SQL 脳から見た Ruby

@yancya

大江戸 Ruby 会議 05
2015-11-08

Powered by Rabbit 2.1.8 and COZMIXNG

これから話すひと

@yancya

- ３児の父 Rubyist
- PostgreSQL と BigQuery が好き
- つくばエクスプレスユーザー

大江戸 Ruby 会議 04

- ビッグっぽいデータについて
- すごく良かったです

大江戸 Ruby 会議 04

- つまり
- Ruby 会議で SQL の話をするのは間違ってなかった

大江戸 Ruby 会議 04

- SQL 枠が出来たと見て、よろしいですね？

SQL と Ruby

- SQL におけるリレーションの実体はタプルの集合（ということにしておく）
- タプルの集合っぽい２次元配列なら Ruby でも扱えるハズ
- 同等の事が出来るかどうか、ちょっとくらべてみよう

SQL と Ruby

サンプルテーブル　users

name	birthday
ちゆ	1999-02-14
ちゆ	2003-02-14
もなみ	2002-02-09
もなみ	2006-02-09
メモリ	2003-11-02
メモリ	2004-11-02

SQL と Ruby

```
users = [
    ['ちゆ', '1999-02-14'],
    ['ちゆ', '2003-02-14'],
    ['もなみ', '2002-02-09'],
    ['もなみ', '2006-02-09'],
    ['メモリ', '2003-11-02'],
    ['メモリ', '2004-11-02'],
]
```

SQL と Ruby

```
SELECT name, birthday
  FROM users
```

- SQL の SELECT

SQL と Ruby

```
users.
  map { |name, birthday| [name, birthday] }
```

- SELECT 相当の処理は Ruby ではこう書く

SQL と Ruby

```
SELECT name, birthday
  FROM users
 WHERE name = 'もなみ'
```

- SQL の WHERE

SQL と Ruby

```
users.select { |name, _| name == 'もなみ' }
#=> [["もなみ", "2002-02-09"], ["もなみ", "2006-02-09"]]
```

- WHERE 相当の処理は Ruby ではこう書く

SQL と Ruby

```
-- name 毎の件数
   SELECT name, COUNT(*)
     FROM users
GROUP BY name
```

- SQL の GROUP BY

SQL と Ruby

```
users.
  group_by { |name, _| name }.
  map { |name, tuples| [name, tuples.size] }
#=> [["ちゆ", 2], ["もなみ", 2], ["メモリ", 2]]
```

- GROUP BY 相当の処理は Ruby ではこう書く

SQL と Ruby

```
-- birthday で並び替え
   SELECT name, birthday
     FROM users
ORDER BY birthday
```

- SQL の ORDER BY

SQL と Ruby

```
users.sort_by { |_, birthday| birthday }
#=> [["ちゆ", "1999-02-14"],
#=>  ["もなみ", "2002-02-09"],
#=>  ["ちゆ", "2003-02-14"],
#=>  ["メモリ", "2003-11-02"],
#=>  ["メモリ", "2004-11-02"],
#=>  ["もなみ", "2006-02-09"]]
```

- ORDER BY 相当の処理は Ruby ではこう書く

SQL と Ruby

```
-- 件数が2件以上の行に絞り込む
   SELECT name
     FROM users
GROUP BY name
   HAVING count(*) > 1
```

- SQL の HAVING

SQL と Ruby

```
users.
  group_by { |name, _| name }.
  select { |_, tuples| tuples.size > 1 }.
  map { |name, _| [name] }
```

- HAVING 相当の処理は Ruby ではこう書く

SQL と Ruby

```
SELECT *
  FROM users
 LIMIT 2
```

- SQL の LIMIT

SQL と Ruby

```
users.take(2)
```

- LIMIT 相当の処理は Ruby ではこう書く

SQL と Ruby

```
SELECT users.name, users.birthday, urls.url
  FROM users
  JOIN urls
    ON users.name = urls.name
```

- SQL の JOIN
- INNER JOIN は属性を付加するだけじゃなくて、絞り込みにも使える

SQL と Ruby

```
urls = [
  ['ちゆ', 'http://tiyu.to'],
  ['もなみ', 'http://www.tokyo-nazo.net/tester/'],
  ['メモリ', 'http://homepage1.nifty.com/GANTSU/memo/index.html'],
]

users.
  product(urls).
  select { |(users_name, _), (urls_name, _)| users_name == urls_name }.
  map { |(name, birthday), (_, url)| [name, birthday, url] }
```

- JOIN 相当の処理は Ruby ではこう書く
- 結構キツい

SQL と Ruby

```
-- name 毎に、birthday 順で1つ前の行と name を '+' で結合
SELECT string_agg(name, '+')
          over(partition by name order by birthday rows 1 preceding)
  FROM users
-- ちゆ
-- ちゆ+ちゆ
-- もなみ
-- もなみ+もなみ
-- メモリ
-- メモリ+メモリ
```

- SQL の WINDOW 関数

SQL と Ruby

```ruby
users.
  group_by { |name, _| name }.
  map { |_, tuples|
    tuples.
      sort_by { |_, birthday| birthday }.
      map.with_index { |tuple, i|
        [tuples[(i.zero? ? i : (i-1))..i].map { |name, _| name }.join("+")]
      }
  }.
  flatten(1)
```

- WINDOW 関数は Ruby ではこう書く
- かなりキツい

どう書く勉強会

- 話は変わって
- オフラインリアルタイムどう書く勉強会
  - http://nabetani.sakura.ne.jp/hena/ord30taxi/
- 鍋谷さんの良問を味わう会

どう書く勉強会

とある世界のタクシー料金 横へな 2015.4.18 問題

問題

とある世界のタクシーの運賃を計算してほしい。
何もかもだと大変なので、円來市と炭州市という2つの市の範囲だけでよい。
乗り降りする場所はA～Gだけを考える。
右図の通り。
経路は客が指定する。理不尽な経路（例：ABABCBA）でも気にしない。

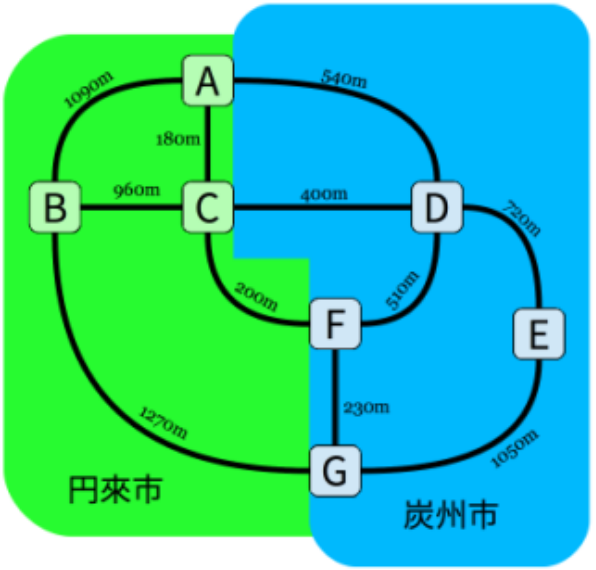
で。
料金体系が円來市と炭州市で異なる。下表の通り。

市	初乗り	距離加算
円來市	995m までは 400円	200m 進むごとに 60円加算
炭州市	845m までは 350円	200m 進むごとに 50円加算

細かいことを書くと。

- 乗車地点での自治体で初乗りの距離と値段が決まる。
  例：C→D→E という経路だと、円來市 の C で乗ったので初乗り995m となる。
- 初乗りの距離が終わったタイミングで加算。
- 加算される位置の自治体で加算の値段が決まる。

ということになっている。



どう書く勉強会

- 第３０回
- 架空のタクシー料金体系について、経路情報に応じた料金を計算
- 初乗り料金があり、距離や課金地点のエリアによって異なる課金額

どう書く勉強会

```
CREATE OR REPLACE FUNCTION pg_temp.calc_charge(route text) RETURNS table(num bigint) AS
  $$
    BEGIN
      RETURN QUERY
```

- 当日は Ruby で解いたが、後日 SQL で解いてみた
- SQL で書いた function プロシージャ（経路文字列を受け取る）

どう書く勉強会

```
WITH point_vias AS (
SELECT row_number() over() as no, *
  FROM unnest(string_to_array(route, NULL)) AS point
```

- 受け取った経路文字列を１文字ずつに分解して配列にする
- 配列を行に展開する
- 行番号を着けたリレーションを返す

どうかく勉強会

- WITH ？
  - サブクエリに名前を付けられる仕組み
  - Common Table Expressions（共通表式）と呼ばれる（CTEと略す）
  - メインの SELECT の前に書いておくと、そのクエリ中にだけ有効な VIEW が見えるようになるというイメージ

どう書く勉強会

```
), point_pairs AS (
SELECT no, point AS f, lead(point) over(order by no) AS t
  FROM point_vias
```

- LEAD 関数で、直後の地点の値を呼び出す

どう書く勉強会

```
), point_pairs AS (
```

- WITH ？（2）
  - WITH はカンマ区切りで、いくらでも書ける（何億とかは無理だと思うので空気読んで）
  - それ以前の CTE を参照する事が可能（逆は無理）

どう書く勉強会

```
), paths AS (
SELECT no, f, t
  FROM point_pairs
 WHERE t IS NOT NULL
```

- 直後の地点が無い行（最終地点）は課金されないので除く

どう書く勉強会

```
), distances AS (
SELECT no, path, distance
  FROM paths
  JOIN path_patterns ON ARRAY[paths.f, paths.t] <@ string_to_array(path_patterns.path, NULL)
```

- path_patterns はマスタとして永続化済みのテーブル
  - 地点間の距離と、地点の地域が入ってる
- それを JOIN して、経路に距離情報を付与する

どう書く勉強会

```
), distance_and_prices AS (
SELECT no, distances.distance, price
  FROM distances
  JOIN path_patterns using(path)
  JOIN extend_charges using(city)
```

- 地点毎に違う追加料金単価が入ったテーブルを JOIN する

どう書く勉強会

```
), distance_progress AS (
SELECT sum(distance) over(order by no) AS terminattion, price
  FROM distance_and_prices
```

- 地点毎の積算距離と追加料金単価を算出

どう書く勉強会

```
), extend_charge_terms AS (
SELECT int8range(coalesce(lag(terminattion) over(), 0) + 1, terminattion, '[]') AS term, price
  FROM distance_progress
```

- 地点毎の積算距離の範囲（範囲型の値）と追加料金単価のペア

どう書く勉強会

```
), extend_charge_start AS (
SELECT distance + 1 AS distance
  FROM first_charges
  JOIN points USING(city)
  JOIN paths ON points.point = paths.f AND paths.no = 1
```

- 初乗り料金が終わって、追加料金が発生し始めるる地点を算出

どう書く勉強会

```
), extend_charge_points AS (
SELECT generate_series(
          (SELECT distance FROM extend_charge_start),
          (SELECT sum(distance) FROM distances),
          200
        ) AS point
```

- 追加料金が発生する地点から終点までの間をを追加料金が発生する距離（200m）毎に区切って課金地点にする

どう書く勉強会

```
), extend_charges AS (
SELECT price AS charge
  FROM extend_charge_points AS ecp
  JOIN extend_charge_terms AS ect
    on ecp.point <@ ect.term
```

- 課金地点テーブルに追加料金単価のテーブルを JOIN
- 課金地点の適用単価を算出

どう書く勉強会

```
), first_charge AS (
SELECT price AS charge
  FROM first_charges
  JOIN points using(city)
  JOIN paths ON paths.f = points.point AND paths.no = 1
```

- 初乗り料金

どう書く勉強会

```
), all_charges AS (
SELECT charge FROM first_charge
 UNION ALL
SELECT charge FROM extend_charges
      )
```

- 初乗り料金と追加料金を UNION

どう書く勉強会

```
SELECT sum(charge) FROM all_charges;
```

- 全ての料金を足すと最終料金が算出される

どう書く勉強会

```
END
   $$ LANGUAGE plpgsql;
```

- function プロシージャー終わり

どう書く勉強会

```
SELECT id, route, pg_temp.calc_charge(route) as actual, expected
  FROM test_cases;
-- id |  route   | actual | expected
-- ----+----------+--------+----------
--  0 | ADFC     |   510 |    510
--  1 | CFDA     |   500 |    500
--  2 | AB       |   460 |    460
--  3 | BA       |   460 |    460
--  4 | CD       |   400 |    400
--  5 | DC       |   350 |    350
--  6 | BG       |   520 |    520
--  7 | GB       |   530 |    530
--  8 | FDA      |   450 |    450
--  9 | ADF      |   450 |    450
```

どう書く勉強会

```
--  10 | FDACB    |   750 |    750
--  11 | BCADF    |   710 |    710
--  12 | EDACB    |   800 |    800
--  13 | BCADE    |   810 |    810
--  14 | EGFCADE  |   920 |    920
--  15 | EDACFGE  |   910 |    910
--  16 | ABCDA    |   960 |    960
--  17 | ADCBA    |  1000 |   1000
--  18 | BADCFGB  |  1180 |   1180
--  19 | BGFCDAB  |  1180 |   1180
--  20 | CDFC     |   460 |    460
--  21 | CFDC     |   450 |    450
```

どう書く勉強会

```
--  22 | ABGEDA    |  1420 |    1420
--  23 | ADEGBA    |  1470 |    1470
--  24 | CFGB      |   640 |     640
--  25 | BGFC      |   630 |     630
--  26 | ABGEDFC   |  1480 |    1480
--  27 | CFDEGBA   |  1520 |    1520
--  28 | CDFGEDABG |  1770 |    1770
--  29 | GBADEGFDC |  1680 |    1680
```

- テストケースを全部パスした

どう書く勉強会

- ファイルはこちらです
  - https://gist.github.com/yancya/a437cf424242dbe9cab9
- ご興味があれば動かしてみてください
- PostgreSQL に喰わせるだけで動きます

ActiveRecord

- 話を Ruby っぽいところに戻す
- 有名な ORM
- 他にも Sequel などがある

ActiveRecord

サンプルテーブル　users

name	birthday
ちゆ	1999-02-14
ちゆ	2003-02-14
もなみ	2002-02-09
もなみ	2006-02-09
メモリ	2003-11-02
メモリ	2004-11-02

ActiveRecord

- たとえば「ちゆ１２歳」と「もなみ９歳」だけを抜き出したいとする

ActiveRecord

```
SELECT *
  FROM users
 WHERE (name = 'ちゆ' AND date_part('year', age(birthday)) = 12)
    OR (name = 'もなみ' AND date_part('year', age(birthday)) = 9)
```

- SQL で書くと、こんなかんじ
- 対象が増える程 OR が増える
- Repeat myself 感（Not DRY）

ActiveRecord

```
SELECT *
  FROM users
 WHERE (name = 'ちゆ' AND date_part('year', age(birthday)) = 12)
    OR (name = 'もなみ' AND date_part('year', age(birthday)) = 9)
```

- "(〜 AND 〜)" みたいなのを生成して join(" OR ") とかで結合すれば、なんとかできそう
- ただ、他の方法を考えてみよう

Shapeshifter

```
class CreateShapeshifters < ActiveRecord::Migration
  def change
    create_table :shapeshifters do |t|
    end

    # INSERT forbidden
    execute <<-SQL
   ALTER TABLE shapeshifters
ADD CONSTRAINT forbid_to_insert_shapeshifters
          CHECK (false)
    SQL
  end
end
```

Shapeshifter

```ruby
class Shapeshifter < ActiveRecord::Base
  def self.metamorphose(schema:, tuples:)
    with(table_name => sanitize_sql_array(<<-SQL.chomp, tuples.flatten))
SELECT *
  FROM #{to_values(tuples)}
    AS t(#{primary_key}, #{schema.join(', ')}) -- SQL インジェクションやべぇ
    SQL
  end

  def self.to_values(tuples)
    tuples.
      map.with_index(1) { |tuple, i| "(#{i}, #{tuple.map { '?' }.join(', ')})" }.
      join(", ").
      tap { |values| break "(VALUES #{values})" }
  end

  def self.sanitize_sql_array(sql, values)
    ActiveRecord::Base.send(:sanitize_sql_array, [sql, *values])
  end
end
```

Shapeshifter

```
with(table_name => sanitize_sql_array(<<-SQL.chomp, tuples.flatten))
```

- その with は一体...
- postgres_ext という gem の機能です
  - https://github.com/dockyard/postgres_ext
  - Arel::Nodes::As のインスタンスか { tablename => sql } みたいな Hash を渡します

Shapeshifter

```
Shapeshifter.all
#=> #<ActiveRecord::Relation []>

Shapeshifter.create
#=> ActiveRecord::StatementInvalid:
#=> PG::CheckViolation: ERROR:  new row for relation "shapeshifters"
#=> violates check constraint "forbid_to_insert_shapeshifters"
```

- 中身も無い、INSERT も出来ない、属性が id のみのモデル
- これをこうじゃ

Shapeshifter

```
ar = Shapeshifter.
  metamorphose(
    schema: %w{code message},
    tuples: [[404, 'Not Found'], [200, 'OK']]
  )
#=> #<ActiveRecord::Relation [#<Shapeshifter id: 1>, #<Shapeshifter id: 2>]>

ar.map(&:attributes)
#=> [{"id"=>1, "code"=>404, "message"=>"Not Found"},
#=>  {"id"=>2, "code"=>200, "message"=>"OK"}]

ar.where(code: 200)
#=> #<ActiveRecord::Relation [#<Shapeshifter id: 2>]>
```

- アプリケーションから注入したタプルが SQL 界を旅してから AR として戻ってくる

Shapeshifter

```
-- 発行される SQL
  WITH "shapeshifters" AS (
SELECT *
  FROM (VALUES (1, 404, 'Not Found'), (2, 200, 'OK'))
    AS t(id, code, message) )
SELECT "shapeshifters".*
  FROM "shapeshifters"
--  id | code |  message
-- ----+------+-----------
--   1 |  404 | Not Found
--   2 |  200 | OK
```

- 一時的に共通表式(CTE)を参照

User モデル

```
class User < ActiveRecord::Base
  scope :on_name_and_ages, -> (tuples) {
    with(
      target_users: Shapeshifter.
        metamorphose(schema: %w{name age}, tuples: tuples)
    ).
    joins(<<-SQL)
JOIN target_users
  ON users.name = target_users.name
 AND age = date_part('year', age(birthday))::integer
    SQL
  }
end
```

- scopeにShapeshifterを仕込む

User モデル

```
User.on_name_and_ages(
  [['ちゆ', 12], ['もなみ', 9]]
).map(&:attributes)
#=> [{"id"=>2, "name"=>"ちゆ", "birthday"=>Fri, 14 Feb 2003},
#=>  {"id"=>4, "name"=>"もなみ", "birthday"=>Thu, 09 Feb 2006}]
```

- タプルを使った絞り込みをする scope になってる

User モデル

```
-- 発行される SQL
  WITH "target_users" AS (
      WITH "shapeshifters" AS (
    SELECT *
      FROM (VALUES (1, 'ちゆ', 12), (2, 'もなみ', 9))
        AS t(id, name, age))
SELECT "shapeshifters".*
  FROM "shapeshifters")
SELECT "users".*
  FROM "users"
  JOIN target_users
    ON users.name = target_users.name
   AND age = date_part('year', age(birthday))::integer
```

まとめ

- SQL で出来る事の殆どは Ruby でもできる
- VALUES 句を使うとクエリに任意のリレーションを埋め込める
- Rails を騙して共通表式（CTE）を読ませたった
- 以上、ご質問があればどうぞ

どれくらい入れられるの

```
Shapeshifter.
  metamorphose(
    schema: [:name],
    tuples: 1_000_000.times.map { [%i{a b c}.sample] }
  ).
  group(:name).
  count
#=> {"c"=>333407, "b"=>332869, "a"=>333724}
```

- 100 万件の GRUOP BY 集計
- ローカルホストで５秒くらい

どれくらい入れられるの

```
# comment
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=37500.00..37502.00 rows=200 width=32) (actual time=630.957..630.958 rows=3 loops=1)
   CTE shapeshifters
     ->  Values Scan on "*VALUES*"  (cost=0.00..12500.00 rows=1000000 width=36) (actual time=0.003..168.647 rows=1000000 loops=1)
   ->  CTE Scan on shapeshifters  (cost=0.00..20000.00 rows=1000000 width=32) (actual time=0.005..448.934 rows=1000000 loops=1)
 Total runtime: 680.796 ms
(5 rows)
```

- クエリの実行自体は 0.7 秒
- 14MB あるとパースが重いかも
  - EXPLAIN に出てこないっぽい
  - パース時間プロファイリングの方法を調べたい