Audit-Alembic

Release 0.2.0

Aug 03, 2017

Contents

CHAPTER 1

Overview

docs	
tests	|
package	

An Alembic plugin to keep records of upgrades and downgrades.

- Free software: MIT license

Installation

```
pip install Audit-Alembic
```

Getting started

Quickstart

Create an Alembic environment if you don't already have one. Edit its `env.py` to include the following:

```
# ... imports ...
import audit_alembic
from myapp import version

if not audit_alembic.alembic_supports_callback():
    raise audit_alembic.exc.AuditSetupError(
        'This Alembic version does not have on_version_apply')
```

```
auditor = audit_alembic.Auditor.create(version)

def run_migrations_offline():
    ...
    context.configure(
        ...
        on_version_apply=auditor.listen,
    )
    ...

def run_migrations_offline():
    ...
    context.configure(
        ...
        on_version_apply=auditor.listen
    )
...
```

More involved

`Auditor.create()` is a factory method: it creates an Alembic history table for you and merely asks you to specify your application version (though it allows much else to be customized as well). If you are already maintaining a table you wish to add records to whenever an Alembic operation takes place, and you have a callable that creates a row for that table, you can instantiate `Auditor` directly:

```
auditor = Auditor(HistoryTable, HistoryTable.alembic_version_applied)
```

In this case `alembic_version_applied` must return a dictionary that can serve as binds for an INSERT statement on `HistoryTable`. It has the same signature as documented for Alembic's `on_version_apply` hook.

Full Documentation

Once the 0.2.0 release is complete, the docs will be accessible here: https://Audit-Alembic.readthedocs.io/

Development

Status

The bulk of the test suite is complete and passing for Postgres, mysql, and SQLite. Travis does not appear to support MSSQL or Oracle so test status for those DB backends is not known. If you find that it does not work for your backend, pull requests to make it so will be happily accepted.

Please feel free to expand from there. See the issues for a list of known issues to work on.

Testing

To run basic tests:

```
$ virtualenv venv && source venv/bin/activate
(venv) $ python setup.py install
(venv) $ pip install pytest psycopg2
(venv) $ pytest
```

To run all tests (i.e. py2 + py3, across all database drivers), run:

```
$ tox
```

See CONTRIBUTING.rst for more detail. Also see our Travis setup.

CHAPTER 2

Installation

At the command line:

```
pip install Audit-Alembic
```

CHAPTER 3

Usage

To use Audit-Alembic in a project:

```
import audit_alembic
```

CHAPTER 4

Reference

audit_alembic

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

These suggestions are largely adapted from those provided by the cookiecutter template used for this project. I rewrote a great deal, and I stand what I haven't changed. Acknowledgment nonetheless is due to others who gave me something to tweak.

Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Documentation improvements

Audit-Alembic could always use more documentation, whether as part of the official Audit-Alembic docs, in docstrings, or even on the web in blog posts, articles, and such.

Feature requests and feedback

The best way to send feedback is to file an issue at gh-issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

- Remember that I am just this guy, you know? If at all possible, please try implementing yourself; if it doesn't work, submit as a pull request and we'll see what we can make out of it.

Development

To set up Audit-Alembic for local development:

1. Fork Audit-Alembic (look for the "Fork" button).
2. Clone your fork locally:

   ```
   git clone git@github.com:your_name_here/Audit-Alembic.git
   ```

3. Create a branch for local development:

   ```
   git checkout -b name-of-your-change
   ```

   Now you can make your changes locally.

4. When you're done making changes, make sure tests pass – see *Testing*.
5. Commit your changes and push your branch to GitHub:

   ```
   git add .
   git commit -m "Your detailed description of your changes."
   git push origin name-of-your-bugfix-or-feature
   ```

   Please observe Tim Pope's guidelines for a good commit message.

6. Submit a pull request through the GitHub website.

Testing

Testing is required – no patch will be accepted without tests that fail if implemented without the fix.

Trigger your virtual environment and install the test dependencies:

```
$ pip install flake8  # required for linting
$ pip install pytest mock  # required for testing
$ pip install psycopg2  # enables running tests with postgresql
$ pip install mysql-python  # enables running tests with mysql
$ pip install pytest-cov  # enables coverage tracking
```

First run `flake8` to ensure no facepalms. Please keep the library PEP8-compliant; deviations are likely to be nitpicked in code review.

Now to run tests:

```
$ python setup.py install &&
$ pytest -k test_my_change  # substitute the name of the new test you wrote
$ pytest --cov  # runs all available tests
```

The above commands run the tests using SQLite and whatever version of python you're running. The full test suite runs several database drivers and several versions of python. You can get this running by just submitting a pull request and Travis will do the rest. It's nonetheless (gently) encouraged, and not all that difficult, to run them yourself before submitting a pull request. It's one easy way to assure yourself the change can be accepted.

Database backends Running with SQLite is trivial, as we've seen. Running with postgresql or mysql requires a working installation of the respective database and its respective Python driver (for the latter see the pip installs above). The principles are pretty simple and ought to work with any SQL dialect that also works for Alembic (e.g. Oracle, MSSQL), but no working examples are known other than those directly supported by Travis: sqlite, postgresql, mysql.

Running tests on postgresql and mysql backends is nearly identical, so let's say you want to run with postgresql. As mentioned above, install a postgresql database if you don't have one, and `pip install psycopg2`.

By default, when asked to run with postgresql or mysql, the test suite expects a database named `test`, usable by a user named `scott` who has a password of `tiger` (run `grep psql .travis.yml` to see how travis creates it, or `grep mysql .travis.yml` for the same with mysql). This default can be overridden if you have your own configuration you want to work with. Either way, beware that the tests include `DROP ALL TABLES` as a teardown command, so make sure the database you're using, `test` or otherwise, contains no data you need to preserve.

Running `pytest` with different backends To use the default setup, database `test` and user `scott` with password `tiger`:

```
$ pytest --cov --db postgresql
$ pytest --cov --db mysql
```

Valid inputs for the `--db` option can be seen by running `pytest --dbs=` [sic]. (Please note, while configurations exist for mssql and oracle, the author has no access to those databases; if you are able to run tests on these, please let me know what you find, pass **or** fail!)

To use a different DB configuration, use a SQLAlchemy URI string:

```
$ pytest --cov --dburi postgresql://scott:tiger@127.0.0.1:5432/test
```

The `--dburi` option can point to any database on any backend, as long as a SQLAlchemy Dialect for it can be found on your system.

You can combine as many of these options as you like to run multiple backends in a single `pytest` invocation:

```
$ pytest --cov --db mysql --db postgresql --dburi sqlite:///my-file.db
```

tox *tox* is a standard tool used to run several distinct testing environments. It creates a new virtual environment for each combination of Python interpreter and database backend it's asked to run. It can be installed inside or outside your own virtual environment.

Install `tox` using `pip`, and on your system install whatever extra interpreters you want to test with. (You should test at a minimum with 2.7 and one of 3.4, 3.5, or 3.6. Whatever you're unable to test, Travis will handle when you create a pull request.)

Also install and set up whatever *database servers* you'll be testing against. If you want want `tox` to use a database configuration other than the default, export as follows:

```
$ export POSTGRESQL='--dburi postgresql://scott:tiger@127.0.0.1:5432/test'
```

The default list of environments is found in `tox.ini` or by running `tox -l`. You can adjust it by exporting to `TOXENV` or using `tox -e` to specify environments. Again, what you're unable to test yourself.

When the environment variables are set to your liking, run:

```
tox
```

It will run all the tests in all available environments and report on results. Furthermore, it will track coverage automatically.

To run just your test on all environments, run like this:

```
tox -- -k test_my_change
```

For more granular control on what tests *tox* runs, read its *docs*, and our `tox.ini`.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code, just submit the pull request.

When ready for merging, you should:

1. Create new tests covering your change, and use *tox* to assure yourself the change works.[1]. If you can write one that fails without your patch, even better! If your patch includes new code and does not include a test, it is unlikely to be accepted; if any tests fail in any Travis environment, they will have to be resolved before the change is accepted.
2. Update documentation when there's new API, functionality etc. Docstrings are usually the best way to do this.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

Simple `pytest -k test_myfeature` will not work; the package must be installed first. `tox` does that for you.

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

(Nota bene: the author has never done this.)

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.

CHAPTER 6

Authors

- John Passaro

CHAPTER 7

Changelog

0.2.0 (TBD)

Alpha release, pending a patch in alembic without which we cannot support stamps.

- Creates a listener for Alembic's `on_version_apply` callback hook which records information from that callback to a SQL table of the user's choosing.
- Test setup making use of SQLAlchemy testing plugins and utilities and Alembic testing utilities.
- Support for running with –sql as well as online mode.
- Tests covering stamps, branches, and a couple of other complex use cases.
- Test setup to cover multiple DB backends. Known to work: SQLite, Postgresql, mysql.

0.1.0 (2017-06-21)

- First release on PyPI. (powered by cookiecutter-pylibrary)

CHAPTER 8

Acknowledgements

Many thanks are of course due to Mike Bayer, the author of the amazing libraries SQLAlchemy and Alembic which this present effort seeks to extend.

Zeconomy enabled the writing of this library and is one of its first adopters.

Thanks also to ionelmc, who wrote a template that vastly sped up the initial development process.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index