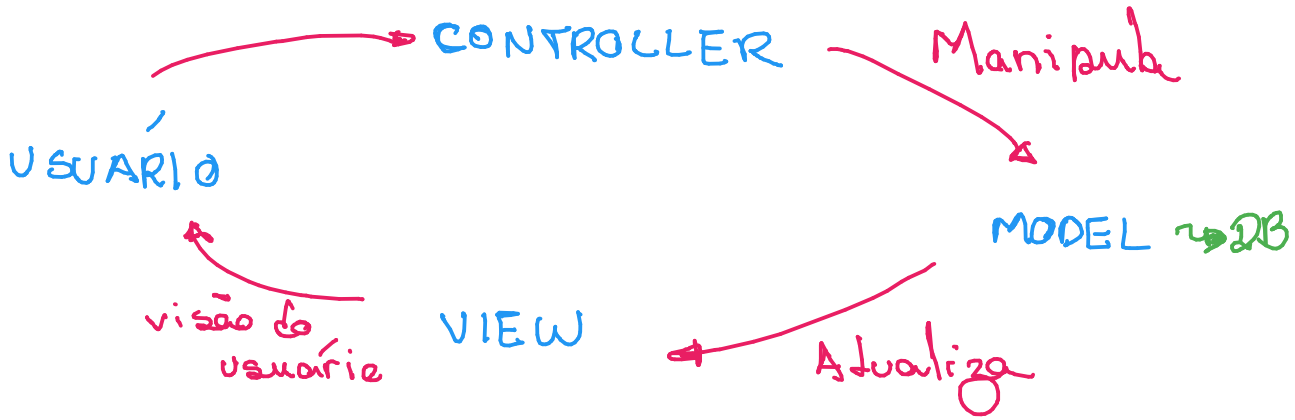
Resumo básico de Ruby On Rails

Autor: Paulo Akira

2018.2

Aula 1 : arquitetura MVC

→ MVC :



Criamos um exemplo de aplicação (rails new appexample).
O "rails" coloca na aplicação criada um servidor simples (PUMA) para que a aplicação possa ser testada.

↳ acesso : localhost : 3000

Aula 2 : pastas na aplicação

→ Pastas de um projeto rails

As pastas do rails são iguais em todos os projetos rails! Facilita quando temos projetos feitos em equipes.

- app : onde se localiza o view, model e controler
- bin : contém scripts que irão inicializar a aplicação (bundle, rails, rake, setup, update)
- config : contém as configurações (idiomas, ambientes, inicializadores, ...)
- dB : banco de dados (podemos trabalhar com SQL, SQLlite, etc.)
- lib : módulos estendidos da aplicação

→ log: gera arquivos com informações do que está acontecendo na aplicação
↳ usar para debugar!

→ public: única pasta disponível para o usuário na versão final da apl.
↳ acesso feito pela web! (e.g. criar um arquivo html do public e testar).

→ test: simular testes da aplicação

→ tmp: todos os arquivos temporários que a aplicação necessita para rodar ficam nesta pasta (cache, socket, etc.)

→ vendor: pasta destinada a terceiros (por exemplo, caso esteja usando um plugin)

Obs: Gemfile contém todos as gems utilizadas no projeto! Podemos adicionar novas gems através deste arquivo.

Aula 3: Linguagem Ruby

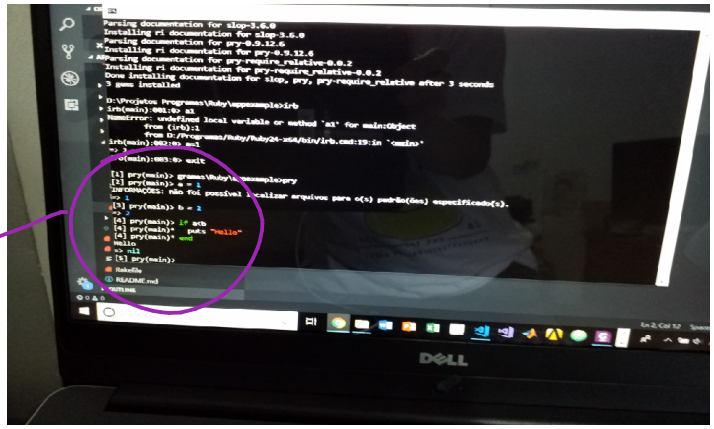
Ruby é uma linguagem interpretada e orientada a objetos!

puts → mostra a string

Obs: testar arquivos ruby nativos dentro da pasta tmp do app example.

Podemo usar um ambiente rápido para interpretar comandos em Ruby. No caso vamos usar o Interactive Ruby Shell (IRB) que é um REPL (read-eval-print loop → faz a leitura, avalia e mostra o resultado em um loop).

Uma alternativa ao IRB é Pry que nada mais é que um IRB melhorado. O Pry é uma gem.



Utilização do Pry!

• array ⇝ a = [ ] ⟶ outra forma de iniciar o array é Array.new

a.push(89) ⇝ a = [89]

a.push("ruby") ⇝ a = [89, "ruby"] a.size ⇝ 2

b = %w(Paulo Akira) ⇝ b = ["Paulo", "Akira"]

• hash : é uma coleção de pares chaves-valor. Os índices são definidos de forma arbitrária.

meu_hash = {"abc" => "rails", "def" => "ruby"}

chave valor chave valor

meu_hash ["abc"] ⇝ "rails"

Podem enxergar arrays como uma classe que possui métodos (.size, .eql, .each

Podemos usar parâmetros em ruby :

array.size.eql?(5) ⇝ false

parâmetro é um determinado valor associado a uma saída

Também podemos utilizar blocos :

O .each pega um elemento do array de cada vez. O do associa este elemento a variável elemento (um elemento do array de cada vez!)

```
a.each do |elemento|
    puts elemento
end
```

do e end formam um bloco de algoritmo

⇝ 89
"ruby"

Pega o 89 associa ao "elemento" e mostra o conteúdo de "elemento". Faz a mesma coisa com o próximo elemento ("ruby").

→ Classes no Ruby :

Como utilizar uma classe em Ruby ?

```
class Pessoa
  def falar
    puts "Estou falando"
  end
end
```

Atleta herda todos os métodos e atributos de pessoa !

```
class Atleta < Pessoa
  def correr
    puts "Correndo ..."
  end
end
```

herança !

O que é um módulo ?

É uma coleção de métodos e constantes.

"É uma espécie classe"

```
module Configurações
    NOME_DO_SISTEMA = "Sistema da academia"
    VERSÃO = "1.2.4.5"
    def calcular
        puts "O resultado final é tanto..."
    end
end
```

} Constantes!

Configurações :: NOME_DO_SISTEMA → "Sistema da Academia"

O Ruby não permite heranças múltiplas. Para solucionar este problema utilizaremos os módulos (Mixin).

```
module Correio
    def enviar
        puts "Enviado"
    end
end
```

Mixin →

```
class MeuMixin
    include Configurações
    include Correio
end
```

Aula 4: Banco de Dados e Rake

Para usar o database com o MySQL:

```
rails new appexample --database=mysql
```

Teremos três ambientes diferentes (desenvolvimento, produção e teste). A cada ambiente, eu terei um banco de dados diferente: development.db, production.db e test.db

development.db: banco de dados "falso" para desenvolver o aplicativo

production.db: banco de dados real

test.db: banco de dados "falso" para realizar testes na aplicação.

Como mudar o ambiente? rails server -e
- development (default)
- production
- test

→ Rake

Rake é uma ferramenta para gerenciar tarefas. → podemos criar nossas próprias tarefas rake!

↳ para ver todos os comandos do Rake: rake -T

Vamos utilizar o "rake db:create" para criar um DB com as configurações

especificados em "config/database.yml".

↳ quando utilizo o sqlite3, "development database" já vem criado.

CRUD - Create Read Update Delete

Quatro operações básicas de um banco de dados!

→ Scaffold (andaime) → o Controller e a View gerados são referentes ao model

Refere-se a autogeração de um mo, uma view e um controller usado para um modelo de banco de dados determinado pelo projetista.

CRUDs são gerados!

Como utilizar o scaffold? rails generate scaffold customer name:string ...

nome do modelo no singular

pode ser o nome que eu quiser! (de preferência em inglês)

campos a serem cadastrados (CRUDs a serem gerados)

Novidades nas pastas após a utilização do scaffold:

customers no plural refere-se as tabelas

- em app/controller surgiu um customers_controller.rb

CRUD está aqui (como métodos da classe Customers)

- em app/models surgiu um customer.rb

no singular

Aula 6: Migrations

Ao criarmos uma tabela o rails cria, por convenção, um campo chamado id para ser a chave primária.

t.timestamp → este campo se tornará dois campos na tabela: "created_at" e "updated_at"

rails irá atualizar com data e hora:

quando o registro for criado

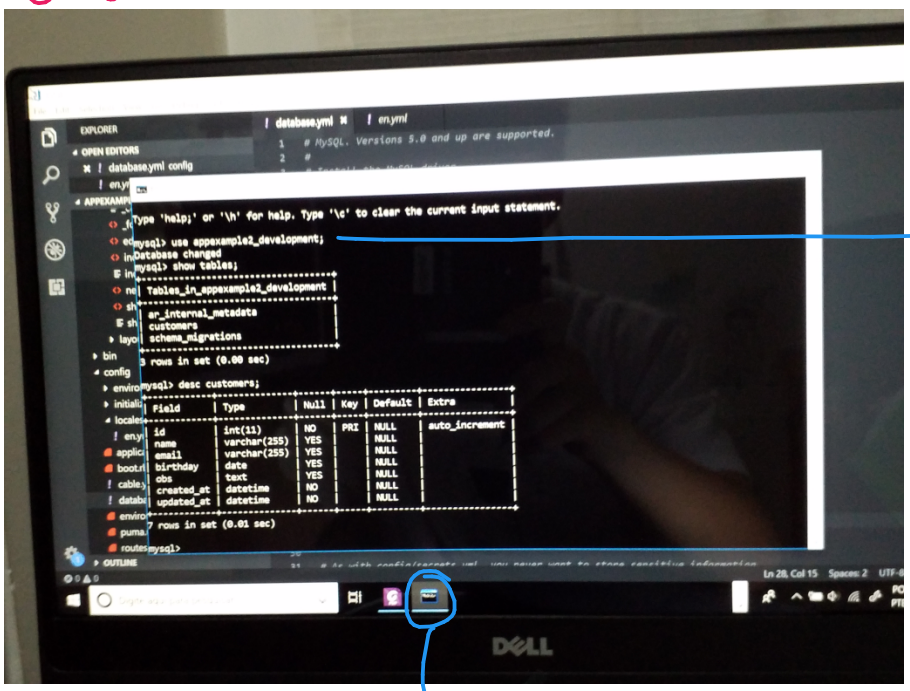
quando o registro for atualizado

Para que servem os migrations?

Utilizamos o migration para atualizar o banco de dados inserindo ou removendo novas colunas. O rails permite que estas mudanças sejam feitas em Ruby, evitando que o desenvolvedor tenha que conhecer detalhes da linguagem de banco de dados (SQLite, MySQL, etc.) ficando a cargo do rails.

↳ Não preciso ir no db e altera-lo manualmente!

Dentro das da basta db/migrate temos o arquivo schema.rb que controla as migrações feitas!



usar no MySQL command line xxx_development;

Sempre que usar o SQL no rails lembrar de colocar a senha no config!

usando o MySQL como database!

Para acessar no servidor: localhost 3000/customers

Aula 6: Embedded Ruby (eRuby) e Views

eRuby → sistema de templates que permite utilizar ruby em um documento de texto (e.g. HTML)

↳ utilizamos a extensão erb para imbutir o ruby no HTML. O arquivo fica xxx.html.erb.

↳ essa "mistura" será particularmente interessante nas Views

O eRuby possui tags especiais para escrever código Ruby no HTML.

Obs: o arquivo index (na View) é um arquivo HTML que é responsável pelo que vemos logo que abrimos a aplicação.

Na index:

<% %> → serve para códigos Ruby que não terão saída para o HTML, ou seja, que não irão modificar o HTML.

<%= %> → logo utilizado quando queremos realizar modificações no HTML.

<% = -%> → quando adicionamos o "-" tiramos a quebra de linha no HTML

<#%= %> → só serve para fazer comentários no próprio código. (não vai aparecer nada no HTML

Interpolação de Variáveis: utilizado para colocar variáveis em um texto que já é Ruby. #{ }

Exemplo:

```
<% a="Ruby" %>
<% b="Rails" %>
<%= "teste #{a} on #{b} teste" %>
```

→ já é um texto em Ruby

Aula 7: Active Record

Nome do model criado com o Scaffold.

Na pasta model, temos o xxx.rb que é o model (lembrando que logo que criamos um model com o scaffold, temos que mandar a migração para o DB de forma a criar as tabelas).

xxx = costumer

Dentro do arquivo costumer.rb (model):

```
Model < Active Record::Base
```

É a classe nova criada — É um módulo — é um atributo do módulo

é uma arquitetura!

O "Model" é herdeiro de Active Record (é uma dos frameworks que o Rails é criada)

O Active Record :: Base é o responsável por correlacionar o banco de dados real com a classe Model, ou seja, ele verifica que existe uma tabela associada ao modelo no banco de dados real e mapeia tudo que tem lá para o Rails.

↳ se temos um "Model" chamado "Customer", automaticamente o Active Record vai saber que existe uma tabela chamada "Customers" no DB real. Isso é feito só devido ao fato do Model herdar o Active Record :: Base.

O Rails Console é um ambiente que carrega todas as classes que tenho disponível no meu projeto, além das classes do irb.

Como utilizar o rails console? No prompt de cmd: rails console ou rails c.

Dessa forma, consigo acessar neste ambiente a tabela através da classe Model (digitamos comandos em Ruby e comandos em SQL são processados na tabela)

Exemplo: a = Customer.first ↳ pega o primeiro elemento da tabela (com id = 1)

b = Customer.all ↳ pega toda a tabela ("será um vetor")

Aula 8: Controllers

O controller será onde colocaremos as ações do nosso sistema

```
def action
  # code
end
```

} arquivo xxx_controller.rb na pasta controller

Convenção do Rails: ↳ basta eu criar a ação e a View com mesmo nome, que o controller automaticamente associará a View a ação

↳ para cada ação pode haver uma View de mesmo nome.

→ Variáveis de instância:

A utilização do @ faz com que a variável fique disponível tanto para o controller quanto para a View.

Obs*: se tenho um model chamado "Customer", os arquivos associados a este model na View estará na pasta "customer" e o arquivo associado no controller será "customer_controller.rb".

Exemplo:

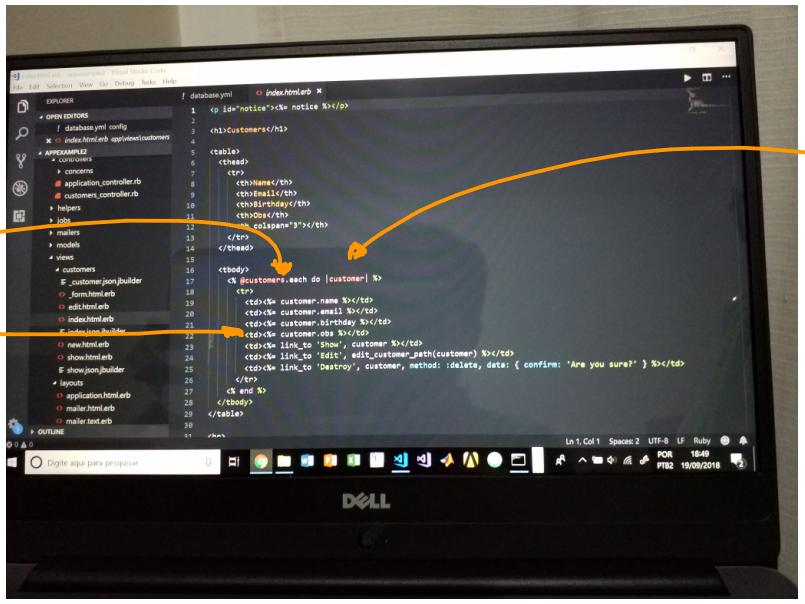
no customer_controller.rb:

```
def index
  @customers = Customer.all
end
```

pega toda a tabela coloca em um vetor!

na view/customer/index.html.erb:

@customers é um vetor!



elementos associados!

tag <td>: fonte normal e alinhamento a esquerda

→ Rotas:

Endereços digitados na URL podem ser tratados pelo rails de alguma forma. Logo que geramos o scaffold "customer" a primeira rota criada foi "customers"

Sempre que digito algo na URL, estou utilizando um verbo http. O verbo GET serve para eu pegar algo do meu servidor e o verbo POST para eu gravar algo no meu servidor (e.g. enviar um formulário para o servidor).

Como definir uma rota?

A View estará associada!

get 'nome_da_rota' => 'nome_do_controller#nome_da_action'

indica para onde a rota está apontando

Associa o que foi escrito na URL com o que deve ser processado pela ação do controller.

Podemos criar uma rota padrão (localhost:3000 vai sair nessa rota) fazendo

root 'nome_do_controller#nome_da_action'

→ Helpers

Proveniente da framework Action View :: Helpers

Helpers são comandos rails para simplificar a codificação.

Gerando um link com o helper link_to:

utilizado no xxx.html.erb

<%= link_to "Nome do link", PATH %>

e.g. /customers (rota para o modelo)

ou

localhost:3000/rails/info/routes → customers_path (comando rails)

→ REST ou RESTFUL

O REST é um conjunto de princípios que visa otimizar a semântica do protocolo http. Apenas os verbos GET e POST não estavam representando bem todas as ações que são realizadas na internet. Dessa forma, foi proposto a criação de mais dois métodos. Podemos fazer uma comparação com o método CRUD do banco de dados.

CRUD		HTTP
Create	→	POST
Read	→	GET
Update	→	PUT
Delete	→	DELETE

} Métodos Novos

No rails o REST está explicito nas rotas onde temos resources :xxx

e.g. customers

O "resources" cria para o "customer" rotas com os novos métodos (ou verbos)

↳ Lembrando que as rotas estão relacionadas com os controllers. Logo o verbo http estará relacionado com uma ação do controller.

→ Filtros

Filtros são métodos que são rodados antes ou depois da ação de um controller. Também pode ser rodado antes e depois da ação.

before_action → antes de qualquer ação ser executada no controller através de uma requisição feita pelo browser, será rodado o que está no before_action.

↳ a grande vantagem é economizar código

Como usar o "before_action" e/ou "after_action"?

```
befor_action :nome_da_ação, only [:ação 1, :ação 2, ..., :ação n]
```

Só vai executar para estas ações!

Obs: outro modo de economizar código é utilizando os "partials".
Partials são Views que utilizam um arquivo em comum. Dessa forma, caso necessitemos realizar alguma alteração, podemos fazer neste arquivo e não em cada View.

→ Helpers II:

- form_for: helper que ajuda a criar formulários → cria um elemento vazio com os dados da tabela para ser preenchido com dados a serem recebidos
- text_field: cria um input de dados
- date_select: cria um input de datas com botões
- text_area: cria uma área para texto

→ Normalmente utilizo form_for (@customer) do f|f, ou seja, passo todos os parâmetros de customer para o elemento f (form builder)

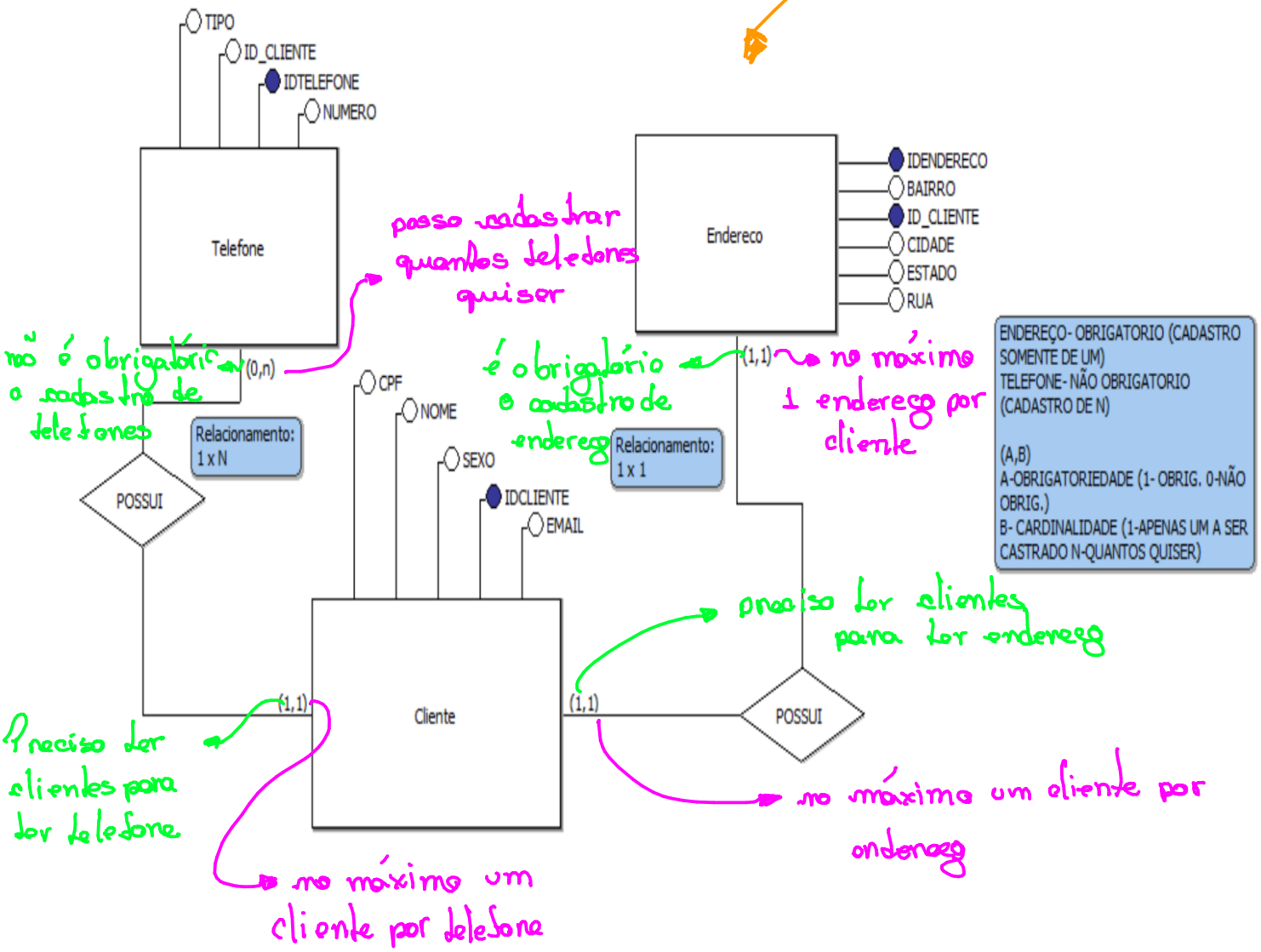
→ Params:

São os parâmetros que chegam pela URL ou pelo verbo POST.

Apêndice: Banco de Dados

→ Relacionamento entre tabelas

É apenas o modelo lógico!



→ No modelo físico:

O que é a chave primária?

É um campo que identifica cada registro como sendo único. Todo elemento de uma tabela terá chaves primárias.

↳ Lembrando que o Rails cria este campo por padrão quando um modelo é criado.

O que é a chave estrangeira (Foreign Key)?

É a chave primária de uma tabela que vai até uma outra tabela de forma a associar os dados desta segunda tabela a primeira tabela.

→ em um relacionamento 1x1 a chave estrangeira fica na tabela mais fraca.

→ em um relacionamento 1xN, a chave estrangeira fica sempre na tabela N.

Exemplo:

→ A tabela telefones é que vai receber as chaves estrangeiras.

→ A tabela endereço é que vai receber as chaves estrangeiras.

é a chave primária de cliente!