ascetic Documentation

Release 0

Ivan Zakrevsky

Mar 11, 2018

Contents

Ascetic exists as a super-lightweight datamapper ORM (Object-Relational Mapper) for Python.

- Home Page: https://bitbucket.org/emacsway/ascetic
- Docs: https://ascetic.readthedocs.io/
- Browse source code (canonical repo): https://bitbucket.org/emacsway/ascetic/src
- GitHub mirror: https://github.com/emacsway/ascetic
- Get source code (canonical repo): `git clone https://bitbucket.org/emacsway/ascetic.git`
- Get source code (mirror): `git clone https://github.com/emacsway/ascetic.git`
- PyPI: https://pypi.python.org/pypi/ascetic

CHAPTER 1

About

Ascetic ORM based on "Data Mapper" pattern. It also supports "Active Record" pattern, but only as a wrapper, the model class is fully free from any service logic. Ascetic ORM follows the KISS principle. Has automatic population of fields from database (see the example below) and minimal size. You do not have to specify the columns in the class. This follows the DRY principle. Ascetic ORM as small as possible.

Inside ascetic.contrib (currently under development) you can find the next solutions:

- multilingual
- polymorphic relations
- polymorphic models (supports for "Single Table Inheritance", "Concrete Table Inheritance" and "Class Table Inheritance" aka Django "Multi-table inheritance")
- "Materialized Path" implementation to handle tree structures
- versioning (which stores only diff, not content copy)

All extensions support composite primary/foreign keys.

"Identity Map" has SERIALIZABLE isolation level by default.

What Ascetic ORM does not? Ascetic ORM does not make any data type conversions (use connection features like this), and does not has "Unit of Work". I recommend using a Storm ORM, if you need these features.

Ascetic ORM is released under the MIT License (see LICENSE file for details).

This project is currently under development, and not stable. If you are looking for stable KISS-style ORM, pay attention to Storm ORM.

Contents:

Table of Contents

CHAPTER 2

PostgreSQL Example

Using these tables:

```
CREATE TABLE ascetic_tests_models_author (
    id serial NOT NULL PRIMARY KEY,
    first_name VARCHAR(40) NOT NULL,
    last_name VARCHAR(40) NOT NULL,
    bio TEXT
);
CREATE TABLE books (
    id serial NOT NULL PRIMARY KEY,
    title VARCHAR(255),
    author_id integer REFERENCES ascetic_tests_models_author(id) ON DELETE CASCADE
);
```

CHAPTER 3

Configuring

You can configure in one the following ways:

1. Put in your PYTHONPATH file ascetic_settings.py with your settings. See file ascetic/settings.py for more details.

2. Define settings module in environment variable ASCETIC_SETTINGS.

3. Call ascetic.settings.configure(), for example:

```
import ascetic.settings.configure
ascetic.settings.configure({
    'DATABASES': {
        'default': {
            'engine': "postgresql",
            'user': "devel",
            'database': "devel_ascetic",
            'password': "devel",
            'debug': True,
            'initial_sql': "SET NAMES 'UTF8';",
        }
    }
})
```

CHAPTER 4

Model declaration

There is two way to declare models as DataMapper or ActiveRecord.

4.1 Datamapper way

```
class Author(object):
    def __init__(self, id=None, first_name=None, last_name=None, bio=None):
        self.id = id
        self.first_name = first_name
        self.last_name = last_name
        self.bio = bio


class AuthorMapper(Mapper):
        defaults = {'bio': 'No bio available'}
        validations = {'first_name': (
            lambda v: len(v) > 1 or "Too short first name",
            lambda self, key, value: value != self.last_name or "Please, enter␣
↪another first name",
        )}

AuthorMapper(Author)


class Book(object):
    def __init__(self, id=None, title=None, author_id=None):
        self.id = id
        self.title = title
        self.author_id = author_id


class BookMapper(Mapper):
    db_table = 'books'
```

```
    relationships = {
        'author': ForeignKey(Author, related_name='books')
    }

BookMapper(Book)
```

4.2 ActiveRecord way

Indeed, it's not an ActiveRecord, - it's just a wrapper over DataMapper.

```
from ascetic.model import Model
from ascetic.mappers import get_mapper
from ascetic.relations import ForeignKey, OneToMany

class Author(Model):
    class Mapper(object):
        defaults = {'bio': 'No bio available'}
        validations = {'first_name': (
            lambda v: len(v) > 1 or "Too short first name",
            lambda self, key, value: value != self.last_name or "Please, enter␣
↪another first name",
        )}

class Book(Model):
    author = ForeignKey(Author, related_name='books')

    class Mapper(object):
        db_table = 'books'
```

Now we can create, retrieve, update and delete entries in our database. Creation

```
james = Author(first_name='James', last_name='Joyce')
get_mapper(Author).save(james)  # Datamapper way

u = Book(title='Ulysses', author_id=james.id)
u.save()  # Use ActiveRecord wrapper
```

CHAPTER 5

Retrieval

```
a = Author.get(1)
a.first_name # James
a.books      # Returns list of author's books

# Returns a list, using LIMIT based on slice
a = Author.q[:10]   # LIMIT 0, 10
a = Author.q[20:30] # LIMIT 20, 10
```

CHAPTER 6

Updating

```
a = Author.get(1)
a.bio = 'What a crazy guy! Hard to read but... wow!'
a.save()
```

CHAPTER 7

Deleting

```
a.delete()
```

CHAPTER 8

SQLBuilder integration

```
object_list = Book.q.tables(
    (Book.s & Author.s).on(Book.s.author_id == Author.s.id)
).where(
    (Author.s.first_name != 'James') & (Author.s.last_name != 'Joyce')
)[:10]
```

Query object based on sqlbuilder.smartsql, see more info.

CHAPTER 9

Signals support

- pre_init
- post_init
- pre_save
- post_save
- pre_delete
- post_delete
- class_prepared

CHAPTER 10

Web

You can use Ascetic ORM with lightweight web-frameworks, like wheezy.web, Bottle, Tornado, pysi, etc.

CHAPTER 11

Gratitude

Forked from https://github.com/lucky/autumn
Thanks to Jared Kuolt (lucky)

CHAPTER 12

Other projects

See also:

- Storm (properties from class) - excellent and simple ORM!
- Article (in English) "Why I prefer Storm ORM for Python"
- Article (in Russian) " Storm ORM Python"
- Article (in English) "Implementation of Repository pattern for browser's JavaScript"
- Article (in Russian) " Repository   JavaScript"
- SQLAlchemy (scheme from class or database, see "autoload" option)
- Openorm (lightweight datamapper), miror
- SQLObject (scheme from class or database, see "fromDatabase" option)
- Peewee (scheme from class)
- Bazaar ORM
- Twistar (scheme from database), provides asynchronous DB interaction
- Activemodel (scheme from database)
- ActiveRecord like ORM under 200 lines (scheme from database)
- A Query Language extension for Python: Query files, objects, SQL and NoSQL databases with a built-in query language
- simpleql SQL table using nothing but Python to build the query
- Generator expressions for database requests (Python recipe)
- Object Relational Mappers (ORMs)

CHAPTER 13

Indices and tables

- genindex
- modindex
- search