SQLAlchemy Fixture Factory Documentation

Release 0.1.0

Michael Pickelbauer

March 23, 2015

Contents

Automatic testing of services which involves database access gets quite quickly cumbersome, especially the preparation of the database. By the time the system matures, complex scenarios need to be covered. A predefined test scenario helps with the setup, but is not very flexible. On the search for a solution quite quickly factory_girl appeared on the radar. But nothing similar could be found for Python.

With this library quick, obvious and easily understandable test scenarios can be created which are flexible, easily to maintain and to extend.

Word of truth must be spoken: *it only works with SQLAlchemy's ORM mapper!* (see doc)

CHAPTER 1

Installation

The library is hosted on PyPI and can be installed via

```
pip install sqlalchemy-fixture-factory
```

CHAPTER 2

Quick Example

Assume following ORM definitions:

```
class Account(Base):
    __tablename__ = 'account'

    id = Column(Integer, primary_key=True)
    name = Column('name', Unicode)

class Person(Base):
    __tablename__ = 'person'

    id = Column(Integer, primary_key=True)
    first_name = Column('first_name', Unicode)
    account_id = Column(Integer, ForeignKey('account.id'))
    account = relationship(Account)
```

definition of a fixture for a person. It includes an account:

```
class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"

class ArnoldPerson(BaseFix):
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)
```

now the usage (I assume SQLAlchemy is properly initialized):

```
# initialize the fixture factory
fix_fact = SqlaFixFact(db_session)

# create a fxiture
arnold_fix = ArnoldPerson(fix_fact).create()

# or more of them
while i in xrange(3):
    arnold_fix = ArnoldPerson(fix_fact).create()

# test
assert 4 == db_session.query(Person).count()
```

CHAPTER 3

Table of Contents

3.1 Instantiating Fixtures

For for the following examples, assume this fixture

```
class ArnoldPerson(BaseFix):
    MODEL = Person

    first_name = "Arnold"
    family_name = "Schwarz"
```

First you need to instantiate the fixture class and pass the `SqlaFixFact` instance as fist parameter:

```
fixture = ArnoldPerson(fix_fact)
```

You can add additional `kwargs` to alter the default values. For more details see *Substitution of Values*

To get a SQLAlchemy ORM model instance there are 3 ways

- `BaseFix.create()`
- `BaseFix.get()`
- `BaseFix.model()`

3.1.1 create()

Adds this model to the session. This instance is not registered and thus can never be referred to via `BaseFix.get()`. Thus each call to `create()` adds a new instance of the fixture to the DB.

```
arnold_fix_1 = ArnoldPerson(fix_fact).create()
arnold_fix_2 = ArnoldPerson(fix_fact).create()
arnold_fix_3 = ArnoldPerson(fix_fact).create()

db_session.query(Person).count()  # is 3
```

3.1.2 get()

Returns an already existing model instance, or creates one and registers it to be able to find it later and then returns the instance.

Note: Changed properties via the `kwargs` parameter are recognized and result in a new instance.

```
arnold_fix_1 = ArnoldPerson(fix_fact).get()
arnold_fix_2 = ArnoldPerson(fix_fact).get()
arnold_fix_3 = ArnoldPerson(fix_fact).get()

db_session.query(Person).count()  # is 1
```

3.1.3 model()

Returns a model instance of this fixture which is ready to be added. The model itself is not added to the DB but all dependencies are.

```
arnold_fix = ArnoldPerson(fix_fact).model()

db_session.query(Person).count()  # is 0

db_session.add(arnold_fix)
db_session.query(Person).count()  # is 1
```

3.2 Referencing other Fixtures

To build complete scenarios, referencing other fixtures becomes necessary.

To reference other fixtures, use following functions:

- `subFactoryGet()`
- `subFactoryCreate()`
- `subFactoryModel()`

These function behave the same as the methods of the `BaseFix` class.

```
class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"

class ArnoldPerson(BaseFix):
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)
```

Prefer the usage of `subFactoryModel()` for referencing other fixtures.

You can also add `kwargs` to alter properties of the factory as described in *Substitution of Values*

```
class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"

class ArnoldPerson(BaseFix):
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)

class ArnoldPersonAdmin(BaseFix):
    MODEL = Person
```

```
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount, name="arney-admin")
```

Do not use the substitutions extensively at referencing other fixtures. In most cases an own fixture should be defined. Like in the example from above.

```
class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"

class ArnoldPerson(BaseFix):
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)

class ArnoldAdminAccount(BaseFix):
    MODEL = Account
    name = "arney-admin"

class ArnoldPersonAdmin(BaseFix):
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAdminAccount)
```

To save you the burden of copying to much information, you could of course inherit from other fixtures. This inherits all properties except those you overwrite.

```
class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"

class ArnoldPerson(BaseFix):
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)

class ArnoldAdminAccount(ArnoldAccount):
    name = "arney-admin"

class ArnoldPersonAdmin(ArnoldPerson):
    account = sqla_fix_fact.subFactoryModel(ArnoldAdminAccount)
```

You can also add sub-factories in lists

```
class ViewRole(BaseFix):
    MODEL = Role
    name = "View Role"

class EditRole(BaseFix):
    MODEL = Role
    name = "Edit Role"

class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"
    # Use get to reference to the roles, as only one instance in the DB is desired
    roles = [sqla_fix_fact.subFactoryGet(ViewRole), sqla_fix_fact.subFactoryGet(EditRole)]

class ArnoldPerson(BaseFix):
```

```
    MODEL = Person
    name = "Arnold"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)
```

3.3 Substitution of Values

To generate a slightly different fixture than a defined one, does not need a new fixture definition. It is possible to alter properties on instantiation time.

Assume following model and fixture for this page

```
class Account(Base):
    __tablename__ = 'account'

    id = Column(Integer, primary_key=True)
    name = Column('name', Unicode)

class Person(Base):
    __tablename__ = 'person'

    id = Column(Integer, primary_key=True)
    first_name = Column('first_name', Unicode)
    family_name = Column('first_name', Unicode)
    account_id = Column(Integer, ForeignKey('account.id'))
    account = relationship(Account)

# Fixtures
class ArnoldAccount(BaseFix):
    MODEL = Account
    name = "arney"

class ArnoldAdminAccount(BaseFix):
    MODEL = Account
    name = "arney-admin"

class ArnoldPerson(BaseFix):
    MODEL = Person

    first_name = "Arnold"
    family_name = "Schwarz"
    account = sqla_fix_fact.subFactoryModel(ArnoldAccount)
```

To substitute a property simply add a key word argument to the constructor

```
arnold_fix = ArnoldPerson(fix_fact, first_name="Franz").create()

assert arnold_fix.first_name == "Franz"
```

Of course more than one substitution is possible

```
arnold_fix = ArnoldPerson(fix_fact, first_name="Franz", family_name="Egger").create()

assert arnold_fix.first_name == "Franz"
assert arnold_fix.family_name == "Egger"
```

Even references to other factories can be substituted. In this case you need to use one of the sub-factory definition functions!

```
arnold_fix = ArnoldPerson(fix_fact, account=sqla_fix_fact.subFactoryModel(ArnoldAdminAccount)).create

assert arnold_fix.account.name == "arney-admin"
```

3.4 API

class sqla_fix_fact.**SqlaFixFact**(*db_session*)
Fixture factory manager

sqla_fix_fact.**subFactoryGet**(*fixture*, ***kwargs*)
To be used in fixture definition (or in the kwargs of the fixture constructor) to reference a other fixture using the `BaseFix.get()` method.

Parameters

- **fixture** – Desired fixture
- **kwargs** – *Optional:* key words to overwrite properties of this fixture

Returns Proxy object for the desired fixture including the altered properties

sqla_fix_fact.**subFactoryCreate**(*fixture*, ***kwargs*)
To be used in fixture definition (or in the kwargs of the fixture constructor) to reference a other fixture using the `BaseFix.create()` method.

Parameters

- **fixture** – Desired fixture
- **kwargs** – *Optional:* key words to overwrite properties of this fixture

Returns Proxy object for the desired fixture including the altered properties

sqla_fix_fact.**subFactoryModel**(*fixture*, ***kwargs*)
To be used in fixture definition (or in the kwargs of the fixture constructor) to reference a other fixture using the `BaseFix.model()` method.

Parameters

- **fixture** – Desired fixture
- **kwargs** – *Optional:* key words to overwrite properties of this fixture

Returns Proxy object for the desired fixture including the altered properties

class sqla_fix_fact.**BaseFix**(*fix_fact*, ***kwargs*)

Parameters

- **fix_fact** – instance of `SqlaFixFact`
- **kwargs** – *Optional:* key words to overwrite properties of this fixture

create()
Adds this model to the session. This instance is not registered and thus can never be referred to via get

Returns SQLAlchemy Model instance

get()
returns an already existing model instance or creates one, registers it to be able to find it later and then returns the instance

Returns SQLAlchemy Model instance

model()

Returns a model instance of this fixture which is ready to be added. The model itself is not added to the DB but all dependencies are.

Returns SQLAlchemy Model instance

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

S

A

B

C

G

I

M

R

S