Correcting Sequelize Model Assumptions

Sequelize allows for multitudes of different syntaxes to accomplish much of what the tool is capable of. This flexibility can be very useful, but does have drawbacks.

A significant drawback of this flexibility is that Sequelize makes assumptions based on provided information to determine values that are not provided.

For your Mod 4 project, this issue is likely to rear it's head in a couple of ways.

Associations

Association foreign keys are one of the areas in which Sequelize can make assumptions. If a foreign key is not provided in the association, Sequelize will determine what it believes the foreign key should be based on the table names.

For instance, if we have an association between a User and Post model, we would build a one to many relationship. A User has many Posts, and a Post belongs to a User.

```
// ./db/models/user.js
User.hasMany(models.Post);

// ./db/models/post.js
Post.belongsTo(models.User);
```

Given these associations, Sequelize will assume that, because a Post belongs to a User, the Posts table must have a foreign key pointing to the Users table. Further, it will assume that the foreign key uses the table name the value points to, with "Id" tacked on at the end. In this case, Sequelize will assume a foreign key `UserId`.

This can cause one (or both) of two issues.

First, we use camel casing to name our columns, which means that our foreign key would actually be `userId`.

An added layer to this issue is that SQLite does not care about casing, so locally, Sequelize won't have an issue with this assumption. When we deploy using a more robust RDBMS like PostgreSQL, this assumption is no longer safe, as Postgres *does* enforce casing.

Second, there will be times when the foreign key has a different name other than the name of the referenced table. It is somewhat common to name foreign keys something a little more descriptive of the relationship that foreign key creates.

For instance with our AirBnB clones, we'll have a one to many association between a Users table and Spots table, with an `ownerId` foreign key on Spots. This helps indicate that the referenced User is specifically the owner of the Spot.

In this case, if we left out our foreign key in the association, Sequelize would assume `UserId`, when instead we have `ownerId`.

Both of these issues can be solved by simply making sure to declare your foreign keys in all associations. This includes both the `foreignKey` and `otherKey` properties in many to many associations.

Models with Multiple Foreign Keys

When a table/model contains more than one foreign key, Sequelize assigns that model a primary key value that it derives from a combination of the foreign key column names.

This isn't always an issue, but if you happen to need the id value of a record from that table, we will run into problems.

Like above, this can be solved by simply specifying for Sequelize what the primary key should be.

Inside of the `init()` method call in your model, add the following property and nested object to the object containing the model's attributes:

```
id: {
  type: DataTypes.INTEGER,
  allowNull: false,
  primaryKey: true,
  autoIncrement: true
},
```

This will instruct Sequelize to use `id` as the primary key.