
OPEN SOURCE
@ IFTTT

build unknown



Polo

Polo travels through your database and creates sample snapshots so you can work with real world data in any environment.

Polo takes an `ActiveRecord::Base` seed object and traverses every whitelisted `ActiveRecord::Association` generating SQL `INSERTs` along the way.

You can then save those SQL `INSERTs` to .sql file and import the data to your favorite environment.

Motivation

Read our blog post or check out this presentation.

Usage

Given the following data model:

```
1 class Chef < ActiveRecord::Base
2   has_many :recipes
3   has_many :ingredients, through: :recipes
4 end
5
6 class Recipe < ActiveRecord::Base
7   has_many :recipes_ingredients
8   has_many :ingredients, through: :recipes_ingredients
9 end
10
```

```
11 class Ingredient < ActiveRecord::Base
12 end
13
14 class RecipesIngredient < ActiveRecord::Base
15   belongs_to :recipe
16   belongs_to :ingredient
17 end
```

Simple ActiveRecord Objects

```
1 inserts = Polo.explore(Chef, 1)
```

```
1 INSERT INTO `chefs` (`id`, `name`) VALUES (1, 'Netto')
```

Where `Chef` is the seed object class, and 1 is the seed object id.

Simple Associations

```
1 inserts = Polo.explore(Chef, 1, :recipes)
```

```
1 INSERT INTO `chefs` (`id`, `name`) VALUES (1, 'Netto')
2 INSERT INTO `recipes` (`id`, `title`, `num_steps`, `chef_id`) VALUES
  (1, 'Turkey Sandwich', NULL, 1)
3 INSERT INTO `recipes` (`id`, `title`, `num_steps`, `chef_id`) VALUES
  (2, 'Cheese Burger', NULL, 1)
```

Complex nested associations

```
1 inserts = Polo.explore(Chef, 1, :recipes => :ingredients)
```

```
1 INSERT INTO `chefs` (`id`, `name`) VALUES (1, 'Netto')
2 INSERT INTO `recipes` (`id`, `title`, `num_steps`, `chef_id`) VALUES
  (1, 'Turkey Sandwich', NULL, 1)
3 INSERT INTO `recipes` (`id`, `title`, `num_steps`, `chef_id`) VALUES
  (2, 'Cheese Burger', NULL, 1)
4 INSERT INTO `recipes_ingredients` (`id`, `recipe_id`, `ingredient_id`)
  VALUES (1, 1, 1)
5 INSERT INTO `recipes_ingredients` (`id`, `recipe_id`, `ingredient_id`)
  VALUES (2, 1, 2)
6 INSERT INTO `recipes_ingredients` (`id`, `recipe_id`, `ingredient_id`)
  VALUES (3, 2, 3)
7 INSERT INTO `recipes_ingredients` (`id`, `recipe_id`, `ingredient_id`)
  VALUES (4, 2, 4)
```

```
8 INSERT INTO `ingredients` (`id`, `name`, `quantity`) VALUES (1, 'Turkey', 'a lot')
9 INSERT INTO `ingredients` (`id`, `name`, `quantity`) VALUES (2, 'Cheese', '1 slice')
10 INSERT INTO `ingredients` (`id`, `name`, `quantity`) VALUES (3, 'Patty', '1')
11 INSERT INTO `ingredients` (`id`, `name`, `quantity`) VALUES (4, 'Cheese', '2 slices')
```

Advanced Usage

Occasionally, you might have a dataset that you want to refresh. A production database that has data that might be useful on your local copy of the database. Polo doesn't have an opinion about your data; if you try to import data with a key that's already in your local database, Polo doesn't necessarily know how you want to handle that conflict.

Advanced users will find the `on_duplicate` option to be helpful in this context. It gives Polo instructions on how to handle collisions. *Note: This feature is currently only supported for MySQL databases. (PRs for other databases are welcome!)*

There are two possible values for the `on_duplicate` key: `:ignore` and `:override`. Ignore keeps the old data. Override keeps the new data. If there's a collision and the `on_duplicate` param is not set, Polo will simply stop importing the data.

Ignore

A.K.A the Ostrich Approach: stick your head in the sand and pretend nothing happened.

```
1 Polo.configure do
2   on_duplicate :ignore
3 end
4
5 Polo::Traveler.explore(Chef, 1, :recipes)
```

```
1 INSERT IGNORE INTO `chefs` (`id`, `name`) VALUES (1, 'Netto')
2 INSERT IGNORE INTO `recipes` (`id`, `title`, `num_steps`, `chef_id`)
  VALUES (1, 'Turkey Sandwich', NULL, 1)
3 INSERT IGNORE INTO `recipes` (`id`, `title`, `num_steps`, `chef_id`)
  VALUES (2, 'Cheese Burger', NULL, 1)
```

Override

Use the option `on_duplicate: :override` to override your local data with new data from your Polo script.

```
1 Polo.configure do
2   on_duplicate :override
3 end
4
5 Polo::Traveler.explore(Chef, 1, :recipes)
```

```
1 INSERT INTO `chefs` (`id`, `name`) VALUES (1, 'Netto')
2 ON DUPLICATE KEY UPDATE id = VALUES(id), name = VALUES(name)
3 ...
```

Sensitive Fields

You can use the `obfuscate` option to obfuscate sensitive fields like emails or user logins.

```
1 Polo.configure do
2   obfuscate :email, :credit_card
3 end
4
5 Polo::Traveler.explore(AR::Chef, 1)
```

```
1 INSERT INTO `chefs` (`id`, `name`, `email`) VALUES (1, 'Netto', '
   eahorctmaagfo.nitm@l')
```

Warning: This is not a security feature. Fields can still easily be rearranged back to their original format. Polo will simply scramble the order of strings so you don't accidentally end up causing side effects when using production data in development. It is not a good practice to use highly sensitive data in development.

Advanced Obfuscation For more advanced obfuscation, you can pass in a custom obfuscation strategy. Polo will take in a lambda that can be used to transform sensitive data.

Using a `:symbol` as an obfuscate key targets all columns of that name. Passing an SQL selector as a `String` will target columns within the specified table.

```
1 Polo.configure do
2
3   email_strategy = lambda do |email|
4     first_part = email.split("@")[0]
5     "#{first_part}@test.com"
6   end
```

```
7
8  credit_card_strategy = lambda do |credit_card|
9    "4123 4567 8910 1112"
10 end
11
12 # If you need the context of the record for its fields, it is
13 # accessible
14 # in the second argument of the strategy
15 social_security_strategy = lambda do |ssn, instance|
16   sprintf("%09d", instance.id)
17 end
18
19 obfuscate({
20   'chefs.email' => email_strategy, # This only applies to the "email"
21   # column in the "chefs" table
22   :credit_card => credit_card_strategy, # This applies to any column
23   # named "credit_card" across every table
24   :ssn_strategy => social_security_strategy
25 })
26 end
27
28 Polo::Traveler.explore(AR::Chef, 1)
```

```
1 INSERT INTO `chefs` (`id`, `name`, `email`) VALUES (1, 'Netto', '
  netto_test@example.com')
```

Installation

Add this line to your application's Gemfile:

```
1 gem 'polo'
```

And then execute:

```
1 $ bundle
```

Or install it yourself as:

```
1 $ gem install polo
```

Contributing

Bug reports and pull requests are welcome on GitHub at <https://github.com/IFTTT/polo>. This project is intended to be a safe, welcoming space for collaboration, and contributors are expected to adhere to the Code of Conduct.

To run the specs across all supported version of Rails, check out the repo and follow these steps:

```
1 $ bundle install
2 $ bundle exec appraisal install
3 $ bundle exec appraisal rake
```

License

The gem is available as open source under the terms of the MIT License.