Assignment for Module #2: Active Record Relationships

The overall goal of this assignment is to assess your ability to implement and use Active Record model relationships. This includes:

- Creating Active Record models and relationships between the models using a rails-provided generator (`rails g model` or `rails g scaffold`)
- Providing validations for models (using built-in Active Record validations as well as custom validations)
- Implementing a grandparent relationship with a `:through` option
- Providing bootstrap data using a `seeds.rb` file
- Implementing `default_scope` queries
- Implementing aggregation queries
- Implementing advanced queries (e.g., SQL snippets)
- Implementing model/database (DB) cascades

The functional goal of this assignment is to implement relationships and query behavior for the following four (4) model classes

1. User
2. Profile
3. TodoList
4. TodoItem

Functional Requirements

1. Create several Active Record model classes - some of which will have relationships defined when they are created
    - User
    - Profile
    - TodoList
    - TodoItem
2. Create database migrations for relationships not present when the model classes were created
3. Create a DB schema for each model class and their relationships
4. Populate the DB with bootstrap data
5. Implement relationship features
6. Implement model class features

Getting Started

1. Create a new Rails application called `todolists`. This will look quite a bit like what you did in the module 1 assignment.

2. Add the following specification to your Gemfile.

```
group :test do
 gem 'rspec-rails', '~> 3.0'
end
```

3. Run the `bundle` command to resolve new gems

4. From the `todolists` application root directory, initialize the rspec tests using `rails generate rspec:install` command.

```
[todolists]$ rails generate rspec:install
create  .rspec
create  spec
create  spec/spec_helper.rb
create  spec/rails_helper.rb
```

   Add the following line to `.rspec` to add verbose output to test results.

```
--format documentation
```

5. Download and extract the starter set of boostrap files.

```
|-- Gemfile
`-- spec
    `-- assignment_spec.rb
```

   - overwrite your existing Gemfile with the Gemfile from the bootstrap fileset. They should be nearly identical, but this is done to make sure the gems and versions you use in your solution can be processed by the automated Grader when you submit. Any submission should be tested with this version of the file.
   - add the `spec/assignment_spec.rb` file provided with the bootstrap fileset to the corresponding `spec` directory that already exists under your application root directory (e.g., `application-root-directory/spec/assignment_s`. This file contains tests that will help determine whether you have completed the assignment.

6. Run the rspec test(s) to receive feedback. `rspec` must be run from the root directory of your application. All tests will (obviously) fail until you complete the specified solution.

```
$ rspec
...
Finished in 0.02831 seconds (files took 1.39 seconds to load)
56 examples, 1 failure, 54 pending
```

   To focus test feedback on a specific step of the requirements, add "-e rq##" to the rspec command line to only evaluate that requirement. Pad all step numbers to two digits.

```
$ rspec -e rq01
Run options: include {:full_description=>/rq01/}

Assignment
  rq01
    Generate Rails application
      must have top level structure of a rails application

Finished in 0.00356 seconds (files took 1.3 seconds to load)
1 example, 0 failures
```

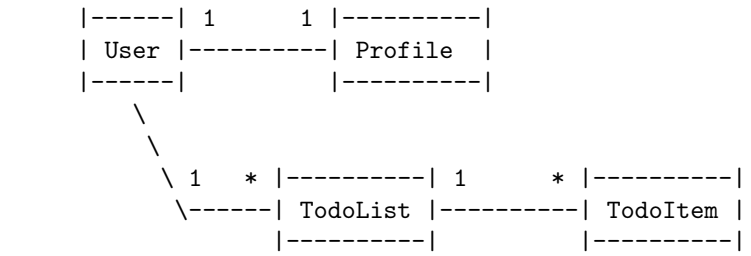
```
$ rspec -e rq02

...
Finished in 0.00422 seconds (files took 2.16 seconds to load)
6 examples, 1 failure, 5 pending

Failed examples:

rspec ./spec/assignment_spec.rb:50 # Assignment rq02 User Model: User class created
```

7. Implement your Models and use the rspec tests to help verify your completed solution.
8. Submit your Rails app solution for grading.

Technical Requirements

1. Create a new Rails app called `todolists`. Use the Gemfile provided in the boostrap files. Do not change the Gemfile from what is provided or your submitted solution may not be able to be processed by the grader (i.e., do not add any additional gems or change gem versions).

   An Entity Relationship (ER) diagram is provided below to help depict each Model's relationship:



2. Re-use the `User` model class and database table creation commands implemented in Assignment 1. (i.e., you may re-use the exact `rails generate` command you used to generate this class from the previous assignment)

   - User
     - username - a string to hold account identity
     - password_digest - a string to hold password information

   Reminder: Ruby/Rails conventions are that class names are CamelCase and file and method names are snake_case.

   You must migrate the database schema at this time and in between each of the following steps to be able to pass the rspec tests incrementally.

```
$ rake db:migrate
$ rspec -e rq02
Run options: include {:full_description=>/rq02/}

Assignment
  rq02
    User Model:
      User class created
      User database structure in place
      User class properties added
        should respond to #username
        should respond to #password_digest
        should respond to #created_at
        should respond to #updated_at

Finished in 0.19157 seconds (files took 1.27 seconds to load)
6 examples, 0 failures
```

3. Re-use the `Profile` model class and database table creation commands implemented in Assignment 1, except this time create it with a reference to `User` when you create the model class. The grader will look for the relationship to be included in the initial create table migration for this model class.

   - Profile
     - gender - a string to hold the words "male" or "female"
     - birth_year - a number to hold the year the individual was born
     - first_name - a string with given name of user
     - last_name - a string with family name of user
     - user - a 1:1 relationship with User (i.e., Profile `belongs_to` User)

   Also define the 1:1 `has_one` relationship in the `User` model class.

   - User
     - profile - a 1:1 relationship with `Profile` (i.e., User `has_one` profile). Add appropriate options to have the `User` model class delete a `Profile` in a cascading fashion

   Reminder: Relationships are only generated or migrated in the class forming the relationship (e.g., the many-side of a many:1). The inverse side of a relationship (e.g., the one-side of a 1:many) need only have the relationship declared in the model class and not in the database migration. In this case, `Profile` is forming the relationship in the database and `User` is the inverse side.

```
$ rake db:migrate
$ rspec -e rq03
```

4. Re-use the `TodoList` model class implemented in Assignment 1 (i.e., you may re-use the exact `rails generate` command you used to generate this class from the previous assignment). The grader will look for no relations in the initial create table migration for this model class.

   - TodoList
     - list_name - a string name assigned to the list
     - list_due_date - a date when todo items in the list are to be complete. This is a date. We are not concerned with the time of day.

```
$ rake db:migrate
$ rspec -e rq04
```

5. Create a database migration (hint: `rails g migration`) that adds a database reference from the `TodoLists` table to the `Users` table. The grader will look for this relationship to be formed by a database table migration.

   Also define the many:1 `belongs_to` relationship in the `TodoList` model class.

   - TodoList
     - user - a many:1 relationship with `User` (i.e., TodoList `belongs_to` User)

   Also define the 1:many `has_many` relationship in the `User` model class.

   - User
     - todo_lists - a 1:many relationship with `TodoList` (i.e., User `has_many` todo_lists). Add appropriate options to have the `User` model class delete a `TodoList` in a cascading fashion

```
$ rake db:migrate
$ rspec -e rq05
```

6. Re-use the `TodoItem` model class implemented in Assignment 1, except this time create it with a many:1 `belongs_to` relationship with `TodoList` when you create the model class. The grader will look for the relationship to be included in the initial create table migration for this model class.

- TodoItem
  - due_date - date when the specific task is to be complete
  - title - a string with short name for specific task
  - description - a text field with narrative text for specific task
  - completed - a boolean value (default=false), indicating whether item is complete
  - todo_list - a many:1 relationship with TodoList - TodoItem belongs_to TodoList

Also define the 1:many `has_many` relationship in the `TodoList` model class.

- TodoList
  - todo_items - a 1:many relationship with `TodoItem` (i.e., TodoList `has_many` todo_items). Add appropriate options to have the `TodoList` model class delete a `TodoItem` in a cascading fashion

```
shell $ rake db:migrate $ rspec -e rq06
```

7. Migrate all database schema changes at this time if you have not yet done so. At this point you should have:

   - four (4) create table (Users, Profiles, TodoLists, and TodoItems)
   - one (1) update table migrations to add reference (TodoLists to Users)

```
$ rake db:migrate
$ rspec -e rq07
```

8. Implement a 1:many `:through` relationship from `User` to `TodoItem` by using the 1:many relationship from `User` to `TodoLists` as a source.

   - User
     - todo_items - a 1:many through relationship with TodoItem through TodoLists (i.e., User `has_many` todo_items)

```
$ rspec -e rq08
```

9. Create a seeds.rb file that will clear the existing data from the model tables and load the database with

   - The four users below with their birth years:
     - Carly Fiorina, 1954
     - Donald Trump, 1946
     - Ben Carson, 1951
     - Hillary Clinton, 1947
   - Usernames (e.g., their last names) and a password for each `User`
   - A `Profile` for each `User`
   - Exactly one `TodoList` per `User` that is due one year from the date the database is loaded
     - (hint: `Date.today` provides today's date and `1.year` can be used to define one year)
   - Each `TodoList` contains five (5) `TodoItems` (there must be 20 total)
   - Each `TodoItem` having a due date of one year from the time the database is loaded
   - Each `TodoItem` must have an arbitrary `title` and `descriptions`

   (Hint: you may want to consider using loops)

   Once the seeds.rb file is created, populate the database using `rake db:seed`

```
$ rake db:seed
$ rspec -e rq09
```

10. Add `default_scope` to both `TodoList` and `TodoItem` models to always return collectons from the database ordered by due dates with earliest dates first.

    `shell $ rspec -e rq10`

11. Add validation to `User` and `Profile` models

- User
  - Define a validation for `username` to enforce that username be supplied by using a built-in validator
- Profile
  - Define custom validator that permits `first_name` or `last_name` to be null but not both
  - Define a validation for `gender` to be either "male" or "female" by using a built-in validator
  - Define custom validator that prevents anyone that is `male` (gender) from having the `first_name` "Sue" ;)

    `$ rspec -e rq11`

12. Add a cascade of deletes that will remove `Profile`, `TodoList`, and `TodoItem` rows for any `User` removed.

    `$ rspec -e rq12`

13. Add a method to the `User` model class called `get_completed_count`, which:

    - accepts a user as a parameter
    - determines the number of `TodoItems` the `User` has completed using an aggregate query function
      - (Hint: You are looking for the count of `TodoItems` associated with a specific `User` where `completed:true`)
    - returns the count

    `shell $ rspec -e rq13`

14. Add a method to the `Profile` class, called `get_all_profiles`, which:

    - accepts a min and max for the `birth year`
    - issues a `BETWEEN` SQL clause in a `where` clause to locate `Profiles` with birth years that are between min year and max year
    - defends itself against SQL injection when applying the parameters to the SQL clauses
    - returns a collection of Profiles in ASC birth year order

    `$ rspec -e rq14`

Self Grading/Feedback

Some unit tests have been provided in the bootstrap files and provide examples of tests the grader will be evaluating for when you submit your solution. They must be run from the project root directory.

```
$ rspec
...
Finished in 9.56 seconds (files took 1.41 seconds to load)
56 examples, 0 failures
```

You can run as many specific tests you wish be adding `-e rq## -e rq##` `shell $ rspec -e rq01 -e rq02`

Submission

Submit an .zip archive (other archive forms not currently supported) with your solution root directory as the top-level (e.g., your Gemfile and sibling files must be in the root of the archive and *not* in a sub-folder. The grader will replace the spec files with fresh copies and will perform a test with different query terms.

```
|-- app
|   |-- assets
|   |-- controllers
|   |-- helpers
|   |-- mailers
|   |-- models
|   `-- views
|-- bin
|-- config
|-- config.ru
|-- db
|-- Gemfile
|-- Gemfile.lock
|-- lib
|-- log
|-- public
|-- Rakefile
|-- README.rdoc
|-- test
`-- vendor
```

Last Updated: 2015-10-20