temporal$_{sqlalchemy}$Documentation

Release 0.5.0

Clover Health

Aug 24, 2018

Contents

1 Temporal SQLAlchemy **1**
- 1.1 Some Terms . . . 1
- 1.2 Methodologies . . . 1
- 1.3 Testing and Development . . . 1
- 1.4 Usage . . . 2
- 1.5 Using Your Model . . . 3
- 1.6 Updating your instance . . . 4
- 1.7 Inspecting history . . . 4

2 Installation **5**

3 Release Notes **7**

4 Contributing Guide **11**
- 4.1 Setup . . . 11
- 4.2 Testing and Validation . . . 11
- 4.3 Documentation . . . 11
- 4.4 Releases and Versioning . . . 12

CHAPTER 1

Temporal SQLAlchemy

1.1 Some Terms

Temporal time dimension. A property can have spatial and/or temporal dimensions, e.g. what it was and when it was that.

Bi-temporal 2x time dimensions. Allows you to separate the "valid time" of a property from "when we knew about it"

Version Clock a simple, incrementing integer that allows you track "generation" of changes (also known as "vclock")

Version the state of an entity at a given version clock

Low Velocity state changes infrequently or you don't need to walk the history regularly

High Velocity state changes frequently or you need to walk the history regularly.

1.2 Methodologies

There exist several ways to add a temporal dimension to your data. SQLAlchemy Continuum uses a shadow or history table for each versioned entity. You can add a `date_created` and `date_modified` columns to your model.

Temporal SQLAlchemy uses a table per property, with an entity clock – all state is written to the entity table as expected, but additionally recorded into series of history tables (per property) and clock entries (per entity).

1.3 Testing and Development

1.3.1 Setup

To set up your development environment, you'll need to install a few things. For Python version management, we use pyenv-virtualenv. Follow the installation instructions there, and then in the *root directory* of this repo run: `make setup` Please note, temporal-sqlalchemy only supports python versions >= 3.5.

1.3.2 Running the Tests

To run the unit tests for all supported versions of Python, run `make test`. If you made a change to the package requirements (in `setup.py` or `test_requirements.txt`) then you'll need to rebuild the environment.

1.4 Usage

I'll start with the caveats, as there are a few:

- you cannot track the history of properties with `onupdate`, `server_default` or `server_onupdate`

```
import sqlalchemy as sa
import temporal_sqlalchemy as temporal

class MyModel(temporal.TemporalModel, SomeBase):
    id = sa.Column(sa.BigInteger, primary_key=True)
    # this will throw an error
    prop_a = sa.Column(sa.Text, onupdate='Some Update Value')
    # this will also throw an error
    prop_b = sa.Column(sa.Text, server_default='Some Server Default')
```

- you have to "temporalize" your session

```
import sqlalchemy.orm as orm
import temporal_sqlalchemy as temporal

sessionmaker = orm.sessionmaker()
session = sessionmaker()
temporal.temporal_session(session)

foo = MyModel(prop_a='first value', prop_b='also first first value')
session.add(foo)
session.commit()


with foo.clock_tick():
    foo.prop_a = 'new value'
    foo.prop_b = 'also new value'

session.commit()
```

- default values are tricky. If you need to record the state of a default value, or need `None` to have historical meaning you must be deliberate. Additionally, we cannot currently handle callable defaults that take a context described here

```
import sqlalchemy as sa
import sqlalchemy.orm as orm
import temporal_sqlalchemy as temporal

sessionmaker = orm.sessionmaker()
session = sessionmaker()
temporal.temporal_session(session)


class MyModel(temporal.TemporalModel, SomeBase):
    __tablename__ = 'my_model_table'
```

```
    __table_args__ = {'schema': 'my_schema'}

    id = sa.Column(sa.BigInteger, primary_key=True)
    description = sa.Column(sa.Text)

    class Temporal:
        track = ('description', )


m = MyModel()
session.add(m)
session.commit()

assert m.vclock == 1
assert m.description == None

description_hm = temporal.get_history_model(MyModel.description)

history = session.query(description_hm).filter(description_hm.entity==m)

# no history entry is created!
assert history.count() == 0

# do this instead
m2 = MyModel(description=None)
session.add(m2)
session.commit()

assert m2.vclock == 1
assert m2.description == None

history = session.query(description_hm).filter(description_hm.entity==m2)

# history entry is now created
assert history.count() == 1
```

1.5 Using Your Model

```
import sqlalchemy.orm as orm
import temporal_sqlalchemy as temporal

sessionmaker = orm.sessionmaker()
session = sessionmaker()

temporal.temporal_session(session)
instance = MyModel(description="first description")

assert instance.vclock == 1

session.add(instance)
session.commit()
```

1.6 Updating your instance

```
with instance.clock_tick():
    instance.description = "second description"

assert instance.vclock = 2
session.commit()
```

1.7 Inspecting history

```
import temporal_sqlalchemy as temporal

description_hm = temporal.get_history_model(MyModel.description)

history = session.query(description_hm).filter(description_hm.entity==instance)

assert history.count() == 2
assert history[0].description == 'first description'
assert history[1].description == 'second description'
```

CHAPTER 2

Installation

temporal_sqlalchemy can be installed with:

```
pip3 install temporal_sqlalchemy
```

CHAPTER 3

Release Notes

```
0.5.0
-----

* bump to 0.5.0 for publishing
* bump version manually
* publish to pypi
* back fill ye ol changelog
* remove eval calls
* drop python 3.4 support
* remove bugbear from 3.4 build
* add postgres and older flake8 requirement
* add test\_py34 to jobs workflow
* empty commit to trigger rebuild
* lol nice typo joey
* woops accidentally added a bunch of doc builds
* gitignore from temple
* tests pass now locally
* make validate passes
* starting flak8 and pylint fixes
* temple scaffolding

0.4.7
-----

* Bump version: 0.4.6 → 0.4.7
* Use list per transaction instead of one big list for persist\_on\_commit (#48)
* remove 3.3 support because pytest dropped it
* remove behavior flag for persist\_on\_commit

0.4.6
-----

* Bump version: 0.4.5 → 0.4.6
* more stylish code
* more tests and a couple of guards
```

```
* add nested transaction fix
* clean up and tests
* opt in to new behavior
* no magic strings
* batch history construction to commit instead of flush
* Bump version: 0.4.4 → 0.4.5
* Forgot to tell Travis and CircleCI to install 3.3
* Fix documentation to reflect 3.3 fix
* Fix broken support for Python 3.3
* Bump version: 0.4.3 → 0.4.4
* Fix README formatting
* Remove unnecessary --current-version arg
* Bump version: 0.4.2 → 0.4.3
* Travis: install multiple Python versions?
* Override default Python version in Travis
* Fix filename in README, eliminate redundant bumpversion
* Fix Travis?
* Fix incorrect installation instructions
* Forgot to change dev-requirements.txt in circle.yml
* Install requirements in Travis
* Be consistent with dashes and underscores
* Fix Travis?
* Add dev instructions
* Forgot to install dev requirements
* Remove leading \`v\` from version tag
* Copy-pasta error
* Need to manually set version here, bumpversion will handle it for us
* Messed up the bumpversion config
* Forgot bumpversion.cfg
* Change circle.yml to use tox
* Fix wrong version number
* Randomize test execution order to reveal interdependencies
* Configure tox
* Prepare to use tox
* No support for Python2 so this isn't a universal wheel
* Remove unnecessary dynamic code for dependencies
* Move installation dependencies into setup()
* Move descriptions into setup call instead of reading file
* Remove support for Python 3.3 and add 3.6
* Don't put install requirements in requirements.txt

0.4.2
-----

* redo index fixes for better history ready performance

0.4.1
-----

* Use lists instead of sets, which SHOULD BE safe, and preserve order
* Revert "Add indices on history table foreign keys"
* bump version
* get rid of python 3.3
* WHAT THE HECK PYTHON 3.3
* fix nullable migration breaking, as well as unintentional unique constraint rename
* Add indices on history table foreign keys

0.4.0
```

```
-----

* new version
* fix relationship handling

0.3.3
-----

* Fix issue #26: Remove compound primary keys on temporal clock tables (#31)
* unify everything
* ClockedOption to TemporalOption

0.3.2
-----

* Raise an assertion if a flush() occurs on changes without a proper clock tick (#24)
* WHAT IS HAPPEN -- the event isn't properly handled in all cases [#19]
* Detect if sessions are already temporalized (#20)
* Update README.rst
* Update README.rst
* simple usage doc

0.3.0
-----

* cleanup and actually fix py 3.3
* whoops really maybe fix python 3.3
* add python 3.5.3 and 3.6.0 to build matrix
* fix python 3.3
* fix python 3.4
* cleanup the bootstrap portion of temporal
* more tests, ensure the first\_tick and last\_tick only load one result
* test the session behavior around deletes
* speed up tests considerably
* TemporalModel extends Clocked for great victory
* stylistic changes
* new style models with better init clock
* support arbitrary shaped relationships (as long as they have real foreign keys)
* pending deprecate bad named properties on ClockedOption -- they're models not tables
* simplify test models
* bump version
* new style temporal declaration first pass
* truncate\_identifier a public method
* Add Travis and Coveralls, to provide complex build matrix (#11)
* Because I'm vain
* pep8
* bump version number
* support for simple joined table inheritance
* bump sqlalchemy to 1.1.2
* port over activity tests
* test via setup.py
* Rearrange things and reformat to flake8 standards
* Tweak and clean up setuptools setup.py script
* clover does open source
* Include version information in package and link setup version to package version
* Activity tests
* related model tests
* concrete base tests
```

```
* updated tests, added actual db tests
* it actually works with the api service now
* six of nine
* remove tox
* non-persistent tests
* tox + setup + pytest
* first pass at setup.py
* get tox prepped
* update readme
* skeleton
* Init circle ci config
* Initial commit
```

CHAPTER 4

Contributing Guide

This project was created using temple. For more information about temple, go to the Temple docs.

4.1 Setup

Set up your development environment with:

```
git clone git@github.com:CloverHealth/temporal-sqlalchemy.git
cd temporal-sqlalchemy
make setup
```

`make setup` will setup a virtual environment managed by pyenv and install dependencies.

Note that if you'd like to use something else to manage dependencies other than pyenv, call `make dependencies` instead of `make setup`.

4.2 Testing and Validation

Run the tests with:

```
make test
```

Validate the code with:

```
make validate
```

4.3 Documentation

Sphinx documentation can be built with:

```
make docs
```

The static HTML files are stored in the `docs/_build/html` directory. A shortcut for opening them is:

```
make open_docs
```

4.4 Releases and Versioning

Anything that is merged into the master branch will be automatically deployed to PyPI. Documentation will be published to ReadTheDocs.

The following files will be generated and should *not* be edited by a user:

- `ChangeLog` - Contains the commit messages of the releases. Please have readable commit messages in the master branch and squash and merge commits when necessary.
- `AUTHORS` - Contains the contributing authors.
- `version.py` - Automatically updated to include the version string.

This project uses Semantic Versioning through PBR. This means when you make a commit, you can add a message like:

```
sem-ver: feature, Added this functionality that does blah.
```

Depending on the sem-ver tag, the version will be bumped in the right way when releasing the package. For more information, about PBR, go the the PBR docs.