0
Рейтинг

Evrone

Подписаться

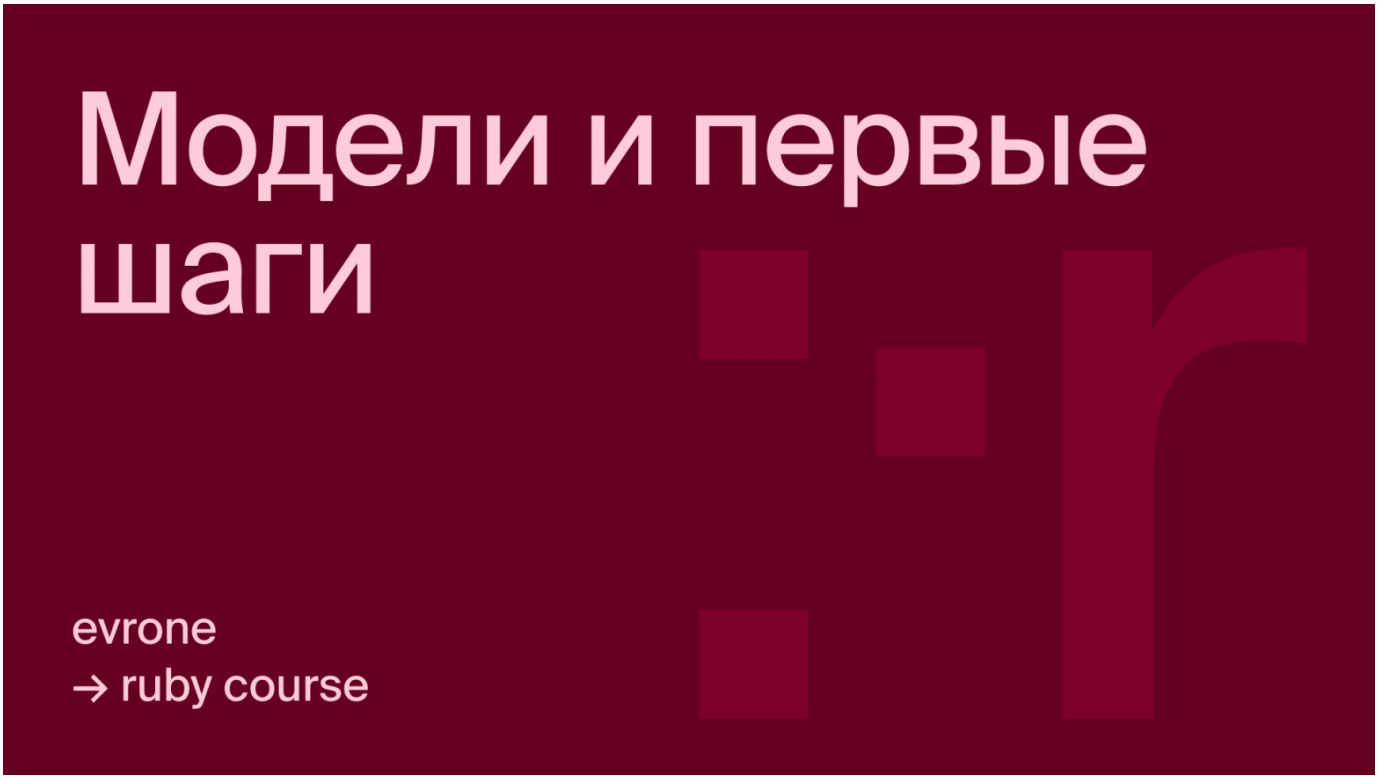
Evrone
19 мая в 19:04

Курс по Ruby+Rails. Часть 8. Модели и первые шаги

8 мин 1.8K

Блог компании Evrone, Ruby*, Ruby on Rails*

Туториал



Миграции — это механизм ActiveRecord, который позволяет вносить изменения в структуру базы данных: создавать и удалять таблицы, переименовывать, добавлять и удалять поля. Так как в них описываются все изменения, они обеспечивают консистентность структуры БД. Они, как и модели, представляют собой классы, написанные на Ruby и хранящиеся в папке `db/migrate` . Каждая миграция определяется временной меткой, с которой начинается имя файла. Эти метки должны быть уникальны, так как используются для контроля миграций в таблице `schema_migrations` , которая хранит все выполненные

миграции. Эти метки указывают на момент создания миграции и позволяют отслеживать хронологию их применения, что необходимо, например, в случае разворачивания приложения на новом сервере.

Создание миграций

Есть несколько способов создания миграций. Во-первых, если вы используете генератор для создания модели, по умолчанию он создает и миграцию. В этом случае вам остается только выполнить её.

Иногда бывает необходимость создать только миграцию (например, если вы хотите внести изменения в структуру уже существующей таблицы, для которой уже есть модель). Для этого можно воспользоваться генератором миграций. Его синтаксис совпадает с синтаксисом генератора моделей, то есть вам всего лишь нужно выполнить `rails g migration`, указав имя миграции и необходимые атрибуты.

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [option
```

Имя миграции должно отражать те изменения, которые она приносит. Поэтому, если, например, вы хотите создать таблицу posts, идеальным названием будет `CreatePosts`. Давайте продолжим работу с таблицей `projects`, которую создали в лекции о моделях, и добавим в неё поля `active` и `description`, при этом для `active` укажем тип `boolean`, а для `description` – `text`.

```
rails g migration AddDescriptionAndActiveToProject description:text active:boolean
```

Если правильно подойти к имени миграции, ActiveRecord поймет, что именно мы хотим, и сгенерирует необходимый код. Например, в нашем случае не только создастся файл миграции, но в нём появится сгенерированный код, который добавит нужные колонки в таблицу `projects`. Это происходит благодаря принципу «соглашения над конфигурацией».

```
class AddDescriptionAndActiveToProject < ActiveRecord::Migration[5.2]
  def change
    add_column :projects, :description, :text
    add_column :projects, :active, :boolean
  end
end
```

Если вы следуете соглашениям ActiveRecord, он возьмет большую часть работы на себя. Таким образом можно прямо из консоли создавать готовые миграции. Например, для добавления таблиц, просто используйте формулу `Create<Table>` и далее список атрибутов, а для добавления колонки в уже существующую таблицу формула выглядит, как `Add<Column>To<Table>`. Для удаления столбца, соответственно, используется `Remove<Column>From<Table>`. Если вы хотите добавить несколько столбцов в имени достаточно указать один из них, а остальные прописать в атрибутах.

Давайте попробуем сгенерировать миграцию для создания таблицы `users`, а затем попробуем добавить и удалить из неё столбцы. Для создания воспользуемся командой `rails g migration CreateUsers` и укажем для начала атрибуты `name` и `email` типа `string`. Кстати, тип `string` является типом по умолчанию, поэтому его можно не указывать при генерации. Он и так автоматически подставится в миграции. Кроме `string`, допустимыми является ряд других типов. Отдельного внимания заслуживает тип `references`, так как он указывает на наличие связи `belongs_to`, то есть используется для хранения внешних ключей. Например, если мы захотим создать таблицу `profiles`, которую решим связать с таблицей `users`, для этого нам пригодится столбец `user_id` типа `references`.

Возможные типы:

- integer
- primary_key

- decimal
- float
- boolean
- binary
- string
- text
- date
- time
- datetime

Итак, у нас готова команда для создания таблицы `users`. После её выполнения появится файл миграции, который будет содержать инструкцию для создания новой таблицы с указанными нами полями. Как видите, ActiveRecord нас правильно понял и использовал хелпер `create_table`. Но помимо указанных нами и прописанных в классе миграции полей, создано ещё одно — поле первичного ключа, которое называется `id`, и хранит в себе уникальный ключ записи. Обратите внимание, что в отличие от случая, когда миграции создаются с помощью генератора моделей, в этой миграции нет временных меток! Об этом стоит помнить если, например, вы решите создать модель вручную, а миграцию к неё сгенерировать. Потому что таким образом вы можете незаметно для себя потерять возможность отслеживать время создания и изменения записей.

```
class CreateUsers < ActiveRecord::Migration[5.2]
  def change
    create_table :users do |t|
      t.string :name
      t.string :email
    end
  end
end
```

Теперь давайте создадим миграцию, которая добавит новые поля пользователям. Для этого назовем миграцию по схеме «Добавить-что-то-куда-то», то есть `Add<Column>To<Table>`. Например, добавим поля `birthday` с типом `date` и `moderator` с типом `boolean`. И снова Rails нас понимает и создает миграцию с готовым кодом, добавив теперь методы `add_column`, а всё, что нам остается — это её выполнить.

```
# rails g migration AddBirthdayToUser birthday:date moderator:boolean
class AddBirthdayToUser < ActiveRecord::Migration[5.2]
  def change
    add_column :users, :birthday, :date
    add_column :users, :moderator, :boolean
  end
end
```

Как видите, создавать миграции с помощью генератора очень просто, но ничто не мешает вам делать это вручную или вносить изменения в уже созданные миграции, но только до того, как они будут применены. Если вы планируете вручную создавать миграции, то стоит поближе познакомиться с миграционными методами.

Написание миграций

Вы уже успели познакомиться с методами `create_table` и `add_column`. И наверняка заметили, что они вызываются внутри метода `change`. Этот метод является основным при написании миграций. Дело в том, что иногда изменения в базе приходится отменять, и для этого производится откат миграций. Так вот метод `change` позволяет нам использовать ряд миграционных методов, которые умеет самостоятельно откатывать. К ним относятся уже известные вам `create_table` и `add_column`, а также ряд других. С полным списком лучше познакомиться в документации, но к основным можно отнести методы для добавления индекса, временных меток и внешних ключей, удаления таблиц, колонок, индексов и ключей, и их переименования.

Когда вы пишете миграции, всегда нужно задумывать о том, сможет ли Rails безболезненно откатить её, поэтому даже при написании миграции для удаления столбца важно указать не только его имя, но и тип, значение по умолчанию и другие опции, если они присутствуют. Это может в случае отката удаления столбца создать его заново с необходимыми параметрами.

Иногда бывают ситуации, когда на автоматический откат миграции рассчитывать не приходится. Например, если вы переносите данные из одного столбца в другой или между таблицами. В этом случае вместо метода `change` вам лучше обратиться к двум другим. Первый - это метод `up`. В нём описываются те изменения, которые необходимо выполнить миграции. Метод `down` описывается в той же миграции, но он содержит код, после выполнения которого схема базы данных должна прийти к исходному состоянию. То есть, если вы, например, создаете таблицу в методе `up`, то в методе `down` должны её удалить. Таким образом, метод `up` выполняется в момент выполнения миграции, а метод `down` в момент её отката.

```ruby
class AddHomePageUrlToUsers < ActiveRecord::Migration[5.2]
  def up
    add_column :users, :home_page_url, :string
    rename_column :users, :email, :email_address
  end

  def down
    rename_column :users, :email_address, :email
    remove_column :users, :home_page_url
  end
end
```

Раз уж мы заговорили об откате миграций, то давайте разберем, как это сделать. Откат последней миграции возможет с помощью команды `rails db:rollback`. Это либо обратит метод `change`, либо вызовет метод `down`. Если вы хотите отменить несколько миграций, сделать это можно указав параметр `STEP`, передав ему количество отменяемых миграций.

Миграционные методы

Теперь давайте подробнее остановимся на миграционных методах. Начнем с тех, которые относятся к таблицам. Прежде всего, это `create_table`, который позволяет создать новую таблицу. Внутри блока перечисляются поля таблицы с указанием их типов.

```ruby
create_table :users do |t|
  t.string :name
  t.string :email, null: false
end
```

Изменять уже существующие таблицы можно с помощью метода `change_table`. Он умеет не добавлять столбцы, переименовывать их и удалять. В нашем примере мы удаляем столбцы `birthday` и `moderator`, добавляем столбец `address` и переименовываем `name` в `nickname`. И всё это одной миграцией.

```ruby
change_table :users do |t|
  t.remove :birthday, :moderator
  t.string :address
  t.rename :name, :nickname
end
```

Для работы со столбцами есть свои методы. Прежде всего, это `add_column`, которому сообщается имя таблицы, в которую стоит добавить столбец, имя этого столбца, его тип и ряд модификаторов. В качестве модификатора можно, например, использовать `default`, что позволит задать значение по умолчанию для создаваемой колонки. А модификатор `null` со значением `false` сделает невозможным сохранение записей с нулевым значением в соответствующем атрибуте. Модификатор `index`, в свою очередь, добавляет индекс для столбца.

```
add_column :users, :moderator, :boolean, deafult: false, null: false
```

Обратным ему методом является `remove_column`. Как уже говорилось ранее, при написании миграции, удаляющей столбец, важно прописывать все имеющиеся у столбца модификаторы. Это поможет в случае необходимости откатить миграция и вернуть столбец в исходное состояние (а вернее, создать новый с теми же характеристиками).

По аналогии с `change_table` существует метод `change_column`. Как вы уже поняли, он позволяет изменять столбцы. Например, мы можем изменить тип столбца `address` на `text`. Но обратите внимание, что в этом методе указан только новый тип, и нет информации о предыдущем. Поэтому метод `change_table` является необратимым и его следует использовать только за пределами метода `change` в связке `up` и `down` методов.

```
add_column :users, :change_column, :text
```

Если вы хотите изменить значение по умолчанию для поля или модификатор `null`, то для этого есть два отдельным метода – `change_column_default` и `change_column_null` соответственно.

```
change_column_null :users, :nickname, false
```

```
change_column_default :users, :moderator, from: false, to: true
```

Best practices

Т.к. миграции работают с базой данных, они могут оказывать влияние на функционирование приложения. Например, переименование названия или удаление колонки

для таблицы, которая уже используется, приведет к ошибкам в работе.

Хорошим местом для начала может послужить проект strong migrations. В нем описаны типичные проблемы, которые могут возникнуть при использовании миграций, и способы их решения.

Например, для случая удалении колонки:

- сначала нужно объявить колонку как неиспользуемую

```
class Project < ApplicationRecord
  self.ignored_columns = ['description']
end
```

  Таким образом весь код ActiveRecord будет ее игнорировать, и мы сможем убедиться, что все работает должным образом;

- и только в следующем релизе удалить колонку из таблицы.

Также популярная проблема – добавления индекса в существующую таблицу. В Postgres этот процесс вызывает блокировку таблицы на запись, что для активных таблиц может быть неприемлемо.

Для того, чтобы избежать этого, необходимо добавлять индекс конкурентно:

```
class AddCodeIndexToProjects < ActiveRecord::Migration[6.1]
  disable_ddl_transaction!

  def change
    add_index :projects, :code, algorithm: :concurrently
  end
end
```

strong_migrations можно установить как gem, он будет находить проблемы в миграции и сообщать о них заблаговременно.

Это большая тема заслуживающая большого внимания, но как минимум теперь вы знаете, для чего они нужны и как их создавать. Миграции - это механизм ActiveRecord, который позволяет вносить изменения в схему базы данных не используя SQl. ActiveRecord

предоставляет широкие возможности для манипуляций со базой, беря на себя большую часть работы, автоматизируя многие вещи благодаря принципу соглашения над конфигурацией. Это позволяет генерировать готовые миграции, сразу содержащие код, вносящий необходимые изменения, и минимизировать код для отката миграций.

Теги: обучение программированию, ruby, миграции, rubyonrails, ruby on rails

Хабы: Блог компании Evrone, Ruby, Ruby on Rails

Evrone

Компания

Подписаться

Сайт Сайт Сайт Сайт

24 Карма 0 Рейтинг

Evrone @Evrone

Пользователь

Подписаться

Комментировать

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

mr-pickles
18 часов назад

16-, 8- и 4-битные форматы чисел с плавающей запятой

Средний 15 мин 6.7K

+68 88 28

ru_vds
15 часов назад

Искусство создания понятных графиков

Средний 7 мин 3.5K

Туториал Перевод

+38 79 2

dlinyj
20 часов назад

Измерение скорости чтения-записи носителей с помощью утилиты dd

Средний 11 мин 4K

Кейс

+38 48 29

SLY_G
10 часов назад

Известная, но очень странная кошачья повадка: кошки, приносящие игрушку — это эволюционная загадка

5 мин 7.6K

Перевод

+30 22 38

alizar
19 часов назад

Автономия разработчиков. Как устроены компании нового типа

Простой 7 мин 4K

Мнение

+25 20 21

Petr0v1
14 часов назад

Марс всё ближе: несмотря на проблемы, запуск Starship можно считать успешным

4 мин 3.9K

+16 10 11

Doctor_IT
14 часов назад

Сможет ли високосная минута решить проблему синхронизации часов?

6 мин 2.7K

+15 14 14

ne_volkov
19 часов назад

Как в Ozon следят за чувствительной информацией в логах и при чем тут Толкиен?

Простой 10 мин 3.2K

Кейс

+15 22 3

badcasedaily1
12 часов назад

Garbage Collection и JVM

Простой 17 мин 2.1K

Обзор

+13 67 3

SLY_G
16 часов назад

Дайджест научпоп-новостей за неделю, о которых мы ничего не писали

8 мин 1.6K

Дайджест

+13 5 1

Показать еще

ИНФОРМАЦИЯ

Сайт	evrone.ru
Дата регистрации	2 августа 2022
Дата основания	2008
Численность	101–200 человек
Местоположение	Россия

БЛОГ НА ХАБРЕ

19 мая в 19:04

Курс по Ruby+Rails. Часть 8. Модели и первые шаги

1.8K 0

25 апр в 14:29

Что нового в Proxmox 7.4

6.9K 23

6 апр в 14:00

Как добавить сторонние драйверы в установочный образ VMware ESXi 8

4.9K 18

22 мар в 19:40

Курс по Ruby+Rails. Часть 7. Модели и ActiveRecord

2.6K 1

27 фев в 19:55

Подробный гайд по Docker на M1

13K 6