Project 8: ORMs

Objectives
Understand what an Object-Relational Mapper is, and how it benefits OO programs. Using Ruby, learn how to write a simple program that uses the ActiveRecord ORM to abstract common database operations.

Overview
Using Ruby, ActiveRecord, and the mysql2 gems, create a simple program that successfully connects to a MySQL server hosted on AWS. Your application will present a simple menu to the user, and execute specific database operations. *However, your application will not contain any SQL*.

Here is a simple example interaction:

```
Snacks!!!! I love them.
Main Menu
A. List Buildings
B. List Machines
C. List Snacks
E. List Users
F. Find a Snack
F. Add a New Snack
Q. Quit
```

When the user chooses to *List* any entity shown above, it should display a list of all Buildings/Machines/Snacks or Users chosen and appropriate details such as id, name, description and so on (varies per entity -- see the schema).

When the user chooses “List Buildings,” you should display each building and the *number of machines* associated with the building. For example:

```
Buildings:
Brown Hall (2 machines)
Marquez Hall (1 machine)
```

When the user chooses “List Machines,” you should display each machine’s details and the building in which it is found.

```
Machines:
abc123, across from 340 (Brown Hall)
zyz123, across from 339 (Brown Hall)
```

When the user chooses “List Snacks,” you should display each snack’s details and the machine it can be found in, including the building associated with that machine.

```
Snacks:
Grossitos
- abc123 in Brown Hall
- xyz456 in Brown Hall
Gummi Worms
- qcy223 in Marquez Hall
```

When the user chooses “Find a Snack,” your program should ask for the snack’s name, and display a list of all the machines and buildings where that snack may be found.

When the user chooses “Add a New Snack,” your program should ask for the snack’s details, such as name, manufacturer, and number of calories. It should then present a list of all the machines per each building, and prompt the user to specify which machine the new snack should be associated with. The program should then save the new snack information (along with the relationship to the specified machine).

After any of A - F are chosen, the Main Menu should be redisplayed.

If the user chooses to Quit, the program should then terminate.

Prerequisites
You should have an account on the remote RDS/MySQL server with permissions on the *snacks* database.

Setup
For this project, you will need to set up a simple Ruby environment. This environment will be used for subsequent projects that talk to NoSQL databases, so the time you put in now will be recouped later (that said, this setup will not take long).

1. Install Ruby
2. Install the mysql client library
3. Install the mysql2 gem
4. Install the activerecord and adapter gems

1. Installing Ruby

Using the VM?
You're done.

OSX
you likely already have a version of Ruby installed. Verify this with:
`ruby -v`
However, you will need to install gcc and/or the Apple Developer Tools (XCode & Command Line Tools) in order to build other libraries. The "Command Line Tools" are the important part. This is straightforward, but ask on Piazza if you get stuck.

Linux
Use a package manager to install Ruby 1.9.3 or install from source.

Windows
There are two things to install: Ruby itself, and the "Developer Tools." Use the Ruby 1.9.3 Installer and install Ruby first. Be sure to select "Add Ruby executable to your PATH" during the installation.
Next, download the Ruby Installer DevKit using this link. Be absolutely certain you installed Ruby 1.9.3 (linked above) and download this particular DevKit. Next, double-click the installer and edit the extraction target to C:\Ruby193\devkit, then click "Extract." Next, follow the Quick Start instructions here:

```
cd C:\Ruby193\devkit
ruby dk.rb init
ruby dk.rb install
```

Do not proceed if you face errors during this step!

2. Installing the mysql client library

Using the VM?
You're done.

OSX
Use brew to install the mysql client libraries, or download and install MySQL server from the official MySQL site.

Linux
Use a package manager to install libmysqlclient-dev. For example:

```
sudo apt-get install libmysqlclient-dev
```

Windows
Download this zip file, and place the files inside C:\Ruby193\lib **and** in C:\windows\system32. Be sure you've placed *the files themselves* and *not* a folder inside C:\Ruby193\lib.

3. Install the mysql2 gem

```
gem install mysql2
```

Do not proceed if you do not see "Successfully installed mysql2..."

Verify that the gem has been installed correctly:

```
irb
require 'mysql2'
```

You should see `true`. Type `exit` to quit the interactive Ruby interpreter. If you do not see `true`, post the exact command and error output on Piazza.

4. Install the activerecord gem and the mysql2 adapter

```
gem install activerecord
gem install activerecord-mysql2-adapter
```

This will take a couple minutes. If you see an error, be sure to post the exact error message on Piazza.

Test
Make sure everything you need is working by running this simple Ruby script. Create a directory called project08 and place the ar_test.rb file inside. You will need to edit the script and change the configuration information (username and password). Run the script:

```
cd path/to/project08
ruby ar_test.rb
```

In addition to some logging output, you should see:

```
Hello, I'm fee and I think you should take the red pill.
```

Do not proceed until this test script runs successfully. If you see an error, be sure to post the exact error message on Piazza.

Look at the Schema
Be sure to take a look at the relational schema on the server. To do so, you must connect to the server using either the mysql command-line client or a GUI tool of your choice.

```
mysql -h csci403.c99q7trvwetr.us-west-2.rds.amazonaws.com -P3306 -u USER -p
```

Be sure to use your username/password as credentials (see "prerequisites" above).

Next, take a look at the schema:

```
use snacks;
show tables;
describe TABLENAME;
```

You will notice a particular convention, chosen in order to match the ORM tool's expectations.

- Tables are plural and lower-case.
- Primary keys are called id.
- Foreign keys use a lower, snake_cased name, matching the foreign table, with id appended (eg `building_id`).
- Join tables use the names of the referenced tables, snake_cased, in alphabetical order (eg, `machines_snacks`).

This is just one convention, followed for simplicity. Realize that ORMs can be configured to fit pre-existing schemas.

Spend time reviewing the schema, making a note of the column names, and notice the relationships inferred by the foreign keys.

Create a Simple Program
Download and save this simple boilerplate program (snacks.rb) inside your project08 directory. Edit the file, adding your MySQL username and password. Now run it:

```
cd path/to/project08
ruby snacks.rb
```

And select option D, to list users. You should see at least one user displayed in a list.

Now implement the remaining features of the program. In order to accomplish this, leverage the features of the ActiveRecord ORM to make your code very lightweight. You will need to declare classes for all of the entities in the database; declare relationships in those class definitions; and declare a few constraints in the Snack class. You'll also need to implement a handful of functions that use the classes that you create.

Spend some time reading the provided `snacks.rb` in an editor. Some functionality has been provided for you, but you must implement the functions whose job it is to interact with the database and display some data (and insert some new data).

Exploring ActiveRecord, Relationships, Finders
In order to implement the features in a sane manner, you'll want to explore the ActiveRecord API, interact with your fellow students in person and on Piazza, and discover how to implement relationships between objects, how to "find" objects given specific criteria (like the snack name) and how to implement a constraint.

Start by reviewing this overview and this API document.

Constraints
There is one last requirement: since MySQL does not allow us to enforce a CHECK constraint, you must make sure that the Snack's calories attribute is not negative. Hint: Do not validate the user input -- see how the ActiveRecord ORM allows you to declare "validations" which are really attribute constraints.

You do not need to spend time handling the cases where the Snack object is not saved due to a validation failing. As long as "no Snack with a negative calories value is saved to the database," then you're all set.

Exploration
A critical part of the value behind this assignment is exploring an unfamiliar API. In order to succeed, you've got to start far before the due date by implementing the Building class; and if you get stuck, interact with your fellow students and instructor!

Once you accomplish the first class implementation, the rest of the project will not take you long to complete.

Presentation
Good writing is important, **especially when it comes to code**. *Writing with poor readability, lack of appropriate whitespace, misspellings, etc will be handed back for re-writing (no grade penalty).*

Submit a zip of your code via blackboard.

Grading Criteria (90 points)
Complete implementation of a simple program that communicates with the remote MySQL server using ActiveRecord, delivering the features described above. (90 total points).
- writing quality (30)
- implementation (60)

Please submit your work via Blackboard by 7AM on the due date.

Bonus Challenges
Add additional constraints to your classes, such as ensuring that the name attribute cannot be blank.

Allow the user to create a new User account or specify a username when starting the program, and to manage a “favorite snacks” list.