Как правильно готовить Ruby on Rails для работы с PostgreSQL

Степан Кирюшкин, 404 Group

Какие еще Ruby еще и on Rails?

Ruby — динамический, рефлексивный, интерпретируемый высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования.

Ruby on Rails — полноценный, многоуровневый фреймворк для построения веб-приложений, использующих базы данных, который основан на архитектуре Модель-Представление-Контроллер (Model-View-Controller, MVC).

PostgreSQL — все еще лучшая Open Source база данных в мире.

Конфигурация базы данных в RoR

```
$ cat config/database.yml
development: # окружение, ниже еще test и production
  adapter: postgresql
  encoding: unicode
  database:
  username:
  password:
  pool: 5 # макс. количество подключений от приложения
  timeout: 5000 # макс. время ожидания подключения
  prepared_statements: false
...
```

Зачем в RoR включены prepared statements?

Плюсы:

1. Один и тот же подготовленный запрос можно использовать несколько
   раз для разных данных, тем самым сокращая код.
2. Запросы со связываемыми переменными лучше кэшируются
   сервером, сокращая время синтаксического разбора.

3. Запросы со связываемыми переменными обладают готовой
   встроенной защитой от SQL-инъекций.

А минусы prepared statements?

Невозможность использовать pool connection manager (ex. PgBouncer)

Почему?

Rails используют prepared statements для компиляции планов исполнения запросов (для повышения производительности).

Скомпилированные планы хранятся в сессии бд, т. е. привязаны к конкретному соединению.

Для нашего приложения раз от раза от PgBouncer может достаться другое соединение с базой, где наших планов нет или на их месте лежат другие.

Особенно больно при использовании асинхронных очередей в БД (для Rails это que, delayed job, active job и тд)

Если вы твердо уверены что pool manager нужен >>

RoR 3.1.x — патч на адаптер, никакие опции в конфигурации базы не помогают.

RoR 3.2.x — добавлена опция prepared_statements: false, но все равно патч на адаптер.

RoR 4.x.x — опция prepared_statements: false работает как ожидается.

>> Если вы твердо уверены что pool manager нужен

Собственно патч для RoR 3.x.x

```
$ cat config/initializers/active_record.rb
module ActiveRecord
  module ConnectionAdapters
    class PostgreSQLAdapter < AbstractAdapter
      def exec_query(sql, name = 'SQL', binds = [])
        n = 1
        binds.map { |num, val| sql.gsub!("$#{n}", "'#{val}'"); n += 1 } unless binds.
empty?
        log(sql, name, binds) do
          result = exec_no_cache(sql, binds)
          ret = ActiveRecord::Result.new(result.fields, result_as_array(result))
          result.clear
          return ret
        end
      …
end
```

Подходы к выкатке изменений в БД

Если вы используете PostgreSQL на полную катушку (хранимые процедуры, arrays, hstore, postgis и другие интересные слова) то обязательно используйте формат схемы данных SQL, вместо стоящего по умолчанию рубиновго DSL.

В противном случае ваши тесты не будут работать.

```
$ cat config/application.rb
class Application < Rails::Application
  ...
  config.active_record.schema_format = :sql
  ...
end
```

Подходы к выкатке изменений в БД

1. Рубиновый DSL

а. Поддержка всего современного в PostgreSQL.

б. Все, чего нет в RoR, можно нарисовать через обычный SQL накатывая его через рубиновые миграции прямо в базу.

в. Повсеместная интеграция с автоматическими инструментами тестирования, раскатки и т.п.

Подходы к выкатке изменений в БД

2. Расовый (PL/) SQL

а. Поддержка вообще всего в PostgreSQL с родной подсветкой.

б. Если ваш новопришедший DBA не видел рубинового языка для создания миграций, то он скажет вам спасибо.

в. Ничего вам не мешает заставить стандартные рубиновые миграции лазить в SQL файлы, читать их и выполнять в базу, тем самым получится интеграция со всеми инструментами из п.1 >>

Подходы к выкатке изменений в БД

3. “Волшебство” (https://github.com/denismilovanov/PostgresqlDeployerGUI)

а. Утилита для построения SQL-миграций из схемы базы данных.

б. Ваша схема должна быть уложена в git.

в. При невозможности автоматизированной раскати сообщает разработчику.

г. Еще плюшки всякие.

4. Ручная раскатка через DBA

а. Многие из нас делают это

Работа с PostgreSQL в RoR

Создание таблиц

```
# SQL
CREATE TABLE models (
    id bigserial PRIMARY KEY CHECK (id > 0),
    int_field integer NOT NULL,
    array_field text[],
    hsore_field hstore
);
# Рубиновый DSL
create_table(:models, id: false) do |t|
  t.column :id, 'bigserial PRIMARY KEY CHECK (id > 0)'
  t.integer :int_field, null: false
  t.text :array_field, array: true
  t.hstore :hstore_field
end
```

Работа с PostgreSQL в RoR через ORM

Общее

```
class Model < ActiveRecord::Base
end
Model.first
# SELECT models.* FROM models ORDER BY id LIMIT 1;
Model.find(10)
# SELECT models.* FROM models WHERE id = 10 LIMIT 1;
Model.take
# SELECT models.* FROM models LIMIT 1;
Model.find_by(field: 'asd')
# SELECT models.* FROM models WHERE field='asd' LIMIT 1;
```

Работа с PostgreSQL в RoR

Хранимые процедуры

Вызов хп в RoR может быть реализован, например, так

```
class Model < ActiveRecord::Base
  def self.my_awesome_procedure(parameter1, parameter2, ...)
    self.select('t.*')
        .from(
          sprintf('schema.my_awesome_procedure(%d, %s, ...) AS t',
            self.sanitize(parameter1),
            self.sanitize(parameter2),
            ...
          )
        )
  end
end
```

Работа с PostgreSQL в RoR

Хранимые процедуры

```
Model.my_awesome_procedure
# SELECT t.* FROM schema.my_awesome_procedure(parameter1, parameter2,
...);
# => [#<Model id: 31, int_field: 1, array_field: [...], hstore_field:
{...}>, #<Model ...
```

Если в хп возвращает записи с полями как в таблице модели, то RoR их распарсит и сделает приведение типов. Остальные поля будут типа String.

```
Model.my_awesome_procedure.first.any_joined_bigint_field.class
# => String
```

Работа с PostgreSQL в RoR

Массивы

RoR поддерживает массивы в PostgreSQL. При работе с базой они разбираются туда (при селекте) и обратно (при записи). Различные дополнительные возможности можно добавлять, например, так

```
class Model < ActiveRecord::Base
  scope :any_in_array, -> (elements) {
    where('array_field && ARRAY[?]', elements)
  }
  scope :all_in_array, -> (elements) {
    where(array_field @> ARRAY[?]', elements)
  }
end
```

Работа с PostgreSQL в RoR

Hstore

Верно все то же самое, что и для массивов. Дополнительно

```
class Model < ActiveRecord::Base
  scope :has_key, -> (key) {
    where('defined(hstore_field, ?)', key)
  }
  scope :has_value, -> (key, value) {
    where('hstore_field -> ? = ?', key, value)
  }
end
```

Работа с PostgreSQL в RoR

Естественно, все это может работать вместе:

```
Model.my_awesome_procedure(parameter1, parameter2, ...)
      .any_in_array('condition1')
      .has_value('condition2', 'value')
```

Генерирует запрос:

```
SELECT t.*
FROM schema.my_awesome_procedure(parameter1, parameter2, ...)
WHERE
  array_field && ARRAY['condition1'] AND
  hstore_field -> 'condition2' = 'value';
```

Работа с PostgreSQL в RoR

Отлов ошибок

```
begin
  # SOME HARD SQL OVER ActiveRecord::Base.connection.execute
rescue => error
  p error.class.name
  p error.original_exception.class.name
end

# "ActiveRecord::StatementInvalid"
# "PG::RaiseException"
```

Помимо RaiseException там еще много всего интересного.

“Проблемки” ORM RoR

N + 1 запросов

```
clients = Client.limit(10)

clients.each do |client|
  puts client.address.postcode
end
```

= 11 запросов, Карл!

“Проблемки” ORM RoR

Решение проблемы N + 1 запросов

```
clients = Client.includes(:address).limit(10)

clients.each do |client|
  puts client.address.postcode
end
```

Для вложенных связей

```
Category.includes(articles: [{ comments: :guest }, :tags]).limit(10)
```

“Проблемки” ORM RoR

Решая проблему N + 1 запросов можно что-нибудь себе отстрелить

```
Article.includes(:comments).where('comments.visible = true')
```

Генерирует запрос:

```
SELECT "articles"."id" AS t0_r0, ... "comments"."updated_at" AS t1_r5
FROM "articles"
LEFT OUTER JOIN "comments" ON "comments"."article_id" = "articles"."id"
WHERE (comments.visible = 1)
```

“А что там может быть за ...?”

“Проблемки” ORM RoR

```
SELECT DISTINCT "addresses"."id" AS t0_r0, "addresses"."latitude" AS t0_r1, "addresses"."
longitude" AS t0_r2, "addresses"."house" AS t0_r3, "addresses"."street" AS t0_r4, "addresses"."
city" AS t0_r5, "addresses"."zip" AS t0_r6, "addresses"."state" AS t0_r7, "addresses"."country"
AS t0_r8, "addresses"."created_at" AS t0_r9, "addresses"."updated_at" AS t0_r10, "addresses"."
street2" AS t0_r11, "addresses"."custom_latitude" AS t0_r12, "addresses"."custom_longitude" AS
t0_r13, "addresses"."name" AS t0_r14, "addresses"."company" AS t0_r15, "addresses"."archived"
AS t0_r16, "orders"."id" AS t1_r0, "orders"."customer_id" AS t1_r1, "orders"."
cleaning_address_id" AS t1_r2, "orders"."billing_address_id" AS t1_r3, "orders"."start_at" AS
t1_r4, "orders"."finish_at" AS t1_r5, "orders"."planned_at" AS t1_r6, "orders"."started_at" AS
t1_r7, "orders"."finished_at" AS t1_r8, "orders"."payed_price" AS t1_r9, "orders"."status" AS
t1_r10, "orders"."pin" AS t1_r11, "orders"."comment" AS t1_r12, "orders"."created_at" AS
t1_r13, "orders"."updated_at" AS t1_r14, "orders"."name" AS t1_r15, "orders"."company" AS
t1_r16, "orders"."phone" AS t1_r17, "orders"."coupon_code" AS t1_r18, "orders"."
coupon_discount" AS t1_r19, "orders"."location_id" AS t1_r20, "orders"."location_discount" AS
t1_r21, "orders"."job_count" AS t1_r22, "orders"."bonus" AS t1_r23, "orders"."organization_id"
AS t1_r24, "orders"."bill_id" AS t1_r25, "orders"."cleaning_count_discount" AS t1_r26,
"orders"."created_by" AS t1_r27, "orders"."price_group_id" AS t1_r28, "orders"."travel_charge"
AS t1_r29, "orders"."departed_at" AS t1_r30, "orders"."arrived_at" AS t1_r31, "orders"."
request_feedback" AS t1_r32, "orders"."feedback_link" AS t1_r33, "orders"."
feedback_requested_at" AS t1_r34, "orders"."rating" AS t1_r35, "orders"."feedback" AS t1_r36,
"orders"."profit_center_code" AS
```

“Проблемки” ORM RoR

```
t1_r37, "orders"."feedback_received_at" AS t1_r38, "orders"."reorder_bonus" AS t1_r39, "orders"."
paid_to" AS t1_r40, "orders"."cleaner_id" AS t1_r41, "orders"."station_id" AS t1_r42, "orders"."
cleaner_comment" AS t1_r43, "orders_for_billings_addresses"."id" AS t2_r0,
"orders_for_billings_addresses"."customer_id" AS t2_r1, "orders_for_billings_addresses"."
cleaning_address_id" AS t2_r2, "orders_for_billings_addresses"."billing_address_id" AS t2_r3,
"orders_for_billings_addresses"."start_at" AS t2_r4, "orders_for_billings_addresses"."finish_at" AS
t2_r5, "orders_for_billings_addresses"."planned_at" AS t2_r6, "orders_for_billings_addresses"."
started_at" AS t2_r7, "orders_for_billings_addresses"."finished_at" AS t2_r8,
"orders_for_billings_addresses"."payed_price" AS t2_r9, "orders_for_billings_addresses"."status" AS
t2_r10, "orders_for_billings_addresses"."pin" AS t2_r11, "orders_for_billings_addresses"."comment"
AS t2_r12, "orders_for_billings_addresses"."created_at" AS t2_r13,
"orders_for_billings_addresses"."updated_at" AS t2_r14, "orders_for_billings_addresses"."name" AS
t2_r15, "orders_for_billings_addresses"."company" AS t2_r16, "orders_for_billings_addresses"."
phone" AS t2_r17, "orders_for_billings_addresses"."coupon_code" AS t2_r18,
"orders_for_billings_addresses"."coupon_discount" AS t2_r19, "orders_for_billings_addresses"."
location_id" AS t2_r20, "orders_for_billings_addresses"."location_discount" AS t2_r21,
"orders_for_billings_addresses"."job_count" AS t2_r22, "orders_for_billings_addresses"."bonus"
```

“Проблемки” ORM RoR

```
AS t2_r23, "orders_for_billings_addresses"."organization_id" AS t2_r24,
"orders_for_billings_addresses"."bill_id" AS t2_r25, "orders_for_billings_addresses"."
cleaning_count_discount" AS t2_r26, "orders_for_billings_addresses"."created_by" AS t2_r27,
"orders_for_billings_addresses"."price_group_id" AS t2_r28, "orders_for_billings_addresses"."
travel_charge" AS t2_r29, "orders_for_billings_addresses"."departed_at" AS t2_r30,
"orders_for_billings_addresses"."arrived_at" AS t2_r31, "orders_for_billings_addresses"."
request_feedback" AS t2_r32, "orders_for_billings_addresses"."feedback_link" AS t2_r33,
"orders_for_billings_addresses"."feedback_requested_at" AS t2_r34,
"orders_for_billings_addresses"."rating" AS t2_r35, "orders_for_billings_addresses"."feedback"
AS t2_r36, "orders_for_billings_addresses"."profit_center_code" AS t2_r37,
"orders_for_billings_addresses"."feedback_received_at" AS t2_r38,
"orders_for_billings_addresses"."reorder_bonus" AS t2_r39, "orders_for_billings_addresses"."
paid_to" AS t2_r40, "orders_for_billings_addresses"."cleaner_id" AS t2_r41,
"orders_for_billings_addresses"."station_id" AS t2_r42, "orders_for_billings_addresses"."
cleaner_comment" AS t2_r43, "purchases"."id" AS t3_r0, "purchases"."product_price_group_id" AS
t3_r1, "purchases"."customer_id" AS t3_r2, "purchases"."shipping_address_id" AS t3_r3,
"purchases"."billing_address_id" AS t3_r4,
```

“Проблемки” ORM RoR

```
"purchases"."created_by" AS t3_r5, "purchases"."token" AS t3_r6, "purchases"."shipping" AS
t3_r7, "purchases"."coupon_code" AS t3_r8, "purchases"."coupon_discount" AS t3_r9,
"purchases"."comment" AS t3_r10, "purchases"."status" AS t3_r11, "purchases"."purchased_at" AS
t3_r12, "purchases"."created_at" AS t3_r13, "purchases"."updated_at" AS t3_r14, "purchases"."
purchase_invoice_id" AS t3_r15, "purchases"."payment" AS t3_r16, "purchases"."pickup" AS
t3_r17, "purchases"."tracking_number" AS t3_r18, "purchases"."paper_invoice_sent_at" AS t3_r19,
"purchases_for_billings_addresses"."id" AS t4_r0, "purchases_for_billings_addresses"."
product_price_group_id" AS t4_r1, "purchases_for_billings_addresses"."customer_id" AS t4_r2,
"purchases_for_billings_addresses"."shipping_address_id" AS t4_r3,
"purchases_for_billings_addresses"."billing_address_id" AS t4_r4,
"purchases_for_billings_addresses"."created_by" AS t4_r5, "purchases_for_billings_addresses"."
token" AS t4_r6, "purchases_for_billings_addresses"."shipping" AS t4_r7,
"purchases_for_billings_addresses"."coupon_code" AS t4_r8, "purchases_for_billings_addresses"."
coupon_discount" AS t4_r9, "purchases_for_billings_addresses"."comment" AS t4_r10,
"purchases_for_billings_addresses"."status" AS t4_r11, "purchases_for_billings_addresses"."
purchased_at" AS t4_r12, "purchases_for_billings_addresses"."created_at" AS t4_r13,
"purchases_for_billings_addresses"."updated_at" AS t4_r14,
```

“Проблемки” ORM RoR

```
"purchases_for_billings_addresses"."purchase_invoice_id" AS t4_r15,
"purchases_for_billings_addresses"."payment" AS t4_r16, "purchases_for_billings_addresses"."
pickup" AS t4_r17, "purchases_for_billings_addresses"."tracking_number" AS t4_r18,
"purchases_for_billings_addresses"."paper_invoice_sent_at" AS t4_r19 FROM "addresses" LEFT
OUTER JOIN "orders" ON "orders"."cleaning_address_id" = "addresses"."id" LEFT OUTER JOIN
"orders" "orders_for_billings_addresses" ON "orders_for_billings_addresses"."
billing_address_id" = "addresses"."id" LEFT OUTER JOIN "purchases" ON "purchases"."
shipping_address_id" = "addresses"."id" LEFT OUTER JOIN "purchases"
"purchases_for_billings_addresses" ON "purchases_for_billings_addresses"."billing_address_id" =
"addresses"."id" WHERE (orders.customer_id = 3282 OR orders_for_billings_addresses.customer_id
= 3282 OR purchases.customer_id = 3282 OR purchases_for_billings_addresses.customer_id = 3282)
```

“Проблемки” ORM RoR

Решаем проблему с решением проблемы...

```
comments_join = <<-SQL
  LEFT JOIN comments ON comments.article_id = articles.id
SQL

Article.select('articles.*, comments.visible')
       .joins(comments_join)
       .where('comments.visible = true')
```

Еще можно устроить тюнинг запросов RoR >>

```
items = Item.where('created_at < ?', 2.days.from_now)
# SELECT * FROM items WHERE created_at < '...';
# => #<ActiveRecord::Relation [#<Item id: 31, name: "Chair" ...
```

```
items = Item.select(:id).where('created_at < ?', 2.days.from_now)
# SELECT id FROM items WHERE created_at < '...';
=> #<ActiveRecord::Relation [#<Item id: 31>, #<Item id: 32>, ...
```

>> Еще можно устроить тюнинг запросов RoR

```ruby
ids = Item.where('created_at < ?', 2.days.from_now).collect(&:id)
# SELECT * FROM items WHERE created_at < '...';
# => [31, 32, 33]
```

```ruby
ids = Item.where('created_at < ?', 2.days.from_now).pluck(:id)
# SELECT id FROM items WHERE created_at < '...';
# => [31, 32, 33]
```

Кеширование повторяющихся запросов в RoR

```
def create
  @client = Client.find(params[:id])

  if @client.update_with_my_procedure(params[:client]) # Сохранение с
помощью хп
    # @client.find(@client.id) Неверно. Запрос в кеше - он не выполнится
    @client.reload

    # Client.uncached { ... }

    # Рисуем новый @client
  else
    # Рисуем ошибки
  end
end
```

Что еще нужно обдумать перед разработкой?

RoR ни в какой версии не поддерживают композитные primary keys. Для этого существует гем “composite_primary_keys”.

1. Поддержка рельсовых миграций, связок has_many ... through и даже отстраивание роутов (sic!)

2. Важно! Если вы складываете в таблицы-связки какие-то дополнительные данные и их нужно редактировать, лучше использовать методы edit связываемых элементов и accepts_nested_attributes_for или принимать данные и складывать их в специальной обработке.

https://github.com/composite-primary-keys/composite_primary_keys

Рекомендательные блокировки

Рекомендательные блокировки (РБ) - это удобный механизм (на уровне бизнес-логики) для организации конкурентного доступа к данным. РБ являются вспомогательным механизмом, с помощью которого приложения могут контролировать доступ к какому-то ресурсу.

Для реализации в RoR используется гем “with_advisory_lock”.

1. Rails 3.x.x, 4.x.x
2. Сам разбирается в РБ уровня сессии и транзакции (нужен патч из issues)
3. Использует SAVEPOINT

Подробнее про РБ: http://pgcookbook.ru/article/advisory_locks_pitfalls.html

https://github.com/mceachen/with_advisory_lock

Спасибо за внимание!

Вопросы?

Степан Кирюшкин, 404 Group
s.kiryushkin@404-group.com