Final Checkpoint Study Guide: SEQUELIZE

https://sequelize.readthedocs.io/en/v3/

SEQUELIZE : INTIAL SETUP

Creating a db instance with new Sequelize and a connection string

http://sequelize.readthedocs.io/en/v3/docs/getting-started/

```
const name = process.env.DATABASE_NAME || pkg.name;
const connectionString = process.env.DATABASE_connectionString || `postgres://localhost:5432/${pkg.name}`;


// create the database instance that can be used in other database files
const db = module.exports = new Sequelize(connectionString, {
  logging: debug, // export DEBUG=sql in the environment to get SQL queries
  native: true    // lets Sequelize know we can use pg-native for ~30% more speed (if you have issues with pg-native feel free to take this out and work it back in later when we have time to help)
});

```

SEQUELIZE : MODEL

Definition
Attribute validation and default values
Class methods, Instance methods and hooks
Eager loading/defaultScope: include keyword

```
const db = require('./db'); //require in the db file where the connection string is or not if in the same file.
const Sequelize = require('sequelize');

var Model = db.define('model',
  {
    title: {
      type: Sequelize.STRING,  //type comes from seqelize so look at docs
      allowNull: false, // attribute given so cann
      validate:{ //validation something to check before submmited to db
        notEmpty:true
      }
    },
    content:{
      type: Sequelize.TEXT
    },
    version:{
      type: Sequelize.INTEGER,
      defaultValue:0
    }
  },
  {
    getterMethod:{
      someFunction(num){
        return this.content.slice(0,num);
      }
    },

    instanceMethods:{
      anotherFunction(thing){
        return this.content = this.content.slice(1);
      }
    },
    classMethods: {
    findByTag: function(tag) {
      return SampleModel.findAll({
        where: {
          tags: { $overlap: [tag] }
        }
      })
    }
  },
  }



  )
```

```
Song.belongsTo(Album);
// Places albumId column on song rows
// Allows song.getAlbum/setAlbum/removeAlbum to exist and function

Album.hasMany(Song);
// Also places albumId column on song rows, which is redundant to Song.belongsTo(Album)
// However, it also allows for the use of album.getSongs/album.setSong(s)/addSong(s)/etc

Artist.hasMany(Album, { as: 'creator' });
// Places creatorId on album rows
// Allows for artist.getAlbums/setAlbum(s)/addAlbum(s)/etc

```

```
User.belongsToMany(Song, { through: 'favorites' });
// Creates a new table in database named "favorites"
// with columns userId and songId
// This allows songs to be associated with many users
// and users to associate with many songs
```