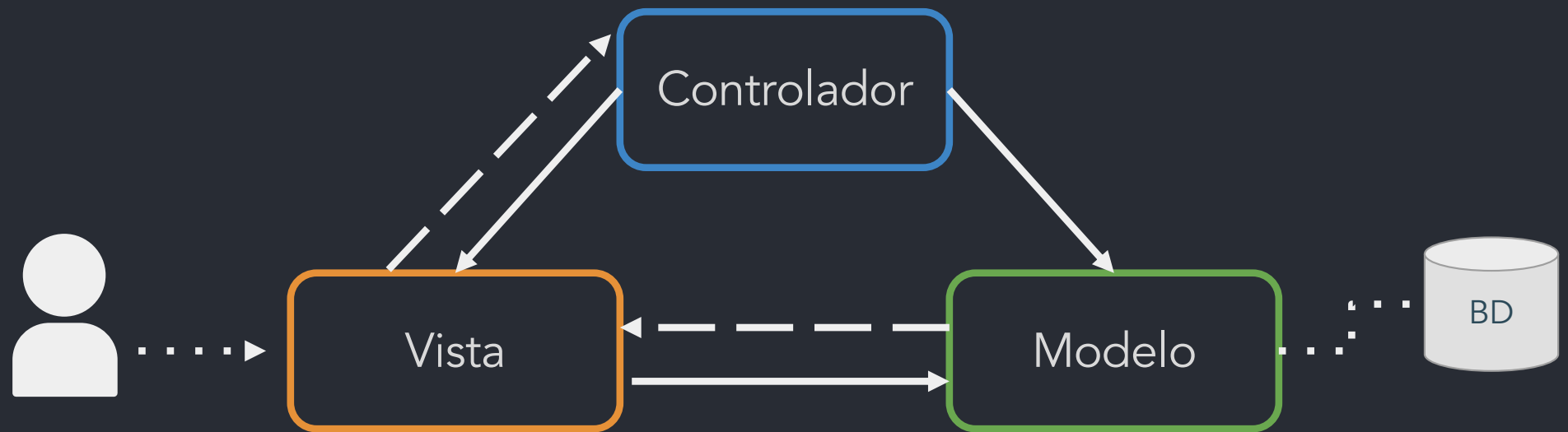


# Active Record

...

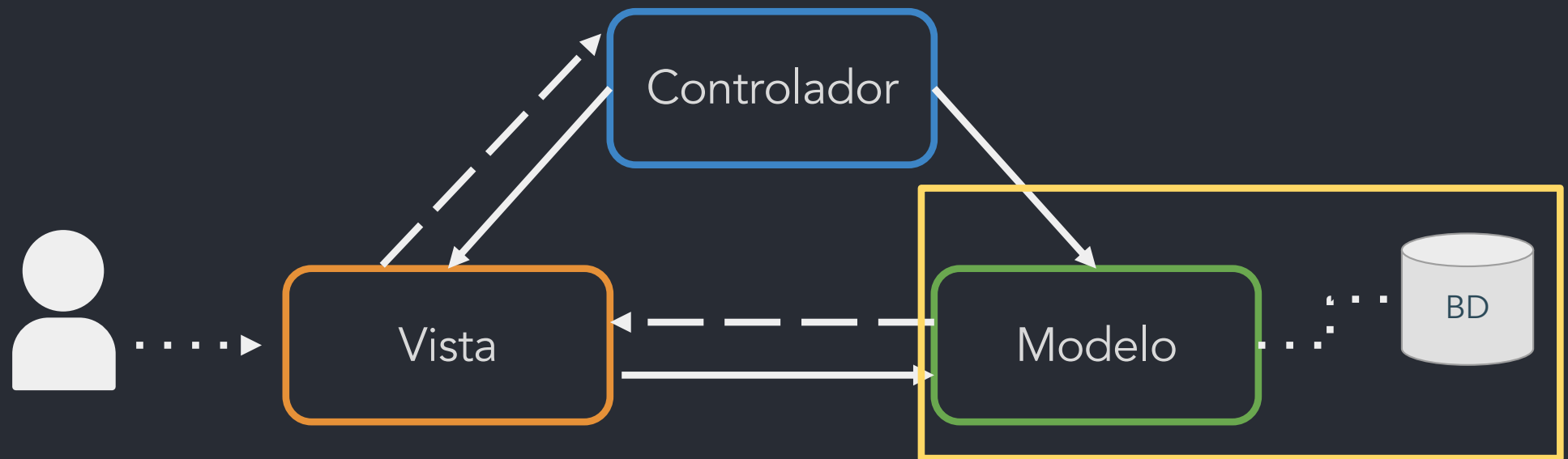
¿Qué es Active Record?

# ¿Qué es Active Record?



MVC

# ¿Qué es Active Record?

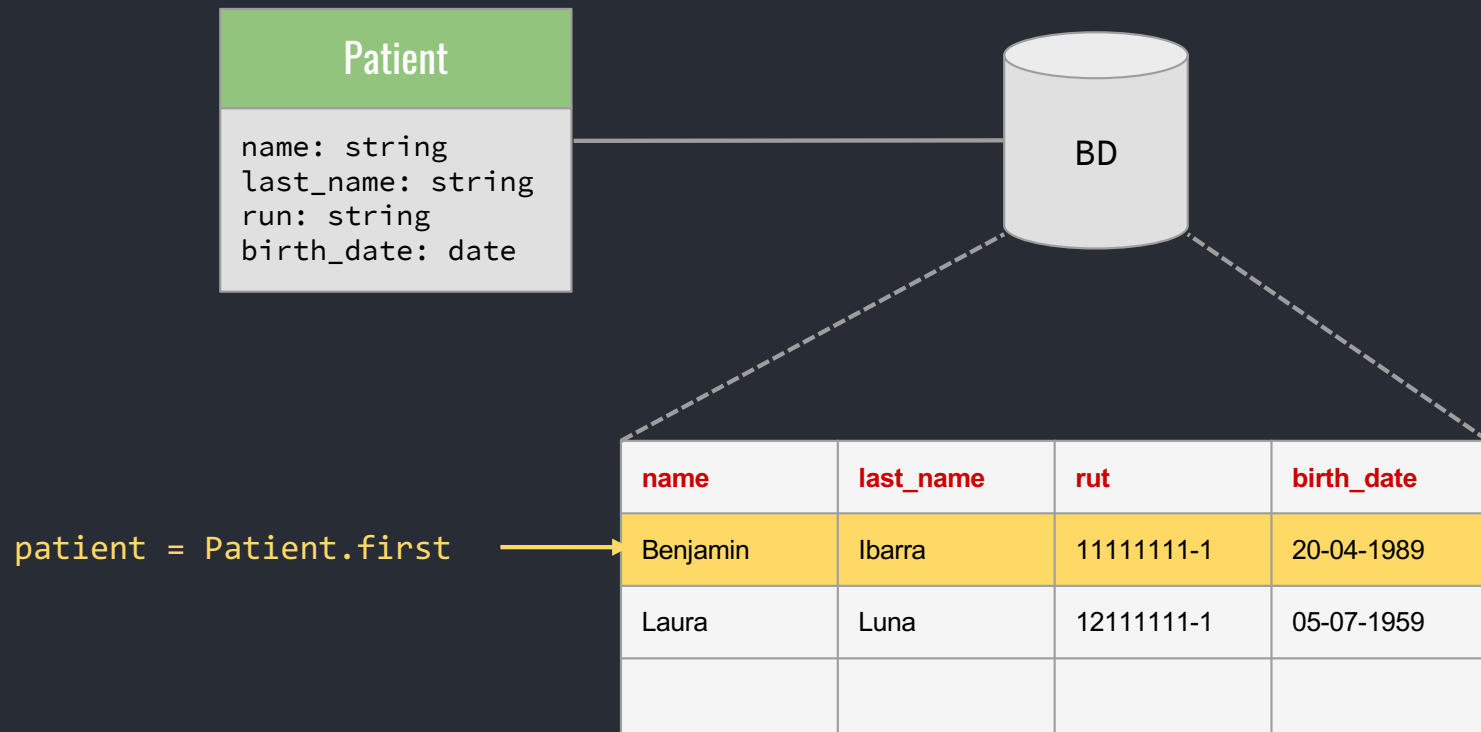


MVC

# ¿Qué es Active Record?

- Patrón de arquitectura
  - Objetos tienen datos persistentes y comportamiento para operar sobre datos
- Utiliza ORM (Object Relational Mapping)
  - Objetos se conectan con los datos. Los objetos pueden acceder a los datos y relaciones sin que el desarrollador escriba SQL directo

# ¿Qué es Active Record?



# Rails utiliza Active Record del lado de los modelos

## Conventions over configurations

- Modelos en singular (regular o irregular)
- Modelos inicial mayúscula y camelCase
- Base de datos plural
- Base de datos minúscula y snake\_case
- Rails genera primary key id automáticamente
- Nombre de las foreign keys deben seguir el formato: singularized\_table\_name\_id (patient\_id)

```
class Patient < ApplicationRecord
class MedicalCenter < ApplicationRecord
  create_table :patients
  create_table :medical_centers
```

[https://guides.rubyonrails.org/v5.2/active\\_record\\_basics.html](https://guides.rubyonrails.org/v5.2/active_record_basics.html)

# Query Interface



# Query Interface

- Interfaz de ActiveRecord que permite hacer operaciones sobre los datos
- Las operaciones se pueden clasificar según CRUD

Read Operations		
<code>.find</code>	<code>.find_by</code>	<code>.all</code>
<code>.first</code>	<code>.last</code>	<code>.take</code>
<code>.where</code>	<code>.order</code>	

# Query Interface

- Method Chaining:
- Invocar múltiples métodos `result = method1().method2().method3()`
- Una query puede devolver:
  - Single objects (<Patient id: 3, ... >)
  - ActiveRecord::Relation
- Solo se pueden encadenar métodos si el resultado de la operación anterior es un ActiveRecord::Relation

Ejemplo:

```
@person = Person.where(name: "Jaime")
```

```
@person = Person.where(name: "Jaime").where(age: 32)
```

# Query Interface

## Create Operations

`.new`  
`.save`  
`.create`

## Update Operations

`.save`  
`.update`

## Delete Operations

`.destroy`  
`Model.destroy_all`

# Validations

- Permiten validar los datos que ingresan en la base de datos
- Validamos lógica de negocio
- No queremos datos erróneos o peligrosos que corrompan la base de datos
- Existen distintos tipos de validación:

# Validations


- Permite validar los datos que ingresan en la base de datos
- Validamos lógica de negocio
- No queremos datos erróneos o peligrosos que corrompa la base de datos
- Existen distintos tipos de validación:

**client-side:** Validaciones en la página/formulario/vista

# Validations

- Permite validar los datos que ingresan en la base de datos
- Validamos lógica de negocio
- No queremos datos erróneos o peligrosos que corrompa la base de datos
- Existen distintos tipos de validación:

**client-side:** Validaciones en la página/formulario/vista

**server-side:**  **Controller:** Validaciones a nivel de controlador

# Validations

- Permite validar los datos que ingresan en la base de datos
- Validamos lógica de negocio
- No queremos datos erróneos o peligrosos que corrompa la base de datos
- Existen distintos tipos de validación:


`client-side:`            Validaciones en la página/formulario/vista

`server-side:` { `Controller:`    Validaciones a nivel de controlador  
                  `Database:`    Constraints/triggers/lógica embebida

# Validations

- Permite validar los datos que ingresan en la base de datos
- Validamos lógica de negocio
- No queremos datos erróneos o peligrosos que corrompa la base de datos
- Existen distintos tipos de validación:

**client-side:** Validaciones en la página/formulario/vista

**server-side:**  **Controller:** Validaciones a nivel de controlador

**Database:** Constraints/triggers/lógica embebida

**Modelos:** Validación a nivel de modelo



# Validations

- Permite validar los datos que ingresan en la base de datos
- Validamos lógica de negocio
- No queremos datos erróneos o peligrosos que corrompa la base de datos
- Existen distintos tipos de validación:

**client-side:**

Validaciones en la página/formulario/vista



**server-side:**

**Controller:**

Validaciones a nivel de controlador



**Database:**

Constraints/triggers/lógica embebida



**Modelos:**

Validación a nivel de modelo



# Active Record Validations in Rails

- Métodos que se corren justo antes de SQL INSERT o SQL UPDATE
- ¿Qué ocurre si falla?
  - Active Record queda inválido
  - No se hace la operación SQL

Validation helpers	
presence	numericality
uniqueness	acceptance
format	confirmation
length	inclusion

# Ejemplos

```
class Person < ApplicationRecord
  validates :name, presence: true
end
```

```
> Person.create(name: "John Doe").valid?
```

```
=> true
```

```
> Person.create(name: nil).valid?
```

```
=> false
```

# Ejemplos

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

```
class Promotion < ApplicationRecord
  validates :start_date, comparison: { greater_than: :end_date }
end
```

# Errores

- Todos los errores pueden obtenerse desde el metodo `errors` que devuelve la colección de todos los errores
- `errors.any` es verdadero si hubo al menos un error

```
class Person < ApplicationRecord
  validates :name, presence: true
end
```

```
p = Person.new
=> #<Person id: nil, name: nil>
p.errors.size
=> 0
p.valid?
=> false
p.errors.objects.first.full_message
=> "Name can't be blank"
```

# Validations in Rails

- `allow_blank`
- `allow_nil`
- `message`

Validation options	
<code>allow_blank</code>	<code>on</code>
<code>allow_nil</code>	<code>if</code>
<code>message</code>	<code>unless</code>

[https://guides.rubyonrails.org/v5.2/active\\_record\\_validations.html](https://guides.rubyonrails.org/v5.2/active_record_validations.html)

# Validations in Rails

¿Qué ocurre cuando un ActiveRecord es invalido?

```
<% if @patient.errors.any? %>
  <div>
    <h2>
      <%= pluralize(@patient.errors.count, "error") %> prohibited
      this patient from being saved:
    </h2>
    <ul>
      <% @patient.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

# Validations in Rails

## errors

- Objeto de la clase ActiveRecord::Errors
- Guarda en messages el atributo como key, y un arreglo de string de errores
- Se pueden agregar errores a nivel de atributo o de clase

```
<% if @patient.errors.any? %>
  <div>
    <h2>
      <%= pluralize(@patient.errors.count, "error") %> prohibited
      this patient from being saved:
    </h2>
    <ul>
      <% @patient.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```



# Validations in Rails

¿Qué pasa si necesitamos una validación más compleja?

Custom Validations con ActiveRecord::Validator

```
class UnderAge < ActiveRecord::Validator
  def validate(record)
    unless !record.birth_date.present? || record.birth_date < 18.years.ago
      record.errors[:birth_date] << 'No under ager can receive the vaccine for the moment'
    end
  end
end

class Patient < ApplicationRecord
  validates_with UnderAge
end
```

# Validations in Rails

¿Qué pasa si necesitamos una validación más compleja? Custom Validations

```
validate :under_age_cannot_be_vaccinated

def under_age_cannot_be_vaccinated
  if birth_date.present? && birth_date > 18.years.ago
    errors.add(:birth_date, "No under age can be vaccinated")
  end
end
```

con custom Methods

# Callbacks

Callback son métodos que se pueden llamar en cualquier momento del ciclo de vida de un objeto

Callbacks		
<b>before_validation</b>	<b>after_validation</b>	<b>after_commit/</b>
<b>before_save</b>	<b>after_save</b>	<b>after_rollback</b>
<b>before_create</b>	<b>after_create</b>	<b>after_initialize</b>
<b>before_update</b>	<b>after_update</b>	<b>after_find</b>
<b>before_destroy</b>	<b>after_destroy</b>	

# Asociaciones

...

# Associations in Rails

- Relación entre dos modelos
- Permiten hacer operaciones más cómodamente

```
@book.create_author(name: 'J.K Rowling')  
@author.books << @book1  
@author.books.each do |book|  
  p book.name  
end
```

# Associations in Rails

- Relación entre dos modelos
- Permiten hacer operaciones más cómodamente

```
@book.create_author(name: 'J.K Rowling')  
@author.books << @book1  
@author.books.each do |book|  
  p book.name  
end
```

¡No es magia! Las claves foráneas se deben agregar mediante migraciones

# Associations in Rails

- Relación entre dos modelos
- Permiten hacer operaciones más cómodamente

```
@book.create_author(name: 'J.K Rowling')
@author.books << @book1
@author.books.each do |book|
  p book.name
end
```

¡No es magia! Las claves foráneas se deben agregar mediante migraciones

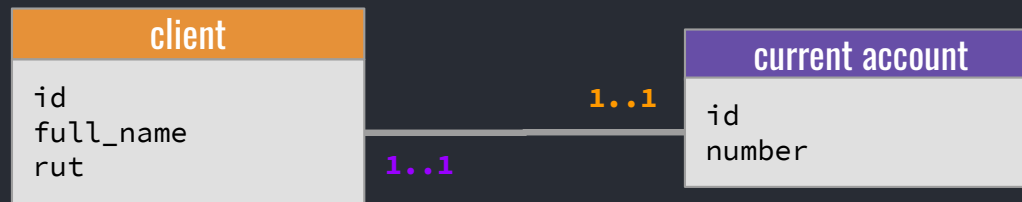
Asociaciones  
disponibles

belongs\_to  
has\_one  
has\_many  
has\_many :through  
has\_one :through  
has\_and\_belongs\_to\_many

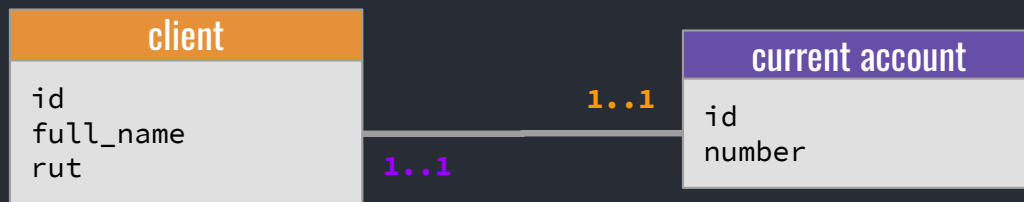
Asociaciones de acuerdo a cardinalidad



1-1



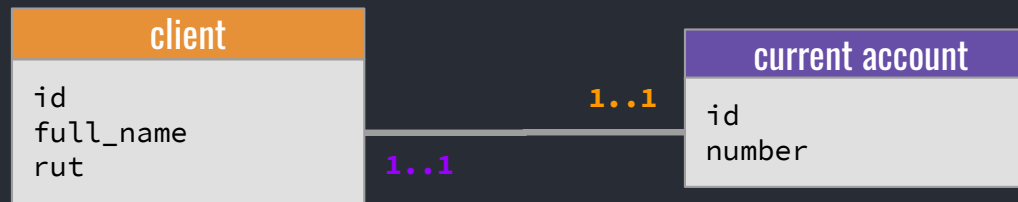
1-1



```
class Client < ActiveRecord
  has_one :current_account
end
```

```
class CurrentAccount < ActiveRecord
  belongs_to :client
end
```

1-1



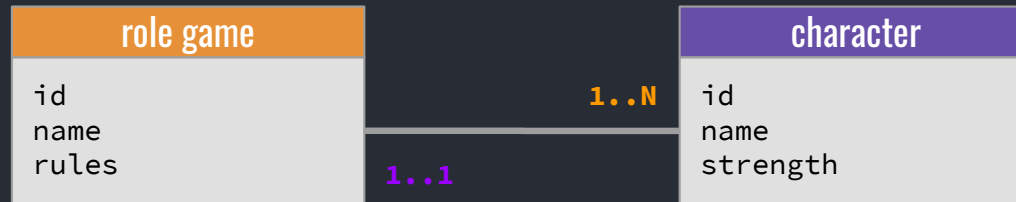
```
class Client < ActiveRecord
  has_one :current_account
end
```

```
class CurrentAccount < ActiveRecord
  belongs_to :client
end
```

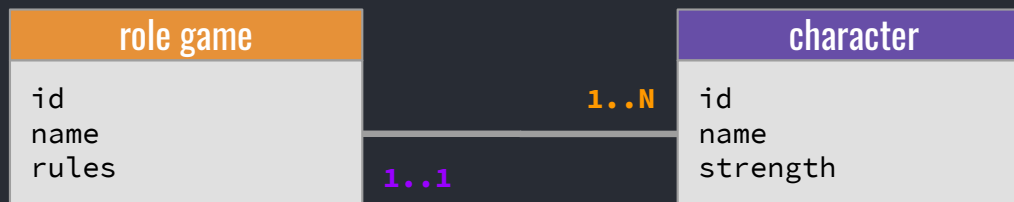
```
class CreateBankEntities < ActiveRecord::Migration[5.0]
  def change
    create_table :clients do |t|
      t.string :full_name
      t.string :rut
      t.timestamps
    end

    create_table :current_accounts do |t|
      t.belongs_to :client, index: true
      t.string :number
      t.timestamps
    end
  end
end
```

# 1-N



# 1-N



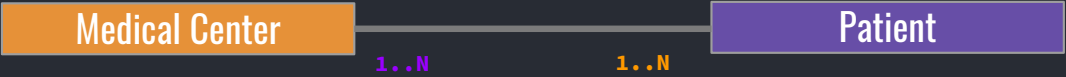
```
class RoleGame < ActiveRecord
  has_many :characters
end
```

```
class Character < ActiveRecord
  belongs_to :role_game
end
```

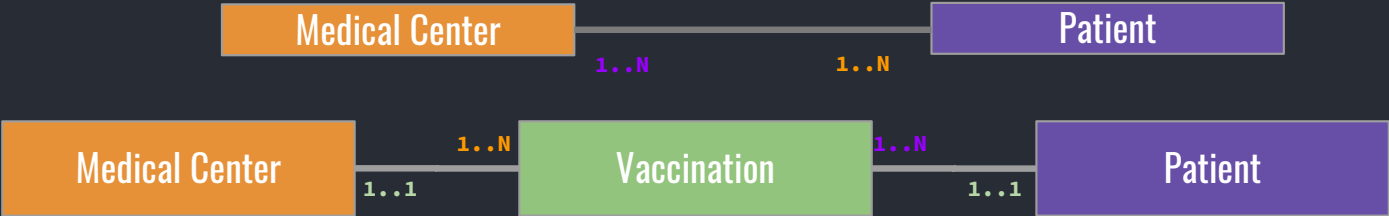
```
class CreateRoleGames < ActiveRecord::Migration[5.0]
  def change
    create_table :role_games do |t|
      t.string :name
      t.text :rules
      t.timestamps
    end

    create_table :characters do |t|
      t.belongs_to :role_game, index: true
      t.string :name
      t.integer :strength
      t.timestamps
    end
  end
end
```

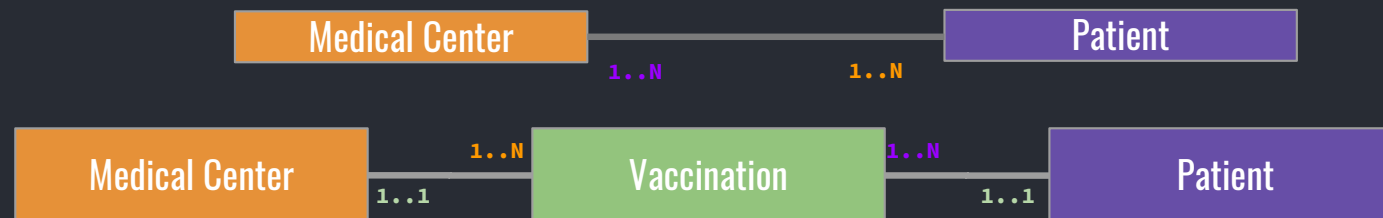
N-N



N-N



# N-N



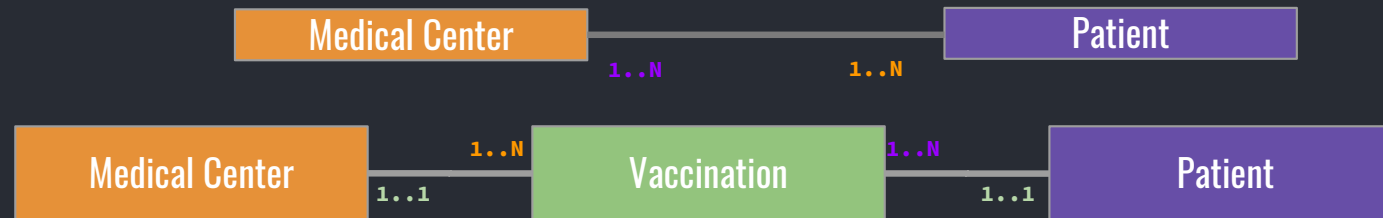
```
class Patient < ActiveRecord
  has_many :vaccinations
  has_many :medical_centers, through: :vaccinations
end
```

```
class MedicalCenter < ActiveRecord
  has_many :vaccinations
  has_many :patients, through: :vaccinations
end
```

```
class Vaccination < ActiveRecord
  belongs_to :patient
  belongs_to :medical_center
end
```



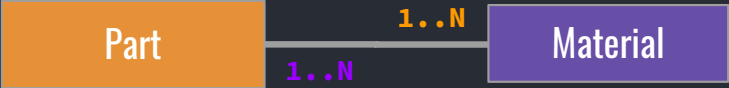
# N-N



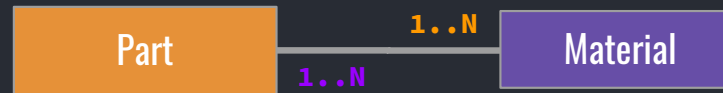
```
class CreateMedicalCentersAndVaccinations < ActiveRecord::Migration[5.0]
  def change
    create_table :medical_centers do |t|
      t.string :name
      t.string :city
      t.string :commune
      t.string :address
      t.timestamps
    end

    create_table :vaccinations do |t|
      t.belongs_to :patient, index: true
      t.belongs_to :medical_center, index: true
      t.datetime :appointment_date
      t.integer :vaccine
      t.boolean :first_dose
      t.timestamps
    end
  end
end
```

N-N



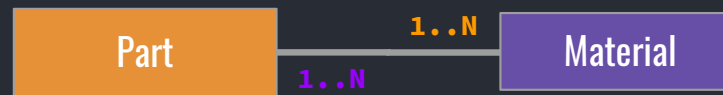
# N-N



```
class Part < ApplicationRecord
  has_and_belongs_to_many :materials
end
```

```
class Material < ApplicationRecord
  has_and_belongs_to_many :parts
end
```

# N-N



```
class Part < ApplicationRecord
  has_and_belongs_to_many :materials
end
```

```
class Material < ApplicationRecord
  has_and_belongs_to_many :parts
end
```

```
class CreateMaterialsAndParts < ActiveRecord::Migration[5.0]
  def change
    create_table :materials do |t|
      t.string :name
      t.timestamps
    end

    create_table :parts do |t|
      t.string :part_number
      t.timestamps
    end

    create_table :materials_parts, id: false do |t|
      t.belongs_to :material, index: true
      t.belongs_to :part, index: true
    end
  end
end
```

1-1



1-1



```
class Client < ActiveRecord
  has_one :current_account
  has_one :debit_card, through: :current_account
end
```

```
class CurrentAccount < ActiveRecord
  belongs_to :client
  has_one :debit_card
end
```

```
class DebitCard < ApplicationRecord
  belongs_to :current_account
end
```

# Self Joins



# Self Joins



Employee

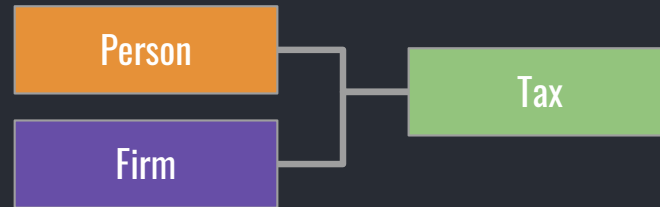
```
class Employee < ApplicationRecord
  has_many :subordinates, class_name: "Employee",
                        foreign_key: "manager_id"
  belongs_to :manager, class_name: "Employee"
end

class CreateEmployees < ActiveRecord::Migration[5.0]
  def change
    create_table :employees do |t|
      t.references :manager, index: true
      t.timestamps
    end
  end
end
```

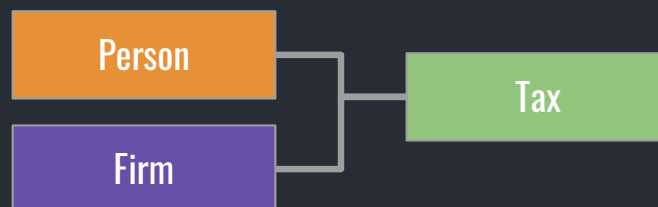
```
@employee.subordinates
@employee.manager
```



Polimórfico



# Polimórfi co



```
class Person < ApplicationRecord
  has_many :taxes, as: :taxable
end
```

```
class Firm < ApplicationRecord
  has_many :taxes, as: :taxable
end
```

```
class Tax < ApplicationRecord
  belongs_to :taxable, polymorphic: true
end
```

```
class CreateTaxes < ActiveRecord::Migration[5.0]
  def change
    create_table :taxes do |t|
      t.string :name
      t.float :fee
      t.references :taxable, polymorphic: true, index: true
      t.timestamps
    end
  end
end
```

## Options

:class_name	:validate
:dependent	:as
:foreign_key	:through
:inverse_of	:primary_key
:polymorphic	

```
class Vaccination < ActiveRecord
  belongs_to :patient, dependent: :destroy
end
```

```
class Employee < ApplicationRecord
  has_many :subordinates, class_name: "Employee",
    foreign_key: "manager_id"
  belongs_to :manager, class_name: "Employee"
end
```

```
class Tax < ApplicationRecord
  belongs_to :taxable, polymorphic: true
end
```

## Options

:class_name	:validate
:dependent	:as
:foreign_key	:through
:inverse_of	:primary_key
:polymorphic	

## Scopes

where	limit
includes	offset
readonly	order
select	

```
class Vaccination < ActiveRecord
  belongs_to :patient, dependent: :destroy
end
```

```
class Employee < ApplicationRecord
  has_many :subordinates, class_name: "Employee",
    foreign_key: "manager_id"
  belongs_to :manager, class_name: "Employee"
end
```

```
class Tax < ApplicationRecord
  belongs_to :taxable, polymorphic: true
end
```

```
class Vaccination < ActiveRecord
  belongs_to :patient, → { where active: true },
    dependent: :destroy
end
```

```
class DebitCard < ApplicationRecord
  belongs_to :current_account, { includes :client }
end
```

# Métodos que agregan las asociaciones

## 1-1

```
association  
association=(associate)  
build_association(attributes)  
create_association(attributes)  
create_association!(attributes)  
reload_association
```

```
@book.create_author(name: 'J.K Rowling')  
@author.books << @book1  
@author.books.each do |book|  
  p book.name  
end
```

## 1-N

```
collection  
collection<<(object, ...)  
collection.delete(object, ...)  
collection.destroy(object, ...)  
collection=(objects)  
collection_singular_ids  
collection_singular_ids=(ids)  
collection.clear  
collection.empty?  
collection.size  
collection.find(...)  
collection.where(...)  
collection.exists?(...)  
collection.build(attributes = {}, ...)  
collection.create(attributes = {})  
collection.create!(attributes = {})  
collection.reload
```

Agregamos un form de una asociación

views/vaccinations/\_form.html.erb

```
<%= form_with(model: [ patient, patient.vaccinations.build ], local: true) do |form| %>
  <p>
    vacuna
    <%= form.select :vaccine, options_for_select(Vaccination.vaccines.keys) %>
  </p>
  <p>
    Fecha de vacunación
    <%= form.date_field :appointment_date, max: Time.now %>
  </p>
  <p>
    ¿Primera dosis?
    <%= form.check_box :first_dose %>
  </p>
  <p>
    <%= form.submit %>
  </p>
<% end %>
```

controllers/vaccinations\_controller.rb

```
class VaccinationsController < ApplicationController
  def create
    @patient = Patient.find_by(id: params[:patient_id])
    if @patient.present?
      if @vaccination = @patient.vaccinations.create(vaccination_params)
        redirect_to patient_path @patient
      else
        render 'patients/show'
      end
    else
      redirect_to patients_path
    end
  end

  private
  def vaccination_params
    params.require(:vaccination).permit(:first_dose, :appointment_date, :vaccine)
  end
end
```



Modelando un chat simple

# Ejemplo de asociaciones

#ChatDelProyecto



**Elena:**  
¿Cuándo es la reunión?



**Jorge:**  
Creo que a las 10

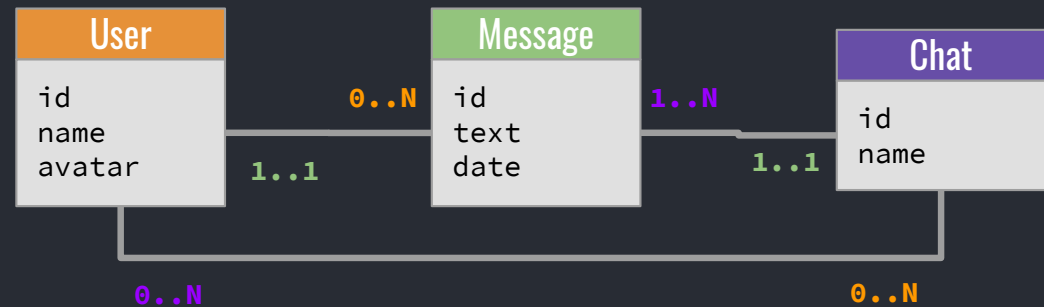


**Hector:**  
Ariel, ¿Sabes dónde nos juntamos?

**Ariel:**  
En el centro de alumnos!



Voy en camino...| send



# Ejemplo de asociaciones

¡Mismo ejemplo que vacunas, centro de vacunación y pacientes!

#ChatDelProyecto

**Elena:**  
¿Cuándo es la reunión?

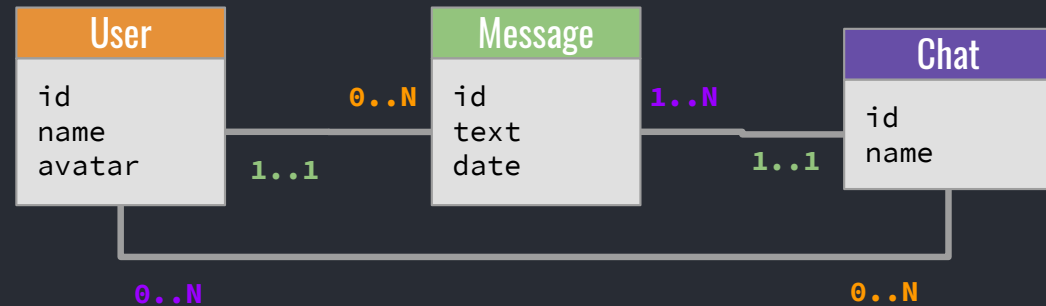
**Jorge:**  
Creo que a las 10

**Hector:**  
Ariel, ¿Sabes dónde nos juntamos?

**Ariel:**  
En el centro de alumnos!



Voy en camino...| send



```
class User < ApplicationRecord
  has_many :messages
  has_many :chats, through: :messages
end
```

```
class Chat < ApplicationRecord
  has_many :messages
  has_many :users, through: :messages
end
```

```
class Message < ActiveRecord
  belongs_to :user
  belongs_to :chat
end
```

# Ejemplo de asociaciones

#ChatDelProyecto

**Elena:**  
¿Cuándo es la reunión?

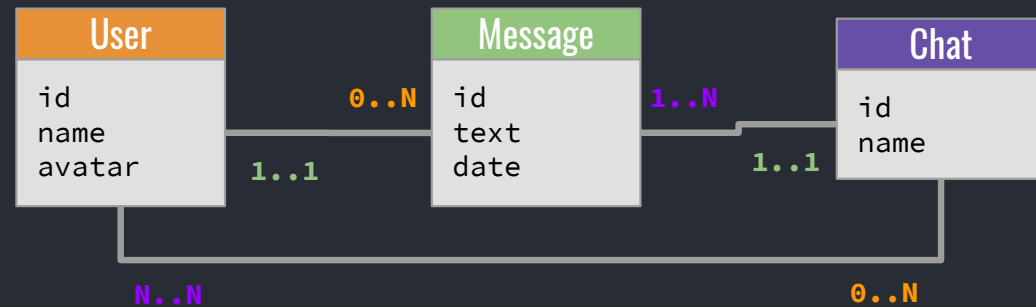
**Jorge:**  
Creo que a las 10

**Hector:**  
Ariel, ¿Sabes dónde nos juntamos?

**Ariel:**  
En el centro de alumnos!



Voy en camino...| send



@user1.messages  
@user1.chats  
@public\_chat.users  
@public\_chat.messages  
@a\_message.user  
@a\_message.chat

# Ejemplo de asociaciones

¿Cómo se relacionan los usuarios?

#ChatDelProyecto

**Elena:**  
¿Cuándo es la reunión?

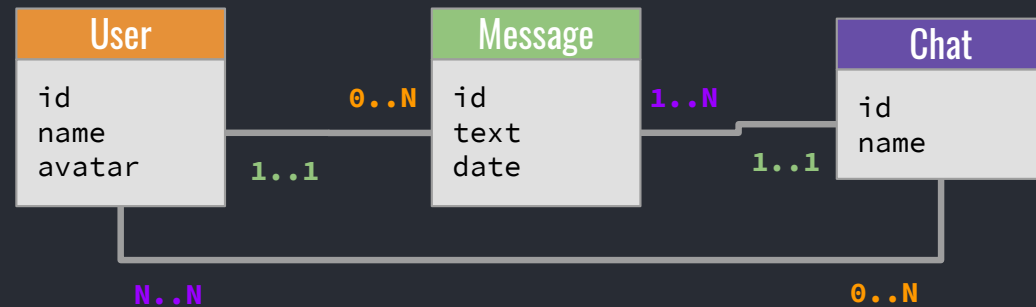
**Jorge:**  
Creo que a las 10

**Hector:**  
Ariel, ¿Sabes dónde nos juntamos?

**Ariel:**  
En el centro de alumnos!



Voy en camino...| send



```
@user1.chats.each do |chat|
  chat.users.each do |user|
    "#{@user1.name} talks to #{user}"
  end
end
```

# Ejemplo de asociaciones

## Mis contactos



Elena



Jorge



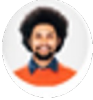


Hector

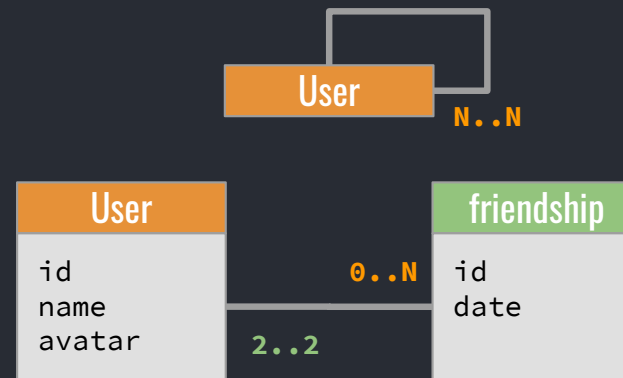
¿Cómo se relacionan los usuarios?



# Ejemplo de asociaciones

Mis contactos	
	Elena
	Jorge
	Hector

¿Cómo se relacionan los usuarios?






```
class User < ApplicationRecord
  has_many :friendships
  has_many :friends, through: :friendships
end
```




```
class Friendship < ApplicationRecord
  belongs_to :user
  belongs_to :friend, class_name: "User"
end
```

# Ejemplo de asociaciones

Contactos de Ariel

Mis contactos		
	Elena	⊗
	Jorge	⊗
	Hector	⊗
		⊕

Contactos de Elena

Mis contactos		
	Ariel	⊗
	Jorge	⊗
	Hector	⊗
		⊕



Friendship

user_id	friend_id






# Ejemplo de asociaciones

Contactos de Ariel

Mis contactos		
	Jorge	⊗
	Hector	⊗
		⊕

Contactos de Elena







Mis contactos		
	Ariel	⊗
	Jorge	⊗
	Hector	⊗
		⊕

Friendship




user_id	friend_id
idAriel	idElena

# Ejemplo de asociaciones

Aplicación de Ariel

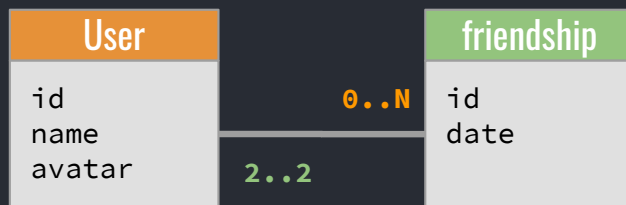
Agregar amigos		
	Jorge	
	Hector	
	Elena	

Aplicación de Elena

Invitaciones	
	Ariel quiere ser tu amiga
	

# Ejemplo de asociaciones

Relación bi-direccional



```
class User < ApplicationRecord
  has_many :friendships
  has_many :friends, through: :friendships
end
```

```
class Friendship < ApplicationRecord
  belongs_to :user
  belongs_to :friend, class_name: "User"
end
```

Users

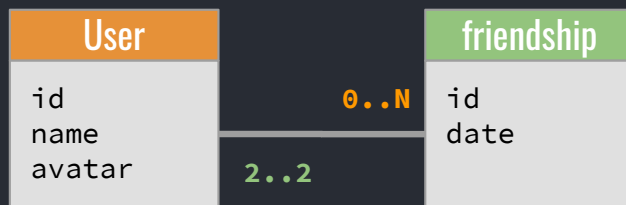
id	name
1	Ariel
2	Elena
3	Hector

Friendship

user_id	friend_id
1	2
...	...
2	1

# Ejemplo de asociaciones

Relación bi-direccional



```
class User < ApplicationRecord
  has_many :friendships
  has_many :friends, through: :friendships
end
```

```
class Friendship < ApplicationRecord
  belongs_to :user
  belongs_to :friend, class_name: "User"
end
```

Users

id	name
1	Ariel
2	Elena
3	Hector

Friendship

user_id	friend_id
1	2
...	...
3	4