MANEJOS DE RELACIONES EN UN ORM: RELACIONES MUCHOS A MUCHOS

TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- ¿Qué es una relación muchos a muchos?
- Definiendo entidades intermedias con más campos.
- Relaciones uno a uno.

¿QUÉ ES UNA RELACIÓN MUCHOS A MUCHOS?

Éstas se producen cuando varios registros de una tabla, se asocian a varios registros de otra tabla. Por ejemplo: existe una relación de muchos a muchos entre una tabla de clientes, y otra tabla de productos; los clientes pueden comprar varios productos, y los productos pueden ser comprados por varios clientes.

Esta relación no se puede representar agregando una clave foránea a una de las tablas, y en su lugar, se utiliza el concepto de modelo de unión. Éste será adicional (y una tabla adicional en la base de datos), tendrá dos columnas de clave foránea, y realizará un seguimiento de las asociaciones. Dicha tabla de unión también se conoce como tabla de paso.

¿CUÁNDO SE UTILIZA?

Se pueden considerar diversas situaciones en las que existan relaciones muchos a muchos. Por ejemplo: pensemos en los modelos Película y Actor. Un actor puede haber participado en muchas películas, y una película tuvo muchos actores involucrados en su producción. El modelo de unión que mantendrá el seguimiento de esta relación, se puede llamar *ActorPeliculas*, y contendrá las claves foráneas: *peliculaId* y *actorId*.

También podemos ejemplificar el uso de esta relación, en el caso de que un estudiante puede inscribirse en varias clases, y a su vez, una clase puede tener varios estudiantes; aquí es posible conocer las clases que ha inscrito un estudiante, y todos los estudiantes inscritos en una clase.

¿CÓMO SE IMPLEMENTA?

Se considera el ejemplo asociado con los modelos Actor y Películas, y se ejemplifica la implementación en el ORM Sequelize, de la siguiente manera:

```
const Pelicula = sequelize.define('Pelicula', { nombre:
DataTypes.STRING });
const Actor = sequelize.define('Actor', { nombre: DataTypes.STRING });
Pelicula.belongsToMany(Actor, { through: 'ActorPeliculas' });
Actor.belongsToMany(Pelicula, { through: 'ActorPeliculas' });
```

Dado que se proporcionó una cadena en la opción directa de la llamada ***belongsToMany***, Sequelize creará automáticamente el modelo ActorPeliculas, el cual actuará como modelo de unión.

DEFINIENDO ENTIDADES INTERMEDIAS CON MÁS CAMPOS

Generalmente, las entidades intermedias o tablas de unión, pueden contener otros atributos además de las claves foráneas, los cuales quizás no tengan sentido en otras tablas. Por ejemplo, si consideramos el caso anterior, puede existir un atributo tipo fecha para mantener un registro de cuando un actor inició en la grabación de la película.

Cuando una tabla unión tiene atributos adicionales, éstos se pueden incluir en el objeto de opciones, de la siguiente manera:

```
ActorPeliculas = sequelize.define('actor_pelicula', {
                role: Sequelize.STRING
});
Actor.belongsToMany(Pelicula, {
                through: ActorPeliculas
});
Pelicula.belongsToMany(Actor, {
                through: ActorPeliculas
}); // through es requerido!

actor.addPelicula(pelicula, {
                through: { role: 'director' }
});
```

RELACIONES UNO A UNO

¿Qué son?

Se presentan cuando un registro de una tabla sólo está relacionado con un registro de otra tabla, y viceversa.

Por ejemplo: consideremos que nuestros empleados deben almacenar su información de contacto. Para este caso, pudiésemos leer la relación de la siguiente manera:

«Un empleado tiene una sola información de contacto», y «Una información de contacto pertenece a un solo empleado».

Dado que la información de contacto es la que depende principalmente del empleado, es en ella donde existirá la clave foránea para representar el vínculo.

¿Cuándo se utiliza?

Existen algunas situaciones en las que se justifica el uso de las relaciones uno a uno, entre las cuales se podrían mencionar:

- Cuando se tienen que añadir campos a una tabla, pero la forma en la que se han escrito las consultas hace que sea muy arriesgado añadir campos en una ya existente, el camino seguro será mantener una tabla adicional con los nuevos campos. Si bien no es la mejor práctica, puede minimizar los riesgos.

- Cuando hay un particionamiento de información, y algunos campos que son poco utilizados, se pueden mantener en una tabla diferente, y por lo tanto, en una partición diferente. Esto se presentaría, por ejemplo, en los datos de un domicilio que puede que no sea muy utilizado, y que se mantienen en una partición separada del resto de los datos de una tabla.

¿Cómo se implementa en un modelo?

Supongamos que tenemos dos modelos: Foo y Bar; y queremos establecer una relación uno a uno entre ambos, donde la tabla Bar contendrá la columna fooId.

La implementación se haría de la siguiente manera:

```
Foo.hasOne(Bar);
Bar.belongsTo(Foo);
```

Como no se agregó ninguna opción al momento de establecer las relaciones, Sequelize inferirá qué hacer a partir de los nombres de los modelos. En este caso, sabe que se debe agregar una columna fooId a la tabla Bar.

Existen varias opciones que pueden ser pasadas como parámetros en las relaciones uno a uno.

Por ejemplo, para configurar el comportamiento de los datos en ON DELETE y ON UPDATE, se puede:

```
Foo.hasOne(Bar, {
 onDelete: 'RESTRICT',
 onUpdate: 'RESTRICT'
});
Bar.belongsTo(Foo);
```

Los posibles valores son: RESTRICT, CASCADE, NO ACTION, SET DEFAULT y SET NULL.

En el caso de querer definir la clave foránea:

Opción 1:

```
        Foo.hasOne(Bar, {
                foreignKey: 'myFooId'
        });
        Bar.belongsTo(Foo);
```

Opción 2:

```
        Foo.hasOne(Bar, {
          foreignKey: {
            name: 'myFooId'
          }
        });
        Bar.belongsTo(Foo);
```

Opción 3:

```
        Foo.hasOne(Bar);
        Bar.belongsTo(Foo, {
```

```
3           foreignKey: 'myFooId'
4         });
```

Opción 4:

```
1 Foo.hasOne(Bar);
2         Bar.belongsTo(Foo, {
3           foreignKey: {
4             name: 'myFooId'
5           }
6         });
```