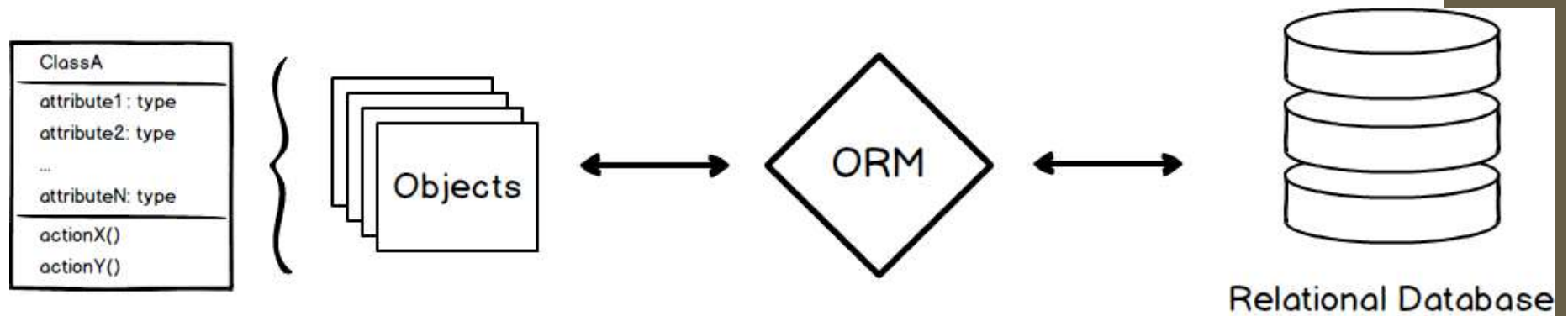


ORM

- Mapeo Objeto Relacional
- Es un Puente entre el paradigma orientado a objetos y las bases de datos relacionales.



Sequelize

- 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`
- `npm install mysql2`

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

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

```
var sequelize = new
```

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

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

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

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

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

```
  Logging: false
```

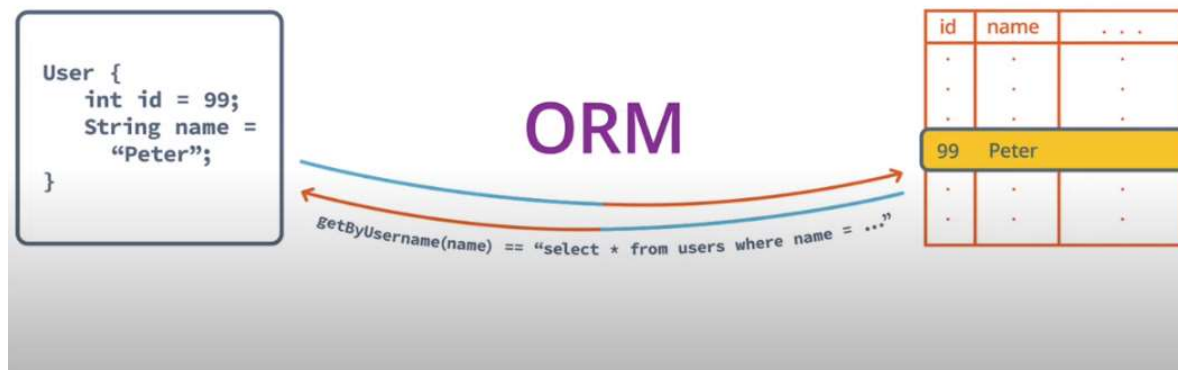
```
});
```

Probando la conexión

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

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**



Definir un Modelo (User.js)

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

```
class Usuario extends Model {
```

```
  Usuario.init({
```

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

```
    nombre: {
```

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

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

```
    },
```

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

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

```
  }, {
```


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

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

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

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

```
  });
```



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

Timestamps

```
class Foo extends Model {}  
Foo.init({ /* attributes */ }, {  
  sequelize,
```

```
  // No olvidar activar timestamps  
  timestamps: true,
```

```
  // Si no queremos el campo createdAt  
  createdAt: false,
```

```
  // Si queremos updatedAt pero con otro nombre  
  updatedAt: 'fechaActualizacion'  
});
```


Sincronización

- `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

```
title: { type: DataTypes.STRING, allowNull: false },  
/* 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 un índice único para la columna.  
someUnique: { type: DataTypes.STRING, unique: true },  
  
// 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 classLevelMethod() { //define métodos de clase  
    return 'foo';  
  }  
  getNombreCompleto() { //define método de instancia  
    return [this.nombre, this.apellido].join(' ');  
  }  
}  
User.init({  
  nombre: DataTypes.STRING,  
  apellido: DataTypes.STRING  
}, { sequelize });  
  
console.log(Usuario.classLevelMethod()); // 'foo'  
const user = Usuario.build({ nombre: 'María', apellido: 'Gomez' }); //instancia modelo  
console.log(Usuario.getNombreCompleto()); // 'María Gomez',
```

Instanciar Modelos

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

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

```
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({ name: "Jose" });  
console.log(joseObj.nombre); // "Jose"  
joseObj.nombre = "Lucas";  
// El nombre todavía es "Jose" en la base de datos  
await joseObj.save();  
// 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();  
// Ahora este registro fue eliminado de la base de datos
```

```
const jane = await User.create({ name: "Jane" });  
console.log(jane.name); // "Jane"  
jane.name = "Ada"; //El nombre todavía es Jane en la BD  
await jane.reload(); //console.log(jane.name); // "Jane"
```

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));
```

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

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

Aplicar cláusulas Where

```
Post.findAll({  
  where: {authorId: 2} // Operador de igualdad por defecto  
}); // SELECT * FROM posts WHERE authorId = 2
```

//Equivalente a

```
const { Op } = require("sequelize");  
Post.findAll({  
  where: { authorId: { [Op.eq]: 2 }  
  }  
}
```

```
Post.findAll({  
  where: { authorId: 12, status: 'active' } //Actua como operador and  
});  
// SELECT * FROM post WHERE authorId = 12 AND status = 'active';
```


Aplicar cláusulas Where (2)

```
const { Op } = require("sequelize");
Post.findAll({
  where: {
    [Op.or]: [
      { 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  
  }  
});
```

Ordenamiento

```
Subtask.findAll({  
  order: [  
    // Ordena por título en orden descendiente  
    ['title', 'DESC'],  
    // Ordena por el máximo de la edad  
    sequelize.fn('max', sequelize.col('age')),  
    // Ordena por el máximo de la edad en forma descendiente  
    [sequelize.fn('max', sequelize.col('age')), 'DESC'],  
    // Ordena por otra función (`col1`, 12, 'lalala') Descendente  
    [sequelize.fn('otherfunction', sequelize.col('col1'), 12, 'lalala'),  
    'DESC'],  
    // 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(`There are ${await Project.count()} projects`);  
const amount = await Project.count({  
  where: {  
    id: {  
      [Op.gt]: 25  
    }  
  }  
});  
console.log(`Hay ${amount} 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 finds

- FindAll, findByPk, findOne

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

```
const project = await Project.findOne({  
  where: { title: 'My Title' } });  
if (project === null) { console.log('Not encontrado!'); }  
else { console.log(project instanceof Project); // true  
  console.log(project.title); // 'My Title'  
}
```

Buscadores – métodos finds

FindOrCreate, findAndCountAll

```
const [user, created] = await User.findOrCreate({
  where: { username: 'sdepold' },
  defaults: { job: 'Technical Lead JavaScript' }
});
console.log(user.username); // 'sdepold'
console.log(user.job); // Podría ser o no 'Technical Lead JavaScript'
if (created) { console.log(user.job); } // 'Technical Lead 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 s
      not: /^[a-z]+$/i,         // does not match this RegExp
      not: ["^[a-z]+$",'i'],     // same as above, but constructing the RegExp from a s
      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 lette
      isAlphanumeric: true,     // will only allow alpha
      isNumeric: true,          // will only allow numbe
      isInt: true,              // checks for valid inte
      isFloat: true,            // checks for valid floa
      isDecimal: true,          // checks for any number
      isLowercase: true,        // checks for lowercase
      isUppercase: true,        // checks for uppercase
      allowNull: true,         // won't allow null
      isNull: true,            // only allows null
      notEmpty: true,          // don't allow empty str
      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.');
```


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  
});
```

Asociaciones

One-To-One, One-To-Many y Many-To-Many

Para implementar estas asociaciones sequelize utiliza 4 tipos de asociaciones

- HasOne
- BelongsTo
- HasMany
- BelongsToMany

`A.hasOne(B);` // una instancia de A tiene una instancia de B

`A.belongsTo(B);` // una instancia de A pertenece a una instancia de B

`A.hasMany(B);` // una instancia de A tiene muchas instancias de B

`A.belongsToMany(B, { through: 'C' });` // una instancia de A pertenece a una instancia de B a través de la table asociativa C

Asociación one-to-one

CHILD MODEL	PARENT MODEL	MAGIC METHODS														
<table><tr><td>id</td><td>name</td><td>age</td><td>parentid</td></tr><tr><td colspan="4">CHILD INSTANCE DATA</td></tr></table>	id	name	age	parentid	CHILD INSTANCE DATA				<table><tr><td>id</td><td>name</td><td>age</td></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<pre>parent.getChild() parent.setChild(child) parent.createChild(child)</pre>
id	name	age	parentid													
CHILD INSTANCE DATA																
id	name	age														
PARENT INSTANCE DATA																
<p>Parent.addChild(child);</p> <table><tr><td>id</td><td>name</td><td>age</td><td>onlyChildid</td></tr><tr><td colspan="4">CHILD INSTANCE DATA</td></tr></table>	id	name	age	onlyChildid	CHILD INSTANCE DATA				<table><tr><td>id</td><td>name</td><td>age</td></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<pre>parent.getOnlyChild() parent.setOnlyChild(child) parent.createOnlyChild(child)</pre>
id	name	age	onlyChildid													
CHILD INSTANCE DATA																
id	name	age														
PARENT INSTANCE DATA																
<p>Parent.addChild(child, {as:'OnlyChild'});</p> <table><tr><td>id</td><td>name</td><td>age</td><td>motherid</td></tr><tr><td colspan="4">CHILD INSTANCE DATA</td></tr></table>	id	name	age	motherid	CHILD INSTANCE DATA				<table><tr><td>id</td><td>name</td><td>age</td></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<pre>parent.getOnlyChild() parent.setOnlyChild(child) parent.createOnlyChild(child)</pre>
id	name	age	motherid													
CHILD INSTANCE DATA																
id	name	age														
PARENT INSTANCE DATA																
<p>Parent.addChild(child, {as:'OnlyChild',foreign-key:'motherId'});</p>																

Asociación one-to-many (1:M)

CHILD MODEL	PARENT MODEL	MAGIC METHODS														
<table><tr><td>id</td><td>name</td><td>age</td><td>parentId</td></tr><tr><td colspan="4">CHILD INSTANCE DATA</td></tr></table>	id	name	age	parentId	CHILD INSTANCE DATA				<table><tr><td>id</td><td>name</td><td>age</td></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<pre>parent.getChildren() parent.countChildren() parent.hasChild() parent.hasChildren() parent.setChildren(children) parent.addChild(child) parent.addChildren(children) parent.removeChild(child) parent.removeChildren(children) parent.createChild(child)</pre>
id	name	age	parentId													
CHILD INSTANCE DATA																
id	name	age														
PARENT INSTANCE DATA																
<p>Child;</p> <table><tr><td>id</td><td>name</td><td>age</td><td>parentId</td></tr><tr><td colspan="4">CHILD INSTANCE DATA</td></tr></table>	id	name	age	parentId	CHILD INSTANCE DATA				<table><tr><td>id</td><td>name</td><td>age</td></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<pre>parent.getOffspring() parent.countOffspring() parent.hasOffspring() parent.setOffspring(children) parent.addOffspring(children) parent.removeOffspring(child) parent.createOffspring(child)</pre>
id	name	age	parentId													
CHILD INSTANCE DATA																
id	name	age														
PARENT INSTANCE DATA																
<p>Child, {as:'offspring'});</p> <table><tr><td>id</td><td>name</td><td>age</td><td>foreignId</td></tr><tr><td colspan="4">CHILD INSTANCE DATA</td></tr></table>	id	name	age	foreignId	CHILD INSTANCE DATA				<table><tr><td>id</td><td>name</td><td>age</td></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<pre>parent.getOffspring() parent.countOffspring() parent.hasOffspring() parent.setOffspring(children) parent.addOffspring(children) parent.removeOffspring(child) parent.createOffspring(child)</pre>
id	name	age	foreignId													
CHILD INSTANCE DATA																
id	name	age														
PARENT INSTANCE DATA																
<p>child, {as:'offspring' foreign key:'foreignId'});</p>																

Parent.hasMany(Child);

Parent.hasMany(Child, {as:'offspring'});

Parent.hasMany(Child, {as:'offspring',foreign-key:'foreignId'});

Asociación many-to-many (N:M)

CHILD MODEL	PARENT MODEL	NEW MODEL	MAGIC METHODS															
<table><tr><th>id</th><th>name</th><th>age</th></tr><tr><td colspan="3">CHILD INSTANCE DATA</td></tr></table>	id	name	age	CHILD INSTANCE DATA			<table><tr><th>id</th><th>name</th><th>age</th></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<table><tr><th>parentid</th><th>childid</th></tr><tr><td colspan="2">PARENT_CHILD INSTANCE DATA</td></tr></table> <pre>parent.getChildren() parent.countChildren() parent.hasChild() parent.hasChildren() parent.setChildren(children) parent.addChild(child) parent.addChildren(children) parent.removeChild(child) parent.removeChildren(children) parent.createChild(child)</pre>	parentid	childid	PARENT_CHILD INSTANCE DATA	
id	name	age																
CHILD INSTANCE DATA																		
id	name	age																
PARENT INSTANCE DATA																		
parentid	childid																	
PARENT_CHILD INSTANCE DATA																		
<table><tr><th>id</th><th>name</th><th>age</th></tr><tr><td colspan="3">CHILD INSTANCE DATA</td></tr></table>	id	name	age	CHILD INSTANCE DATA			<table><tr><th>id</th><th>name</th><th>age</th></tr><tr><td colspan="3">PARENT INSTANCE DATA</td></tr></table>	id	name	age	PARENT INSTANCE DATA			<table><tr><th>sourceFK</th><th>targetFK</th></tr><tr><td colspan="2">PARENT_CHILD INSTANCE DATA</td></tr></table> <pre>parent.getChildren() parent.countChildren() parent.hasChild() parent.hasChildren() parent.setChildren(children) parent.addChild(child) parent.addChildren(children) parent.removeChild(child) parent.removeChildren(children) parent.createChild(child)</pre>	sourceFK	targetFK	PARENT_CHILD INSTANCE DATA	
id	name	age																
CHILD INSTANCE DATA																		
id	name	age																
PARENT INSTANCE DATA																		
sourceFK	targetFK																	
PARENT_CHILD INSTANCE DATA																		

Parent.belongsToMany(Child, {through: 'Parent_Child'});

Parent.belongsToMany(Child, {through: 'Parent_Child', foreignKey: 'sourceFK', otherKey: 'targetFK'})

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.- Preparamos los paths de las carpetas sequelize

Creamos un archivo en la raíz `.sequelizerc` y agregamos:

```
const path = require('path');  
module.exports = {  
  "config": path.resolve('./config', 'config.json'),  
  "models-path": path.resolve('./models'),  
  "seeders-path": path.resolve('./seeders'),  
  "migrations-path": path.resolve('./migrations')  
};
```

Resumen de Pasos para una appMVC Express-Sequelize-Mysql

12.- 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

13.- Crear la Base de Datos Vacía

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

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

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

17.- Generar los Modelos

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

18.- Ejecutar las migraciones para crear las tablas

`sequelize db:migrate`

19.- Agregar las asociaciones en nuestros modelos

20.- 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
```