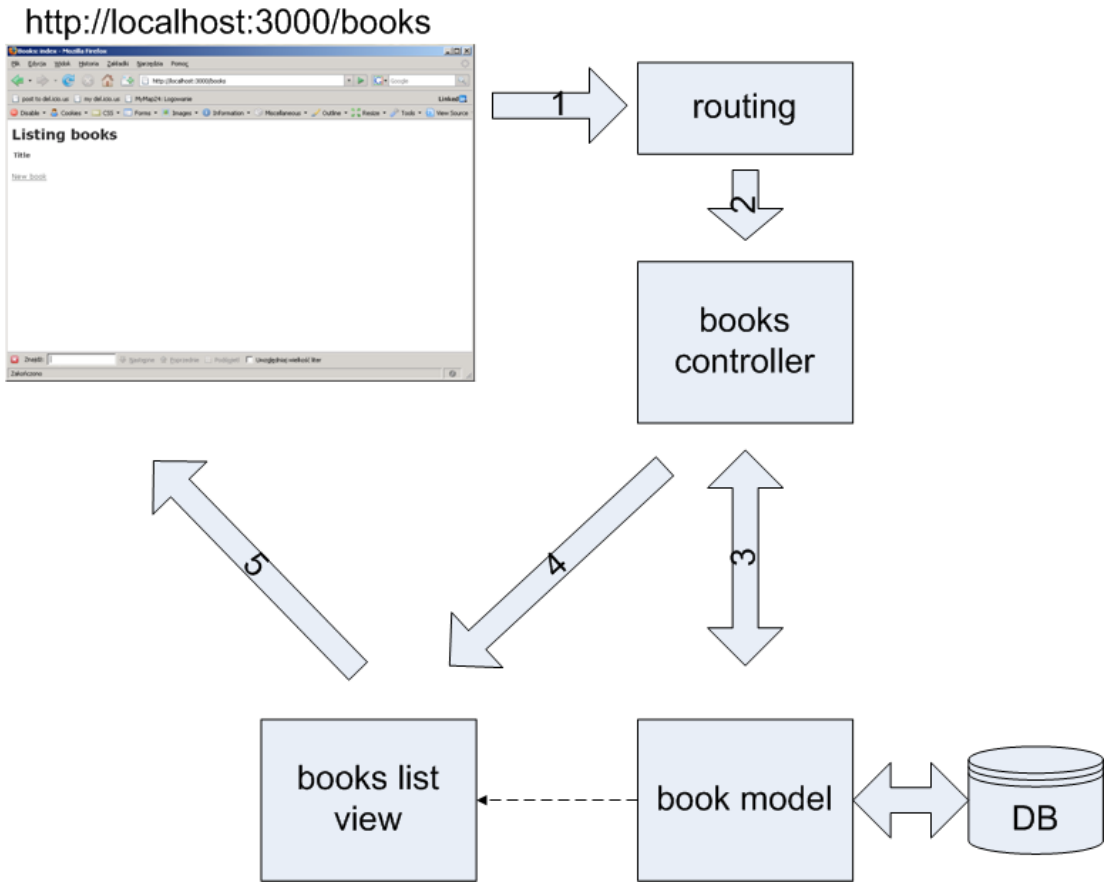
EPI: Interfejs Graficzny
Wykład nr 5
Warstwa modelu – `ActiveRecord`

dr inż. Aleksander Smywiński-Pohl

Elektroniczne Przetwarzanie Informacji
Konsultacje: czw. 14.00-15.30, pokój 3.211

Plan prezentacji

MVC w RoR


http://localhost:3000/books
Listing books
Title
New book
1
routing
2
books controller
3
4
5
books list view
book model
DB

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu
  - niektóre tabele odpowiadają **związkom** między klasami (dla zależności m-do-n)

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu
  - niektóre tabele odpowiadają **związkom** między klasami (dla zależności m-do-n)
  - każdy rekord posiada podstawowy **klucz główny** `id`

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu
  - niektóre tabele odpowiadają **związkom** między klasami (dla zależności m-do-n)
  - każdy rekord posiada podstawowy **klucz główny** `id`
- **zapytania** – zapewniają obiektowy dostęp do danych zgromadzonych w bazie

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu
  - niektóre tabele odpowiadają **związkom** między klasami (dla zależności m-do-n)
  - każdy rekord posiada podstawowy **klucz główny** `id`
- **zapytania** – zapewniają obiektowy dostęp do danych zgromadzonych w bazie
- **migracje** – pozwalają modyfikować schemat bazy danych bez użycia SQL

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu
  - niektóre tabele odpowiadają **związkom** między klasami (dla zależności m-do-n)
  - każdy rekord posiada podstawowy **klucz główny** `id`
- **zapytania** – zapewniają obiektowy dostęp do danych zgromadzonych w bazie
- **migracje** – pozwalają modyfikować schemat bazy danych bez użycia SQL
- **walidacje** – pozwalają weryfikować poprawność danych przed ich zapisaniem

`ActiveRecord` – railsowy ORM

- implementacja wzorca *active record*:
  - **tabelom** bazy danych odpowiadają **klasy**
  - **wierszom** tabeli odpowiadają **obiekty**
  - **komórkom** tabeli odpowiadają **atrybuty** obiektu
  - niektóre tabele odpowiadają **związkom** między klasami (dla zależności m-do-n)
  - każdy rekord posiada podstawowy **klucz główny** `id`
- **zapytania** – zapewniają obiektowy dostęp do danych zgromadzonych w bazie
- **migracje** – pozwalają modyfikować schemat bazy danych bez użycia SQL
- **walidacje** – pozwalają weryfikować poprawność danych przed ich zapisaniem
- **callbacki** – pozwalają wykonywać określone zadania stowarzyszone z cykle życia obiektu

Sposób użycia

- definicja klasy

```
class Book < ActiveRecord::Base
end
```

Sposób użycia

- definicja klasy

```
class Book < ActiveRecord::Base
end
```

- utworzenie nowego rekordu

```
book = Book.new(title: "Dziady")
book.save
```

Sposób użycia

- definicja klasy

```
class Book < ActiveRecord::Base
end
```

- utworzenie nowego rekordu

```
book = Book.new(title: "Dziady")
book.save
```

- odczytanie rekordu

```
book = Book.find(1)
book.title          #=> 'Dziady'
```

Sposób użycia

- definicja klasy

```
class Book < ActiveRecord::Base
end
```

- utworzenie nowego rekordu

```
book = Book.new(title: "Dziady")
book.save
```

- odczytanie rekordu

```
book = Book.find(1)
book.title          #=> 'Dziady'
```

- modyfikacja rekordu

```
book.title = "Dziady IV"
book.save
# lub
book.update_attributes(title: "Dziady IV")
```

Sposób użycia

- definicja klasy

```
class Book < ActiveRecord::Base
end
```

- utworzenie nowego rekordu

```
book = Book.new(title: "Dziady")
book.save
```

- odczytanie rekordu

```
book = Book.find(1)
book.title          #=> 'Dziady'
```

- modyfikacja rekordu

```
book.title = "Dziady IV"
book.save
# lub
book.update_attributes(title: "Dziady IV")
```

- usunięcie rekordu

```
book.destroy
```

Konfiguracja bazy

Plik `config/database.yml`

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000
production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000
```

Generator modelu

```
rails generate model Book title:string
      invoke  active_record
      create    db/migrate/20111108182152_create_books.rb
      create    app/models/book.rb
      invoke    test_unit
      create      test/unit/book_test.rb
      create      test/fixtures/books.yml
```

Plan prezentacji

Proste zapytania

- za pomocą klucza głównego (uwaga!):

```
Book.find(1)
```

Proste zapytania

- za pomocą klucza głównego (uwaga!):

```
Book.find(1)
```

- jw. (wersja bezpieczna):

```
Book.find_by_id(1)
```

Proste zapytania

- za pomocą klucza głównego (uwaga!):

```
Book.find(1)
```

- jw. (wersja bezpieczna):

```
Book.find_by_id(1)
```

- kilka obiektów na raz

```
Book.find(1,2,5)
```

Proste zapytania

- za pomocą klucza głównego (uwaga!):

```
Book.find(1)
```

- jw. (wersja bezpieczna):

```
Book.find_by_id(1)
```

- kilka obiektów na raz

```
Book.find(1,2,5)
```

- pierwszy obiekt

```
Book.first
```

Proste zapytania

- za pomocą klucza głównego (uwaga!):

```
Book.find(1)
```

- jw. (wersja bezpieczna):

```
Book.find_by_id(1)
```

- kilka obiektów na raz

```
Book.find(1,2,5)
```

- pierwszy obiekt

```
Book.first
```

- ostatni obiekt

```
Book.last
```

Proste zapytania

- za pomocą klucza głównego (uwaga!):
  ```
  Book.find(1)
  ```
- jw. (wersja bezpieczna):
  ```
  Book.find_by_id(1)
  ```
- kilka obiektów na raz
  ```
  Book.find(1,2,5)
  ```
- pierwszy obiekt
  ```
  Book.first
  ```
- ostatni obiekt
  ```
  Book.last
  ```
- wszystkie obiekty
  ```
  Book.all
  ```

Złożone zapytania

- określona wartość atrybutu (zwraca obiekt lub `nil`):

```
Book.find_by_title('Ruby')
```

Złożone zapytania

- określona wartość atrybutu (zwraca obiekt lub `nil`):

```
Book.find_by_title('Ruby')
```

- z prostymi opcjami (zwraca tablicę):

```
Author.where('last_name=?',last_name_var).order('first_name ASC')
```

Złożone zapytania

- określona wartość atrybutu (zwraca obiekt lub `nil`):

```
Book.find_by_title('Ruby')
```

- z prostymi opcjami (zwraca tablicę):

```
Author.where('last_name=?',last_name_var).order('first_name ASC')
```

- z prostymi opcjami (zwraca obiekt lub `nil`):

```
Author.where('last_name=?',last_name_var).order('first_name ASC').
  first
```

Złożone zapytania

- określona wartość atrybutu (zwraca obiekt lub `nil`):

```
Book.find_by_title('Ruby')
```

- z prostymi opcjami (zwraca tablicę):

```
Author.where('last_name=?',last_name_var).order('first_name ASC')
```

- z prostymi opcjami (zwraca obiekt lub `nil`):

```
Author.where('last_name=?',last_name_var).order('first_name ASC').
  first
```

- z zakresem (zwraca tablicę):

```
Author.where(birth_date: (40.years.ago..30.years.ago)).
  order(:birth_date)
```

Złożone zapytania

- określona wartość atrybutu (zwraca obiekt lub `nil`):

```
Book.find_by_title('Ruby')
```

- z prostymi opcjami (zwraca tablicę):

```
Author.where('last_name=?',last_name_var).order('first_name ASC')
```

- z prostymi opcjami (zwraca obiekt lub `nil`):

```
Author.where('last_name=?',last_name_var).order('first_name ASC').
  first
```

- z zakresem (zwraca tablicę):

```
Author.where(birth_date: (40.years.ago..30.years.ago)).
  order(:birth_date)
```

- maksymalna liczba wyników (zwraca tablicę):

```
Author.limit(10).offset(20)
```

Zapytania na powiązanych obiektach

```
author = Author.first
author.books.order(:publish_date).first
author.books.where(publish_date: (10.years.ago..Time.now)).
  limit(3)
```

Zapytania na powiązanych obiektach

```
author = Author.first
author.books.order(:publish_date).first
author.books.where(publish_date: (10.years.ago..Time.now)).
  limit(3)

category = Category.find_by_name("Proza")
category.books.order('rating DESC').limit(10)
```

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników
- **first** – zwraca pierwszy wynik

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników
- **first** – zwraca pierwszy wynik
- **last** – zwraca ostatni wynik

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników
- **first** – zwraca pierwszy wynik
- **last** – zwraca ostatni wynik
- **limit** – określa maksymalny rozmiar wyniku

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników
- **first** – zwraca pierwszy wynik
- **last** – zwraca ostatni wynik
- **limit** – określa maksymalny rozmiar wyniku
- **offset** – określa przesunięcie wyniku względem początku

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników
- **first** – zwraca pierwszy wynik
- **last** – zwraca ostatni wynik
- **limit** – określa maksymalny rozmiar wyniku
- **offset** – określa przesunięcie wyniku względem początku
- **select** – zawęża listę zwróconych atrybutów

Opcje zapytań

- **where** – określa warunki zapytania, akceptuje tablicę zwykłą lub asocjacyjną
- **order** – określa kolejność wyników
- **first** – zwraca pierwszy wynik
- **last** – zwraca ostatni wynik
- **limit** – określa maksymalny rozmiar wyniku
- **offset** – określa przesunięcie wyniku względem początku
- **select** – zawęża listę zwróconych atrybutów
- **includes** – zachłanne ładowanie powiązanych obiektów

scope – nazwane zapytania

Dodając do klas makro scope:

```
class Author < Person
  scope :living, where(death_date: nil)
  scope :polish, where(nationality: "polski")
end
```

scope – nazwane zapytania

Dodając do klas makro scope:

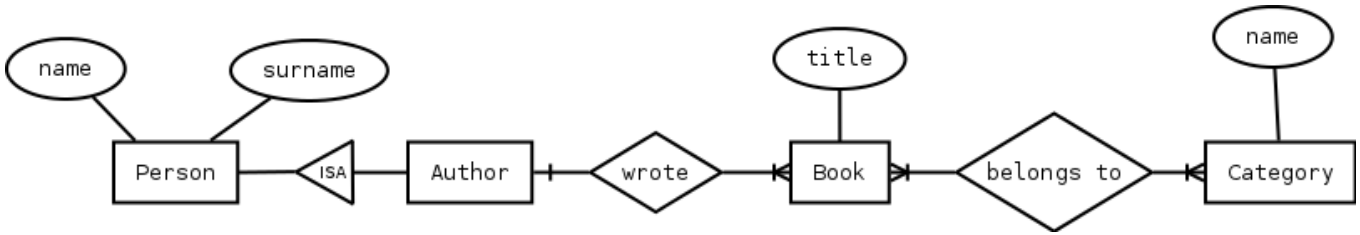
```
class Author < Person
  scope :living, where(death_date: nil)
  scope :polish, where(nationality: "polski")
end
```

możemy pisać bardziej czytelne zapytania:

```
Author.living.polish.order("last_name, first_name")
# zamiast
Author.where(death_date: nil).
  where(nationality: "polski").order("last_name, first_name")
```

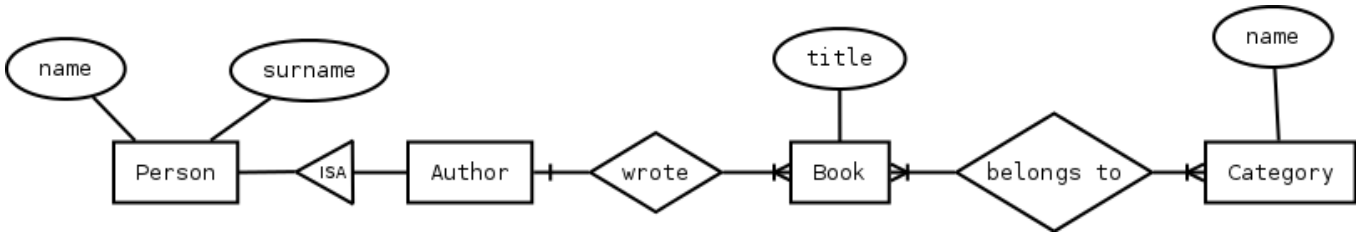
Plan prezentacji

Model konceptualny a model obiektowy




```
class Person < ActiveRecord::Base
end
```

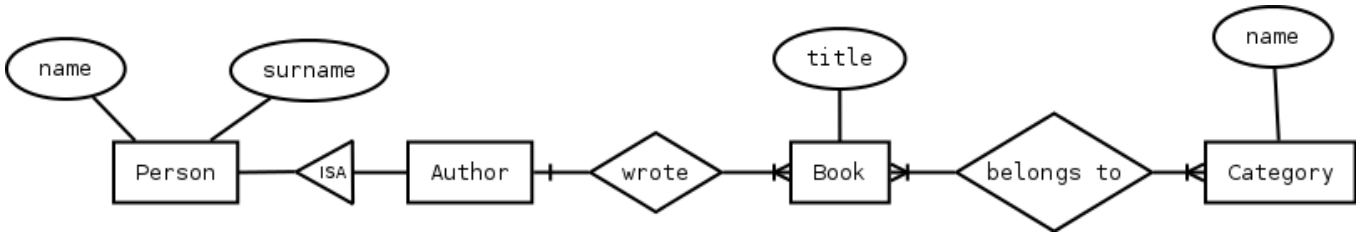
Model konceptualny a model obiektowy




```
class Person < ActiveRecord::Base
end

class Author < Person
  has_many :books
end
```

Model konceptualny a model obiektowy

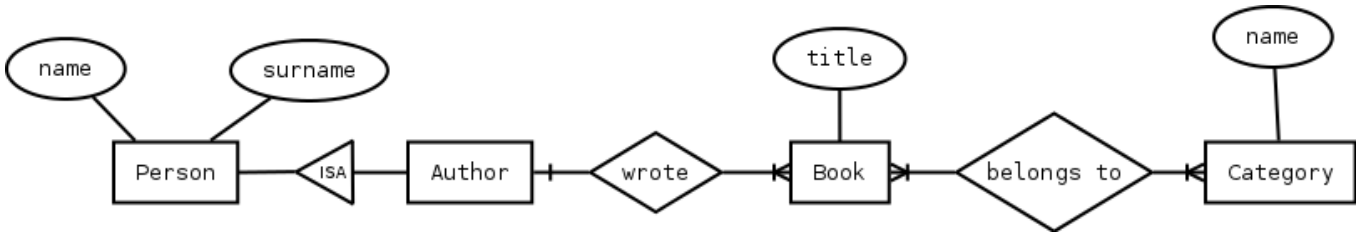



```
class Person < ActiveRecord::Base
end

class Author < Person
  has_many :books
end

class Book < ActiveRecord::Base
  belongs_to :author
  has_and_belongs_to_many :categories
end
```

Model konceptualny a model obiektowy




```
class Person < ActiveRecord::Base
end

class Author < Person
  has_many :books
end

class Book < ActiveRecord::Base
  belongs_to :author
  has_and_belongs_to_many :categories
end

class Category < ActiveRecord::Base
  has_and_belongs_to_many :books
end
```

Związek jeden-do-wiele po stronie wiele

```
class Book < ActiveRecord::Base
  belongs_to :author
end
```

Związek jeden-do-wiele po stronie wiele

```
class Book < ActiveRecord::Base
  belongs_to :author
end
```

Obiekty klasy Book posiadają teraz następujące dodatkowe metody:

```
book.author
book.author=(author)
book.author?(some_author)
book.author.nil?
book.build_author(...)
book.create_author(...)
```

Związek jeden-do-wiele po stronie jeden

```ruby
class Author < ActiveRecord::Base
  has_many :books
end
```

Związek jeden-do-wiele po stronie jeden

```
class Author < ActiveRecord::Base
  has_many :books
end
```

Obiekty klasy `Author` posiadają teraz następujące dodatkowe metody:

```
author.books
author.books<<(book)
author.books.delete(book)
author.books=[book1,book2,...]
author.book_ids
author.book_ids=[id1,id2,...]
author.books.clear
author.books.empty?
author.books.size
author.books.find(...)
author.books.exists?(...)
author.books.build(...)
author.books.create(...)
```

Związek wiele-do-wiele

```ruby
class Category < ActiveRecord::Base
  has_and_belongs_to_many :books
end
```

Związek wiele-do-wiele

```
class Category < ActiveRecord::Base
  has_and_belongs_to_many :books
end
```

Obiekty klasy Category posiadają teraz następujące dodatkowe metody:

```
category.books
category.books<<(book)
category.books.delete(book)
category.books=[book1,book2,...]
category.book_ids
category.book_ids=[id1,id2,...]
category.books.clear
category.books.empty?
category.books.size
category.books.find(...)
category.books.exists?(...)
category.books.build(...)
category.books.create(...)
```

Konwencje bazy danych

1. każda tabela (z wyjątkiem złączeniowej) posiada sztuczny klucz główny o nazwie `id`

Konwencje bazy danych

1. każda tabela (z wyjątkiem złączeniowej) posiada sztuczny klucz główny o nazwie `id`
2. w bazie danych wszystkie nazwy tabel zaczynają się małą literą i są w liczbie mnogiej, np. `authors`, `books`, `categories`

Konwencje bazy danych

1. każda tabela (z wyjątkiem złączeniowej) posiada sztuczny klucz główny o nazwie `id`
2. w bazie danych wszystkie nazwy tabel zaczynają się małą literą i są w liczbie mnogiej, np. `authors`, `books`, `categories`
3. związek jeden-do-wiele:

Konwencje bazy danych

1. każda tabela (z wyjątkiem złączeniowej) posiada sztuczny klucz główny o nazwie `id`
2. w bazie danych wszystkie nazwy tabel zaczynają się małą literą i są w liczbie mnogiej, np. `authors`, `books`, `categories`
3. związek jeden-do-wiele:
   3.1 po stronie „jeden" nazwa związku w liczbie mnogiej, np. `author.books`, brak pola w bazie danych

Konwencje bazy danych

1. każda tabela (z wyjątkiem złączeniowej) posiada sztuczny klucz główny o nazwie `id`
2. w bazie danych wszystkie nazwy tabel zaczynają się małą literą i są w liczbie mnogiej, np. `authors`, `books`, `categories`
3. związek jeden-do-wiele:
   - 3.1 po stronie „jeden" nazwa związku w liczbie mnogiej, np. `author.books`, brak pola w bazie danych
   - 3.2 po stronie „wiele" nazwa związku w liczbie pojedynczej, np. `book.author`, nazwa pola w bazie: nazwa powiązanego modelu + `_id`, np. `author_id`

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce
5. relacja IS-A:

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce
5. relacja IS-A:
   - 5.1 realizowana jest poprzez mechanizm dziedziczenia

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce
5. relacja IS-A:
   - 5.1 realizowana jest poprzez mechanizm dziedziczenia
   - 5.2 wszystkie klasy hierarchii mają przypisaną jedną tabelę (STI)

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce
5. relacja IS-A:
   - 5.1 realizowana jest poprzez mechanizm dziedziczenia
   - 5.2 wszystkie klasy hierarchii mają przypisaną jedną tabelę (STI)
   - 5.3 nazwa tabeli tworzona jest na podstawie klasy bazowej

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce
5. relacja IS-A:
   - 5.1 realizowana jest poprzez mechanizm dziedziczenia
   - 5.2 wszystkie klasy hierarchii mają przypisaną jedną tabelę (STI)
   - 5.3 nazwa tabeli tworzona jest na podstawie klasy bazowej
   - 5.4 typ obiektu zapamiętywany jest w polu tekstowym `type`

Konwencje bazy danych – cd.

4. związek wiele-do-wiele:
   - 4.1 tabela złączeniowa: nazwa pierwszego modelu w liczbie mnogiej + nazwa drugiego modelu w liczbie mnogiej, kolejność nazw modeli zgodna z porządkiem alfabetycznym, np. `books_categories`
   - 4.2 pola tabeli złączeniowej: nazwa pierwszego modelu + `_id`, nazwa drugiego modelu + `_id`, np. `book_id`, `category_id`
   - 4.3 tabela złączeniowa nie musi posiadać klucza `id`, powinna posiadać klucz unikalny obejmujący oba klucze obce
5. relacja IS-A:
   - 5.1 realizowana jest poprzez mechanizm dziedziczenia
   - 5.2 wszystkie klasy hierarchii mają przypisaną jedną tabelę (STI)
   - 5.3 nazwa tabeli tworzona jest na podstawie klasy bazowej
   - 5.4 typ obiektu zapamiętywany jest w polu tekstowym `type`
   - 5.5 wszystkie klasy mają dostęp do tych samych atrybutów

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić,

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:
  - **set_table_name**
    ustala odmienną nazwę tabeli

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:
  - **set_table_name**
    ustala odmienną nazwę tabeli
  - **belongs_to :creator, class_name: "Author"**
    zmiana powiązanej klasy

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:
  - **set_table_name**
    ustala odmienną nazwę tabeli
  - **belongs_to :creator, class_name: "Author"**
    zmiana powiązanej klasy
  - **belongs_to :author, foreign_key: "creator_id"**
    zmiana klucza obcego

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:
  - **set_table_name**
    ustala odmienną nazwę tabeli
  - **belongs_to :creator, class_name: "Author"**
    zmiana powiązanej klasy
  - **belongs_to :author, foreign_key: "creator_id"**
    zmiana klucza obcego
- pozostała ważniejsze opcje:
  - **:dependent** – zachowanie obiektu powiązanego przy usuwaniu obiektu nadrzędnego

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:
  - **set_table_name**
    ustala odmienną nazwę tabeli
  - **belongs_to :creator, class_name: "Author"**
    zmiana powiązanej klasy
  - **belongs_to :author, foreign_key: "creator_id"**
    zmiana klucza obcego
- pozostała ważniejsze opcje:
  - **:dependent** – zachowanie obiektu powiązanego przy usuwaniu obiektu nadrzędnego
  - **:polymorphic** – związek polimorficzny

Opcje konfiguracyjne

- wszystkie domyślne ustawienia można zmienić, nie należy jednak robić tego bez powodu
- odstępstwa od konwencji określa się w klasie, np.:
  - **set_table_name**
    ustala odmienną nazwę tabeli
  - **belongs_to :creator, class_name: "Author"**
    zmiana powiązanej klasy
  - **belongs_to :author, foreign_key: "creator_id"**
    zmiana klucza obcego
- pozostała ważniejsze opcje:
  - **:dependent** – zachowanie obiektu powiązanego przy usuwaniu obiektu nadrzędnego
  - **:polymorphic** – związek polimorficzny
  - **:through** – związek zapożyczony

Plan prezentacji

Motywacja

- pozwalają na przyrostowe budowanie schematu bazy danych

Motywacja

- pozwalają na przyrostowe budowanie schematu bazy danych
- upraszczają synchronizację schematu bazy danych w środowisku programisty i na zdalnym serwerze

Motywacja

- pozwalają na przyrostowe budowanie schematu bazy danych
- upraszczają synchronizację schematu bazy danych w środowisku programisty i na zdalnym serwerze
- pozwalają dzielić się z innymi programistami zmianami, które muszą wprowadzić w swojej bazie

Motywacja

- pozwalają na przyrostowe budowanie schematu bazy danych
- upraszczają synchronizację schematu bazy danych w środowisku programisty i na zdalnym serwerze
- pozwalają dzielić się z innymi programistami zmianami, które muszą wprowadzić w swojej bazie
- pozwalają na wycofywanie wprowadzonych zmian

Motywacja

- pozwalają na przyrostowe budowanie schematu bazy danych
- upraszczają synchronizację schematu bazy danych w środowisku programisty i na zdalnym serwerze
- pozwalają dzielić się z innymi programistami zmianami, które muszą wprowadzić w swojej bazie
- pozwalają na wycofywanie wprowadzonych zmian
- automatycznie wykrywają, które zmiany muszą być wprowadzone w bazie

Motywacja

- pozwalają na przyrostowe budowanie schematu bazy danych
- upraszczają synchronizację schematu bazy danych w środowisku programisty i na zdalnym serwerze
- pozwalają dzielić się z innymi programistami zmianami, które muszą wprowadzić w swojej bazie
- pozwalają na wycofywanie wprowadzonych zmian
- automatycznie wykrywają, które zmiany muszą być wprowadzone w bazie
- pisane są w Rubim – są niezależne od konkretnej implementacji bazy danych

Model konceptualny a migracje

```ruby
class CreatePeople <
  ActiveRecord::Migration
  def change
    create_table :people do |t|
      t.string :name
      t.string :surname
      t.string :type
    end
  end
end
```

Model konceptualny a migracje

```ruby
class CreatePeople <
  ActiveRecord::Migration
  def change
    create_table :people do |t|
      t.string :name
      t.string :surname
      t.string :type
    end
  end
end
```

```ruby
class CreateBooks <
  ActiveRecord::Migration
  def change
    create_table :books do |t|
      t.string :title
      t.references :author
      # to samo co
      # t.integer :author_id
    end
  end
end
```

Model konceptualny a migracje – cd.

```ruby
class CreateCategories < ActiveRecord::Migration
  def change
    create_table :categories do |t|
      t.string :name
    end
  end
end
```

Model konceptualny a migracje – cd.

```ruby
class CreateCategories < ActiveRecord::Migration
  def change
    create_table :categories do |t|
      t.string :name
    end
  end
end

class CreateBooksCategories < ActiveRecord::Migration
  def change
    create_table :books_categories, id: false do |t|
      t.references :book
      t.references :category
    end
    add_index :books_categories, [:book_id, :category_id], unique: true
  end
end
```

Pliki migracji – `db/migrate/*`

- plik migracji opisuje zmianę schematu bazy danych

Pliki migracji – `db/migrate/*`

- plik migracji opisuje zmianę schematu bazy danych
- metoda **change** lub 2 metody: **up** i **down** opisują zmianę schematu

Pliki migracji – `db/migrate/*`

- plik migracji opisuje zmianę schematu bazy danych
- metoda **change** lub 2 metody: **up** i **down** opisują zmianę schematu
- posiadają numer, który jest generowany na podstawie czasu powstania pliku

Pliki migracji – db/migrate/*

- plik migracji opisuje zmianę schematu bazy danych
- metoda **change** lub 2 metody: **up** i **down** opisują zmianę schematu
- posiadają numer, który jest generowany na podstawie czasu powstania pliku
- pusty plik migracji generowany jest przez polecenie: **rails generate migration migration_name**

Pliki migracji – db/migrate/*

- plik migracji opisuje zmianę schematu bazy danych
- metoda **change** lub 2 metody: **up** i **down** opisują zmianę schematu
- posiadają numer, który jest generowany na podstawie czasu powstania pliku
- pusty plik migracji generowany jest przez polecenie: **rails generate migration migration_name**
- plik migracji tworzony jest też automatycznie w trakcie tworzenia nowego modelu

Pliki migracji – db/migrate/*

- plik migracji opisuje zmianę schematu bazy danych
- metoda **change** lub 2 metody: **up** i **down** opisują zmianę schematu
- posiadają numer, który jest generowany na podstawie czasu powstania pliku
- pusty plik migracji generowany jest przez polecenie: **rails generate migration migration_name**
- plik migracji tworzony jest też automatycznie w trakcie tworzenia nowego modelu
- w obu poleceniach można dodać pary **nazwa_pola:typ**, które zostaną automatycznie uwzględnione w migracji

Wprowadzanie zmian w bazie

- **schema_info** jest tablą w bazie danych posiadającą jedno pole tekstowe – **version**

Wprowadzanie zmian w bazie

- **schema_info** jest tablą w bazie danych posiadającą jedno pole tekstowe – **version**
- `schema_info` zawiera numery wszystkich wprowadzonych migracji

Wprowadzanie zmian w bazie

- **schema_info** jest tablą w bazie danych posiadającą jedno pole tekstowe – **version**
- `schema_info` zawiera numery wszystkich wprowadzonych migracji
- polecenie **rake db:migrate** sprawdza numery w bazie danych i porównuje je z numerami migracji w katalogu `db/migrate`

Wprowadzanie zmian w bazie

- **schema_info** jest tablą w bazie danych posiadającą jedno pole tekstowe – **version**
- `schema_info` zawiera numery wszystkich wprowadzonych migracji
- polecenie **rake db:migrate** sprawdza numery w bazie danych i porównuje je z numerami migracji w katalogu `db/migrate`
- jeśli w bazie danych brakuje jakiś numerów migracji, są one aplikowane w kolejności odpowiadającej czasowi ich powstania

Wprowadzanie zmian w bazie

- **schema_info** jest tablą w bazie danych posiadającą jedno pole tekstowe – **version**
- `schema_info` zawiera numery wszystkich wprowadzonych migracji
- polecenie **rake db:migrate** sprawdza numery w bazie danych i porównuje je z numerami migracji w katalogu `db/migrate`
- jeśli w bazie danych brakuje jakiś numerów migracji, są one aplikowane w kolejności odpowiadającej czasowi ich powstania
- w celu wycofania ostatniej zmiany wprowadzamy
  rake db:rollback
  powodzenie tego procesu zależy od poprawnej implementacji metody `down`

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty
- **db:create:all** – tworzy bazę danych dla wszystkich środowisk

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty
- **db:create:all** – tworzy bazę danych dla wszystkich środowisk
- **db:migrate:redo** – wycofuje i wprowadza ostatnią wprowadzoną migrację

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty
- **db:create:all** – tworzy bazę danych dla wszystkich środowisk
- **db:migrate:redo** – wycofuje i wprowadza ostatnią wprowadzoną migrację
- **db:migrate:reset** – czyści bazę i od początku przeprowadza wszystkie migracje

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty
- **db:create:all** – tworzy bazę danych dla wszystkich środowisk
- **db:migrate:redo** – wycofuje i wprowadza ostatnią wprowadzoną migrację
- **db:migrate:reset** – czyści bazę i od początku przeprowadza wszystkie migracje
- **db:migrate:status** – wyświetla status poszczególnych migracji

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty
- **db:create:all** – tworzy bazę danych dla wszystkich środowisk
- **db:migrate:redo** – wycofuje i wprowadza ostatnią wprowadzoną migrację
- **db:migrate:reset** – czyści bazę i od początku przeprowadza wszystkie migracje
- **db:migrate:status** – wyświetla status poszczególnych migracji
- **db:seed** – wczytuje dane inicjujące z pliku `db/seed.rb`

Inne polecenia `rake` do zarządzania bazą

- **db:create** – tworzy bazę danych dla środowiska programisty
- **db:create:all** – tworzy bazę danych dla wszystkich środowisk
- **db:migrate:redo** – wycofuje i wprowadza ostatnią wprowadzoną migrację
- **db:migrate:reset** – czyści bazę i od początku przeprowadza wszystkie migracje
- **db:migrate:status** – wyświetla status poszczególnych migracji
- **db:seed** – wczytuje dane inicjujące z pliku `db/seed.rb`
- **db:version** – wyświetla numer ostatniej wprowadzonej migracji

Plan prezentacji

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych
- istotnie rozszerza zestaw więzów dostępnych w bazie danych

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych
- istotnie rozszerza zestaw więzów dostępnych w bazie danych
- zawiera zestaw predefiniowanych walidatorów, często wykorzystywanych w aplikacjach internetowych

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych
- istotnie rozszerza zestaw więzów dostępnych w bazie danych
- zawiera zestaw predefiniowanych walidatorów, często wykorzystywanych w aplikacjach internetowych
- pozwala na weryfikację wartości tekstowych, liczbowych, logicznych, itp.

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych
- istotnie rozszerza zestaw więzów dostępnych w bazie danych
- zawiera zestaw predefiniowanych walidatorów, często wykorzystywanych w aplikacjach internetowych
- pozwala na weryfikację wartości tekstowych, liczbowych, logicznych, itp.
- pozwala na wyabstrahowanie bardziej skomplikowanych walidacji (np. adresu e-mail)

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych
- istotnie rozszerza zestaw więzów dostępnych w bazie danych
- zawiera zestaw predefiniowanych walidatorów, często wykorzystywanych w aplikacjach internetowych
- pozwala na weryfikację wartości tekstowych, liczbowych, logicznych, itp.
- pozwala na wyabstrahowanie bardziej skomplikowanych walidacji (np. adresu e-mail)
- umożliwia ograniczenie działania walidacji do tworzenia/modyfikowanie obiektu

Motywacja

- deklaratywny mechanizm weryfikacji poprawności wprowadzanych danych
- istotnie rozszerza zestaw więzów dostępnych w bazie danych
- zawiera zestaw predefiniowanych walidatorów, często wykorzystywanych w aplikacjach internetowych
- pozwala na weryfikację wartości tekstowych, liczbowych, logicznych, itp.
- pozwala na wyabstrahowanie bardziej skomplikowanych walidacji (np. adresu e-mail)
- umożliwia ograniczenie działania walidacji do tworzenia/modyfikowanie obiektu
- może być używany do obiektów niezwiązanych z bazą danych

Przykład walidatorów

```
class Book < ActiveRecord::Base
  validates :title, presence: true, length: {minimum: 3}
end
```

Przykład walidatorów

```
class Book < ActiveRecord::Base
  validates :title, presence: true, length: {minimum: 3}
end

class Person < ActiveRecord::Base
  validates :shoe_size, numericality: true
  validates :first_name, presence: true,
    length: {maximum: 30}
  validates :user_name, uniqueness: true
    length: {within: 6..20}
  validates :email, format: {
    with: /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i,
  }
  validates :password, confirmation: {
    message: "should match confirmation"
  }
  validates :terms_of_service, acceptance: {on: :create}
end
```

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości
- **inclusion** – przynależność do określonej grupy wartości, np. płeć – mężczyzna/kobieta

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości
- **inclusion** – przynależność do określonej grupy wartości, np. płeć – mężczyzna/kobieta
- **format** – sprawdzenie formatu wartości tekstowej przy użyciu wyrażenia regularnego, np. kod pocztowy

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości
- **inclusion** – przynależność do określonej grupy wartości, np. płeć – mężczyzna/kobieta
- **format** – sprawdzenie formatu wartości tekstowej przy użyciu wyrażenia regularnego, np. kod pocztowy
- **length** – sprawdzenie długości wartości tekstowej

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości
- **inclusion** – przynależność do określonej grupy wartości, np. płeć – mężczyzna/kobieta
- **format** – sprawdzenie formatu wartości tekstowej przy użyciu wyrażenia regularnego, np. kod pocztowy
- **length** – sprawdzenie długości wartości tekstowej
- **numericality** – sprawdzenie wartości numerycznej

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości
- **inclusion** – przynależność do określonej grupy wartości, np. płeć – mężczyzna/kobieta
- **format** – sprawdzenie formatu wartości tekstowej przy użyciu wyrażenia regularnego, np. kod pocztowy
- **length** – sprawdzenie długości wartości tekstowej
- **numericality** – sprawdzenie wartości numerycznej
- **uniqueness** – unikalność wartości, np. loginu

Predefiniowane walidatory

- **presence** – sprawdzenie niepustości atrybutu
- **acceptance** – sprawdzenie wartości logicznej, np. zatwierdzenie regulaminu
- **confirmation** – porównanie identyczności dwóch atrybutów, np. hasła i jego powtórzenia
- **exclusion** – wykluczenie należenia do określonej grupy wartości
- **inclusion** – przynależność do określonej grupy wartości, np. płeć – mężczyzna/kobieta
- **format** – sprawdzenie formatu wartości tekstowej przy użyciu wyrażenia regularnego, np. kod pocztowy
- **length** – sprawdzenie długości wartości tekstowej
- **numericality** – sprawdzenie wartości numerycznej
- **uniqueness** – unikalność wartości, np. loginu
- **associated** – weryfikowanie powiązanego obiektu

Sposób działania

```
person = Person.new()
if person.save
  #...                    # wszystkie walidacje były pozytywne
else
  person.errors           # zawiera informacje o błędach
end
```

Sposób działania

```ruby
person = Person.new()
if person.save
  #...                    # wszystkie walidacje były pozytywne
else
  person.errors           # zawiera informacje o błędach
end

begin
  person = Person.new()
  person.save!
rescue ActiveRecord::RecordInvalid
  # obsługa błędu
end
```

Model – więcej informacji

- `guides.rubyonrails.org/association_basics.html`
  asocjacje (związki) modeli
- `guides.rubyonrails.org/migrations.html`
  migracje schematu bazy danych
- `guides.rubyonrails.org/active_record_validations_callbacks.html`
  walidacje
- `guides.rubyonrails.org/active_record_querying.html`
  wyszukiwanie obiektów w bazie danych

Pytania

PYTANIA?