Проектирование архитектуры нашего приложения

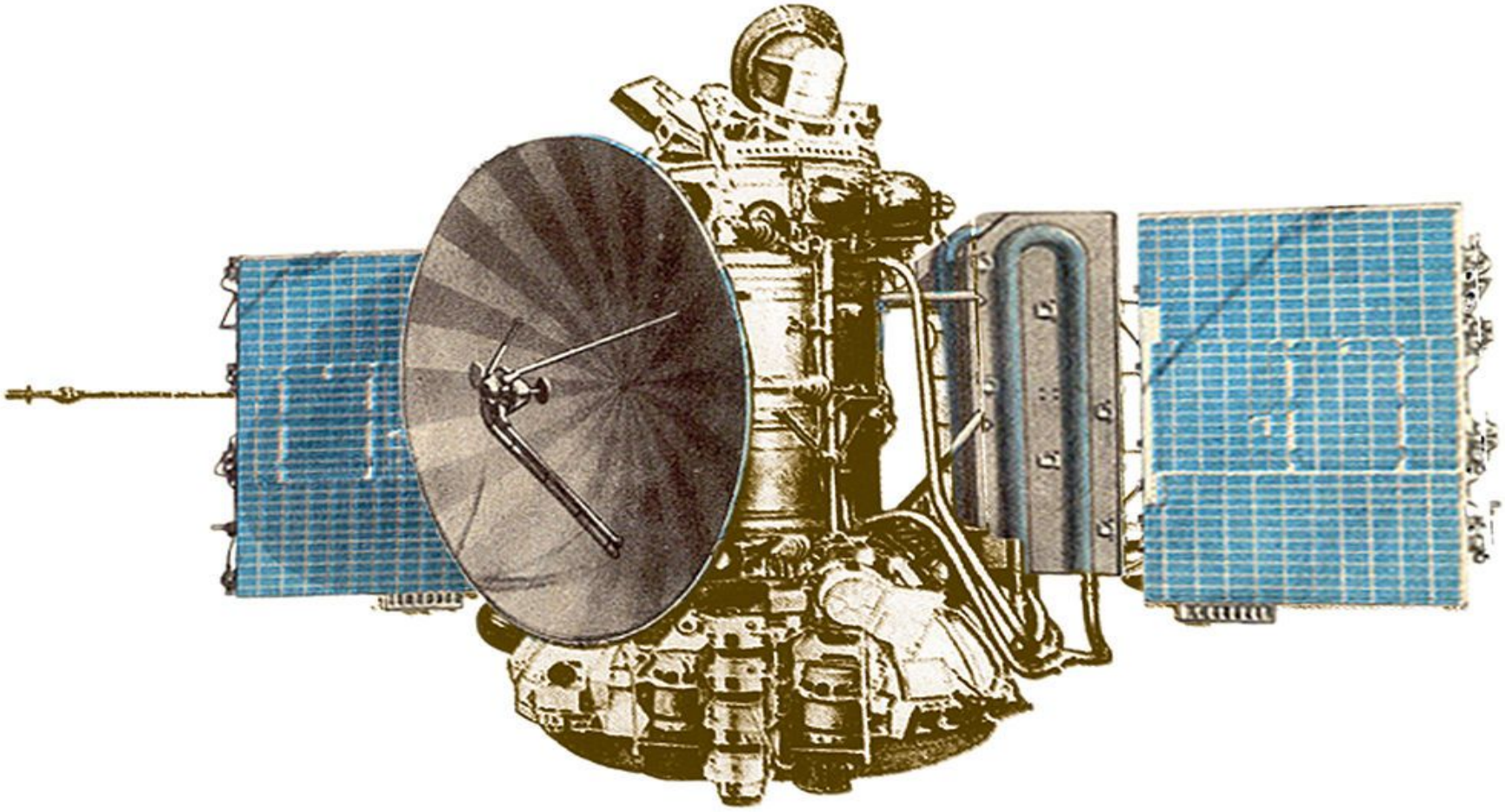

bomz.org
ЗНАНИЕ - СИЛА
незнание - рабочая сила
bomz.org

Знания

- Архитектура — модели и связи между ними
- Модели
- ORM, ActiveRecord
- Миграции
- СУБД PostgreSQL
- Отношения между моделями
- Валидация для наших моделей
- Методы обратного вызова
- Запросы к БД
- Группа именованных условий (scope)

Архитектура

Архитектура



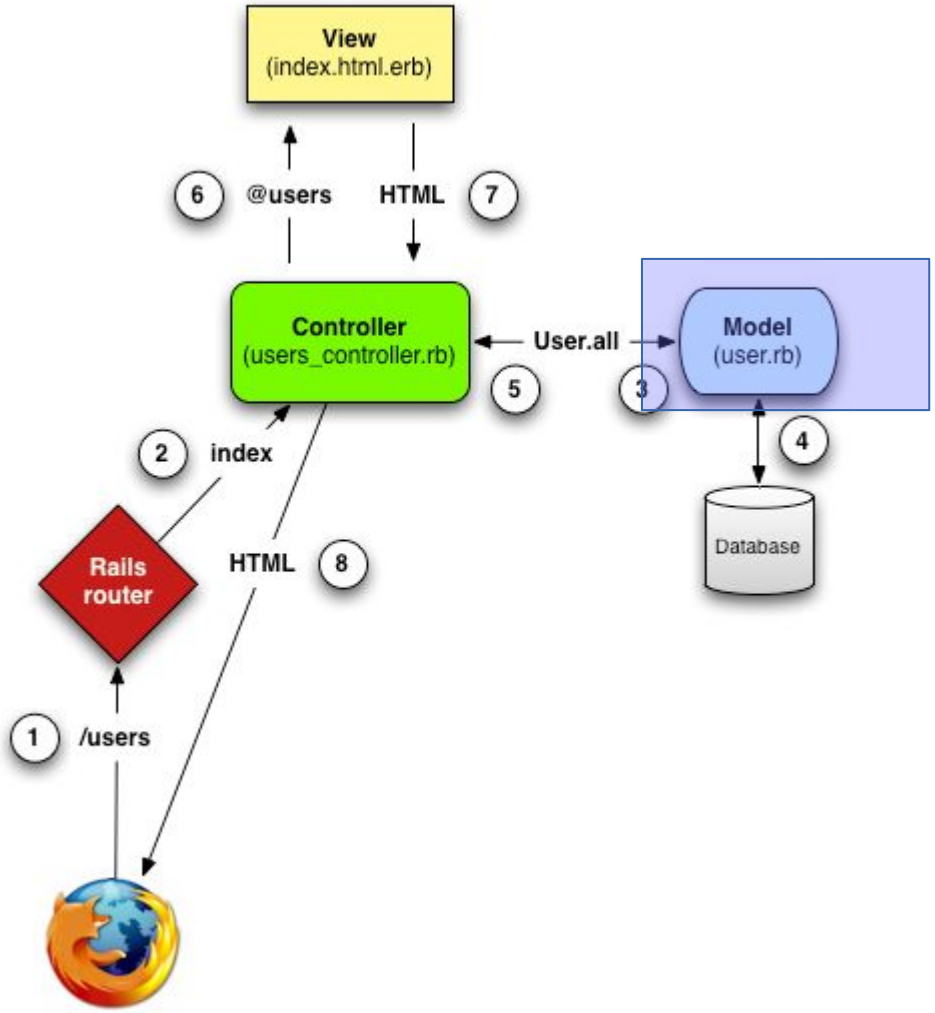
Архитектура базируется на требованиях, целях

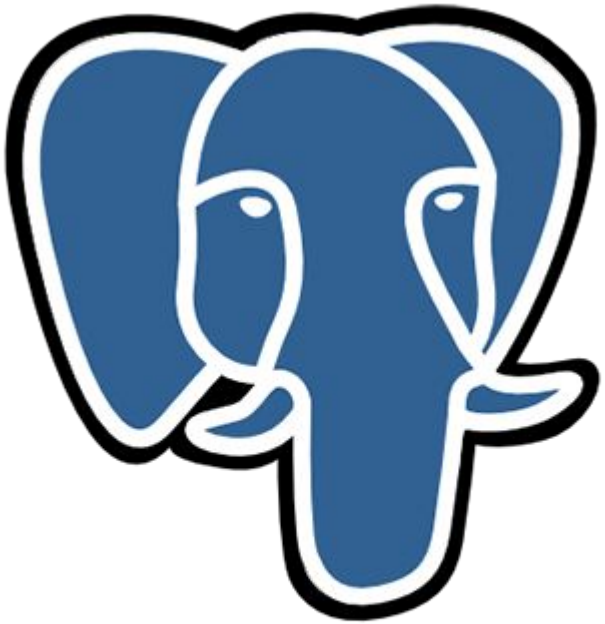
Модель

Модели

- Компонент MVC, который предназначен для реализации логики веб-приложения
- Чаще всего отображает таблицу БД в ОО-стиле
- $ rails g model Model column:type[column:type]*

Связь модели с контроллером





PostgreSQL

PostgreSQL

- Объектно-реляционная СУБД
  - Наследовать таблицы
  - Создавать пользовательские типы данных
- Поддержка многочисленных типов данных
  - Хранение массивов
  - Поддержка JSON/JSONB
  - …
- Транзакционный DDL
- Размеры данных
  - 32 Тб — таблица
  - 1 Гб — поле
  - 250-1600 столбцов в таблице

Создавать пользовательские типы данных

```
CREATE TYPE competence as (
  title VARCHAR(40),
  rate FLOAT
);

CREATE TYPE specialization as ENUM ('mobile', 'web',
'embedded');

CREATE TABLE professionals (
   compy competence
);

INSERT INTO professionals VALUES(ROW('Монтаж
электрооборудования', 0.8));
```

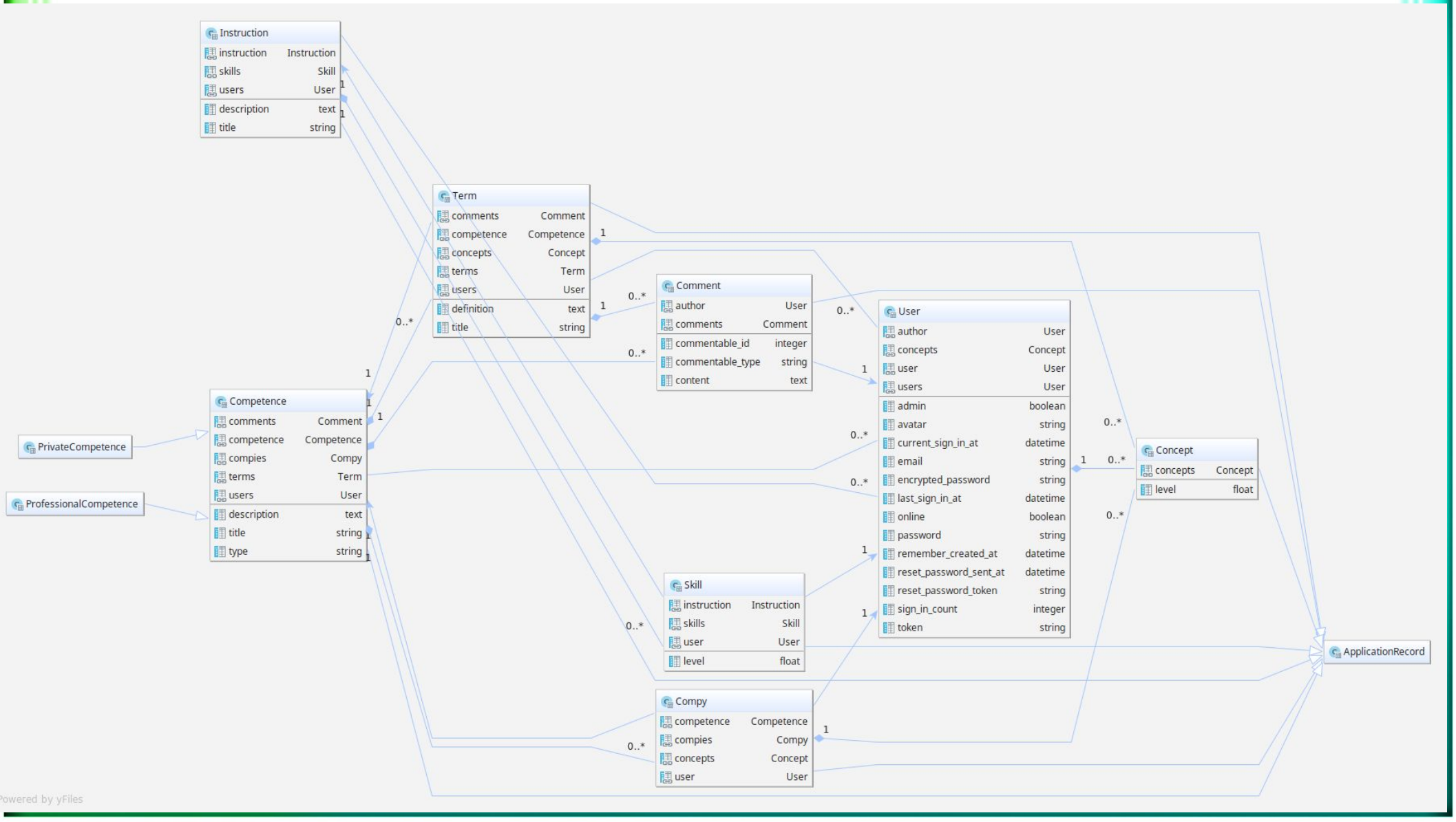
Наследовать таблицы
Использовать массивы

```
CREATE TABLE programmers(
  type specialization,
  IDE VARCHAR(50)[]
) INHERITS (professionals);
INSERT INTO programmers(type, compy, ide) VALUES
('web', ROW('Подключение Bootstrap', 0.8),
ARRAY['RubyMine', 'Atom']);

SELECT ide[2] from programmers;
```

Миграция

- Механизм фреймворка, который позволяет управлять структурой БД в ОО-стиле
- Генерация: $ rails generate migration <name>
- Папка db/migration
- ! Должны выполняться в обе стороны

Информационная модель БД



ORM

Реляционная модель	ОО-модель
Таблица	Модель (класс)
Столбец	Поле
Запись	Объект

ActiveRecord — шаблон проектирования

AR

```
competence = Competence.new
competence.name = "Веб-разработка на Rails"
competence.save
```

SQL

```
INSERT INTO competences (name)
VALUES ('Веб-программирование на Rails')
```

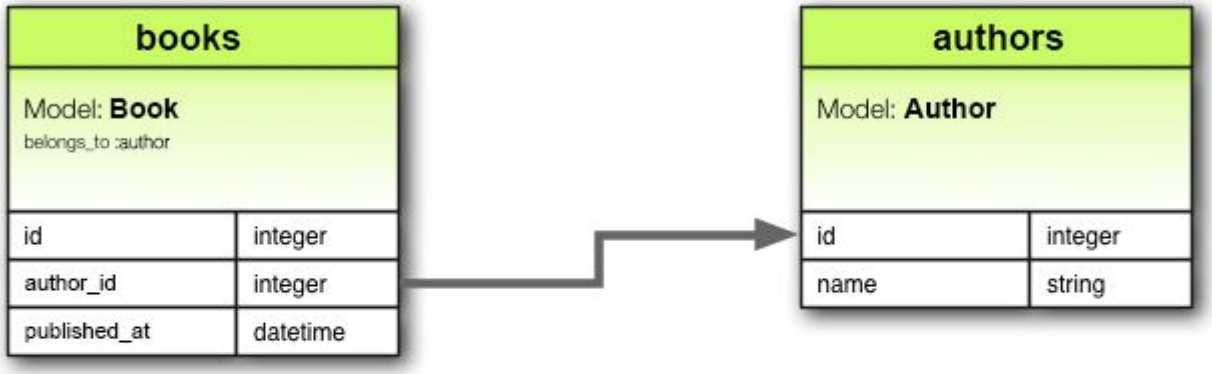
Отношения между моделями

```
belongs_to
has_one
has_many
has_many :through
has_one :through
has_and_belongs_to_many
```

Варианты отношений

```
1 к 1:
belongs_to к has_one
has_one :through

1 к N:
has_many к belongs_to

N к M:
has_many :through
has_and_belongs_to_many+
```

Отношения между моделями

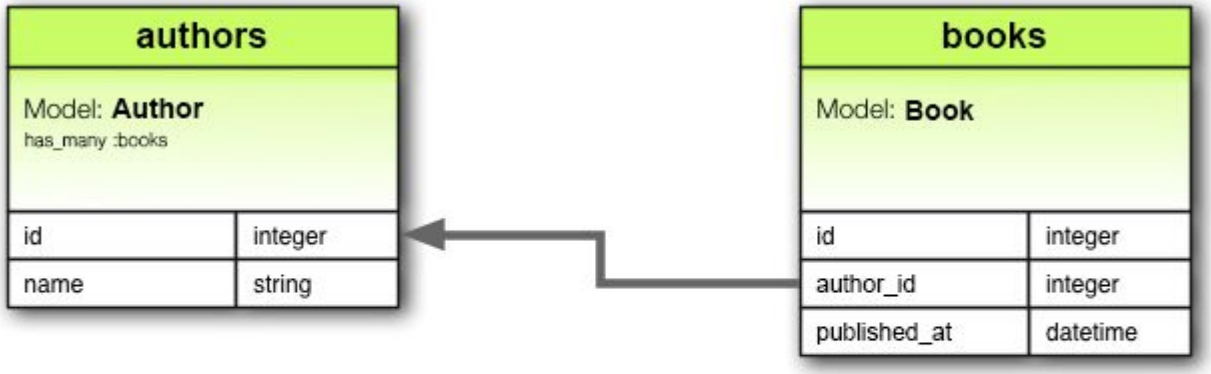



```
class Book < ApplicationRecord
  belongs_to :author
end

class Author < ApplicationRecord
  has_one :book
end
```

```
create_table :books do |t|
      t.belongs_to :author, foreign_key: true
      t.datetime :published_at
      t.timestamps null: false
    end
```

```
create_table :authors do |t|
      t.string :name
      t.timestamps null: false
    end
```

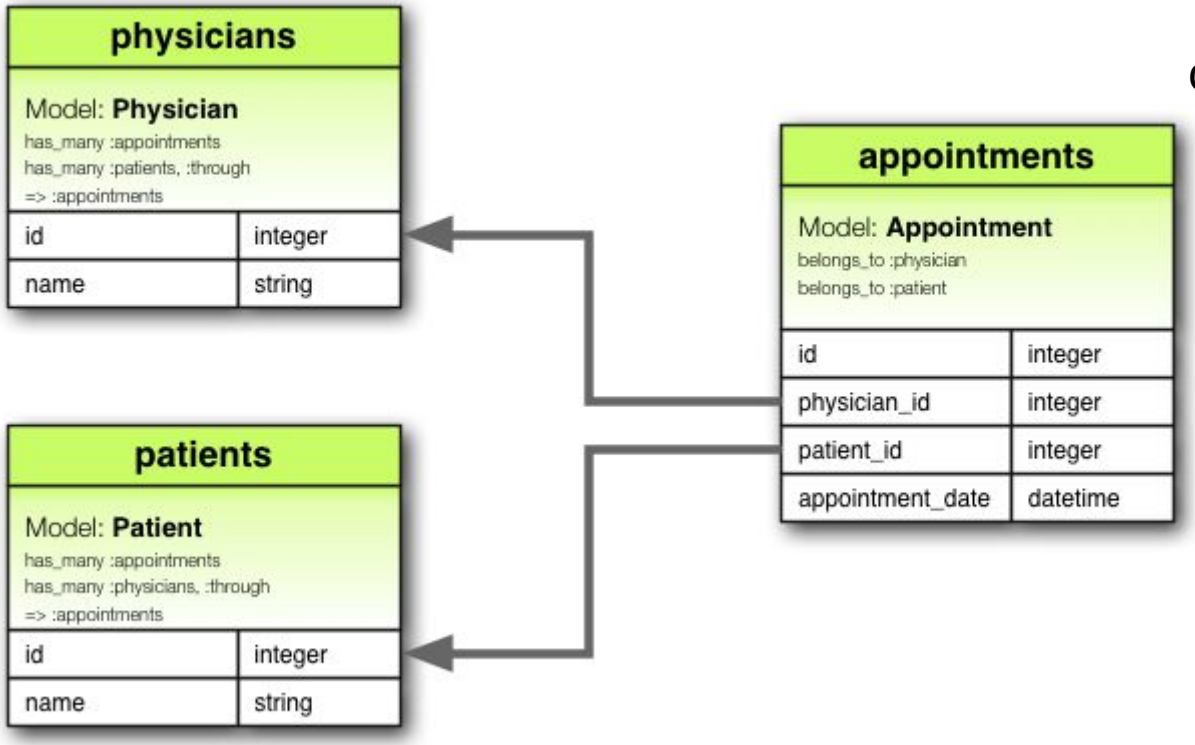
Отношения между моделями




```
class Book < ApplicationRecord
  belongs_to :author
end

class Author < ApplicationRecord
  has_many :books
end
```

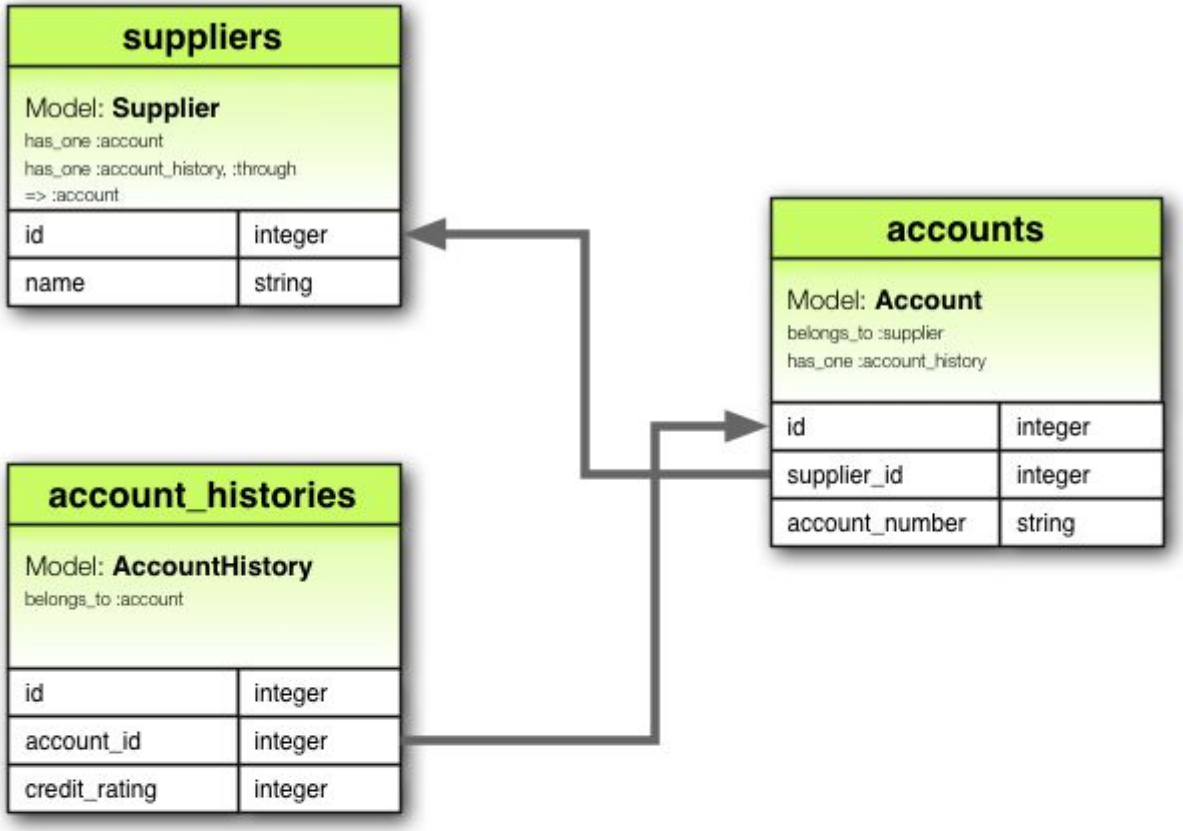
```
create_table :books do |t|
      t.belongs_to :author, foreign_key: true
      t.datetime :published_at
      t.timestamps null: false
    end
```

```
create_table :authors do |t|
      t.string :name
      t.timestamps null: false
    end
```



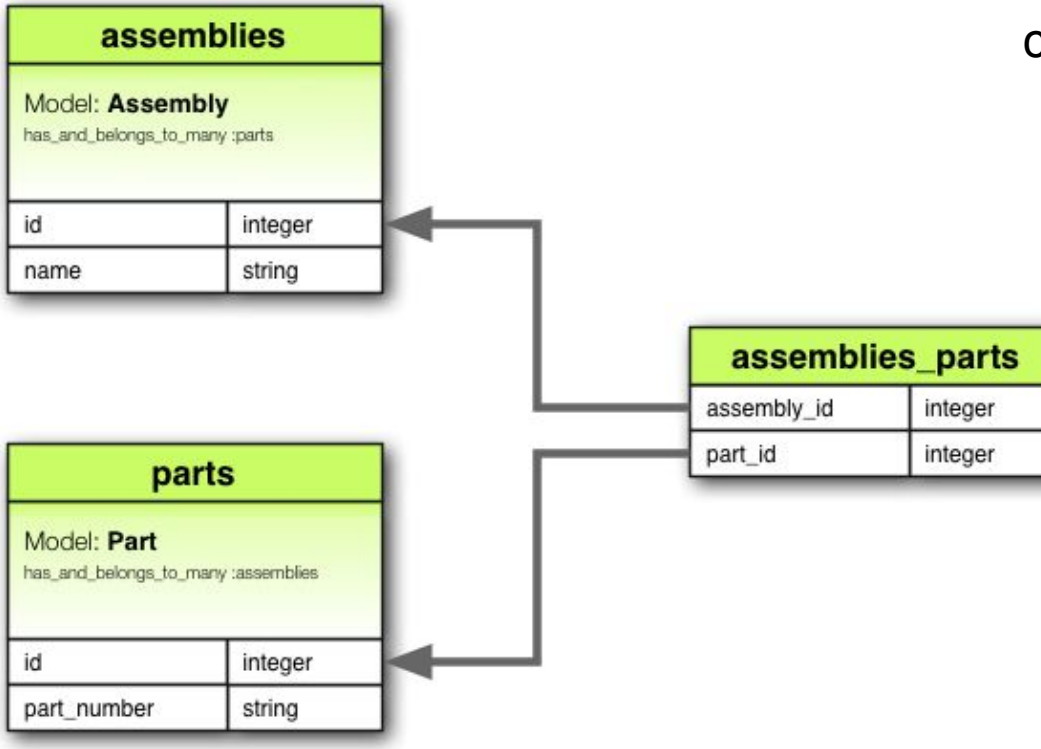

```
create_table :physicians do |t|
  t.string :name
  t.timestamps null: false
end

create_table :patients do |t|
  t.string :name
  t.timestamps null: false
end

create_table :appointments do |t|
  t.belongs_to :physician,
        foreign_key: true
  t.belongs_to :patient,
        foreign_key: true
  t.datetime :appointment_date
  t.timestamps null: false
end
```

```
class Physician < ApplicationRecord
  has_many :appointments
  has_many :patients, through: :appointments
end

class Appointment < ApplicationRecord
  belongs_to :physician
  belongs_to :patient
end

class Patient < ApplicationRecord  # ActiveRecord::Base до Rails 5.0
  has_many :appointments
  has_many :physicians, through: :appointments
end
```




```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, :through => :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```

```
create_table :suppliers do |t|
  t.string :name
  t.timestamps null: false
end

create_table :accounts do |t|
  t.belongs_to :supplier,
        foreign_key: true
  t.string :account_number
  t.timestamps null: false
end

create_table :account_histories do |t|
  t.belongs_to :account,
        foreign_key: true
  t.integer :credit_rating
  t.timestamps null: false
end
```




```
create_table :assemblies do |t|
    t.string :name
    t.timestamps null: false
  end

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

create_table :assemblies_parts, id: false do |t|
  t.belongs_to :assembly,
        foreign_key: true
  t.belongs_to :part,
        foreign_key: true
end
```

```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```

Основные запросы к БД

- CRUD
- Использование связей при создании
- Поиск
- …

CRUD

- Создать новую запись
- User.create email: 'tester@test.ru', password: '52344234'
- Считать существующую запись:
- User.first
- User.find[15, 23]
- Обновить существующую запись:
- user.update_attribute email, 'lang.sha@mail.ru'
- User.last.update(params[:user]) # params = {User: {email: 'forrester@mail.ru', password: 'norender'} }
- Удалить существующую запись
- User.last.delete

- Использование связей при создании

- misha = User.create email: 'misha@mail.ru', password: '123123'
- portfolio = Portfolio.create user: misha
- user = User.create email: 'test1@mail.ru', password: '123123123', competences: [Competence.create(name: 'Rails'), Competence.create(name: 'PHP')]

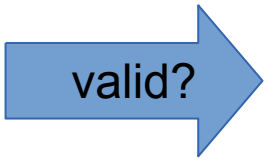
Поиск

- user = User.find_by(email: 'misha@profport.ru')
- User.where('password like ? or email like ?', "%#{pattern}%", '@profport.ru')
- User.where(created_at: (Time.now.midnight-1.month..Time.now.midnight+1.month) # BETWEEN '2016-10-27 19:00:00' AND '2016-12-27 19:00:00')

Валидация

```
class User < ApplicationRecord
# ...
  validates :email, presence: true, strict: StandardError
  validates :email, format: { with: /\A\w+@\w+\.\w{2,6}\z/, message: '%{value} is not email' },
on: :custom_scenario
  validates :password, length: {in: 6..20, too_short: '%{attribute} is too short'}
# ...
end
```

create
create!
save
save!
update
update!



model.errors

Валидаторы

- Обязательные поля:
- **validates** :email, :password, presence: true
- Определённая длина:
- **validates** :password, length: {in: 6..20, too_short: '%{attribute} is too short'}
- Валидация всех связанных моделей:
- **validates_associated** :competences
- Валидация уникальности:
- **validates** :email, uniqueness: true
- Соответствие формату:
- **validates** :email, format: { with: /\A\w+@\w+\.\w{2,6}\z/, message: '%{value} is not email' }
- ...

- Методы обратного вызова (код выполняется в транзакции)

Методы обратного вызова


```
before_save do |user|
  puts «Собираемся сохранить»
end
```

```
user.save
```

```
after_save do |user|
  puts "Сохранили!"
end
```

Скоупы в ActiveRecord

- Специальные запросы
- Реализуются с помощью лямбда-выражений
- Например:
  - scope :published, -> { where(published: true) }
  - scope :newest, ->(date) { where('created_at > ?', date }
- Использование:
  - Post.published
  - Post.published.newest(DateTime.current - 1.week)


Ты молод, креативен, талантлив? Амбициозен, уверен в себе, полон свежих идей? А делать хоть что-нибудь умеешь?!
Atkritka.com

Умения

- Подключить СУБД PostgreSQL
- Создать соответствующие модели
- Прописать ограничения на столбцы БД в миграциях
- Прописать валидации
- Реализовать отношения между моделями
- Использовать методы обратного вызова
- Выполнять основные запросы к БД
- Создать scope для модели Achievement (status, created_at)

- Создать другие модели

- $ rails g model User email:string password:string
- $ rails g model Portfolio
- $ rails g Achievement description:text status:integer

Подключить СУБД PostgreSQL

- gem 'pg'
- Настроить доступ (пока для root)
- $ rails db:create
- Создать пользователя и назначить ему права
- Указать этого пользователя в настройках доступа

- Настроить доступ пока для root (config/database.yml)

- default: &default
- adapter: postgresql
- pool: 5
- timeout: 5000
- server: localhost

- development:
- <<: *default
- database: 20161127_profport_development
- username: root
- password: root

Прописать ограничения на столбцы БД в миграциях

- Для пользователя:
- t.string :email, limit: 50, null: false
- t.string :password, limit: 20, null: false
- Для достижения (achievement):
- t.text :description, null: false
- t.integer :status, default: 0

Реализовать отношения между моделями

- Создать миграции с внешними ключами и промежуточной таблицей
- $ rails g migration AddRelationBetweenPortfolioAndUser
- Прописать отношения на уровне моделей

Нюансы

- t.references — пишем «модель»:
- t.references :**user**, foreign_key: true
- add_foreign_key — пишем таблицу:
- add_foreign_key :portfolios, :**users**
- **t.references :user — не делается уникальным**
  - t.references :user, index: { unique: true }
- remove_column — указывать тип (для отката)
- remove_column :competences, author_id, :**string**

- Создать метод обратного вызова

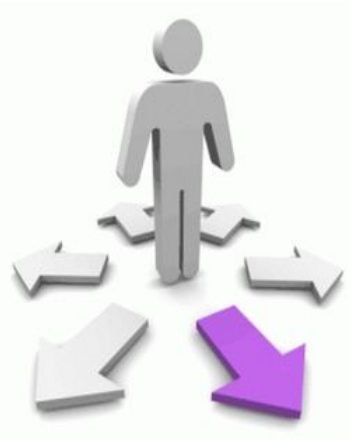
- # модель User
- after_create do
- Portfolio.create user: self
- end


ТУДА
100м
СЮДА
200м
ВЕЧНАЯ НЕОПРЕДЕЛЕННОСТЬ!
КАК ЖИТЬ?
DEMOTIVATORS.RU

Неопределённости

- Почему не использовали create_join_table для создания кросс-таблицы между competences и portfolios? Не создаётся первичный ключ, не на что опереться, если мы хотим ссылаться на записи кросс-таблицы (portfolio_competences)
- Альтернатива шаблону ActiveRecord? Data Mapper. Hibernate (Java), Doctrine (PHP), Ruby Object Mapper/Sequel

Самостоятельно

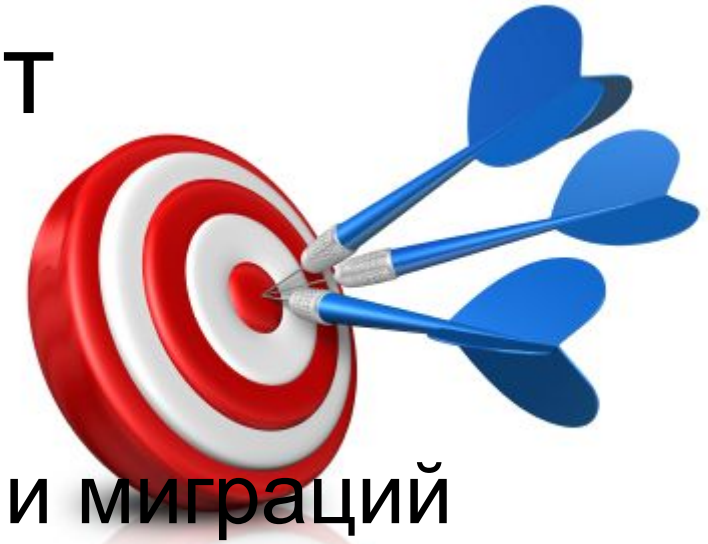


- Создание ограничений (внешних ключей) на уровне СУБД с помощью SQL
- Полиморфные связи
- Понятие scope в запросах
  - Получение записей из БД относительно scope
- Полезные советы

Результат


Это же так просто!
Я это сделаю!
Я могу это сделать!
Я попытаюсь...
Как мне это сделать?
Я хочу это сделать!
Я не могу это сделать (не умею, не получится)
Я не буду этого делать.

Результат



- Изучены создание моделей и миграций
- Изучены различные способы ассоциации моделей
- Изучены различные способы обеспечения целостности БД
- Спроектирована архитектура нашего приложения

Список источников

- Основное
- Миграции базы данных Rails
- Валидации Active Record
- Связи (ассоциации) Active Record
- Интерфейс запросов Active Record
- Дополнительное
- Active Record против Data Mapper-а для сохранения данных