yalchemy Documentation

Release 0.2.15

Clover Health

May 06, 2020

Contents

1 yalchemy **1**
- 1.1 Quick Start . . . 1
- 1.2 Next Steps . . . 3

2 Installation **5**

3 Interface **7**

4 Release Notes **13**

5 Contributing Guide **15**
- 5.1 Setup . . . 15
- 5.2 Testing and Validation . . . 15
- 5.3 Documentation . . . 15
- 5.4 Releases and Versioning . . . 16

Index **17**

CHAPTER 1

yalchemy

yalchemy is a tool for converting YAML (or Python dictionaries) to SQLAlchemy objects.

1.1 Quick Start

yalchemy provides the following intermediate yalchemy objects:

1. *Table* - For converting SQLAlchemy Tables.
2. *Index* - For converting SQLAlchemy Indexes.
3. *ForeignKey* - For converting SQLAlchemy ForeignKeys.
4. *UniqueConstraint* - For converting SQLAlchemy UniqueConstraints.
5. *CheckConstraint* - For converting SQLAlchemy CheckConstraints.
6. *Column* - For converting SQLAlchemy Columns
7. *TableSet* - A utility for representing a list of SQLAlchemy Tables.

Each of these objects implement the following yalchemy object interface:

class yalchemy.table_structures.**Yalchemy**
The base yalchemy object that defines the interface for serialization

classmethod from_dict(*dict_obj*)
Constructs a yalchemy object from a dictionary

classmethod from_sqla(*sqla_obj*)
Constructs a yalchemy object from a sqlalchemy object

Parameters **sqla_obj** – The sqlalchemy object

classmethod from_yaml(*yaml_str*, *loader_factory=None*)
Loads a yalchemy object from a yaml string

Parameters

- **yaml** (*str*) – The yaml string
- **loader_factory** (*function(stream)*) – (optional) constructs a custom PyYaml loader. If not provided and libyaml is installed, it will default to CSafeLoader. If not provided and libyaml is not installed, it will default to SafeLoader.

to_dict()
Converts a yalchemy object to a dictionary.

Note: Wrapper around _to_dict that will order the _to_dict returned dict by the __slots__ order in an OrderedDict. Please remember to override `_to_dict` and not `to_dict` when subclassing yalchemy.

---

to_sqla(***kwargs*)
Converts a yalchemy object to a sqlalchemy object

Parameters **kwargs** – Keyword arguments specific to the yalchemy object being created. Some objects need special parameters for their construction (e.g. sqlalchemy MetaData)

to_yaml()
Converts a yalchemy object to a yaml string

With this interface, it's possible to do the following types of conversions:

```
import yalchemy

# Create a SQLAlchemy table from a dictionary
table = yalchemy.Table.from_dict({
    'name': 'my_table',
    'schema': 'schema',
    'columns': [
        {'name': 'col1', 'datatype': 'varchar', 'format': [123], 'null': '^@'},
        {'name': 'col2', 'datatype': 'integer', 'required': True},
        {'name': 'col3', 'datatype': 'timestamptz', 'required': True,
         'default': {'type': 'function', 'value': 'NOW()'}}
    ],
    'foreign_keys': [
        {'column': 'col2', 'remote_table': 'other_table', 'remote_column': 'other_col
↪'}
    ],
    'indexes': [{'columns': ['col1', 'col2']}],
    'unique_constraints': [{'columns': ['col3']}],
    'primary_keys': ['col2']
}).to_sqla(...)

# Columns, foreign keys, and indexes can also be created / converted in the same way
column = yalchemy.Column.from_dict({
    'name': 'col2',
    'datatype': 'integer',
    'required': True
}).to_sqla(...)

fkey = yalchemy.ForeignKey.from_dict({
    'column': 'col2',
    'remote_table': 'other_table',
    'remote_column': 'other_col'
}).to_sqla(...)

index = yalchemy.Index.from_dict({'columns': ['col1', 'col2']}).to_sqla(...)
```

In yalchemy, the keys used in the dictionaries in the `from_dict` constructors are similar to the constructor arguments. For example,

```
# Create a yalchemy column object directly
column = yalchemy.Column(name='col2', remote_table='other_table', remote_column=
↪'other_col')

# Create the yalchemy object from a dictionary
column = yalchemy.Column.from_dict({'name': 'col2', 'remote_table': 'other_table',
↪'remote_column': 'other_col'})

# Convert it to sqlalchemy
column.to_sqla()
```

When instantiating more complex objects like a *Table*, the keys used in the dictionaries are also similar to the constructor arguments, however, the values are sometimes other yalchemy objects. It is recommended to use `from_dict` constructor when instantiating a yalchemy object whenever possible.

Keep in mind that `to_sqla` methods sometimes require additional arguments for conversion. Please consult the *Interface* section for details on all of the `to_sqla` methods.

1.2 Next Steps

For the full documentation on the interface and the spec used to create yalchemy object from dictionaries, view the *Interface* section.

CHAPTER 2

Installation

yalchemy can be installed with:

```
pip3 install yalchemy
```

To improve performance, it is recommended to install LibYAML to leverage the native C implementation of the PyYaml parser.

CHAPTER 3

Interface

class yalchemy.**Column**(*name*, *datatype*, *format_=None*, *required=False*, *default=None*, *doc=None*)

A yalchemy Column that can be converted to a sqlalchemy Column

classmethod from_dict(*dict_obj*)

Creates a yalchemy Column from a dictionary.

The dictionary has the following specification:

```
{
    # The name of the column
    'name': str(required),

    # The datatype. Can be any of the postgres datatypes listed
    # at https://www.postgresql.org/docs/current/static/datatype.html
↪#DATATYPE-TABLE
    'datatype': str(required),

    # The format of the datatype. This is an array of arguments that is passed
    # into the datatype. For example, a format of [255] for a varchar type␣
↪would
    # create a varchar(255) column datatype
    'format': list(default=[]),

    # If the datatype is required. If False, it will be null
    'required': bool(default=False),

    # The database server-side default for the column. Column defaults follow␣
↪the
    # specification provided in the `ColumnDefault.from_dict` documentation
    'default': dict(optional)
}
```

classmethod from_sqla(*sqla_obj*)

Create a yalchemy Column using a sqlalchemy Column

Parameters **sqla_obj** (Column) – The sqlalchemy column

to_sqla()
Converts the yalchemy Column to a sqlalchemy Column

class yalchemy.**Index**(*columns*, *name=None*)
A yalchemy Index that can be converted to a sqlalchemy Index

from_dict(*dict_obj*)
Creates a yalchemy column collection from a dictionary.

The dictionary has the following specification:

```
{
    # The list of column names. If multiple are
    # provided, a multidimensional index will be created
    'columns': list[str](required),

    # A fixed name for this object
    # (if not specified, will automatically generate a hashed name)
    'name': str(optional)
}
```

classmethod from_sqla(*sqla_obj*)
Create a yalchemy Index from a sqlalchemy Index

to_sqla(*table_name*)
Converts the yalchemy column collection to a sqlalchemy object.

Parameters **table_name** (*str*) – The name of the table to which the sqlalchemy object is applied

class yalchemy.**ForeignKey**(*column*, *remote_table*, *remote_column*)
A yalchemy ForeignKey that can be converted to a sqlalchemy ForeignKeyIndex

classmethod from_dict(*dict_obj*)
Creates a yalchemy ForeignKey from a dictionary.

The dictionary has the following specification:

```
{
    # The name of the column
    'column': str(required),

    # The name of the remote table
    'remote_table': str(required),

    # The name of the remote column
    'remote_column': str(required),
}
```

Note: The `column` and `remote_column` arguments can be list of strings in the case of creating foreign keys on combinations of columns.

classmethod from_sqla(*sqla_obj*)
Create a yalchemy ForeignKey from a sqlalchemy ForeignKeyConstraint

Parameters **sqla_obj** – The sqlalchemy ForeignKeyConstraint object.

to_sqla()
Converts the yalchemy ForeignKey to a sqlalchemy ForeignKeyConstraint

class yalchemy.**UniqueConstraint**(*columns*, *name=None*)
A yalchemy UniqueConstraint that can be converted to a sqlalchemy UniqueConstraint

from_dict(*dict_obj*)
Creates a yalchemy column collection from a dictionary.

The dictionary has the following specification:

```
{
    # The list of column names. If multiple are
    # provided, a multidimensional index will be created
    'columns': list[str](required),

    # A fixed name for this object
    # (if not specified, will automatically generate a hashed name)
    'name': str(optional)
}
```

classmethod from_sqla(*sqla_obj*)
Create a yalchemy UniqueConsraint from a sqlalchemy UniqueConstraint

to_sqla(*table_name*)
Converts the yalchemy column collection to a sqlalchemy object.

Parameters **table_name** (*str*) – The name of the table to which the sqlalchemy object is applied

class yalchemy.**CheckConstraint**(*name*, *check*)
A yalchemy CheckConstraint that can be converted to a sqlalchemy CheckConstraint

---

Note: While other yalchemy objects perform equality comparisons on all attributes of their classes, CheckConstraint does not do comparison on the 'check' attribute. This is because Postgres may format the check value when storing it.

---

classmethod from_dict(*dict_obj*)
Creates a yalchemy CheckConstraint from a dictionary.

The dictionary has the following specification:

```
{
    # The check is the string of the check we want to
    # have on the constraint. It should be verbatium
    # what the check would be in SQL
    'name': str(required),
    'check': str(required),
}
```

classmethod from_sqla(*sqla_obj*)
Create a yalchemy CheckConstraint from a sqlalchemy CheckConstraint

to_sqla()
Converts the yalchemy CheckConstraint to a sqlalchemy CheckConstraint.

class yalchemy.**Table**(*name*, *schema*, *columns=None*, *foreign_keys=None*, *primary_keys=None*, *indexes=None*, *check_constraints=None*, *unique_constraints=None*, *doc=None*)
A yalchemy Table that can be converted to a sqlalchemy Table

classmethod from_dict(*dict_obj*)
Creates a yalchemy Table from a dictionary.

The dictionary has the following specification:

```
{
    # The table name
    'name': str(required),

    # The schema
    'schema': str(required),

    # The list of table column dictioniaries. Column dictionaries
    # follow the specification provided in the `Column.from_dict`␣
↪documentation
    'columns': list[dict](optional),

    # The list of column foreign key dictionaries. Foreign keys follow the
    # specification provided in the `ForeignKey.from_dict` documentation
    'foreign_keys': list[dict](optional),

    # The list of index dictionaries. Indexes follow the
    # specification provided in the `Index.from_dict` documentation
    'indexes': list[dict](optional),

    # The list of column names (as strings) that are primary keys. The unique
    # constraint is made with the columns in the given order.
    'primary_keys': list[str](optional),

    # The list of unique constraints.  Unique constraints follow the
    # specification provided in the `UniqueConstraint.from_dict` documentation

    # The list of check constraints.  Check constraints follow the
    # specification provided in the `CheckConstraint.from_dict` documentation
    'check_constraints': list[dict](optional)

    # Documentation for the table. Not read from sqlalchemy objs
    # or written to them until SA version 1.2
    'doc': str(optional)
}
```

A full example of a table looks like this:

```
{
    'name': 'my_table',
    'schema': 'schema',
    'doc': 'Some document string'
    'columns': [
        {'name': 'col1', 'datatype': 'varchar', 'format': [123], 'required':␣
↪False},
        {'name': 'col2', 'datatype': 'integer', 'required': True,
         'doc': 'col2 is a good column'},
    ],
    'foreign_keys': [
        {'column': 'col2', 'remote_table': 'other_table', 'remote_column':
↪'other_col'}
    ],
    'indexes': [{'columns': ['col1', 'col2']}],
    'unique_constraints': [{'name': 'my_unique_constraint', 'columns': ['col2
↪']}],
    'check_constraints': [{'name': 'my_check', 'check': 'char_length(col1) =␣
↪col2'}]
```

```
    'primary_keys': ['col2']
}
```

classmethod from_sqla(*sqla_obj*)
Create a yalchemy Table from a sqlalchemy Table.

Parameters **sqla_obj** – The sqlalchemy Table object

to_sqla(*metadata=None*, *include_indexes=True*)
Converts the yalchemy Table to a sqlalchemy Table.

Parameters

- **metadata** (*MetaData*) – The sqlalchemy metdata object. If None, creates a new sqlalchemy MetaData instance.
- **include_indexes** (*bool, default=True*) – Include indexes when creating the sqlalchemy Table object

class yalchemy.**TableSet**(*tables=None*)
A yalchemy TableSet that can be converted to a list of sqlalchemy Tables

classmethod from_dict(*dict_obj*)
Creates a yalchemy TableSet from a dictionary.

The dictionary has the following specification:

```
{
    # The list of table dictionaries. The specification for a table
    # dictionary can be found in the `Table.from_dict` documentation.
    'tables': list[dict](optional)
}
```

classmethod from_sqla(*sqla_objs*)
Create a yalchemy TableSet from a list of sqlalchemy Tables.

Parameters **sqla_objs** (*list[Table]*) – A list of sqlalchemy tables

to_sqla(*metadata=None*, *include_indexes=True*, *add_unique_constraints=True*)
Converts the yalchemy TableSet to a list of sqlalchemy Tables.

Parameters

- **include_indexes** (*bool, default=True*) – Include the indexes in the returned sqlalchemy tables.
- **add_unique_constraints** (*bool, default=True*) – Add unique constraints to the targets of foreign keys. Requires the target tables of foreign keys to be in this table set.

CHAPTER 4

Release Notes

```
0.2.15
------

* SRE-544 Unpin the minor version

0.2.14
------

* [SRE-544] Upgrade the dependency (#17)

0.2.13
------

* update deploy key

0.2.12
------

* respect column ordering in yalchemy indexes and primary keys (#15)

0.2.11
------

* primary keys should be ordered (#13)

0.2.10
------


0.2.9
-----

* [DI-8] Temple update 2018-07-27 (#11)
* [DI-8] Temple update 2018-07-26 (#10)
```

```
0.2.8
-----

* Improve yaml loader for Yalchemy.from\_yaml() (#7)

0.2.7
-----

* Fix AUTHORS auto-update by pbr (#5)

0.2.6
-----

* Relax core dependencies and fix AUTHORS (#3)

0.2.5
-----

* Remove temporary README.md which is taking precedence over README.rst (#2)

0.2.4
-----

* Initial open source release of yalchemy (#1)
* first commit
```

CHAPTER 5

Contributing Guide

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

5.1 Setup

Set up your development environment with:

```
git clone git@github.com:CloverHealth/yalchemy.git
cd yalchemy
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`.

5.2 Testing and Validation

Run the tests with:

```
make test
```

Validate the code with:

```
make validate
```

5.3 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
```

5.4 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.

Index

C

CheckConstraint (class in yalchemy), 9
Column (class in yalchemy), 7

F

ForeignKey (class in yalchemy), 8
from_dict() (yalchemy.CheckConstraint class method), 9
from_dict() (yalchemy.Column class method), 7
from_dict() (yalchemy.ForeignKey class method), 8
from_dict() (yalchemy.Index method), 8
from_dict() (yalchemy.Table class method), 9
from_dict() (yalchemy.table_structures.Yalchemy class method), 1
from_dict() (yalchemy.TableSet class method), 11
from_dict() (yalchemy.UniqueConstraint method), 9
from_sqla() (yalchemy.CheckConstraint class method), 9
from_sqla() (yalchemy.Column class method), 7
from_sqla() (yalchemy.ForeignKey class method), 8
from_sqla() (yalchemy.Index class method), 8
from_sqla() (yalchemy.Table class method), 11
from_sqla() (yalchemy.table_structures.Yalchemy class method), 1
from_sqla() (yalchemy.TableSet class method), 11
from_sqla() (yalchemy.UniqueConstraint class method), 9
from_yaml() (yalchemy.table_structures.Yalchemy class method), 1

I

Index (class in yalchemy), 8

T

Table (class in yalchemy), 9
TableSet (class in yalchemy), 11
to_dict() (yalchemy.table_structures.Yalchemy method), 2
to_sqla() (yalchemy.CheckConstraint method), 9
to_sqla() (yalchemy.Column method), 7
to_sqla() (yalchemy.ForeignKey method), 8
to_sqla() (yalchemy.Index method), 8
to_sqla() (yalchemy.Table method), 11
to_sqla() (yalchemy.table_structures.Yalchemy method), 2
to_sqla() (yalchemy.TableSet method), 11
to_sqla() (yalchemy.UniqueConstraint method), 9
to_yaml() (yalchemy.table_structures.Yalchemy method), 2

U

UniqueConstraint (class in yalchemy), 8

Y

Yalchemy (class in yalchemy.table_structures), 1