SQLAlchemy-i18n Documentation

Release 1.0.3

Konsta Vesterinen

January 30, 2017

SQLAlchemy-i18n is an internationalization extension for SQLAlchemy.

CHAPTER 1

Installation

This part of the documentation covers the installation of SQLAlchemy-i18n.

1.1 Supported platforms

SQLAlchemy-i18n has been tested against the following Python platforms.

- cPython 2.6
- cPython 2.7
- cPython 3.3
- cPython 3.4
- cPython 3.5

1.2 Installing an official release

You can install the most recent official SQLAlchemy-i18n version using pip:

```
pip install sqlalchemy-i18n
```

1.3 Installing the development version

To install the latest version of SQLAlchemy-i18n, you need first obtain a copy of the source. You can do that by cloning the git repository:

```
git clone git://github.com/kvesteri/sqlalchemy-i18n.git
```

Then you can install the source distribution using the `setup.py` script:

```
cd sqlalchemy-i18n
python setup.py install
```

1.4 Checking the installation

To check that SQLAlchemy-i18n has been properly installed, type `python` from your shell. Then at the Python prompt, try to import SQLAlchemy-i18n, and check the installed version:

```
>>> import sqlalchemy_i18n
>>> sqlalchemy_i18n.__version__
1.0.3
```

CHAPTER 2

QuickStart

In order to make your models use SQLAlchemy-i18n you need two things:

1. Assign get_locale function sqlalchemy_utils.i18n module. The following example shows how to do this using flask.ext.babel:

```
import sqlalchemy_utils
from flask.ext.babel import get_locale


sqlalchemy_utils.i18n.get_locale = get_locale
```

2. Call make_translatable() before your models are defined.
3. Define translation model and make it inherit mixin provided by translation_base function

```
import sqlalchemy as sa

from sqlalchemy_i18n import (
    make_translatable,
    translation_base,
    Translatable,
)


make_translatable(options={'locales': ['fi', 'en']})


class Article(Translatable, Base):
    __tablename__ = 'article'
    __translatable__ = {'locales': ['fi', 'en']}

    locale = 'en'  # this defines the default locale

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
    author = sa.Column(sa.Unicode(255))


class ArticleTranslation(translation_base(Article)):
    __tablename__ = 'article_translation'

    name = sa.Column(sa.Unicode(255))

    content = sa.Column(sa.UnicodeText)
```

```
article = Article()
article.name = u'Some article'

session.add(article)
session.commit()
```

CHAPTER 3

Basic usage

3.1 Current translation

Current translation is a hybrid property in parent object that returns the associated translation object for current locale.:

```
article = Article()
article.current_translation.name = 'Some article'
```

You can also directly set the current translation:

```
article.current_translation = ArticleTranslation(name='Some article')
```

Articles and translations can be efficiently fetched using various SQLAlchemy loading strategies:

```
session.query(Article).options(joinedload(Article.current_translation))
```

3.2 Fallback translation

If there is no translation available for the current locale then fallback locale is being used. Fallback translation is a convenient hybrid property for accessing this translation object.:

```
article = Article()
article.translations.en.name = 'Some article'

article.fallback_translation.name  # Some article
```

Fallback translation is especially handy in situations where you don't necessarily have all the objects translated in various languages but need to fetch them efficiently.

```
query = (
    session.query(Article)
    .options(joinedload(Article.current_translation))
    .options(joinedload(Article.fallback_translation))
)
```

3.3 Translatable columns as hybrids

For each translatable column SQLAlchemy-i18n creates a hybrid property in the parent class. These hybrid properties always point at the current translation.

Example:

```
article = Article()
article.name = u'Some article'

article.translations['en'].name  # u'Some article'
```

If the there is no translation available for current locale then these hybrids return the translation for fallback locale. Let's assume the current locale here is 'fi':

```
article = Article()
article.translations.fi.name = ''
article.translations.en.name = 'Some article'

article.name  # 'Some article'
```

3.4 Accessing translations

Dictionary based access:

```
article.translations['en'].name = u'Some article'
```

Attribute access:

```
article.translations.en.name = u'Some article'
article.translations.fi.name = u'Joku artikkeli'
```

CHAPTER 4

Queries

4.1 Joinedload current translation

```
import sqlalchemy as sa


articles = (
    session.query(Article)
    .options(sa.orm.joinedload(Article.current_translation))
)

print articles[0].name
```

4.2 Joinedload arbitrary translations

```
import sqlalchemy as sa


articles = (
    session.query(Article)
    .options(sa.orm.joinedload(Article.translations['fi']))
    .options(sa.orm.joinedload(Article.translations['en']))
)
```

You can also use attribute accessors:

```
articles = (
    session.query(Article)
    .options(sa.orm.joinedload(Article.translations.fi))
    .options(sa.orm.joinedload(Article.translations.en))
)
```

4.3 Joinedload all translations

```
articles = (
    session.query(Article)
    .options(sa.orm.joinedload(Article.translations))
)
```

Configuration

Several configuration options exists for SQLAlchemy-i18n. Each of these options can be set at either manager level or model level. Setting options an manager level affects all models using given translation manager where as model level configuration only affects given model.

5.1 Dynamic source locale

Sometimes you may want to have dynamic source (default) locale. This can be achieved by setting *dynamic_source_locale* as *True*.

Consider the following model definition:

```
class Article(Base):
    __tablename__ = 'article'
    __translatable__ = {
        'locales': [u'en', u'fi'],
        'dynamic_source_locale': True
    }

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)

    author = sa.Column(sa.Unicode(255))

    def get_locale(self):
        return 'en'


class ArticleTranslation(translation_base(Article)):
    __tablename__ = 'article_translation'

    name = sa.Column(sa.Unicode(255))
    content = sa.Column(sa.UnicodeText)
```

Now you can use the dynamic source locales as follows:

```
article = Article(locale='fi', name=u'Joku artikkeli')
article.name == article.translations['fi'].name  # True

article2 = Article(locale='en', name=u'Some article)
article2.name == article.translations['en'].name  # True
```

As with regular translations, the translations using dynamic source locales can even be fetched efficiently using good old SQLAlchemy loading constructs:

```
articles = (
    session.query(Article)
    .options(sa.orm.joinedload(Article.current_translation))
)  # loads translations based on the locale in the parent class
```

5.2 Other options

- locales

    Defines the list of locales that given model or manager supports

- auto_create_locales

    Whether or not to auto-create all locales whenever some of the locales is created. By default this option is True. It is highly recommended to leave this as True, since not creating all locales at once can lead to problems in multithreading environments.

    Consider for example the following situtation. User creates a translatable Article which has two translatable fields (name and content). At the first request this article is created along with one translation table entry with locale 'en'.

    After this two users edit the finnish translation of this article at the same time. The application tries to create finnish translation twice resulting in database integrity errors.

- fallback_locale

    The locale which will be used as a fallback for translation hybrid properties that return None or empty string.

- translations_relationship_args

    Dictionary of arguments passed as defaults for automatically created translations relationship.:

    class Article(Base): __tablename__ = 'article' __translatable__ = {

        'locales': [u'en', u'fi'], 'translations_relationship_args': {

            'passive_deletes': False

        }

    }

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)

    author = sa.Column(sa.Unicode(255))

    def get_locale(self): return 'en'

    class ArticleTranslation(translation_base(Article)): __tablename__ = 'article_translation'

    name = sa.Column(sa.Unicode(255)) content = sa.Column(sa.UnicodeText)

CHAPTER 6

License