5 Работа с моделями

Продолжение предыдущего занятия

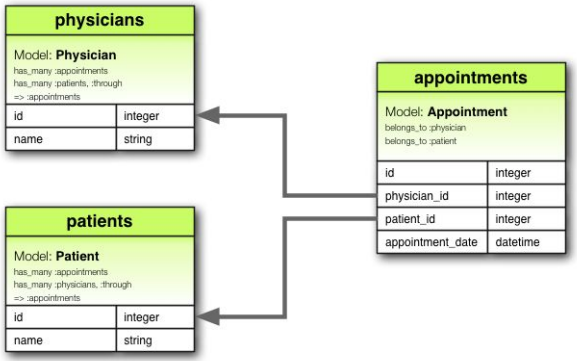
На прошлом занятии

1. Миграции
2. Генератор миграций и моделей
3. Реализация связи один-ко-многим

Подготовка к занятию

1. Кто скачал демо-приложение
   a. В консоли выполнить **git pull**
2. У кого нет демо-приложения
   a. cd ~
   b. Выполняем инструкции по адресу https://github.com/DenisKem/ruby_lesson1

Связь многие-ко-многим




```
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, :through => :appointments
end
```

Связь многие-ко-многим. Алгоритм

Создать соединительную модель. Она часто имеет название Model1ToModel2

Связать ее с двумя моделями, с помощью связи многие-к-одному.

Связать первую и вторую модели через соединительную таблицу, при помощи has_many: through

Задание 4

Допустим что предприятие для которого мы разрабатываем реестр сотрудников занимается разработкой проектов в какой-нибудь области. В таком случае в одном проекте могут участвовать несколько сотрудников, а сотрудник может быть привлечен одновременно к нескольким проектам.

Получается что сотрудники и проекты связаны связью многие-ко-многим.

Задание 4

Связать сотрудников с их проектами. Для это требуется:

1. Создать модель Project (проект)
2. Создать модель EmployeeToProject
3. Настроить связи
4. Запустить тестовый скрипт
    a. В консоли rake demo: employees_projects
    b. lib/tasks/demo.rake

На след. слайде краткая памятка :)

Напоминаем

Создание модели

bundle exec rails g model ModelName field1:datatype field2:datatype

Связь многие-к-одному

belongs_to :authors

has_many :books

Создание сущностей

1. Без сохранения в БД
    a. Model.new params. # Данный объект можно сохранить потом, вызвав метод save/save!
2. С сохранением
    a. Model.create(params) # вернет boolean-значение сохранилась ли запись
    b. Model.create!(params) # выкинет исключение, если провалена валидация

Создание сущностей через связи

object.another_models.create(params)

object.another_models.build(params) # Без сохранения в БД.

Без этих удобных методов нам пришлось бы создать объект AnotherModel и вручную задать значение AnotherModel#project_id.

В случае связи многие-ко-многим пришлось бы совершить гораздо больше действий.

Взаимодействие с объектом

```
employee = Employee.new

employee.position = Position.all.sample # или

employee.assign_attributes position: Position.all.sample

employee.save
```

Тестовые данные. db/seeds.rb

В начале файла происходит очистка БД путем вызова для каждой модели метода delete_all

Далее создаем тестовые данные. Первыми заполняются данные, от которых зависят другие данные (например, сперва категории блога, а затем посты)

Полезным будет использовать гем faker

Задание 5

Доработать “сиды” проекта:

1. В начало файла добавить удаление проектов
2. Добавить тестовые проекты
3. Привязать случайным образом сотрудников к тестовым проектам

Запросы к БД

Для получения объектов из базы данных нет необходимости писать вручную SQL-запросы. ORM Active Record выполнит запросы в базу данных за вас, он совместим с большинством СУБД (MySQL, PostgreSQL и SQLite - это только некоторые из них). Независимо от того, какая используется СУБД, формат методов Active Record будет всегда одинаковый.

Запросы. Получение одного объекта

Model#find(id) поиск по первичному ключу

Model#find_by_field(field) поиск по конкретному полю

Методы с восклицательным знаком, а также метод **Model#find** будут выбрасывать исключение ActiveRecord::RecordNotFound, если в базе отсутствует такая запись.

Запросы. Получение коллекции

Model#all - получить все записи данной модели

Model#where(condition) - добавляет условие выборк. Является цепочечным методом, то есть можно собирать следующего вида конструкции:

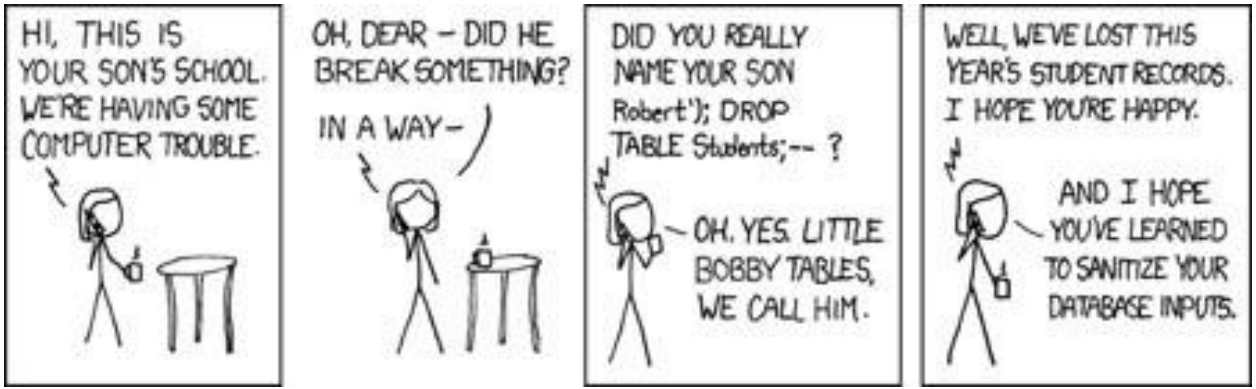
Model.where(...).where(...)

Model#where.not(condition) используется для отрицания (предикат **NOT**)

Примеры можной найти в сгенерированном скаффолде.

Запросы. Как не надо делать

Client.where("first_name LIKE '%#{params[:first_name]}%'") - место для атаки SQL-инъекцией.



Запросы. Безопасные условия

Client.where("first_name LIKE :first_name”, first_name: “%#{params[:first_name]}%")

Client.where("first_name LIKE ?”, “%#{params[:first_name]}%")

Для условия можно использовать и массивы.

Client.where(first_name: [‘Саша’, ‘Маша’, ‘Петя’])

http://rusrails.ru/ruby-on-rails-security-guide#sql-in-ektsii

Сортировка

Чтобы получить записи из базы данных в определенном порядке, можете использовать метод order.

```
Client.order(:created_at)
# ИЛИ
Client.order("created_at")

Client.order(created_at: :desc)
# ИЛИ
Client.order(created_at: :asc)
```

Задание 6

Отсортировать сотрудников по продолжительности работы на предприятии.

Взаимодействие с существующими объектами

employee = Employee.find_by_name ‘Иван’

employee.position = Position.all.sample

employee.save # или

employee.update position: Position.all.sample

Валидации моделей

Объект считается валидным, когда все его поля соответсвуют каким-либо определенным условиям. Например, сущность Employee валидна - когда заполнены имя работника и его должность. Хранение сотрудника в базе сотрудника без имени бессмысленно и вредит системе

Подключение валидаций

В контексте класса модели пишется следующая конструкция

validates<пробел>:field1, [:field2, ...], helper1, [helper2]

Helper - presence: true - обычная пара ключ-значение

Встроенные хэлперы валидации

Перечислены здесь:

http://rusrails.ru/active-record-validations

Можно определить кастомные валидаторы

Обработка ошибок валидации

Валидация вызывается

1. перед сохранением записи (create, create!, save, save!, update)
2. Прямым вызовом #valid?

Методы с восклицательным знаком при провале валидации выкидывают исключения с текстом содержащим ошибки.

Методы без восклицательного знака при провале валидации возвращают false. После этого у объекта можно вызвать метод errors, содержащий ошибки в формате field_name => [‘error1’, ‘error2’]

Задание 7

В справочнике должностей все записи должны быть с уникальным названием. Не должно быть должностей без названия.

У записи о работнике обязательно должны быть указаны его имя и должность.