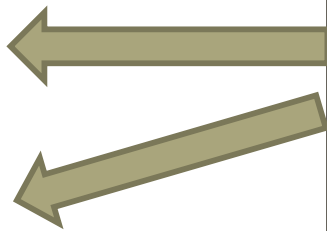


Definir la asociación **hasOne**

```
Parent.hasOne(Child);  
Persona.hasOne(Usuario);
```

```
Parent.hasOne(Child, {as:'OnlyChild'});  
Persona.hasOne(Usuario, {as:'cuenta'});
```



Deben ser
definidas en el
modelo Padre

Importante!!!

No es necesario definir en los modelos los atributos de Claves Primarias y Foraneas a excepción de tablas intermedias

```
Correo.hasOne(Persona, { as:  
'Remitente' });  
Correo.hasOne(Persona, { as:  
'Receptor' });
```

...

...

```
objCorreo.getRemitente()  
objCorreo.getReceptor()
```

```
objCorreo.setReceptor()  
objCorreo.createRemitente()
```

Donde definir la Asociación:

Si usamos sequelize-cli

```
Const {Model} = require("sequelize")
Module.exports = (sequelize, DataTypes) => {
  class Foo extends Model {
    static associate(models) {
      1 Foo.hasOne(models.Bar)
        //Foo.hasOne(models.Bar, {foreignKey:'aliasfk'})
    }
  }
}
```

Usar Asociacion: Foo.hasOne(Bar)

```
instanciafoo.getBar()  
instanciafoo.setBar()  
instanciafoo.createBar()
```

```
const foo = await Foo.create({ name: 'the-foo' });  
const bar1 = await Bar.create({ name: 'some-bar' });  
const bar2 = await Bar.create({ name: 'another-bar' });  
console.log(await foo.getBar()); // null  
await foo.setBar(bar1);  
console.log((await foo.getBar()).name); // 'some-bar'  
await foo.createBar({ name: 'yet-another-bar' });  
const newlyAssociatedBar = await foo.getBar();  
console.log(newlyAssociatedBar.name); // 'yet-another-bar'  
await foo.setBar(null); // Un-associate  
console.log(await foo.getBar()); // null
```

Definir la asociación belongsTo

```
Parent.belongsTo(Child);  
Persona.belongsTo(Localidad);
```

Usar la asociación belongsTo

```
persona1.getLocalidad()  
persona1.setLocalidad(localidad1)  
persona1.createLocalidad(objLocalidad)
```

Definir la asociacion **hasMany**

```
Parent.hasMany(Child);  
Equipo.hasMany(Jugador);
```

```
Parent.hasMany(Child, {as:'nombreDeAlias'});  
Equipo.hasMany(Jugador, {as:'integrante'});
```

```
Parent.hasMany(Child, {as:'unAlias',foreignkey:'foreignId'});  
Equipo.hasMany(Jugador,  
{as:'integrante'},foreignKey:'myjugadorId' );
```

```
Jugador.belongsTo(Equipo, foreignKey:'myjugadorId')
```

Foo.hasMany(Bar)

- fooInstance.getBars()
- fooInstance.countBars()
- fooInstance.hasBar()
- fooInstance.hasBars()
- fooInstance.setBars()
- fooInstance.addBar()
- fooInstance.addBars()
- fooInstance.removeBar()
- fooInstance.removeBars()
- fooInstance.createBar()

Foo.hasMany(Bar)

```
const foo = await Foo.create({ name: 'the-foo' });
const bar1 = await Bar.create({ name: 'some-bar' });
const bar2 = await Bar.create({ name: 'another-bar' });
console.log(await foo.getBars()); // []
console.log(await foo.countBars()); // 0
console.log(await foo.hasBar(bar1)); // false
await foo.addBars([bar1, bar2]);
console.log(await foo.countBars()); // 2
await foo.addBar(bar1); //es omitido por ya estar relacionado
console.log(await foo.countBars()); // 2
console.log(await foo.hasBar(bar1)); // true
await foo.removeBar(bar2);
console.log(await foo.countBars()); // 1
await foo.createBar({ name: 'yet-another-bar' });
console.log(await foo.countBars()); // 2
await foo.setBars([]); // Desasocia todas las barras asociadas
console.log(await foo.countBars()); // 0
```


Métodos getter

```
const easyTasks = await projectA.getTasks({  
  where: {  
    difficulty: {  
      [Op.lte]: 5  
    }  
  }  
});  
  
const taskTitles = (await projectB.getTasks({  
  attributes: ['title'],  
  raw: true  
})).map(task => task.title);
```

Definir la asociación belongsToMany

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

```
Movie.belongsToMany(Actor, { through: ActorMovies });  
Actor.belongsToMany(Movie, { through: ActorMovies });
```


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

```
Movie.belongsToMany(Actor, { through: ActorMovies,  
foreignKey:'movieId', otherKey:'actorId'});
```

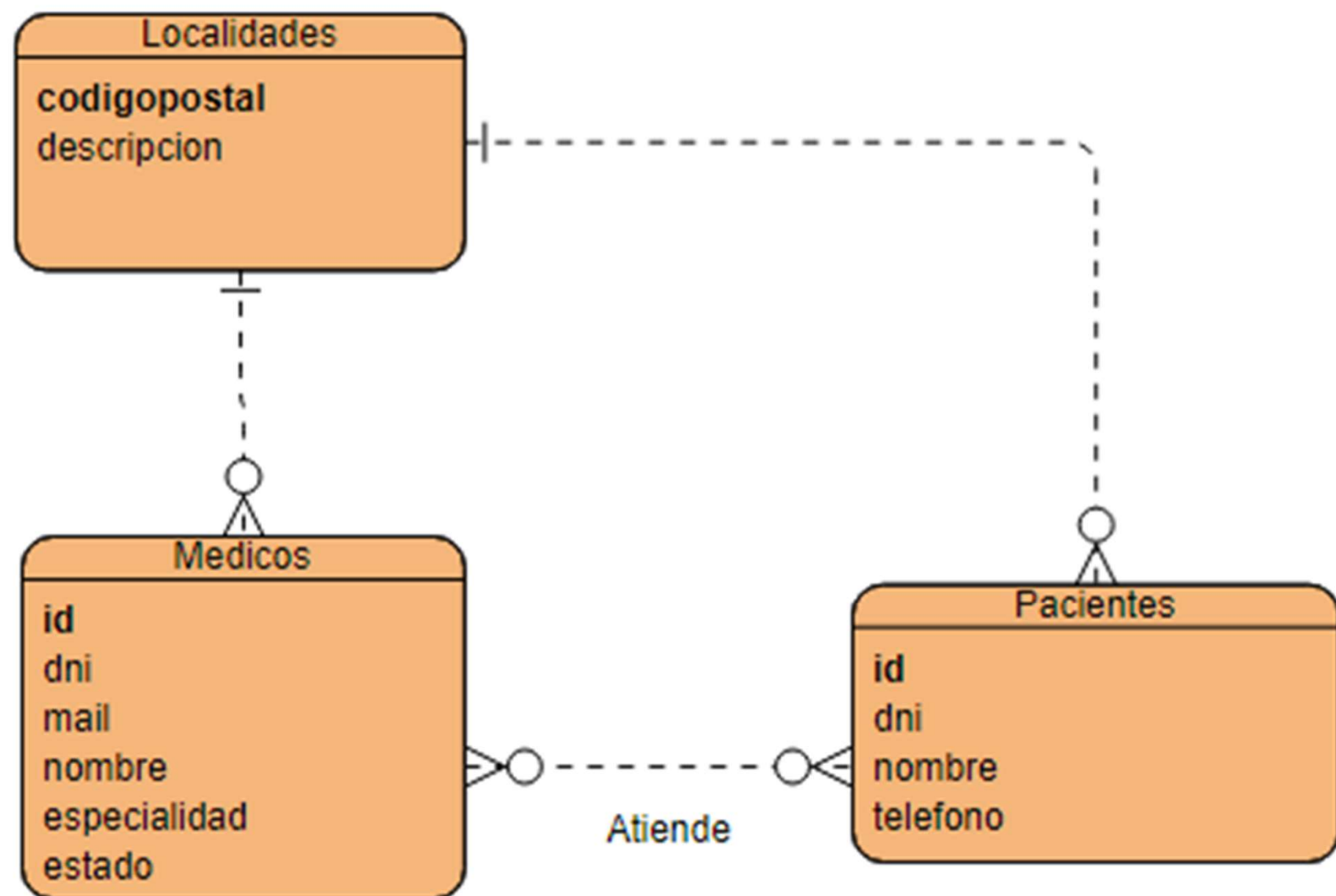
```
Parent.belongsToMany(Child, { through: Parent-Child,  
uniqueKey: 'mi_custom_id' })
```

Foo.belongsToMany(Bar, { through:Baz})

Mismos que para hasMany()

fooInstance.getBars() 
fooInstance.countBars()
fooInstance.hasBar()
fooInstance.hasBars()
fooInstance.setBars()
fooInstance.addBar()
fooInstance.addBars()
fooInstance.removeBar()
fooInstance.removeBars()
fooInstance.createBar()

```
[  
  {"id": 1,  
    "name": "bar1",  
    "baz1": {  
      "idbar": 1,  
      "idfoo": 1,  
      "name": "Baz1",  
    }  
  },  
  {"id": 2,  
    "name": "bar2",  
    "baz2": {  
      "idbar": 2,  
      "idfoo": 1,  
      "name": "Baz2",  
    }  
  }  
]
```



Consultas Usando las asociaciones

Lazy Loading vs Eager Loading

Lazy Loading = Carga Diferida

```
const medico = await Medico.findByPk(5);  
// hacer cosas con el médico  
console.log('Nombre:', medico.nombre);  
console.log('Especialidad:', medico.especialidad);  
// Ahora queremos información sobre su Localidad  
const loc = await medico.getLocalidad();  
// Hacer cosas con la localidad  
console.log('Nombre:', loc.descripcion);
```

Consultas Usando las asociaciones

Lazy Loading vs Eager Loading

Eager Loading = Carga Ansiosa o Temprana

```
const medico = Medico.findByPk(5, {  
  include: Localidad  
});
```

// Ahora la localidad ya viene incluida en el medico

```
console.log('Nombre:', medico.nombre);  
console.log('Especialidad:', medico.especialidad);  
console.log('Localidad:', medico.Localidad.descripcion);
```

Eager Loading

¿Como retorna los datos?

```
const medicos = await Medico.findAll({ include: Localidad });  
console.log(JSON.stringify(medicos, null, 2));
```

```
[  
  {  
    "nombre": "Juan Medico",  
    "id": 1,  
    "dni": 1111,  
    "especialidad": "Cirujano General"  
    "localidadId": 1,  
    "Localidad": {  
      "descripcion": "Capital",  
      "id": 1  
    }  
  }  
]
```

hasOne
belongsTo

medicos[0].localidad
// 1

Medicos[0].Localidad.descripcion
//Capital

Eager Loading

¿Como retorna los datos?

```
const medicos2 = await  
Medico.findAll({ include:  
Paciente });
```

```
console.log(JSON.stringify  
(medicos2, null, 2));
```

hasMany belongsToMany

```
"id": 1,  
  "dni": 1111,  
  "nombre": "Juan Medico",  
  "localidadId": 1,  
  "Pacientes": [  
    {  
      "id": 1,  
      "dni": 3333,  
      "nombre": "Ana",  
      "Atencion": {  
        "medicoid": 1,  
        "pacientId": 1,  
        "fecha": "2021-05-31T21:03:39.000Z",  
      }  
    }, {  
      "id": 2,  
      "dni": 4444,  
      "nombre": "Lucas",  
      "Atencion": {  
        "medicoid": 1,  
        "pacientId": 2,  
        "fecha": "2021-05-30T21:03:39.000Z",  
        "Medicoid": 1  
      }  
    }  
  ]  
}
```


Super Many-to-Many

```
Medico.belongsToMany(Paciente, { through: Atencion });  
Paciente.belongsToMany(Medico, { through: Atencion });
```

```
Paciente.hasMany(Atencion);  
Atencion.belongsTo(Paciente);  
Medico.hasMany(Atencion);  
Atencion.belongsTo(Medico);
```

//Todo esto funcionará

```
Medico.findAll({ include: Paciente });  
Paciente.findAll({ include: Medico });  
Medico.findAll({ include: Atencion });  
Paciente.findAll({ include: Atencion });  
Atencion.findAll({ include: Medico });  
Atencion.findAll({ include: Paciente });
```

//Todo esto funcionará

```
await objmedico.getPacientes();  
await objpaciente.getMedicos();  
await objmedico.getAtencions();  
await objpaciente.getAtencions();  
await objatenc.getMedicos();  
await objatencion.getPacientes();
```