Laboratorio 3 – Modelo de datos y Active Record

## Objetivos

* Entender la abstracción creada por el Active Record de Rails para acceder y manipular información de una base de datos referencial mediante programación orientada a objetos.
* Aprender las funciones básicas proveídas por Active Record.
* Aprender a utilizar la herramienta irb de Ruby y la consola de Rails.

## Requerimientos

En este laboratorio deberá implementar un modelo de datos con el Active Record de Rails para manejar la calendarización de citas medicas en una clínica. Deberá utilizar la consola incluida en Rails para realizar las pruebas de sus clases. Internamente la consola de Rails utiliza la herramienta irb de Ruby para interactuar con los programas creados en Ruby en tiempo de ejecución.

Descripción Debe implementar una clase de Active Record que le permita manejar citas. Esta clase debe soportar las operaciones básicas de crear, editar, mostrar y eliminar. Este conjunto de operaciones básicas se conocen popularmente como CRUD (Create Read Update Delete)

Debido a que estamos utilizando Active Record para abstraer su base de datos, solo deberá preocuparse por crear el esquema de tablas y lograr manipular la información almacenada.

Recuerde que en Active Record una tabla es mapeada a una clase y cada tupla es mapeada a una instancia de clase. Esta abstracción le permite manipular los datos de las tablas mediante métodos de una instancia o clase y ya no directamente con SQL. Esto le permitirá aprovechar las ventajas de la programación orientada a objetos para manipular la información guardada en bases de datos.

El esquema de su tabla de citas debe tener al menos lo siguiente:

* Nombre de la tabla: appointments.
* Identificador único (id)
* Día y Hora de inicio. (begin\_time)
* Día y Hora de finalización. (end\_time)
* Identificador del médico encargado. (medic\_id)
* Identificador del paciente. (patient\_id)
* 3 métodos estáticos para crear, eliminar y modificar una cita.
* Nombre de la tabla: medics.
* Identificador único (id)
* Nombre (name)
* Departamento en el que trabaja (department)
* Un método estático para mostrar sus citas correspondientes: showAppointments().
* Nombre de la tabla: patients
* Identificador único (id)
* Nombre (name)
* Direccion (address)
* Un método estático para mostrar sus citas correpondientes: showAppointments().

Recuerde respetar los nombres de las columnas, los identificadores son integer, día y hora es datetime, nombre, dirección y departamento son text y en todas las tablas el id es auto incrementable además de ser la llave primaria.

Una persona no puede asistir a 2 citas distintas al mismo tiempo y un médico no puede atender 2 citas distintas a la misma hora, esto quiere decir que las citas no se deben traslapar.

Puesto que deberá implementar métodos adicionales a los proveídos por Active Record para manejar las citas y mantener la consistencia de su información. Estos deberán ser implementados con las firmas descritas a continuación:

* boolean assignAppointment(medic\_id, patient\_id, begin\_time, end\_time )
* boolean modifyAppointment( appointment\_id, medic\_id, begin\_time, end\_time)
* boolean deletAppointment ( appointment\_id )

En todos los casos el valor booleano de vuelta indica si la transacción fue completada exitosamente o no.

## Procedimiento

Será necesaria una nueva aplicación que puede crear escribiendo en la terminal

rails new lab3

y hacer un nuevo repositorio de git en github (<https://github.com/repositories/new>) luego repetir los pasos que ya conocen del laboratorio anterior para git. En esta práctica no vamos a usar Heroku.

Configuración

Vamos trabajar esta práctica con MySQL, deben tener instalado todo lo necesario y además la gema para que funcione con Rails. A través de cualquier herramienta (CLI, Query Browser, etc) deben crear una base de datos llamada lab3 y crear las tablas que se describen arriba.

Luego, será necesario configurar Rails para que pueda interactuar con MySQL, la configuración se encuentra a partir de su working directory (lab3) en el archivo

config/database.yml

Vamos a estar trabajando con el ambiente de desarrollo, la sección del archivo que le interesa por defecto dice:

development:

adapter: sqlite3

database: db/development.sqlite3

pool: 5

timeout: 5000

y usted lo reemplazará por

development:

adapter: mysql

database: lab3

username: root

password:

En esta caso, mi instalación de MySQL tiene de nombre de usuario “root” y el password es vacío, coloque allí los valores de su instalación tomando en cuenta que el código lo publicará en un repositorio público y el archivo será visible para todos en internet.

Luego, hay que decirle a Rails que se utilizará la gema de MySQL edite el archivo

Gemfile

y allí hay que agregar

gem 'mysql'

debajo de la referencia a la gema de Rails. Cada vez que modifique este archivo, para asegurarse que las gemas están instaladas ejecute

bundle install

Al ejecutar el comando, se modificará su archivo Gemfile.lock y si estuviera utilizando Heroku es necesario hacer un Git push(seguido de add y commit), recuerde que **para este laboratorio no usamos Heroku**.

Para más información de la forma de manejar las gemas con el bundler y el Gemfile, visite:

<http://yehudakatz.com/2010/04/21/named-gem-environments-and-bundler/>

<http://optimisdev.com/posts/gemfile-lock-explained>

Modelos

Aun que aun no sabemos como funcionan, empezaremos a trabajar con modelos dentro de la aplicación, aun no usamos migraciones(eso lo vemos en la siguiente práctica) si no que creamos las tablas manualmente entonces al generar el modelo hay que decirle que no queremos una migración. Para generar el modelo ejecute en la línea de comandos

rails g model medic --migration=false

Nota: en Rails 3 hay acortadores, por eso puede escribir “rails g” en vez de “rails generate”, pero los 2 comandos dan el mismo resultado.

Si hubiera cometido algún error, puede destruir el modelo con

rails destroy model medic

Note que el nombre del modelo es en singular a pesar de que la tabla está en plural. Siga este convenio. Genere los modelos para los pacientes, los médicos y las citas.

Cada modelo tiene código de Ruby asociado (recuerde ORM es una clase por tabla y un objeto por fila) que encuentra en la carpeta models dentro de app dentro de la raíz de su working directory, por ejemplo, para médicos será

app/models/medic.rb

En ese archivo debe escribir los métodos estáticos descritos en los requerimientos del laboratorio.

Pruebas

Estaremos trabajando con ruby interactivo(irb), específicamente con la versión que tiene todo el ambiente de trabajo del proyecto de rails, para abrirlo debe ejecutar

rails c

Aquí puede agregar sus datos de formas distintas, por ejemplo: (en negrillas el código que ustedes tienen que escribir)

irb(main):001:0> **el\_doctor = Medic.new**

=> #<Medic id: nil, name: nil, department: nil>

irb(main):002:0> **el\_doctor.name = "House"**

=> "House"

irb(main):003:0> **el\_doctor.department = "emergencias"**

=> "emergencias"

irb(main):004:0> **el\_doctor.save**

=> true

otra posible forma de hacerlo es:

irb(main):005:0> **un\_paciente = Patient.new :name => "cookie monster", :address => "sesame street"**

=> #<Patient id: nil, name: "cookie monster", address: "sesame street">

irb(main):006:0> **un\_paciente.save**

=> true

Para agregar una cita es necesario conocer un id de médico y otro de paciente, en este caso utilizaremos los de los ejemplos anteriores.

irb(main):020:0> **una\_cita = Appointment.new do |cita|**

irb(main):021:1\* **cita.begin\_time = Time.now**

irb(main):022:1> **cita.end\_time = DateTime.new(2011,2,3,22,0)**

irb(main):023:1> **cita.medic\_id = el\_doctor.id**

irb(main):024:1> **cita.patient\_id = un\_paciente.id**

irb(main):025:1> **end**

=> #<Appointment id: nil, begin\_time: "2011-02-03 17:45:20", end\_time: "2011-02-03 22:00:00", medic\_id: 2, patient\_id: 1>

irb(main):026:0> **una\_cita.save**

=> true

Note que cada vez que se llama al método save del objeto devuelve true si tuvo éxito y false de lo contrario.

Una vez que ha realizado algunas inserciones de prueba, puede intentar obtener resultados de la base de datos, por ejemplo:

irb(main):027:0> **Medic.all**

=> [#<Medic id: 1, name: "Dr. Eustaquio", department: nil>, #<Medic id: 2, name: "House", department: "emergencias">]

irb(main):028:0> **Medic.first**

=> #<Medic id: 1, name: "Dr. Eustaquio", department: nil>

irb(main):029:0> **Patient.last**

=> #<Patient id: 1, name: "cookie monster", address: "sesame street">

irb(main):030:0> **Appointment.find(1)**

=> #<Appointment id: 1, begin\_time: "2011-02-03 17:45:20", end\_time: "2011-02-03 22:00:00", medic\_id: 2, patient\_id: 1>

Nota: la creación del “Dr. Eustaquio” no se incluye en el código de ejemplo, es algo que agregué, por lo que sus resultados pueden variar.

Métodos estáticos

Para agregar un método estático debe trabajar en la clase del modelo, por ejemplo para los médicos, la clase debe lucir algo así:

class Medic < ActiveRecord::Base

def Medic.showAppointments(medic\_id)

primera\_cita = Appointment.where(:medic\_id => medic\_id).first

puts "Hora: #{primera\_cita.begin\_time}"

end

end

Cada vez que edite un modelo, debe salir de la consola(irb) para que se actualicen los cambios.

## Entrega

Enviar un archivo.zip que contenga el folder de la aplicación, un archivo de texto con el url de su repositorio en Github y el script de SQL utilizado para crear las tablas.