



3. Rails Básico

g. A primeira Migration

- i. Forma “mágica” do rails de resolver um velho problema
 - 1. Sistema pronto a 3 anos e é necessário fazer uma correção numa versão anterior à final
 - 2. Sem o esquema do banco versionado você tem um grande problema em mãos
- ii. Com migrations você tem todas as alterações feitas no banco de dados desde o primeiro dia de desenvolvimento
 - 1. É possível reconstruir o estado do banco para qualquer versão
- iii. Diversas tarefas do rake para lidar com migrations
 - 1. db:migrate: executa todas as migrações ainda não executadas
- iv. Rails mantém uma tabela ‘schema_migrations’
 - 1. Armazena os nomes de todas as migrations já executadas
 - a. Novo no rails 2.1
 - b. Antes só salvava o número da última migração executada
 - i. Problemas se mais de um desenvolvedor criasse migrations ao mesmo tempo

v. Código de uma migration

- 1. db/migrate/<numeros>_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.string :login
      t.string :password
      t.string :name
      t.string :email
      t.timestamps
    end
  end
  def self.down
    drop_table :users
  end
end
```

vi. Toda migration estende ActiveRecord::Migration

vii. Toda migration tem os métodos up e down

- 1. Up: criar esquemas e evoluir o banco
- 2. Down: desfazer as alterações feitas no up
 - a. ActiveRecord::IrreversibleMigration: Exceção no caso da operação não poder ser desfeita



- viii. Métodos da classe ActiveRecord::Migration
 1. create_table(nome, opções): cria uma tabela e a passa como parâmetro para dentro do bloco
 2. drop_table(nome): remove uma tabela
 3. rename_table(nome_antigo, nome_novo): renomear uma tabela
 4. add_column(tabela, campo, tipo, opções): cria uma coluna em uma tabela existente
 5. rename_column(tabela, campo, novo_nome_campo): altera o nome de um campo de uma tabela
 6. change_column(tabela, campo, tipo, opções): altera uma coluna, pode-se mudar o tipo ou outras restrições
 7. remove_column(tabela, campo): remove um campo
 8. add_index(tabela, coluna, opções): cria um índice
 9. remove_index(tabela, colunas): remove um índice
 10. change_table(nome, opções): altera diversos aspectos de uma tabela de uma só vez
- ix. Opções para o create_table
 1. :id: informa se deve ser criado um id automático (padrão true)
 2. :primary_key: nome da coluna que vai representar a chave primária caso o :id seja configurado como false
 3. :temporary: cria uma tabela temporária
 4. :force: apaga a tabela antes de criar
- x. O parâmetro passado para o bloco tem os seguintes métodos
 1. column(nome, tipo, opções): cria um campo na tabela, maior parte dos outros são derivados deste
 2. timestamps: criacolunascreated_at e updated_at
 3. string(nome, opções): column(nome, :string, opções)
 4. text(nome, opções): column(nome, :text, opções)
 5. integer(nome, opções): column(nome, :integer, opções)
 6. float(nome, opções): column(nome, :float, opções)
 7. decimal(nome, opções): column(nome, :decimal, opções)
 8. datetime(nome, opções): column(nome, :datetime, opções)
 9. timestamp(nom, opções): column(nom, :timestamp, opções)
 10. time(nome, opções): column(nome, :time, opções)
 11. date(nome, opções): column(nome, :date, opções)
 12. binary(nome, opções): column(nome, :binary, opções)
 13. boolean(nome, opções): column(nome, :boolean, opções)
 14. index(nome, opções): add_index(tabela, colunas, opções)
 15. references(nome, opções): column(nome_id, :integer, opções)
 16. belongs_to(nome, opções): atalho para references



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

- xi. Tipos de dados
 - 1. `:primary_key`: cria uma chave primária
 - 2. outros: `:string`, `:text`, `:integer`, `:float`, `:decimal`, `:datetime`, `:timestamp`, `:time`, `:date`, `:binary`, `:boolean`
- xii. Opções
 - 1. `:limit`: tamanho do campo (`:string`, `:text`, `:binary` e `:integer`)
 - 2. `:default`: valor padrão para o campo
 - 3. `:null`: informa se aceita valores nulos
 - 4. `:precision`: precisão para o campo do tipo `:decimal`
 - 5. `:scale`: escala para o campo do tipo `:decimal`
- xiii. Os parâmetros `nome:tipo` passados para o scaffold são para a criação da migration
- xiv. Padrões
 - 1. Nomes de entidades sempre em inglês
 - 2. Nomes de tabelas serão nomes de entidades no plural
 - 3. Campos que referenciam outras tabelas tem o formato `<nome_entidade>_id`
- h. O primeiro model
 - i. Representa uma tabela no banco
 - ii. Descendem da classe `ActiveRecord::Base`
 - iii. Lugar certo para escrever regras de negócios e validações
 - iv. Código gerado pelo scaffold
 - 1.

```
class User < ActiveRecord::Base
  end
```
 - v. Apenas a declaração da classe
 - 1. Por padrão usa o CamelCase
 - a. Tudo junto com primeiras letras maiúsculas
 - vi. Validações disponíveis
 - 1. `validates_exclusion_of(campos)`: contrário de `validates_inclusion_of`
 - 2. `validates_format_of(campos, :with => /.../)`: valida o texto contra a expressão regular
 - 3. `validates_inclusion_of(campos, :in => [])`: verifica se o valor do campo está dentro do array
 - 4. `validates_length_of(campos, opções)`: comprimento do texto, usando as seguintes opções:
 - a. `:minimum`: tamanho mínimo
 - b. `:maximum`: tamanho máximo
 - c. `:is`: tamanho exato
 - d. `:within`: usando um intervalo
 - e. `:allow_nil`: pode não ser informado



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

- f. `:allow_blank`: pode ser uma string vazia
- 5. `validates_numericality_of(campos)`: verifica se é número
- 6. `validates_presence_of(campos)`: verifica se foi preenchido
- 7. `validates_size_of(campos)`: alias para `validates_length_of`
- 8. `validates_uniqueness_of(campos)`: verifica se é único
- vii. Adicionaremos algumas validações ao model user
 1.

```
class User < ActiveRecord::Base
  validates_presence_of :login, :password, :name, :email
  validates_length_of :login, :in => 5..100
  validates_length_of :password, :in => 5..50
  validates_format_of :email, :with => /\A([\^@\s]+)@((?[-a-z0-9]+\.)+[a-z]{2,})\Z/
  validates_uniqueness_of :login
  validates_uniqueness_of :email
end
```
 2. Todos os campos são obrigatórios
 3. Login precisa ter entre 5 e 100 caracteres
 4. Senha precisa ter entre 5 e 50 caracteres
 5. Expressão regular valida o formato de email
 6. Login e email precisam ser únicos
 7. Mensagens de erro em inglês
 - a. Campo `:message` em cada validação
- viii. Não precisa parar o servidor para verificar as alterações
- i. O primeiro controlador
- j. Métodos básicos do ActiveRecord
- k. Recebendo parâmetros nos controladores
- l. Respondendo a requisições
- m. As primeiras views
- n. Precisando de ajuda para limpar o código das views?
- o. Configuração de rotas, um nome bonito para URLs
- p. Continuando o desenvolvimento
- q. Conferindo as migrations geradas
- r. Escrevendo modelos
- s. Gerando todo o código
- t. Um pouco de segurança na aplicação
- u. Um layout menos confuso para a aplicação
- v. Associando usuários a projetos
- w. Adicionando tipos de tarefas a um projeto
- x. Cadastro das horas trabalhadas
- y. Um relatório para a aplicação
- z. Limpando um pouco o código e se livrando um pouco do “inglês”