**Conceito de ORM no Ruby**

1. **Mapeamento de Tabelas para Classes**: Em ORM, cada tabela no banco de dados é representada por uma classe no seu código Ruby. Cada linha da tabela é uma instância dessa classe, e cada coluna é um atributo.
2. **Abstração das Consultas SQL**: Em vez de escrever SQL diretamente, você usa métodos Ruby para criar consultas, o que torna o código mais limpo e menos propenso a erros.
3. **Associações**: ORM facilita o gerenciamento das relações entre tabelas, como "one-to-many" e "many-to-many", através de declarações simples.
4. **Validações**: ORM permite definir regras de validação para os dados antes deles serem salvos no banco.

**ActiveRecord no Ruby on Rails**

O ActiveRecord é o ORM padrão do Ruby on Rails e oferece uma interface poderosa para interagir com bancos de dados. Vamos ver como isso funciona com alguns exemplos:

**Exemplo de Configuração de Modelo**

Considere uma aplicação com um modelo Post que mapeia para uma tabela posts no banco de dados.

ruby

class Post < ApplicationRecord

# Validações

validates :title, presence: true

validates :content, length: { minimum: 10 }

# Associações

has\_many :comments

end

**Criação de Registros**

Você pode criar registros no banco de dados usando métodos como new e create:

ruby

# Criar um novo post (não salva no banco)

post = Post.new(title: "Meu Primeiro Post", content: "O conteúdo deste post é fascinante.")

# Salvar no banco de dados

post.save

# Criar e salvar de uma só vez:

Post.create(title: "Outro Post", content: "Este é outro exemplo de conteúdo de post.")

**Consulta de Registros**

ActiveRecord permite consultas simples e complexas sem precisar de SQL:

ruby

# Encontrar um post por ID

post = Post.find(1)

# Encontrar todos os posts

posts = Post.all

# Consultar com condições

posts\_com\_titulo = Post.where(title: "Meu Primeiro Post")

# Ordernação

posts\_ordenados = Post.order(created\_at: :desc)

**Atualização de Registros**

Atualizar registros também é direto com ActiveRecord:

ruby

post = Post.find(1)

post.update(title: "Título Atualizado")

**Exclusão de Registros**

Excluir registros é simples:

ruby

post = Post.find(1)

post.destroy

**Associações**

ActiveRecord torna o trabalho com associações muito fácil:

ruby

# Supondo que um post tem muitos comentários

class Comment < ApplicationRecord

belongs\_to :post

end

# Criar um comentário para um post

post = Post.find(1)

post.comments.create(content: "Este é um ótimo post!")

# Acessar comentários de um post

comentarios = post.comments

**Vantagens do ORM**

* **Produtividade**: Reduz a quantidade de código necessário para operações comuns de banco de dados.
* **Segurança**: Minimiza a configuração manual de SQL, diminuindo o risco de injeção de SQL.
* **Portabilidade**: Facilita a troca do sistema de banco de dados sem alterar o código.