django-sabridge Documentation

Release 0.0.1

John Paulett

Nov 09, 2017

Contents

1 Motivation 1

2 Usage 3

3 Caveats 5
- 3.1 Transactions . . . 5
- 3.2 Performance . . . 5

4 Contents 7
- 4.1 sabridge API . . . 7
- 4.2 Developing django-sabridge . . . 7
- 4.3 Change Log . . . 8
- 4.4 License . . . 8
- 4.5 Ideas . . . 8

5 Links 11

CHAPTER 1

Motivation

Django's ORM is wonderful and easy to use. When the standard ORM operations are insufficient for an application, Django provides multiple methods for more directly interacting with the database, including `django.db.models.Manager.raw()` and `django.db.connection.cursor()`. However, using these methods can easily lead to vendor lock-in. Additionally, programmatically building SQL is a difficult task (especially to do so securely).

SQLAlchemy offers an excellent SQL rendering engine, which allows programmatic generation of SQL. By exposing Django models via SQLAlchemy's Expression Language, a developer can build extremely complex queries while retaining database-independence (vendor-specific features are still available).

Other efforts have aimed to replace Django's ORM with SQLAlchemy's ORM. django-sabridge instead leaves Django's ORM in place, while allowing SQLAlchemy Expression Language to easily access those Django models.

django-sabridge addresses a specific need. It may not be the ideal solution, so please *contribute* better approaches. Please also be aware of the *Caveats*.

CHAPTER 2

Usage

To demonstrate sabridge, we will access `django.contrib.auth.models.User` through SQLAlchemy.

First, import and initialize the *sabridge.Bridge*:

```
>>> from sabridge import Bridge
>>> bridge = Bridge()
```

We use the model's class to obtain the SQLAlchemy version of the table:

```
>>> from django.contrib.auth.models import User
>>> table = bridge[model]
```

The *sabridge.Bridge* returns an instance of `sqlalchemy.schema.Table`. If we write data in Django, we can then view that data via SQLAlchemy:

```
>>> User.objects.create(username='alice')
>>> result = list(table.select().execute())
>>> len(result)
1
>>> result[0][table.c.username]
u'alice'
```

CHAPTER 3

Caveats

3.1 Transactions

sabridge does not re-use Django's connection to the database, thus if executing in a transaction, any data modified by either Django or SQLAlchemy will not be visible to the other, until the transaction is committed.

Practically, this means that any test cases that uses both Django and SQLAlchemy will have to inherit from `django.test.TransactionTestCase` instead of the more typical `django.test.TestCase`. The TransactionTestCase does not wrap each test in a transaction, thus the data modified by SQLAlchemy and Django is not isolated. Unfortunately, the TransactionTestCase is significantly slower than the normal TestCase. Refer to the TransactionTestCase documentation.

3.2 Performance

sabridge uses SQLAlchemy's reflection (`autoload=True`) to discover the schema of the requested Django model. Efforts are made to reduce the number of times introspection occurs, but a user of django-sabridge should make sure that it fits within any performance requirements.

CHAPTER 4

Contents

4.1 sabridge API

class `sabridge.`**`Bridge`**

`__getitem__`(*model_cls*)

Returns the `sqlalchemy.schema.Table` representation of `model_cls`, a `django.db.models.Model` subclass.

Use dict-notation to obtain the `Table`:

```
>>> from myapp.models import mymodel
>>> brige = Bridge()
>>> mytable = bridge[mymodel]
>>> print type(mytable)
<class 'sqlalchemy.schema.Table'>
```

`Bridge` stores the `Table` for the lifetime of the `Bridge`, thus table reflection only occurs once per model for the `Bridge`.

`connection_url`()

Build a URL for `sqlalchemy.create_engine()` based upon the database defined by `django.db.connection`

`meta`

`sqlalchemy.schema.MetaData` instance bound to the current Django database connection.

4.2 Developing django-sabridge

Get the code: http://github.com/johnpaulett/django-sabridge

Setup the environment:

```
virtualenv --no-site-packages env
source env/bin/activate
pip install -r dev_requirements.txt
python setup.py develop
```

Run the test suite:

```
./runtests.py
```

4.3 Change Log

4.3.1 0.0.1 - July 4th, 2011

- Initial release
- Basic mapping from Django's Model class into SQLAlchemy's Table class using SQLAlchemy's table introspection.

4.4 License

```
Copyright (c) 2011, John Paulett
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of django-sabridge nor the
      names of its contributors may be used to endorse or promote products
      derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL JOHN PAULETT BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
```

4.5 Ideas

A collection of some random ideas & thoughts for future implementations

- Easier access for the *through* table for `django.db.models.ManyToManyFields` than requiring the user to manually access `Model.column._through()`

- We could use SQLAlchemy to generate the SQL for `django.db.connection.cursor()`, allowing reuse of Django's current transaction. This should currently work, but we could add some sugar for making it easier & documenting it. It would mean that you no longer get a SQLAlchemy ResultProxy back.

CHAPTER 5

Links

- https://github.com/johnpaulett/django-sabridge
- http://django-sabridge.readthedocs.org

Index

Symbols

__getitem__() (sabridge.Bridge method), 7

B

Bridge (class in sabridge), 7

C

connection_url() (sabridge.Bridge method), 7

M

meta (sabridge.Bridge attribute), 7