Gemas y Active Records Sin Rails

Gemas

Introducción

- Una gema es un formato simple para publicar y compartir código Ruby.
- Cada gema tiene un nombre, versión y plataforma.

Gemas

CLI

- **Instalar una gema:** `gem install rake`
- **Buscar una gema:** `gem search sinatra`
- **Listar gemas instaladas:** `gem list`

Bundler

Introducción

- Mantiene un entorno consistente para las aplicaciones ruby
- Asegura que la aplicación que lo use tenga las dependencias necesarias para que se ejecute sin errores.
- Bundler es una gema: `gem install bundler`

Bundler

Ejemplo de uso

Definimos las dependencias en el archivo `Gemfile`

```
source 'https://rubygems.org'

gem 'sinatra'
```

Luego instalamos las dependencias con **`bundle install`** o simplemente **`bundle`**.

Bundler

Comandos

```
# Instalar dependencias:
bundle install

# Actualizar dependencias a sus últimas versiones:
bundle update

# Ejecutar un script en el contexto del bundle actual:
bundle exec

# Ver las gemas instaladas en el bundle actual:
bundle list

# Ver donde está ubicada una gema:
bundle show NOMBRE_GEMA
```

Bundler

Uso de gemas

Con declarar las dependencias en el `Gemfile` no basta, hay que invocar a bundler desde el código.

```
require 'bundler'
Bundler.require
```

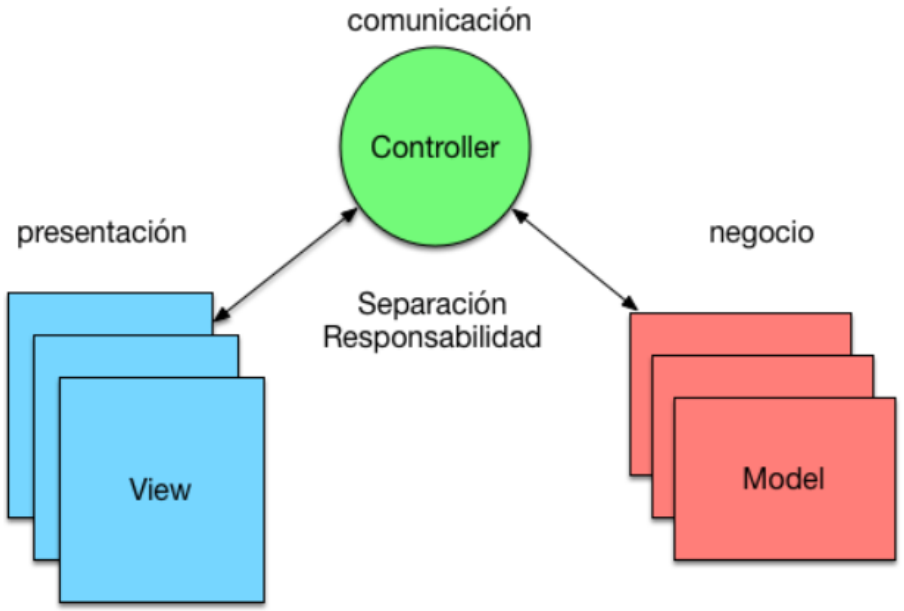
Bundler requiere todas las dependencias.

```
require 'bundler'
Bundler.setup
require 'sinatra'
```

Bundler configura pero los require deben ser explícitos.

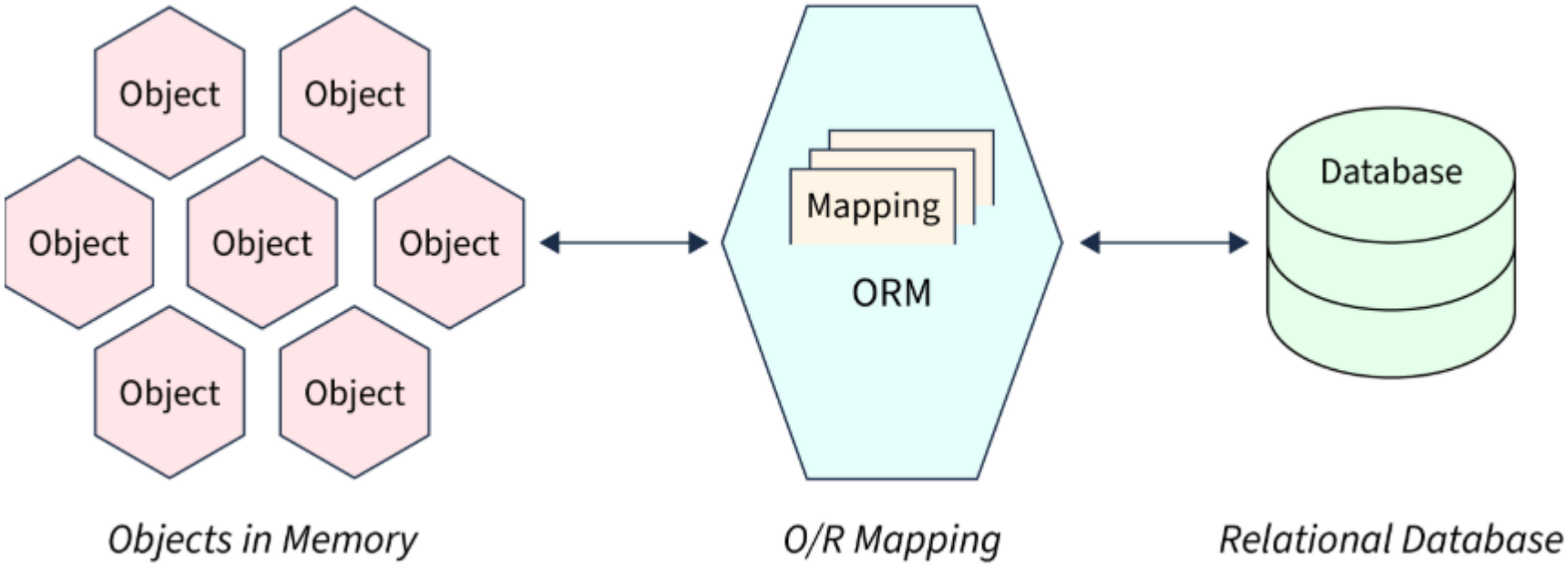
¿Qué es Active Record?

Active Record es parte de la **M** en el patrón MVC en una aplicación Ruby
Es la capa del sistema responsable de representar datos y lógica de negocio.
Active Record te ayuda a crear y usar objetos Ruby cuyos atributos requieren almacenamiento persistente en una base de datos.



El Patrón Active Record

El patrón Active Record es descrito por Martin Fowler en el libro Patterns of Enterprise Application Architecture como "**un objeto que envuelve una fila en una tabla de base de datos, encapsula el acceso a la base de datos y añade lógica de dominio a esos datos.**"



Active Record como un Marco ORM

Active Record nos da la capacidad de hacer lo siguiente usando objetos Ruby:

- Representar modelos y sus datos.
- Representar asociaciones entre modelos.
- Representar jerarquías de herencia a través de modelos relacionados.
- Validar modelos antes de que se persistan en la base de datos.
- Realizar operaciones de base de datos de manera orientada a objetos.

Convenciones de Nomenclatura - Ejemplos

Modelo / Clase	Tabla / Esquema
`Article`	`articles`
`LineItem`	`line_items`
`Product`	`products`
`Person`	`people`

Ejemplo gema Active Records

Inicializar bundler

```
PS C:\Users\USUARIO> bundle init
```

Editar gemfile

```
source "https://rubygems.org"

gem "activerecord"
gem "sqlite3"
```

Instalar las gemas

```
PS C:\Users\USUARIO> bundle install
```

Ejemplo: Relación uno a uno

```ruby
require "bundler/setup"
require "active_record"

# Conexión a SQLite
ActiveRecord::Base.establish_connection(
  adapter: "sqlite3",
  database: "test.db"
)

# Definir modelos
class User < ActiveRecord::Base
  has_one :profile
end

class Profile < ActiveRecord::Base
  belongs_to :user
end
```

Ejemplo: Relación uno a uno

```ruby
# Crear tablas si no existen
ActiveRecord::Schema.define do
  unless User.table_exists?
    create_table :users do |t|
      t.string :name
    end
  end

  unless Profile.table_exists?
    create_table :profiles do |t|
      t.string :bio
      t.references :user   # crea la foreign key user_id
    end
  end
end
```

Ejemplo: Relación uno a uno

```ruby
# Crear usuario y perfil
user = User.create(name: "Ana")
profile = Profile.create(bio: "Desarrolladora Ruby", user: user)

puts user.profile.bio     # => "Desarrolladora Ruby"
puts profile.user.name    # => "Ana"
```

Ejemplo: Relación muchos a muchos

```ruby
class User < ActiveRecord::Base
  has_many :memberships
  has_many :groups, through: :memberships
end

class Group < ActiveRecord::Base
  has_many :memberships
  has_many :users, through: :memberships
end

class Membership < ActiveRecord::Base
  belongs_to :user
  belongs_to :group
end
```

Ejemplo: Relación muchos a muchos

```ruby
ActiveRecord::Schema.define do
  unless User.table_exists?
    create_table :users do |t|
      t.string :name
    end
  end

  unless Group.table_exists?
    create_table :groups do |t|
      t.string :title
    end
  end

  unless Membership.table_exists?
    create_table :memberships do |t|
      t.references :user
      t.references :group
    end
  end
end
```

Ejemplo: Relación muchos a muchos

```ruby
# Crear datos
user1 = User.create(name: "Ana")
user2 = User.create(name: "Juan")

group1 = Group.create(title: "Admins")
group2 = Group.create(title: "Editores")

# Asignar usuarios a grupos
user1.groups << group1
user1.groups << group2
user2.groups << group2

# Consultas
puts user1.groups.map(&:title).inspect   # ["Admins", "Editores"]
puts group2.users.map(&:name).inspect    # ["Ana", "Juan"]
```

Ejemplo: Relación uno a muchos

```ruby
# Definir modelos
class User < ActiveRecord::Base
  has_many :posts
end

class Post < ActiveRecord::Base
  belongs_to :user
end
```

Ejemplo: Relación uno a muchos

```ruby
# Crear tablas si no existen
ActiveRecord::Schema.define do
  unless User.table_exists?
    create_table :users do |t|
      t.string :name
    end
  end

  unless Post.table_exists?
    create_table :posts do |t|
      t.string :title
      t.text :content
      t.references :user   # crea columna user_id
    end
  end
end
```

Ejemplo: Relación uno a muchos

```ruby
# Crear usuario y posts
user = User.create(name: "Ana")
Post.create(title: "Primer post", content: "Hola mundo", user: user)
Post.create(title: "Segundo post", content: "Aprendiendo ActiveRecord", user: user)

# Consultas
puts "Posts de Ana:"
user.posts.each do |post|
  puts "- #{post.title}: #{post.content}"
end

puts "El autor del primer post es: #{Post.first.user.name}"
```