**http://www.guiasrails.es/active\_record\_basics.html**

**1 ¿Que es Active Record?**

Active Record es la M en [MVC](http://www.guiasrails.es/getting_started.html#the-mvc-architecture) - el modelo - el cual es la capa del sistema responsable de representar los datos y la lógica de nogocio para manipularlos. Active Record facilita la creación y manipulación de objetos de negocio quienes requieren ser almacenados persistentemente en una base de datos. Esta es una implementación del patrón de Active Record el cual en si mismo es una descripción de un sitema de Mapeo de Objetos Relacionales.

**1.1 El Patrón Active Record**

[Active Record fue descripto por Martin Fowler](http://www.martinfowler.com/Catalog/activeRecord.html) en su libro *Patterns of Enterprise Application Architecture*. En Active Record, los objetos soportan tanto la persistencia y el comportamiento que opera con los datos. Active Record toma la opinion que al asegurar el acceso lógico a los datos como parte del objeto educará a los usuarios de ese objeto sobre como escribir y leer en la base de datos.

**1.2 Mapeo de Objetos Relacionales**

El Mapeo de Objetos Relacionales (Object-Relational Mapping), comunmente nombrado por sus siglas ORM, es una técnica que conecta la riquesa de los objetos de una aplicación con las tablas de un sistema de base de datos relacional. Utilizando ORM, las propiedades y las relaciones de los objetos en una aplicación pueden ser facilmente guardadas y recuperadas desde la base de datos sin escribir sentencias SQL directamente y con el mínimo código en general de acceso a la base de datos.

**1.3 Active Record como un Framework ORM**

Active Record brinda varios mecanismos, los más importantes nos la capacidad para:

* Representar modelos y sus datos.
* Representar asociaciones entre esos modelos.
* Representar jerarquías de herencia a través de modelos relacionados.
* Validar modelos antes the que sean guardados o cambiados en la base de datos.
* Mantener las operaciones de la base de datos orientadas a objetos.

**2 Convención sobre Configuración en Active Record**

Cuando escribimos aplicaciones usando otros lenguajes de programacion o frameworks, es necesario escribir una gran cantidad de código de configuración. Esto es particularmente verdad para los frameworks ORM en general. Sin embargo, si sigues las convenciones adoptadas por Rails, necesitarás escribir muy poca configuración (en algunos casos ninguna) cuando creas modelos Active Record. La idea es que si configuras tus aplicaciones de la misma manera la mayoría de las veces, entonces ese debería ser la manera por defecto. Por consiguiente, podríamos necesitar configuración explícita solo en los casos donde no podemos seguir la convención estandard.

**2.1 Convenciones sobre Nombres**

Por defecto, Active Record utiliza algunas convenciones para encontrar algunas convenciones para conocer con detalle como el mapeo entre los modelos y las tablas de la base de datos debería crearse. Rails convertirá al plural los nombres de tus clases para encontrar la respectiva tabal en la base de datos. Entonces, para una clase Book, deberías tener una tabla de base de datos llamada **books**. Los mecanismos de pluralizar the Rails son muy potentes, tiene la capacidad de pluarizar (y singularizar) ambos en palabras regulares e irregulares. Cuando usamos nombres de clases compuestos de dos o más palabras, el nombre de la clase del modelo debe debería seguir las convenciones Ruby, usando la forma CamelCase, mientras que el nombre de la tabla debe contener las palabras separadas por guiones bajos. Ejemplos:

* Tabla de Base de Datos - Plural con guiones bajos separando las palabras (ej: book\_clubs).
* Clase del Modelo - Singular con la primera letra de cada palabra en mayúsculas (ej: BookClub).

| **Modelo / Clase** | **Tabla / Esquema** |
| --- | --- |
| Article | articles |
| LineItem | line\_items |
| Deer | deers |
| Mouse | mice |
| Person | people |

**2.2 Convenciones del Esquema**

Active Record utiliza convenciones de nombres para las columnas en las tablas de la base de datos, dependiendo de los propósitos de esas columnas.

* **Claves foráneas** - Estos camos deberían ser nombrados siguiendo el patrón nombre\_de\_tabla\_en\_singular\_id (ej: item\_id, order\_id). Estos son los campos que Active Record buscará cuando crees asociaciones entre tus modelos.
* **Claves primarias** - Por defecto, Active Record utilizará una columna entero llamada idcomo la clave primaria de la tabla. Cuando utilizas [Active Record Migrations](http://www.guiasrails.es/migrations.html) para crear tus tablas, esta columna será creada automáticamente.

También hay nombres de columnas opcionales que podrán dar características adicionales a las instancias de Active Record:

* created\_at - Automáticamente guarda el dia y la hora actual el momento en que se crea el objeto.
* updated\_at - Automáticamente guarda el día y la hora actual del momento en que se actualiza un registro.
* lock\_version - Añade [bloqueo optimista](http://api.rubyonrails.org/classes/ActiveRecord/Locking.html) a un modelo.
* type - Especifica el modo de utilizar el modelo [La Herencia Simple de Tables](http://api.rubyonrails.org/classes/ActiveRecord/Base.html#class-ActiveRecord::Base-label-Single+table+inheritance).
* (association\_name)\_type - Graba el tipo para [asociaciones polimórficas](http://www.guiasrails.es/association_basics.html#polymorphic-associations).
* (table\_name)\_count - Utilizado para cachear el número de objetos que pertencientes en una asociación. Por ejemplo, una columna comments\_count en una clase Articles que tiene muchas instancias de Comment guardará en cache el número de los comentarios existentes de cada artículo.

Mientras estos nombres de columnas son opcionales, están en realidad reservados por Active Record. Evita el uso de las palabras reservadas si no quieres funcionalidades extra. Por ejemplo, type es una palabra reservada utilizada para definir que una tabla está utilizando Herencia Simple de Tabla (STI). Si no estás utilizando STI, intenta con una palabra análoga como "context", que puede aún así mantener la descripción de los datos que estás modelando.

**3 Creando Modelos Active Record**

Es muy fácil crear modelos Active Record. Todo lo que necesitas hacer es una subclase de la clase ActiveRecord::Base y listo:

|  |
| --- |
| class Product < ActiveRecord::Base  end |

Esto creará una clase modelo Product, mapeada a la tabla products de la base de datos. Para hacer esto también tendrás que tener la posibilidad de mapear columnas de cada fila son los atributos de cada instancia del modelo. Supón que la tabla products fue creada utilizando una sentencia SQL como:

|  |
| --- |
| CREATE TABLE products (     id int(11) NOT NULL auto\_increment,     name varchar(255),     PRIMARY KEY  (id)  ); |

Siguiendo el esquema de arriba, tendrías la capacidad de escribir código como el que sigue:

|  |
| --- |
| p = Product.new  p.name = "Some Book"  puts p.name # "Some Book" |

**4 Sobrescribiendo Las Convensiones de Nombres**

¿Que sucede si quieres seguir diferenes convenciones de nombres o necesitas utilizar una aplicación Rails con una base de datos heredada? No hay problema, puedes facilmente sobrescribir las convenciones por defecto.

Utilizando el método ActiveRecord::Base.table\_name= para especificar el nombre de la tabla que debería ser utilizada:

|  |
| --- |
| class Product < ActiveRecord::Base    self.table\_name = "PRODUCT"  end |

Si tu haces esto, tendrás que definir manualmente el nombre de la clase a contiene los fixtures (class\_name.yml) utilizando el método the set\_fixture\_class en la definición de los test.

|  |
| --- |
| class FunnyJoke < ActiveSupport::TestCase    set\_fixture\_class funny\_jokes: Joke    fixtures :funny\_jokes    ...  end |

También es posible cambiar la columna clave primaria de la tabla con el método ActiveRecord::Base.primary\_key=:

|  |
| --- |
| class Product < ActiveRecord::Base    self.primary\_key = "product\_id"  end |

**5 CRUD: Leyendo y Escribiendo Datos**

CRUD es el acronismo para las cuatro verbos que utilizamos para operar con los datos: **C**reate,**R**ead, **U**pdate and **D**elete. Active Record automáticamente crea métodos que permiten a una aplicación leer y manipular los datos guardados dentro de las tablas.

**5.1 Create**

Los objetos Active Record pueden crearse desde un hash, un bloque o configurar sus atributos manualmente antes de la creación. El método new retornará un nuevo objeto mientras createretornará retornará el objeto y lo guardará en la base de datos.

Por ejemplo, dado un modelo User con los atributos name y occupation, el método llamado createcreará y guardará un nuevo registro en la base de datos:

|  |
| --- |
| user = User.create(name: "David", occupation: "Code Artist") |

Utilizando el método new, un objeto puede ser instanciado sin haber sido guardado:

|  |
| --- |
| user = User.new  user.name = "David"  user.occupation = "Code Artist" |

Una llamada a user.save guardará el registro en la base de datos.

Finalmente, si un bloque es proveído, ambos create y new producirá el nuevo objeto inicializado dentro de un bloque:

|  |
| --- |
| user = User.new do |u|    u.name = "David"    u.occupation = "Code Artist"  end |

**5.2 Read**

Active Record provee una rica API para acceder a los datos dentro de una base de datos. Debajo hay algunos ejemplos de diferentes métodos provistos por Active Record.

|  |
| --- |
| # devuelve una colección de usuarios  users = User.all |
| # devuelve el primer usuario  user = User.first |

|  |
| --- |
| # devuelve el primer usuario llamado David  david = User.find\_by(name: 'David') |
| # encontrar todos los usuarios llamados David que tienen de ocupación Code Artists y ordenado por created\_at en sentido cronologicamente inverso  users = User.where(name: 'David', occupation: 'Code Artist').order('created\_at DESC') |

Puedes aprender más acerca de consultar un modelo Active Record en la guía [Interface de Consultas Active Record](http://www.guiasrails.es/active_record_querying.html).

**5.3 Update**

Una vez que un objeto Active Record ha sido recuperado, sus atributos pueden ser modificados y volver a ser guardados en la base de datos.

|  |
| --- |
| user = User.find\_by(name: 'David')  user.name = 'Dave'  user.save |

Un camino corto para este uso es mapear un hash con los nombres de los atributos que se desean modificar y su valor, como por ejemplo:

|  |
| --- |
| user = User.find\_by(name: 'David')  user.update(name: 'Dave') |

Esta es la forma más poderosa de actualizar varios atributos a la vez. Si, por otro lado, quieres actualizar varios registros a la vez, encontrarás muy útil el método de clase update\_all:

|  |
| --- |
| User.update\_all "max\_login\_attempts = 3, must\_change\_password = 'true'" |

**5.4 Delete**

Asimismo, una vez que se recupera el objeto Active Record también puede ser destruído, lo cual lo borrará de la base de datos.

|  |
| --- |
| user = User.find\_by(name: 'David')  user.destroy |

**6 Validaciones**

Active Record te permite validar el estado del modelo antes que se escriba en la base de datos. Hay varios metodos que puedes utilizar para comprobar tu modelo y validar que un atributo no está vacio, es único y no está aún en la base de datos, que siga un formato específico, y muchos más. La validación es una tarea muy importante a considerar cuando se guardan datos en una base de datos, entonces los métodos save y update toman esto en cuenta cuando se ejecutan: ellos retornan false cuando una validación falla y no mantuvieron ninguna operación en la base de datos. Todos estos tienen una contrapartida bang (esta es , save! y update!), la cuales son estrictas y arrojan una excepciónn ActiveRecord::RecordInvalid si la validación falla. Un rápido ejemplo para ilustralo:

|  |
| --- |
| class User < ActiveRecord::Base    validates :name, presence: true  end    user = User.new  user.save  # => false  user.save! # => ActiveRecord::RecordInvalid: Validation failed: Name can't be blank |

Puedes aprender más acerca de validaciones en la [Guía de Validaciones Active Record](http://www.guiasrails.es/active_record_validations.html).

**7 Callbacks**

Las retrollamadas Active Record (o callbacks) te permiten adjuntar ciertos eventos en el ciclo de vida de tus modelos. Esto te faculta a añadir comportamintos a tus modelos de forma transparente en la ejecución cuando estos eventos ocurren. Como cuando tu quieres crear un nuevo registro, actualizarlo, destruírlo, etc. Puedes aprender más de retrollamadas en la [Guía de Active Record Callbacks](http://www.guiasrails.es/active_record_callbacks.html).

**8 Migraciones**

Rails provee un lenguaje de dominio específico para manejar un esquema de base de datos llamado migraciones (migrations). Las migraciones son ficheros guardados que se ejectutan contra cualquier base de datos que Active Record soporte utilizando rake. Aquí hay una migración que crea una tabla:

|  |
| --- |
| class CreatePublications < ActiveRecord::Migration    def change      create\_table :publications do |t|        t.string :title        t.text :description        t.references :publication\_type        t.integer :publisher\_id        t.string :publisher\_type        t.boolean :single\_issue          t.timestamps null: false      end      add\_index :publications, :publication\_type\_id    end  end |

Rails mantiene el historial sobre que fichero fue actualizado en la base de datos y provee características para deshacer los cambios. Para realmente crear la tabla, deberías ejecutar rake db:migrate y para deshacerlo rake db:rollback.