Associations

Sequelize supports the standard associations: One-To-One, One-To-Many and Many-To-Many.

To do this, Sequelize provides **four** types of associations that should be combined to create them:

- The `HasOne` association
- The `BelongsTo` association
- The `HasMany` association
- The `BelongsToMany` association

The guide will start explaining how to define these four types of associations, and then will follow up to explain how to combine those to define the three standard association types (One-To-One, One-To-Many and Many-To-Many).

Defining the Sequelize associations

The four association types are defined in a very similar way. Let's say we have two models, `A` and `B`. Telling Sequelize that you want an association between the two needs just a function call:

```
const A = sequelize.define('A', /* ... */);
const B = sequelize.define('B', /* ... */);

A.hasOne(B); // A HasOne B
A.belongsTo(B); // A BelongsTo B
A.hasMany(B); // A HasMany B
A.belongsToMany(B, { through: 'C' }); // A BelongsToMany B through the junction
```

They all accept an options object as a second parameter (optional for the first three, mandatory for `belongsToMany` containing at least the `through` property):

```
A.hasOne(B, { /* options */ });
A.belongsTo(B, { /* options */ });
A.hasMany(B, { /* options */ });
A.belongsToMany(B, { through: 'C', /* options */ });
```

The order in which the association is defined is relevant. In other words, the order matters, for the four cases. In all examples above, `A` is called the **source** model and `B` is