Основы Rails

Идеальная База Данных

- Содержит данные пользователя — поэтому не тестируй на ней приложение!
- Rails совет: у разработки, продакшна и тестов - раздельные БД
- Разные типы БД используются в разных случаях
- Как изменять БД, если необходимо скопировать изменения в БД продакшна?
- Rails совет: миграция — скрипт, описывающий изменения, подходящий для всех ДБ

Плюсы миграции

- Можно определить любую миграцию, и узнать, какая и когда применялась
- Миграции могут быть обратимыми
- Можно управлять через контроль версий
- Автоматизирована == надежно повторяема
- Сравните: использовать Bundler и вручную установить библиотеки/гемы
- Совет: автоматизируйте
- Определитесь, что делать и создайте инструменты автоматизации

Знакомство с генератором кода

- Он просто осуществляет миграцию.

- Применить миграцию к разработке:rake db:migrate

- К продакшну: heroku rake db:migrate

- Применение миграции также записывает в БД, какие миграции сделаны

Rails Cookery #1

Дополнения к функциональности приложения ==
добавление модели, представления, действия контроллера

• Чтобы добавить новую модель в Rails app:

1. Создайте миграцию, описывающую изменения:

rails создает миграцию

2.Примените миграцию: rake db:migrate

3.Если модель новая, создайте для нее файл app/models/model.rb

• Обновляйте схему тестовой DB: rake db:test:prepare

На основании знаний о Rails, скажите, какой объект, скорее всего, присутствует в миграционном коде:

```
def up
  create_table 'movies' do |t|
    t.datetime 'release_date' ...
  end
end
```

☐ Объкт, отображающий БД

☐ Объект, отображающий переменную модели

☐ Объект, отображающий таблицу

☐ Все может быть правильно

CRUD in SQL

- Язык структурированных запросов (SQL) является языком запросов, используется по RDBMS

- Rails генерирует SQL выражения для выполнения на Руби

- 4 основные операции на строке таблицы: создать (C), читать (R), обновить атрибуты (U), удалить (D)

Ruby-сторона модели

- Наследование от ActiveRecord::Base
- связывает модель с БД
- производит CRUD операции с моделью
- Имя таблицы БД основывается на имени модели: Movie➔movies
- Названия колонок БД являются getters и setters
- Методы получения и установки не изменяют переменные!

Создание: new ≠ save

- Обязательный вызов save или save! на экземпляре AR model, чтобы сохранить изменения в БД
- '!' версия опасна: не сохраняет изменения при неудачной операции
- create просто объединяет создание новой сущности и сохранение
- После создания объект приобретает первичный ключ
- если x.id is nil or x.new_record? истинно, то x не сохранялся
- Поведение скопировано с ActiveRecord

Read: поиск в БД

- метод класса where ищет объекты по атрибутам

```
Movie.where("rating='PG'")
Movie.where('release_date < :cutoff and
rating = :rating',
:rating => 'PG', :cutoff => 1.year.ago)
Movie.where("rating=#{rating}") # BAD IDEA!
```

- вызовы where могут быть эффективно связаны в цепочку

```
kiddie = Movie.where("rating='G'")
old_kids_films =
kiddie.where "release_date < ?", 30.years.ago
```

Read: find_*

- поиск по id:

Movie.find(3) #exception if not foundMovie.find_by_id(3) # nil if not found

- динамический поиск по атрибутам

method_missing:Movie.find_all_by_rating('PG')
Movie.find_by_rating('PG')
Movie.find_by_rating!('PG')

Обновить: два варианта

• Изменить атрибуты и сохранить

```
objectm=Movie.find_by_title('The Help')
m.release_date='2011-Aug-10'
m.save!
```

• Изменить существующие атрибуты

```
objectMovie.find_by_title('The Help').
update_attributes!(
:release_date => '2011-Aug-10'
)
```

Обновление транзакционально - или все атрибуты обновлены или ни одного

Пусть у таблицы fortune_cookies есть поле fortune_text
какие из этих методов FortuneCookie <
ActiveRecord::Base не вернут silly fortune?

- def silly_fortune_1

```
@fortune_text + 'in bed'
end
```

- def silly_fortune_2

```
self.fortune_text + 'in bed'
end
```

- def silly_fortune_3

```
fortune_text + 'in bed'
end
```

- Они все вернут silly fortune

Rails Cookery #2

- Чтобы добавить новое действие в Rails app

1.Создайте маршрут в config/routes.rb

2. Добавьте действие в соответствующий app/controllers/*_controller.rb

3. Удостоверьтесь, что действию есть на что опираться в app/views/model/action.html.haml

4. Попробуем определить show действие и представление для него

Зоны ответственности MVC

• Модель

методы, чтобы управлять данными

Movie.where(...), Movie.find(...)

• Контроллер: получить данные от модели, сделать доступными для представление

```
def show
  @movie = Movie.find(params[:id])
end
```

Цель: показать данные, обеспечить взаимодействие с пользователем

show @movie

• Что еще пользователь может делать на этой странице?

• Как пользователь на нее попадет?

Как мы сюда попали: URI helpers

link_to movie_path(3)
index.
html.
haml
...
def show
  @movie = Movie.find(params[:id])
end
GET /movies/:id
{:action=>"show", :controller=>"movies"}
params[:id]←3

Что еще можно сделать?

- Как насчет того, чтобы дать возможность пользователю вернуться в список фильмов?

- RESTful URI:

- movies_path без аргументов ссылается на индексное действие

= link_to 'Back to List', movies_path

A) Маршрут состоит из URI и HTTP method

B) URI часть маршрута должна быть создана через Rails URI helpers

C) URI часть маршрута может быть создана через Rails URI helpers

- ☐ Только (A) is true
- ☐ Только (C) is true
- ☐ Только (A) and (B) are true
- ☐ Только (A) and (C) are true

Работа с формами

Создание ресурса обычно состоит из двух действий

- new: создать пустую форму
- create: подвердить заполненную форму

- Как создавать?
- Как заполнять?
- Что возвращать?

Чтобы создать новую подтверждаемую форму:

1. Определить действие, которое показывает форму

2. Определить действие, которое получает данные формы

3. Создайте маршруты, действия, представления для каждого из действий

• В представлении формы, элемент формы name перечисляет и контролирует что появится в результирующих параметрах

• Хелперы есть для многих используемых элементов

Создание формы

• анатомия формы в HTML

• действие и атрибуты метода (маршрут)

• только названные элементы формы будут отправлены в параметрах

Создание формы в Rails

• часто можно использовать URI helper для действия, так как это просто URI часть маршрута

• хелперы для полей формы создают приемлимые наименования для этих полей

Переадресация, Flash и Сессия

Какой вид должен присутствовать в создании действия?

- Перенаправлять пользователя на более удобную страницу:
- например, список фильмов, если создание успешно
- например, форма нового фильма, если неуспешно
- Redirect запускает новый HTTP запрос
- Как информировать пользователя о том, почему он был перенаправлены?
- Совет: flash[] — ведет себя как хэш, который существует до конца следующего запроса
- flash[:notice] по соглашению для информации
- flash[:warning] по соглашению для ошибок

Flash & Сессия

- session[]: как хэш, который всегда присутствует (не только до конца запроса)
- reset_session стирает всю сессию
- session.delete(:some_key), как для хэша
- По дефолту, cookies хранят всё содержимое сессии & flash
- Альтернатива: хранить сессии в таблицах БД
- Альтернатива: хранить сессии в “NoSQL” системе хранения, такой как memcache

Ben Bitdiddle: “Вы можете сохранить произвольные объекты в сессию.”

Что вы думаете?

Да — это сработает!

Да — но неудачная идея!

Нет, потому что нельзя загрузить в хэш случайные объекты

Нет, потому что сессия это не хэш, а только ведет себя похожим образом