Active Record Playground

Let's familiarize ourselves with the ActiveRecord gem.

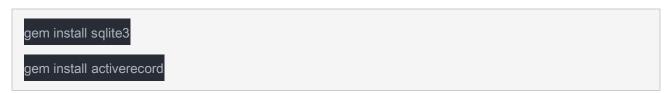
Usually, ActiveRecord is built into a larger framework, e.g. Rails. The configuration files are split into different parts to better organise the code. Here, we have set up everything needed to run ActiveRecord on an existing *sqlite3* database in one file so you can see what is going on. As you can see, it's just a one-liner saying adapter: sqlite and database: "./students.sqlite3.

Note:

- 1. If you are planning to work on our local machine, please git fork the challenge here and git clone it to your local machine.
- If you are using Ronin, please note that it will take approx 12-15 seconds to run the test code.

Installing Gems

We will need the sqlite3 and activerecord gem for this challenge. If these two gems are not installed on your local machine yet, please do the following:



Database

The database given is an sqlite3 database with 20 records of student information(name, email, age). You can access the sqlite3 database directly via the terminal, e.g. sqlite3 students.sqlite3 as per the previous challenges.

Setting Up

As mentioned previously, ActiveRecord is usually built into a larger framework, e.g. Rails where the configuration files are split into different parts to better organise the code. However, in this exercise, we have set up everything needed to run ActiveRecord on an existing *sqlite3* database in one file so you can see what is going on.

In playground.rb, the line below configures ActiveRecord to connect to a sqlite3 database in the same folder named "students.sqlite3":

ActiveRecord::Base.establish_connection(:adapter => "sqlite3", :database => "./students.sqlite3")

We then set up ActiveRecord to use Ruby objects with SQL data by doing:

class Student < ActiveRecord::Base

Now that we have set up the necessary, we can then write some test code in the code editor and see what happens.

Create, Read, Update, Delete (CRUD)

Before we start coding, read the <u>CRUD section in the ActiveRecord Documentation</u>to have an idea of of what we will be doing. Once you are done reading, we will write some test code to get ourselves used to Active Record methods.

Tip: You can also use irb to play with ActiveRecord by using irb -r ./playground.rb

Create (C)

Let's add some new students into our database. There are two ways to do this, we can either use new along with save, or create.

student = Student.new(name: "Harry Potter", email: "harry@example.com", age: 18)
student.save

Now do p student. A Student object will be returned to you. It will look something like that (note that your created_at and updated_at will be different.)

#<Student id: 21, name: "Harry Potter", email: "harry@example.com", age: 18, created_at: "2017-02-26 1 3:05:14", updated_at: "2017-02-26 13:05:14">

Now add another user by using create, i.e.

student = Student.create(name: "Luke Watt", email: "luke@example.com", age: 23)

Do p student again. If you want to, go into your database and check that these two students have indeed been added! Once you have confirmed that the students are added, go do some research on what are the differences between using new, save, and create!

Read (R)

Let's do some data querying.

Querying ALL records

To select all the records in the database, we can use Student.all. If you want to view the query, type p Student.all. You will notice a bunch of code that looks like this:

```
#<ActiveRecord::Relation [#<Student id: 1, name: "Grayson Bednar", email: "emmy@schneider.net", ag
e: 20, created_at: "2017-02-26 12:16:22", updated_at: "2017-02-26 12:16:22">, ... ]>
```

Notice that the object returned to you is an #<ActiveRecord::Relation object and the Student objects are stored in an array-like structure (notice the square brackets []?). We can **count** how many Students objects are there. We can also use the [] to access each object. Give the following a try in your code:

p students.count # did you get 22?

p students[5] # did you get the student information with id 4? Also, what object was returned to you?

pay attention to the object that is returned to you, is it a User object or Active Record relation object?

pay attention to the object that is returned to you, is it a User object or Active Record relation object?

p Student.last

We can also use each to iterate through the ActiveRecord Relation object access each Student object, i.e.



Main takeaway: understand what each method call returns to you - is it an **Active Record Relation** object or is it a **Student** object?

Querying records based on given attribute

If we want to find record(s) based on given attributes, we can use where, or find_by. However, there is a difference between these two methods. The method where return an Active Record relation object, while find_by returns the first object that matches the required attribute. For example, if there are 3 users with the name "John" and with IDs 2, 5, 10 respectively in the database, using find_by will return the user "John" with ID 2.

Let's now test these code. First, let's find the student by the name "Dr. Lois Pfeff" using where.

```
student = Student.where(name: "Dr. Lois Pfeff")
p student
```

Pay attention to the object that is returned to you, is it a Student object or an Active Record relation object?

You can access the student's attributes by doing the following:



p student.age

Now let's use find_by

```
student = Student.find_by(name: "Dr. Lois Pfeff")
p student
```

Pay attention to the object that is returned to you, is it a Student object or an Active Record relation object?

Knowing the difference between where and find_by will help you structure your queries better.

If we want to find a student based on his/her ID, we can use find too. Give the following a try:

```
p Student.find(7)
p Student.find_by(id: 7)
```

What are the objects returned?

Update (U)

Say a student has changed his/her email address. We can update our database by first assigning the value to the attribute and saving it, or use update. Let's try out some code.

Student with ID 5 has changed her email address. Let's first find the student and change her email address:

```
student = Student.find(5)

p student

student.email = "elsie@example.com"

student.save

p student # check that the email has indeed been updated.
```

Let's now learn how to use update to the job. We will now update the student's age to 21.

```
student.update(age: 21)
p student
```

Delete (D)

In some occasions, we might want to remove records from our database. We can use delete or destroy. Let's now remove students with ID 21 and 22 from our database.



Okie dokie! We have now gone though several CRUD methods. Write down somewhere so that you can remember them! :)