3 quan hệ :

* 1 – 1
* 1 – N
* N – N

4 associations :

* hasOne(B) : FK nằm ở B
* belongsTo(B) : FK nằm ở A
* hasMany(B) : FK nằm ở B
* belongsToMany(B, { through: 'C' }) : sử dụng table C (chứa FK : aId & bId)

Thiết lập Relationship

* Tạo 1 – 1 = dùng hasOne + belongsTo cùng nhau
* Tạo 1 – N = dùng hasMany + belongsTo cùng nhau
* Tạo N – N dùng belongsToMany + belongsToMany cùng nhau

## Lý do tại sao ta cần phải defined về association ở cả 2 models

Giải thích :

- Defined ở 1 thì cũng được thôi sẽ không bị lỗi gì cả (khi defined)

Nhưng khi ta sử dụng Model thì sẽ xảy ra trường hợp

Nếu không defined associations ở cả 2 models [chỉ gọi Foo.hasOne(Bar)]

// This works...

await Foo.findOne({ include: Bar });

// But this throws an error:

await Bar.findOne({ include: Foo });

// SequelizeEagerLoadingError: foo is not associated to bar!

Nếu defined associations ở cả 2 models [cả Foo.hasOne(Bar) & Bar.belongsTo(Foo)]

// This works!

await Foo.findOne({ include: Bar });

// This also works!

await Bar.findOne({ include: Foo });

## Quan hệ 1 – 1

Điều ta cần lưu ý ở 1 – 1 là ta sẽ đặt FK ở đâu, vì không như 1 – N hay N – N thì 1 – 1 ta có thể đặt FK ở bất kì 1 trong 2 Model

Đặt ở đâu thì cần phải tự hỏi rằng :

* Model A có thể exist nếu thiếu B không & ngược lại

Còn nếu cả 2 đều không bị phụ thuộc bởi model còn lại 🡪 thì thích FK ở đâu chẳng được

Thông thường, thì tất cả quan hệ đều default là allowNull 🡪 Không có ràng buộc về sự phụ thuộc lẫn nhau như này

Foo.hasOne(Bar, {

foreignKey: {

allowNull: true

}

});

\*Nếu muốn có rằng buộc thì mình đổi lại: true 🡪 false

### Implement

Foo.hasOne(Bar);

Bar.belongsTo(Foo, {

foreignKey: {

name: 'myFooId'

},

type: DataTypes.Interger,

allowNull: false,

defaultValue: …

});

* foreignKey (string | object)

onDelete & onUpdate

Foo.hasOne(Bar, {

onDelete: 'RESTRICT',

onUpdate: 'RESTRICT'

});

Bar.belongsTo(Foo);

Options:

* “RESTRICT”
* “CASCADE”
* “NO ACTION”
* “SET DEFAULT”
* “SET NULL”

1 – 1 Default :

* onDelete: “SET NULL”
* onUpdate: “CASCADE”

## Quan hệ 1 – Nhiều

Team.hasMany(Player);

Player.belongsTo(Team, {

foreignKey: 'clubId'

});

1 – N default :

* onDelete: “SET NULL”
* onUpdate: “CASCADE”

## Quan hệ N - N

const Movie = sequelize.define('Movie', { name: DataTypes.STRING });

const Actor = sequelize.define('Actor', { name: DataTypes.STRING });

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

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

\*mặc định, sqlize tự động tạo 1 table “ActionMovies” như dưới

CREATE TABLE IF NOT EXISTS "ActorMovies" (

"createdAt" TIMESTAMP WITH TIME ZONE NOT NULL,

"updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL,

"MovieId" INTEGER REFERENCES "Movies" ("id") ON DELETE CASCADE ON UPDATE CASCADE,

"ActorId" INTEGER REFERENCES "Actors" ("id") ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY ("MovieId","ActorId")

);

Manually Create Model bằng cách

const Movie = sequelize.define('Movie', { name: DataTypes.STRING });

const Actor = sequelize.define('Actor', { name: DataTypes.STRING });

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

MovieId: {

type: DataTypes.INTEGER,

references: {

model: Movie, // 'Movies' would also work

key: 'id'

}

},

ActorId: {

type: DataTypes.INTEGER,

references: {

model: Actor, // 'Actors' would also work

key: 'id'

}

}

});

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

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

// Kết quả

CREATE TABLE IF NOT EXISTS "ActorMovies" (

"MovieId" INTEGER NOT NULL REFERENCES "Movies" ("id") ON DELETE RESTRICT ON UPDATE CASCADE,

"ActorId" INTEGER NOT NULL REFERENCES "Actors" ("id") ON DELETE RESTRICT ON UPDATE CASCADE,

"createdAt" TIMESTAMP WITH TIME ZONE NOT NULL,

"updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL,

UNIQUE ("MovieId", "ActorId"),

PRIMARY KEY ("MovieId","ActorId")

);

\*special option (chỉ cho N - N)

Project.belongsToMany(User, {

through: UserProjects,

uniqueKey: 'my\_custom\_unique' / false

})

To prevent creating this unique key, use the **unique: false** option.

N – N default :

* onDelete: “CASCADE”
* onUpdate: “CASCADE”

# Create, Update & Delete

## Create

* Direct
* Special methods / mixins (~ method này tự động được thêm bởi sqlize)

### Direct

Bar.create({

name: 'My Bar',

fooId: 5

});

### Methods

### Foo.hasOne(Bar)

fooInstance.getBar()

fooInstance.setBar()

fooInstance.createBar()

\*Foo.belongsTo(Bar) = Foo.hasOne(Bar)

Impl

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

### Foo.hasMany(Bar)

fooInstance.getBars()

fooInstance.countBars()

fooInstance.hasBar()

fooInstance.hasBars()

fooInstance.setBars()

fooInstance.addBar()

fooInstance.addBars()

fooInstance.removeBar()

fooInstance.removeBars()

fooInstance.createBar()

\*Foo.belongsToMany(Bar, { through: Baz }) =  Foo.hasMany(Bar)

Impl

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);

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([]); // Un-associate all previously associated bars

console.log(await foo.countBars()); // 0

\*when using alias

### Task.hasOne(User, { as: 'Author' })

taskInstance.getAuthor()

taskInstance.setAuthor()

taskInstance.createAuthor()

Options

const easyTasks = await project.getTasks({

where: {

difficulty: {

[Op.lte]: 5

}

}

});

const taskTitles = (await project.getTasks({

attributes: ['title'],

raw: true

})).map(task => task.title);