Eve-SQLAlchemy Documentation

Release 0.7.2.dev0

Dominik Kellner

Mar 29, 2020

Contents

Use Eve with SQLAlchemy instead of MongoDB. Re-use your existing SQL data model and expose it via a RESTful Web Service with no hassle.

CHAPTER 1

Documentation

1.1 Installation

This part of the documentation covers the installation of Eve-SQLAlchemy. The first step to using any software package is getting it properly installed.

Installing Eve-SQLAlchemy is simple with pip:

```
$ pip install eve-sqlalchemy
```

1.1.1 Development Version

Eve-SQLAlchemy is actively developed on GitHub, where the code is always available. If you want to work with the development version of Eve-SQLAlchemy, there are two ways: you can either let *pip* pull in the development version, or you can tell it to operate on a git checkout. Either way, virtualenv is recommended.

Get the git checkout in a new virtualenv and run in development mode.

```
$ git clone https://github.com/pyeve/eve-sqlalchemy.git
Cloning into 'eve-sqlalchemy'...
...

$ cd eve-sqlalchemy
$ virtualenv venv
...
Installing setuptools, pip, wheel...
done.

$ . venv/bin/activate
$ pip install .
...
Successfully installed ...
```

This will pull in the dependencies and activate the git head as the current version inside the virtualenv. Then all you have to do is run `git pull origin` to update to the latest version.

To just get the development version without git, do this instead:

```
$ mkdir eve-sqlalchemy
$ cd eve-sqlalchemy
$ virtualenv venv
$ . venv/bin/activate
$ pip install git+https://github.com/pyeve/eve-sqlalchemy.git
...
Successfully installed ...
```

And you're done!

1.2 Tutorial

The example app used by this tutorial is available at `examples/simple` inside the Eve-SQLAlchemy repository.

1.2.1 Schema registration

The main goal of the SQLAlchemy integration in Eve is to separate dependencies and keep model registration depend only on sqlalchemy library. This means that you can simply use something like that:

```
from sqlalchemy import Column, DateTime, ForeignKey, Integer, String, func
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import column_property, relationship

Base = declarative_base()


class CommonColumns(Base):
    __abstract__ = True
    _created = Column(DateTime, default=func.now())
    _updated = Column(DateTime, default=func.now(), onupdate=func.now())
    _etag = Column(String(40))


class People(CommonColumns):
    __tablename__ = 'people'
    id = Column(Integer, primary_key=True, autoincrement=True)
    firstname = Column(String(80))
    lastname = Column(String(120))
    fullname = column_property(firstname + " " + lastname)


class Invoices(CommonColumns):
    __tablename__ = 'invoices'
    id = Column(Integer, primary_key=True, autoincrement=True)
    number = Column(Integer)
    people_id = Column(Integer, ForeignKey('people.id'))
    people = relationship(People, uselist=False)
```

We have used `CommonColumns` abstract class to provide attributes used by Eve, such as `_created` and _updated. These are not needed if you are only reading from the database. However, if your API is also writing to the database, then you need to include them.

1.2.2 Eve settings

All standard Eve settings will work with SQLAlchemy support. However, you need to manually decide which SQLAlchemy declarative classes you wish to register. You can do so using DomainConfig and ResourceConfig, which will give you a default schema (DOMAIN dictionary) derived from your SQLAlchemy models. This is intended as a starting point and to save you from redundant configuration, but there's nothing wrong with customizing this dictionary if you need to!

```
from eve_sqlalchemy.config import DomainConfig, ResourceConfig
from eve_sqlalchemy.examples.simple.tables import Invoices, People

DEBUG = True
SQLALCHEMY_DATABASE_URI = 'sqlite://'
SQLALCHEMY_TRACK_MODIFICATIONS = False
RESOURCE_METHODS = ['GET', 'POST']

# The following two lines will output the SQL statements executed by
# SQLAlchemy. This is useful while debugging and in development, but is turned
# off by default.
# --------
# SQLALCHEMY_ECHO = True
# SQLALCHEMY_RECORD_QUERIES = True

# The default schema is generated using DomainConfig:
DOMAIN = DomainConfig({
    'people': ResourceConfig(People),
    'invoices': ResourceConfig(Invoices)
}).render()

# But you can always customize it:
DOMAIN['people'].update({
    'item_title': 'person',
    'cache_control': 'max-age=10,must-revalidate',
    'cache_expires': 10,
    'resource_methods': ['GET', 'POST', 'DELETE']
})

# Even adding custom validations just for the REST-layer is possible:
DOMAIN['invoices']['schema']['number'].update({
    'min': 10000
})
```

A note about using update

A common mistake is to use update to try to update values in a nested dictionary. This will overwrite the entire dictionary and probably cause KeyErrors.

```
# Instead of this...
DOMAIN['foo'].update({
  'datasource': {  # 'datasource' will only contain 'default_sort'!
    'default_sort': [('id', -1)]
  }
```

(continues on next page)

(continued from previous page)

```
})
# ... do this:
DOMAIN['foo']['datasource']['default_sort'] = [('id', -1)]
```

1.2.3 Authentication example

This example is based on the Token-Based tutorial from Eve Authentication. First we need to create eve-side authentication:

```
"""
Auth-Token
~~~~~~~~~~

Securing an Eve-powered API with Token based Authentication and
SQLAlchemy.

This snippet by Andrew Mleczko can be used freely for anything
you like. Consider it public domain.
"""


from eve import Eve
from eve.auth import TokenAuth
from .models import User
from .views import register_views


class TokenAuth(TokenAuth):
    def check_auth(self, token, allowed_roles, resource, method):
        """First we are verifying if the token is valid. Next
        we are checking if user is authorized for given roles.
        """
        login = User.verify_auth_token(token)
        if login and allowed_roles:
            user = app.data.driver.session.query(User).get(login)
            return user.isAuthorized(allowed_roles)
        else:
            return False


if __name__ == '__main__':
    app = Eve(auth=TokenAuth)
    register_views(app)
    app.run()
```

Next step is the *User* SQLAlchemy model:

```
"""
Auth-Token
~~~~~~~~~~

Securing an Eve-powered API with Token based Authentication and
SQLAlchemy.

This snippet by Andrew Mleczko can be used freely for anything
```

(continues on next page)

(continued from previous page)

```
you like. Consider it public domain.
"""

import hashlib
import string
import random

from itsdangerous import TimedJSONWebSignatureSerializer \
    as Serializer
from itsdangerous import SignatureExpired, BadSignature

from werkzeug.security import generate_password_hash, \
    check_password_hash

from sqlalchemy.orm import validates
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
SECRET_KEY = 'this-is-my-super-secret-key'


class User(Base):
    __tablename__ = 'users'

    login = Column(String, primary_key=True)
    password = Column(String)
    roles = relationship("Role", backref="users")

    def generate_auth_token(self, expiration=24*60*60):
        """Generates token for given expiration
        and user login."""
        s = Serializer(SECRET_KEY, expires_in=expiration)
        return s.dumps({'login': self.login })

    @staticmethod
    def verify_auth_token(token):
        """Verifies token and eventually returns
        user login.
        """
        s = Serializer(SECRET_KEY)
        try:
            data = s.loads(token)
        except SignatureExpired:
            return None # valid token, but expired
        except BadSignature:
            return None # invalid token
        return data['login']

    def isAuthorized(self, role_names):
        """Checks if user is related to given role_names.
        """
        allowed_roles = set([r.id for r in self.roles])\
            .intersection(set(role_names))
        return len(allowed_roles) > 0

    def encrypt(self, password):
        """Encrypt password using werkzeug security module.
```

(continues on next page)

(continued from previous page)

```
        """
        return generate_password_hash(password)

    @validates('password')
    def _set_password(self, key, value):
        """Using SQLAlchemy validation makes sure each
        time password is changed it will get encrypted
        before flushing to db.
        """
        return self.encrypt(value)

    def check_password(self, password):
        if not self.password:
            return False
        return check_password_hash(self.password, password)
```

And finally a flask login view:

```
"""
Auth-Token
~~~~~~~~~~

Securing an Eve-powered API with Token based Authentication and
SQLAlchemy.

This snippet by Andrew Mleczko can be used freely for anything
you like. Consider it public domain.
"""

import json
import base64

from flask import request, jsonify
from werkzeug.exceptions import Unauthorized
from .models import User


def register_views(app):

    @app.route('/login', methods=['POST'])
    def login(**kwargs):
        """Simple login view that expect to have username
        and password in the request POST. If the username and
        password matches - token is being generated and return.
        """
        data = request.get_json()
        login = data.get('username')
        password = data.get('password')

        if not login or not password:
            raise Unauthorized('Wrong username and/or password.')
        else:
            user = app.data.driver.session.query(User).get(login)
            if user and user.check_password(password):
                token = user.generate_auth_token()
                return jsonify({'token': token.decode('ascii')})
        raise Unauthorized('Wrong username and/or password.')
```

1.2.4 Start Eve

That's almost everything. Before you can start Eve you need to bind SQLAlchemy from the Eve data driver:

```
app = Eve(validator=ValidatorSQL, data=SQL)
db = app.data.driver
Base.metadata.bind = db.engine
db.Model = Base
```

Now you can run Eve:

```
app.run(debug=True)
```

and start it:

```
$ python app.py
 * Running on http://127.0.0.1:5000/
```

and check that everything is working like expected, by trying requesting *people*:

```
$ curl http://127.0.0.1:5000/people/1
```

```
{
    "id": 1,
    "fullname": "George Washington",
    "firstname": "George",
    "lastname": "Washington",
    "_etag": "31a6c47afe9feb118b80a5f0004dd04ee2ae7442",
    "_created": "Thu, 21 Aug 2014 11:18:24 GMT",
    "_updated": "Thu, 21 Aug 2014 11:18:24 GMT",
    "_links": {
        "self": {
            "href":"/people/1",
            "title":"person"
        },
        "parent": {
            "href": "",
            "title": "home"
        },
        "collection": {
            "href": "/people",
            "title": "people"
        }
    },
}
```

1.2.5 Using Flask-SQLAlchemy

If you are using Flask-SQLAlchemy, you can use your existing `db` object in the `SQL` class driver, rather than the empty one it creates.

You can do this by subclassing `SQL` and overriding the driver.

```
from eve_sqlalchemy import SQL as _SQL
from flask_sqlalchemy import SQLAlchemy

```

(continues on next page)

(continued from previous page)

```
db = SQLAlchemy(app)

class SQL(_SQL):
   driver = db

app = Eve(validator=ValidatorSQL, data=SQL)
```

1.2.6 SQLAlchemy expressions

With this version of Eve you can use SQLAlchemy expressions such as: *like*, *in*, *any*, etc. For more examples please check SQLAlchemy internals.

Query strings are supported, allowing for filtering and sorting. Both native Mongo queries and Python conditional expressions are supported. For more examples please check SQLAlchemy filtering.

Filtering

Generating 'exact' matches

Here we are asking for all *people* where *lastname* value is *Smith*:

```
/people?where={"lastname":"Smith"}
```

which produces where closure:

```
people.lastname = "Smith"
```

Generating multiple 'exact' matches

Here we are asking for all *people* where *age* value is between *50* and *60*:

```
/people?where=age>50 and age<60
```

which produces where closure:

```
people.age > 50 AND people.age < 60
```

Generating 'like' matches

Here we are asking for all *people* where *lastname* value contains *Smi*:

```
/people?where={"lastname":"like(\"Smi%\")"}
```

which produces where closure:

```
people.lastname LIKE "Smi%"
```

Generating 'in' matches

Here we are asking for all *people* where *firstname* value is *John* or *Fred*:

```
/people?where={"firstname":"in(\"(\'John\',\'Fred\')\")"}
```

or you can also use the other syntax query

```
/people?where={"firstname":['John','Fred']}
```

which produces where closure:

```
people.firstname IN ("John", "Fred")
```

Generating 'similar to' matches

```
/people?where={"firstname":"similar to(\"(\'%ohn\'|\'%acob\')\")"}
```

which produces where closure:

```
people.firstname SIMILAR TO '("%ohn"|"%acob")'
```

Generating 'any' matches

If you have postgresql ARRAY column you can use *any*:

```
/documents?where={"keywords":"any(\"critical\")"}
```

which produces where closure:

```
"critical" = ANY(documents.keywords)
```

Generating 'not null' matches

```
/documents?where={"keywords":"!=null"}
```

which produces where closure:

```
documents.keywords IS NOT NULL
```

Generating 'datetime' matches

Here we are asking for all *documents* that where *_created* after *Mon, 17 Oct 2019 03:00:00 GMT*:

```
/documents?where=_created> \"Mon, 17 Oct 2019 03:00:00 GMT\"
```

which produces where closure:

```
documents._created > 2019-10-17 03:00:00
```

Sorting

Starting from version 0.2 you can use SQLAlchemy ORDER BY expressions such as: *nullsfirst*, *nullslast*, etc.

Using those expresssion is straightforward, just pass it as 3 argument to sorting:

```
/people?sort=[("lastname", -1, "nullslast")]
```

which produces order by expression:

```
people.lastname DESC NULLS LAST
```

You can also use the following python-Eve syntax:

```
/people?sort=lastname,-created_at
```

FAQ

cURL

Keep in mind that every browser or cURL generator can implement its own encoder, and not all produce the same result. So, adding *–data-urlencode* to the curl query should work.

```
curl -iG --data-urlencode where='_created> "Thu, 22 Nov 2018 09:00:00 GMT"'␣
↪localhost:5000/people
```

1.2.7 Embedded resources

Eve-SQLAlchemy support the embedded keyword of python-eve (Eve Embedded Resource Serialization).

```
/people?embedded={"address":1}
```

For example, the following request will list the people and embedded their addresses.

Starting from version 0.4.0a, only the fields that have the projection (Eve Projections) enabled are included in the associated resource. This was necessary to avoid endless loops when relationship between resources were referring each other.

1.3 Simple example

Create a file, called trivial.py, and include the following:

```
''' Trivial Eve-SQLAlchemy example. '''
from eve import Eve
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import column_property

from eve_sqlalchemy import SQL
from eve_sqlalchemy.config import DomainConfig, ResourceConfig
from eve_sqlalchemy.validation import ValidatorSQL

Base = declarative_base()


class People(Base):
    __tablename__ = 'people'
    id = Column(Integer, primary_key=True, autoincrement=True)
    firstname = Column(String(80))
    lastname = Column(String(120))
    fullname = column_property(firstname + " " + lastname)


SETTINGS = {
    'DEBUG': True,
    'SQLALCHEMY_DATABASE_URI': 'sqlite://',
```

(continues on next page)

(continued from previous page)

```
    'SQLALCHEMY_TRACK_MODIFICATIONS': False,
    'DOMAIN': DomainConfig({
        'people': ResourceConfig(People)
    }).render()
}

app = Eve(auth=None, settings=SETTINGS, validator=ValidatorSQL, data=SQL)

# bind SQLAlchemy
db = app.data.driver
Base.metadata.bind = db.engine
db.Model = Base
db.create_all()

# Insert some example data in the db
if not db.session.query(People).count():
    db.session.add_all([
        People(firstname=u'George', lastname=u'Washington'),
        People(firstname=u'John', lastname=u'Adams'),
        People(firstname=u'Thomas', lastname=u'Jefferson')])
    db.session.commit()

# using reloader will destroy in-memory sqlite db
app.run(debug=True, use_reloader=False)
```

Run this command to start the server:

```
python trivial.py
```

Open the following in your browser to confirm that the server is serving:

```
http://127.0.0.1:5000/
```

You will see something like this:

```
<resource>
    <link rel="child" href="people" title="people"/>
</resource>
```

Now try the people URL:

```
http://127.0.0.1:5000/people
```

You will see the three records we preloaded.

```
<resource href="people" title="people">
    <link rel="parent" href="/" title="home"/>
    <_meta>
        <max_results>25</max_results>
        <page>1</page>
        <total>3</total>
    </_meta>
    <_updated>Sun, 22 Feb 2015 16:28:00 GMT</_updated>
    <firstname>George</firstname>
    <fullname>George Washington</fullname>
    <id>1</id>
```

(continues on next page)

(continued from previous page)

```
    <lastname>Washington</lastname>
</resource>
```

1.4 Upgrading

1.4.1 Upgrading from 0.6.0 to 0.7.0

Eve-SQLAlchemy is now based on Eve 0.7, which introduces potentially breaking changes:

- The ETag format was changed to comply with RFC 7232-2.3. Be aware the ETag header values are now enclosed with double-quotes.
- Eve now returns a *428 Precondition Required* instead of a generic *403 Forbidden* when the *If-Match* request header is missing.

For a comprehensive list of changes refer to the official changelog.

1.4.2 Upgrading from 0.5.0 to 0.6.0

There is one potentially breaking change in 0.6.0: Due to a regression 0.5.0 did not return *None/null* values anymore (as Eve does and 0.4.1 did). That means your API might return slightly different responses after upgrading to 0.6.0 than it did before. If it's really a breaking change for you depends on your API specification and your clients.

1.4.3 Upgrading from 0.4.1 to 0.5.0

There are two breaking changes in 0.5.0:

1. Eve-SQLAlchemy now handles related IDs and embedded objects with just one field in the payload, just as Eve does. This will most likely affect your consumers, too!
2. We introduced a new way to register your SQLAlchemy models with Eve. So far there is no backward compatible wrapper for the former `registerSchema` decorator.

Let's look at the needed changes in more detail. To illustrate both changes, we will look at the following models (the full code is in the *examples* directory):

```
class People(CommonColumns):
    __tablename__ = 'people'
    id = Column(Integer, primary_key=True, autoincrement=True)
    firstname = Column(String(80))
    lastname = Column(String(120))
    fullname = column_property(firstname + " " + lastname)


class Invoices(CommonColumns):
    __tablename__ = 'invoices'
    id = Column(Integer, primary_key=True, autoincrement=True)
    number = Column(Integer)
    people_id = Column(Integer, ForeignKey('people.id'))
    people = relationship(People, uselist=False)
```

1. Related IDs and embedding

Getting an invoice in 0.4.1 will return the *people_id* in the payload:

```
{
    "_created": "Sat, 15 Jul 2017 02:24:58 GMT",
    "_etag": null,
    "_id": 1,
    "_updated": "Sat, 15 Jul 2017 02:24:58 GMT",
    "id": 1,
    "number": 42,
    "people_id": 1
}
```

And, if you embed the related `People` object, you will get:

```
{
    "_created": "Sat, 15 Jul 2017 02:39:25 GMT",
    "_etag": null,
    "_id": 1,
    "_updated": "Sat, 15 Jul 2017 02:39:25 GMT",
    "id": 1,
    "number": 42,
    "people": {
        "_created": "Sat, 15 Jul 2017 02:39:25 GMT",
        "_etag": null,
        "_id": 1,
        "_updated": "Sat, 15 Jul 2017 02:39:25 GMT",
        "firstname": "George",
        "fullname": "George Washington",
        "id": 1,
        "lastname": "Washington"
    },
    "people_id": 1
}
```

But this was actually not how Eve itself is handling this. In order to follow the APIs generated by Eve more closely, we decided to adopt the way Eve is doing embedding and use the same field for both the related ID and the embedded document. Which means starting in 0.5.0, the first response looks like this:

```
{
    "_created": "Sat, 15 Jul 2017 02:52:20 GMT",
    "_etag": "26abc30d70f57de186d9f99a7192444fcf538519",
    "_updated": "Sat, 15 Jul 2017 02:52:20 GMT",
    "id": 1,
    "number": 42,
    "people": 1
}
```

And the second one (with embedding):

```
{
    "_created": "Sat, 15 Jul 2017 02:54:44 GMT",
    "_etag": "8a1121cacb77a21f9ff3b5a85cfba0a501a538ea",
    "_updated": "Sat, 15 Jul 2017 02:54:44 GMT",
    "id": 1,
    "number": 42,
    "people": {
```

(continues on next page)

(continued from previous page)

```
        "_created": "Sat, 15 Jul 2017 02:54:44 GMT",
        "_updated": "Sat, 15 Jul 2017 02:54:44 GMT",
        "firstname": "George",
        "fullname": "George Washington",
        "id": 1,
        "lastname": "Washington"
    }
}
```

2. Registering of SQLAlchemy models

In 0.4.1, you were most likely doing something along the following lines in your *settings.py*:

```
ID_FIELD = 'id'
config.ID_FIELD = ID_FIELD

registerSchema('people')(People)
registerSchema('invoices')(Invoices)

DOMAIN = {
    'people': People._eve_schema['people'],
    'invoices': Invoices._eve_schema['invoices']
}
```

There are good news: manually (and globally) setting `ID_FIELD`, including the workaround of setting `config.ID_FIELD`, is not required anymore. The same applies to `ITEM_LOOKUP_FIELD` and `ITEM_URL`. While you can still override them, they are now preconfigured at the resource level depending on your models' primary keys.

The required configuration for the models above simplifies to:

```
from eve_sqlalchemy.config import DomainConfig, ResourceConfig

DOMAIN = DomainConfig({
    'people': ResourceConfig(People),
    'invoices': ResourceConfig(Invoices)
}).render()
```

Note: If you've modified `DATE_CREATED`, `LAST_UPDATED` or `ETAG`, you have to pass their value to `DomainConfig.render()`. They are needed during rendering the final `DOMAIN` configuration.

```
DomainConfig(domain_dict).render(date_created=DATE_CREATED,
                                 last_updated=LAST_UPDATED,
                                 etag=ETAG)
```

1.5 How to Contribute

Contributions are welcome! Not familiar with the codebase yet? No problem! There are many ways to contribute to open source projects: reporting bugs, helping with the documentation, spreading the word and of course, adding new features and patches.

1.5.1 Getting Started

1. Make sure you have a GitHub account.
2. Open a new issue, assuming one does not already exist.
3. Clearly describe the issue including steps to reproduce when it is a bug.

1.5.2 Making Changes

- Fork the repository on GitHub.
- Create a topic branch from where you want to base your work.
- This is usually the `master` branch.
- Please avoid working directly on the `master` branch.
- Make commits of logical units (if needed rebase your feature branch before submitting it).
- Check for unnecessary whitespace with `git diff --check` before committing.
- Make sure your commit messages are in the proper format.
- If your commit fixes an open issue, reference it in the commit message (#15).
- Make sure your code conforms to PEP8 (we're using flake8 for PEP8 and extra checks).
- Make sure you have added the necessary tests for your changes.
- Run all the tests to assure nothing else was accidentally broken.
- Run again the entire suite via tox to check your changes against multiple python versions. `pip install tox; tox`
- Don't forget to add yourself to AUTHORS.

These guidelines also apply when helping with documentation (actually, for typos and minor additions you might choose to fork and edit).

1.5.3 Submitting Changes

- Push your changes to a topic branch in your fork of the repository.
- Submit a Pull Request.
- Wait for maintainer feedback.

1.5.4 Keep fork in sync

The fork can be kept in sync by following the instructions here.

1.5.5 Join us on IRC

If you're interested in contributing to the Eve-SQLAlchemy project or have any questions about it, come join us in Eve's #python-eve channel on irc.freenode.net.

1.5.6 First time contributor?

It's alright. We've all been there. See next chapter.

1.5.7 Don't know where to start?

There are usually several TODO comments scattered around the codebase, maybe check them out and see if you have ideas, or can help with them. Also, check the open issues in case there's something that sparks your interest. There's also a special `contributor-friendly` label flagging some interesting feature requests and issues that will easily get you started - even without knowing the codebase yet. If you're fluent in English (or notice any typo and/or mistake), feel free to improve the documentation. In any case, other than GitHub help pages, you might want to check this excellent Effective Guide to Pull Requests

CHAPTER 2

Changelog

2.1 0.7.2 (unreleased)

- Nothing changed yet.

2.2 0.7.1 (2019-08-10)

- Updated Tutorial to use werkzeug.security module (#196) [Mandar Vaze]
- Require Flask-SQLAlchemy >= 2.4 and SQLAlchemy >= 1.3 due to security issues [Dominik Kellner]
- Support filtering on embedded document fields / across relations (#186) [Dominik Kellner]
- Fix sorting across relations [Dominik Kellner]
- Add Python 3.7 and PyPy3 to supported (and tested) versions [Dominik Kellner]
- Pin SQLAlchemy version due to warnings in Flask-SQLAlchemy [Dominik Kellner]
- Improve documentation (#187, #189) [Marc Vila]

2.3 0.7.0 (2018-10-08)

- Eve 0.7 support (#178) [Nicola Iarocci]

2.4 0.6.0 (2018-08-15)

- Fix querying of list relations using *where* [Dominik Kellner]
- Update Tutorial (#177) [Nicola Iarocci]

- Return None-values again (#155) [Cuong Manh Le]
- Allow to supply own Flask-SQLAlchemy driver (#86) [fubu]
- Support columns with server_default (#160) [Asif Mahmud Shimon]

2.5 0.5.0 (2017-10-22)

- Add DomainConfig and ResourceConfig to ease configuration (#152) [Dominik Kellner]
- Fixes in documentation (#151) [Alessandro De Angelis]
- Fix deprecated import warning (#142) [Cuong Manh Le]
- Configure *zest.releaser* for release management (#137) [Dominik Kellner, Øystein S. Haaland]
- Leverage further automated syntax and formatting checks (#138) [Dominik Kellner]
- Clean up specification of dependencies [Dominik Kellner]
- Added 'Contributing' section to docs (#129) [Mario Kralj]
- Fix trivial app output in documentation (#131) [Michal Vlasák]
- Added dialect-specific PostgreSQL JSON type (#133) [Mario Kralj]
- Fix url field in documentation about additional lookup (#110) [Killian Kemps]
- Compatibility with Eve 0.6.4 and refactoring of tests (#92) [Dominik Kellner]

2.6 0.4.1 (2015-12-16)

- improve query with null values [amleczko]

2.7 0.4.0a3 (2015-10-20)

- *hybrid_properties* are now readonly in Eve schema [amleczko]

2.8 0.4.0a2 (2015-09-17)

- PUT drops/recreates item in the same transaction [goneri]

2.9 0.4.0a1 (2015-06-18)

- support the Python-Eve generic sorting syntax [Goneri Le Bouder]
- add support for *and_* and *or_* conjunctions in sqla expressions [toxsick]
- embedded table: use DOMAIN to look up the resource fields [Goneri Le Bouder]

2.10 0.3.4 (2015-05-18)

- fix setup.py metadata
- fix how embedded documents are resolved [amleczko]

2.11 0.3.3 (2015-05-13)

- added support of SA association proxy [Kevin Roy]
- make sure relationships are generated properly [amleczko]

2.12 0.3.2 (2015-05-01)

- add fallback on attr.op if the operator doesn't exists in the *ColumnProperty* [Kevin Roy]
- add support for PostgreSQL JSON type [Goneri Le Bouder]

2.13 0.3.1 (2015-04-29)

- more flexible handling sqlalchemy operators [amleczko]

2.14 0.3 (2015-04-17)

- return everything as dicts instead of SQLAResult, remove SQLAResult [Leonidaz0r]
- fix update function, this closes #22 [David Durieux]
- fixed replaced method, we are compatible with Eve>=0.5.1 [Kevin Roy]
- fixed jsonify function [Leonidaz0r]
- update documentation [Alex Kerney]
- use id_field column from the config [Goneri Le Bouder]
- add flake8 in tox [Goneri Le Bouder]

2.15 0.2.1 (2015-02-25)

- always wrap embedded documents [amleczko]

2.16 0.2 (2015-01-27)

- various bugfixing [Arie Brosztein, toxsick]
- refactor sorting parser, add sql order by expresssions; please check https://eve-sqlalchemy.readthedocs.org/#sqlalchemy-sorting for more details [amleczko]

2.17 0.1 (2015-01-13)

- First public preview release. [amleczko]