William Montgomery
University of Illinois at Chicago
CS 341 - Fall 2013
Final Project

Ruby vs. C# Database Access Comparison

For this project, I implemented Homework 8 in Ruby. Homework 8 involved building a program in tiers to access a database in C#. I believe I came up with a unique solution that was slightly less brittle to schema changes than most solutions. I did this by allowing the Data Access layer to be given a type, so that it could dynamically generate objects based on the type. The solution allowed a fairly clean interface to the presentation layer, but was clunky. For the final project, I reimplemented the code in Ruby, using the ActiveRecord Ruby gem. This allowed for rapid development with no SQL anywhere in the code.

Running the Program

In order to run the program, Ruby should be installed. I built and tested it on Ruby version 1.9.3p194. It probably will not run on earlier versions. The code requires several gems. The Ruby Bundler is used to manage these gems. To install the gems run the following commands in the source directory:

gem install bundler
bundle install

To run, type:
ruby Runner.rb

Sample Output

```
1. Ability to retrieve total # of movies
1066
############################################################
2. Ability to lookup a movie name from a movie id
Despicable Me
     id => 74
############################################################
3. Ability to lookup a movie id from a movie name
The Harder They Come
```

```
     id => 1003
The Harder They Come
     id => 1004
The Harder They Come
     id => 1005
The Harder They Come
     id => 1006
The Harder They Come
     id => 1007
The Harder They Come
     id => 1008
The Harder They Come
     id => 1009
The Harder They Come
     id => 1010
The Harder They Come
     id => 1011
The Harder They Come
     id => 1012
################################################################
4. Ability to retrieve information about all the movies: ids,
names, number of reviews, average rating, lowest rating, and
highest rating; also the ability to foreach through this data
When Harry Met Sally
     Number of reviews => 180
     Hightst rating => 5
     Lowest rating => 3
The Matrix
     Number of reviews => 100
     Hightst rating => 5
     Lowest rating => 4
Finding Nemo
     Number of reviews => 100
     Hightst rating => 5
     Lowest rating => 4
Despicable Me
     Number of reviews => 100
     Hightst rating => 2
```

```
     Lowest rating => 3
Monsters Inc.
     Number of reviews => 100
     Hightst rating => 4
     Lowest rating => 5
The Matrix
     Number of reviews => 100
     Hightst rating => 1
     Lowest rating => 1
Shark Attack 3
     Number of reviews => 93
     Hightst rating => 2
     Lowest rating => 2
Casablanca
     Number of reviews => 100
     Hightst rating => 3
     Lowest rating => 2
It's a Wonderful Life
     Number of reviews => 95
     Hightst rating => 1
     Lowest rating => 2
Animal House
     Number of reviews => 100
     Hightst rating => 5
     Lowest rating => 2
############################################################
5. Ability to retrieve top-10 movies with the highest average
ratings
Finding Nemo
     Average rating => 4.58
The Matrix
     Average rating => 4.42
Despicable Me
     Average rating => 4.38
Finding Nemo
     Average rating => 4.35
The Matrix
     Average rating => 4.31
```

```
It's a Wonderful Life
     Average rating => 4.3
Finding Nemo
     Average rating => 4.28
Monsters Inc.
     Average rating => 4.26
When Harry Met Sally
     Average rating => 4.26
The Matrix
     Average rating => 4.24
############################################################
6. Ability to retrieve total # of reviews
99206
############################################################
7. Ability to retrieve average rating across all the reviews
3.244531580751164
############################################################
8. Ability to retrieve information about a user based on user
id: number of distinct movies reviewed, number of reviews
submitted, and average rating given by user across all their
reviews
User 2649335
     number of distinct movies => 2
     number of reviews => 11
     average rating => 4
############################################################
9. Ability to retrieve top-10 users who have submitted the
most reviews
User 2439493
     Number of Reviews => 431
User 1664010
     Number of Reviews => 279
User 2625420
     Number of Reviews => 275
User 1314869
     Number of Reviews => 202
User 1001129
     Number of Reviews => 191
```

```
User 305344
     Number of Reviews => 182
User 303948
     Number of Reviews => 160
User 1907667
     Number of Reviews => 155
User 1977959
     Number of Reviews => 150
User 1945809
     Number of Reviews => 141
############################################################
10. Ability to insert a new review into the database
When Harry Met Sally
     5
     User 2649335
```

Code Snippets

This is an example of ActiveRecord's where method. It accepts a hash as input with the parameters to evaluate. This code also shows the use of a block, which is everything between do and end. The block is similar to a lambda function in Ruby, except that it does not throw an error if you give it the wrong number of arguments.

```
Movie.where(name: "The Harder They Come").limit(10).each do
|movie|
     puts movie.name
     puts "    id => #{movie.id}"
end
```

Below is an example of some code used to find information about the reviews submitted by a single user. It is an example of using map with lambda functions. In the third line, it maps across an array to find unique movies. In the last line, it maps across the array again to sum up the ratings in order to compute the average rating.

```
reviews = Review.where(user_id: User.last.id)
puts User.last.name
puts "    number of distinct movies => #{reviews.map{|review|
review.movie_id}.uniq.count}"
puts "    number of reviews => #{reviews.count}"
```

```
puts "    average rating => #{reviews.map{|review|
review.rating}.sum / reviews.count}"
```

I like this code because it chains together a lot of methods, yet is still human readable. The first line groups reviews by the "user_id" field, then counts the number of ratings, sorts them in reverse order, takes the first 10 and iterates through them.

```
Review.group(:user_id).count(:rating).sort_by {|key, value|
value}.reverse.take(10).each do |key, value|
     user = User.find(key)
     puts user.name
     puts "    Number of Reviews => #{value}"
end
```

This is an example of ActiveRecord eager loading data. Instead of doing one query to get all the movies and then iterating through the movies to get the reviews for each, we load the reviews so that we only have to make 2 database queries (instead of N + 1).

```
Movie.all.includes(:reviews).limit(10).each do |movie|
     reviews = movie.reviews.sort{|review| review.rating}
     puts "#{movie.name} "
     puts "    Number of reviews => #{movie.reviews.count}"
     puts "    Highest rating => #{reviews.first.rating}"
     puts "    Lowest rating => #{reviews.last.rating}"
end
```