Exemple de couche de persistence : ActiveRecord

Installation

```
gem install activerecord
sudo apt-get install libsqlite3-dev
gem install sqlite3 (ou mysql ou postgres ou autre BD)
```

Utilisation et définition de la connexion à la base

```
<pre 'code'>
require 'active_record'

ActiveRecord::Base.establish_connection(
  :adapter => "sqlite" | "mysql| "postgres" | ...
)
```

```
<pre 'code'>

#mysql
ActiveRecord::Base.establish_connection(
  :adapter  => "mysql",
  :host     => "localhost",
  :username => "myuser",
  :password => "mypass",
  :database => "somedatabase"
)
# sqlite
ActiveRecord::Base.establish_connection(
  :adapter => "sqlite3",
  :database  => "path/to/dbfile"
)

ActiveRecord::Base.establish_connection(
  YAML::load(File.open('database.yml')))
```

Composants d'ActiveRecord

- interface à la définition d'un schéma de BD : migration
- ORM : `ActiveRecord::Base` = Table, 1 objet = 1 ligne de la table
- relations 1..1, 1..N, N..N entre objets
- validateur : codage de contraintes sur les valeurs

Migration

- abstraction de la définition du schéma d'une BD : définition des tables, du nom et du type des colonnes
- gestion des modifications au fil du temps : ajout de tables, modification de colonnes. Le schéma évolue de version en version.
- indépendant du SGBD !

Définition d'une migration

Un fichier avec :

- classe héritière de `ActiveRecord::Migration`
- une méthode `up`, codant le passage de la version n à n+1
- une méthode `down`, codant le passage de la version n+1 à n
- les méthodes `up` et `down` font références à des transformations.

Exemple :

```
<pre 'code'>
class AddSsl < ActiveRecord::Migration
  def up
    add_column :accounts, :ssl_enabled, :boolean, :default => 1
  end

  def down
    remove_column :accounts, :ssl_enabled
  end
end
```

La colonne `:id` de type `integer` est automatiquement définie comme clef de la table par défaut.

Migration : les transformations

Sur les tabels

- `create_table(name, options)` : création de table, rend un objet disponible dans un block pour des transformations ultérieure. `options` est un Hash pour passer des arguments à la création de la table.
- `drop_table(name)`
- `rename_table(old_name, new_name)`

Sur les colonnes

- `add_column(table_name, column_name, type, options)`: ajout de colonne. Les types disponibles sont: `:string`, `:text`, `:integer`, `:float`, `:decimal`, `:datetime`, `:timestamp`, `:time`, `:date`, `:binary`, `:boolean`. Une valeur par défaut peut être spécifié avec une option, i.e. `{ :default => 11 }`. Autre options: `:limit`,`:null` (e.g. { :limit => 50, :null => false }) cf ActiveRecord::ConnectionAdapters::TableDefinition#column pour plus de précision.
- `rename_column(table_name, column_name, new_column_name)`: renommage de colonne, le type et le contenu sont inchangés.
- `change_column(table_name, column_name, type, options)` : changement du type d'une colonne, même syntaxe que `add_column`
- `remove_column(table_name, column_name)`

Sur les index

- `add_index(table_name, column_names, options)`: indexation d'une colonne. Options disponibles: `:name`, `:unique` e.g. `{ :name => "users_name_index", :unique => true }`.
- `remove_index(table_name, :column => column_name)`
- `remove_index(table_name, :name => index_name)`

Migration : exemple

```
<pre 'code'>
class AddSystemSettings < ActiveRecord::Migration
  def up
    create_table :system_settings do |t|
      t.string  :name
      t.string  :label
      t.text  :value
      t.string  :type
      t.integer  :position
    end

    SystemSetting.create  :name => "notice",
                          :label => "Use notice?",
                          :value => 1
  end

  def down
    drop_table :system_settings
  end
end
```

Toutes les migrations sont mise dans le même répertoire

Les migrations sont jouées dans l'ordre alphabétique des fichiers.

```
<pre 'code'>
ActiveRecord::Migrator.migrate "db/migrate"
# plus finement :
ActiveRecord::Migrator.run(:up, "db/migrate/", version)
ActiveRecord::Migrator.run(:down, "db/migrate/", version)
```

Object Relational Mapping

1 ligne d'une table <=> un object

conventions :

- nom de la table au pluriel `People`, nom de la classe au singulier `Person`, mais `set_table_name "autre_nom"`
- colonne `:id` : clef de la table, mais `set_primary_key "autre_colonne"`

```
<pre 'code'>
class Person < ActiveRecord::Base
# table People, clef primaire :id
# accesseurs créés avec le nom des colonnes de la table
end
```

Creation

```
<pre 'code'>
p = Person.new
p.name = "Eponge"
p.firstname = "Bob"
# p.id => nil
p.save
# p.id => 1
```

Lecture

```
<pre 'code'>
p = Person.find(1)
p = Person.find_by_name("Eponge")
p = Person.find_all_by_firstname("Bob")
```

Mise à jour

```
<pre 'code'>
p.firstname = "Robert"
p.save
```

Effacement

```
<pre 'code'>
Person.all.each{|p| p.destroy}
```

ActiveRecord : gestion des associations entre objets/tables

Les clefs externes dans les tables sont de la forme `objet_id` par convention.

1-1

has_one/belongs_to

```
<pre 'code'>

class Employee < ActiveRecord::Base
  has_one :office
end
class Office < ActiveRecord::Base
  belongs_to :employee    # foreign key - employee_id
end
```

1-N

has_many/belongs_to

```
<pre 'code'>
class Manager < ActiveRecord::Base
  has_many :employees
end
class Employee < ActiveRecord::Base
  belongs_to :manager     # foreign key - manager_id
end
```

ActiveRecord : associations N-N

has_many :trough

On objectifie la table de jointure

```
<pre 'code'>
class Assignment < ActiveRecord::Base
  belongs_to :programmer  # foreign key - programmer_id
  belongs_to :project     # foreign key - project_id
end
class Programmer < ActiveRecord::Base
  has_many :assignments
  has_many :projects, :through => :assignments
end
class Project < ActiveRecord::Base
  has_many :assignments
  has_many :programmers, :through => :assignments
end
```

has_and_belongs_to_many

```
<pre 'code'>
class Programmer < ActiveRecord::Base
  has_and_belongs_to_many :projects       # foreign keys in the join table
end
class Project < ActiveRecord::Base
  has_and_belongs_to_many :programmers    # foreign keys in the join table
end
```

Activerecord : associations et méthodes générées

1-1

generated methods	belongs_to	belongs_to :polymorphic	has_one
other	X	X	X
other=(other)	X	X	X
build_other(attributes={})	X		X
create_other(attributes={})	X		X
create_other!(attributes={})	X		X

1-N et N-N

generated methods	habtm	has_many	has_many :through
others	X	X	X
others=(other,other,...)	X	X	X
other_ids	X	X	X
other_ids=(id,id,...)	X	X	X
others<<	X	X	X
others.push	X	X	X
others.concat	X	X	X
others.build(attributes={})	X	X	X
others.create(attributes={})	X	X	X
others.create!(attributes={})	X	X	X
others.size	X	X	X
others.length	X	X	X
others.count	X	X	X
others.sum(args*,&block)	X	X	X
others.empty?	X	X	X
others.clear	X	X	X
others.delete(other,other,...)	X	X	X
others.delete_all	X	X	X
others.destroy_all	X	X	X
others.find(*args)	X	X	X
others.exists?	X	X	X
others.uniq	X	X	X
others.reset	X	X	X

ActiveRecord : valider ses donnés

Les validateurs permettent de coder des contraintes, permettant de ne sauver dans la BD que les enregistrements vérifiant ces contraintes. Voir la documentation complète pour les détails.

Validateurs fournis : `validates :attribute ...`

Les validateurs sont des implémentations de la classe `ActiveModel::EachValidator`

```
<pre 'code'>
class Person
  validates :terms, :acceptance => true
  validates :password, :confirmation => true
  validates :username, :exclusion => { :in => %w(admin superuser) }
  validates :email, :format => { :with => /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i, :on => :create }
  validates :age, :inclusion => { :in => 0..9 }
  validates :first_name, :length => { :maximum => 30 }
  validates :age, :numericality => true
  validates :username, :presence => true
  validates :username, :uniqueness => true
end
```

Tester la validité : `valid?`, `errors`

La phase de validation est automatiquement faite par les méthodes `create`, `save`, `update` et `update_attributes` On peut forcer la validation en utilisant la méthode `valid?`

```
<pre 'code'>
 p = Person.new
=> #<Person id: nil, password: nil, password_confirmation: nil, username: nil, email: nil, age: nil, first_name: nil, created_at: nil, updated_at: nil>
>> p.valid?
=> false
>> p.errors
=> #<ActiveModel::Errors:0x00000003eb4b70 @base=#<Person id: nil, password: nil, password_confirmation: nil, username: nil, email: nil, age: nil, first_name: nil, created_at: nil, updated_at: nil>, @messages={:email=>["is invalid"], :age=>["is not included in the list", "is not a number"], :username=>["can't be blank"]}>
>> p.errors.messages
=> {:email=>["is invalid"], :age=>["is not included in the list", "is not a number"], :username=>["can't be blank"]}
```