modlit Documentation

Release 0.2.2

Pat Daburu

May 27, 2018

Contents:

Helpful utilities for SQLAlchemy (and GeoAlchemy) data model projects.

CHAPTER 1

API Documentation

1.1 modlit

Helpful utilities for SQLAlchemy (and GeoAlchemy) data model projects.

1.2 modlit.api

1.3 modlit.base

The GeoAlchemy declarative base for the data model is defined in this module along with some other helpful classes.

class modlit.base.**AutoGuidMixin**
Bases: object

This mixin includes an *id* column.

id = Column(None, GUID(), table=None, primary_key=True, nullable=False, default=Columnl

class modlit.base.**Base**(***kwargs*)
Bases: object

The most base type

metadata = MetaData(bind=None)

class modlit.base.**ModelMixin**
Bases: object

This mixin includes columns and methods common to objects within the data model.

classmethod geometry_type() → modlit.geometry.GeometryTypes
Get the geometry type defined for the model class.

Returns the geometry type

Note: The name of the geometry attribute must be 'geometry' to be properly identified.

1.4 modlit.db

1.5 modlit.errors

Something went wrong? OK. Here's what we'll do...

exception modlit.errors.**ModlitError**(*message: str*)
Bases: Exception

A general error pertaining to things that happen in the *modlit* package.

message
Get the error message.

Returns the error message

1.6 modlit.geometry

This module contains stuff pertaining to geometry columns.

class modlit.geometry.**GeometryTypes**
Bases: enum.IntFlag

Supported geometry types.

CURVE = 8
curves

GEOMETRY = 7
generic geometries

GEOMETRYCOLLECTION = 23
geometry collections

LINESTRING = 2
polyline geometries

MULTILINESTRING = 18
multilinestring collections

MULTIPOINT = 17
multipoint collections

MULTIPOLYGON = 20
multipolygon collections

NONE = 0
no geometry type

POINT = 1
point geometries

POLYGON = 4
polygon geometries

1.7 modlit.meta

This module contains metadata objects to help with inline documentation of the model.

modlit.meta.**COLUMN_META_ATTR = '__meta__'**
the property that contains column metadata

class modlit.meta.**ColumnMeta**(*label: str = None*, *description: str = None*, *nena: str = None*, *source: modlit.meta.Source = None*, *target: modlit.meta.Target = None*)
Bases: modlit.meta._MetaDescription

Metadata for table columns.

description
Get the human-friendly description of the column.

Returns the human-friendly description of the column

get_enum(*enum_cls: Type[Union[modlit.meta.Requirement, modlit.meta.Usage]]*) → modlit.meta.Requirement
Get the current value of an attribute defined by an enumeration.

Parameters **enum_cls** – the enumeration class

Returns the value of the attribute

label
Get the human-friendly label for the column.

Returns the human-friendly label for the column

nena
Get the name of the NENA equivalent field.

Returns

the NENA equivalent field.

source
Get the information about the source data contract.

Returns the source data contract

target
Get the information for the target data contract.

Returns the target data contract

class modlit.meta.**Requirement**
Bases: enum.IntFlag

This enumeration describes contracts with source data providers.

NONE = 0
data for the column is neither requested nor required

REQUESTED = 1
data for the column is requested

REQUIRED = 3
data for the column is required

class modlit.meta.**Source**(*requirement: modlit.meta.Requirement = <Requirement.NONE: 0>*, *synonyms: Iterable[str] = None*)

Bases: modlit.meta._MetaDescription

'Source' information defines contracts with data providers.

is_synonym(*name: str*)

Is a given name a synonym for this source column?

Parameters **name** – the name to test

Returns *True* if the name appears to be a synonym, otherwise *False*

requirement

Get the source data contract.

Returns the source data contract

modlit.meta.**TABLE_META_ATTR = '__meta__'**

the property that contains table metadata

class modlit.meta.**TableMeta**(*label: str = None*, *synonyms: Iterable[str] = None*)

Bases: modlit.meta._MetaDescription

Metadata for tables.

label

Get the human-friendly label for the column.

Returns the human-friendly label

class modlit.meta.**Target**(*guaranteed: bool = False*, *calculated: bool = False*, *usage: modlit.meta.Usage = <Usage.NONE: 0>*)

Bases: modlit.meta._MetaDescription

'Target' information describes contracts with data consumers.

calculated

May the column's value be generated or modified by a calculation?

Returns *True* if the column may be generated or modified by a calculation, otherwise *False*

guaranteed

Is the column guaranteed to contain a non-empty value?

Returns *True* if the column is guaranteed to contain a non-empty value, otherwise *False*

usage

Get the *Usage* flag for the column.

Returns a single flag value that indicates the ways in which the data in this column is expected to be used

class modlit.meta.**Usage**

Bases: enum.IntFlag

This enumeration describes how data may be used.

DISPLAY = 2

The data is displayed to users.

NONE = 0

The data is not used.

SEARCH = 1

The data is used for searching.

modlit.meta.**column**(*dtype: Any*, *meta: modlit.meta.ColumnMeta*, **args*, ***kwargs*) → sqlalchemy.sql.schema.Column

Create a GeoAlchemy `Column` annotated with metadata.

Parameters

- **dtype** – the SQLAlchemy/GeoAlchemy column type
- **meta** – the meta data

Returns a GeoAlchemy `Column`

1.8 modlit.model

This module contains general members to help you work with the model.

modlit.model.**IS_MODEL_CLASS = '__model_cls__'**

signifies a model class

class modlit.model.**ModelMap**(*base_cls: type, alt_tablename: str, alt_columns: Dict[str, str]*)

Bases: `object`

A model map describes a set off alternate names that may be used to refer to a modeled object.

modlit.model.**load**(*package*, *skip_modules: List[str] = None*)

Load the data model.

Parameters

- **package** – the package that contains the model classes
- **skip_modules** – a list of names of the modules that should be skipped when importing the package

modlit.model.**model**(*label: str*)

Use this decorator to provide meta-data for your model class.

Parameters **label** – the friendly label for the class

1.9 modlit.modules

Sometimes we need to do things like loading modules dynamically. The tools to help with that are in here.

modlit.modules.**walk_load**(*package*, *skip_modules: List[str] = None*)

Walk and load the modules in a package.

Parameters

- **package** – the package that contains the model classes
- **skip_modules** – a list of names of the modules that should be skipped when importing the package

1.10 modlit.types

This module contains custom GeoAlchemy/SQLAlchemy types.

class modlit.types.**GUID**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

This is a Platform-independent GUID type that uses PostgreSQL's UUID type and otherwise uses CHAR(32), storing as stringified hex values.

See also:

http://docs.sqlalchemy.org/en/latest/core/custom_types.html#backend-agnostic-guid-type

impl
alias of sqlalchemy.sql.sqltypes.CHAR

load_dialect_impl(*dialect*)
Return a TypeEngine object corresponding to a dialect.

This is an end-user override hook that can be used to provide differing types depending on the given dialect. It is used by the TypeDecorator implementation of type_engine() to help determine what type should ultimately be returned for a given TypeDecorator.

By default returns self.impl.

process_bind_param(*value*, *dialect*)
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying TypeEngine object, and from there to the DBAPI execute() method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

process_result_value(*value*, *dialect*)
Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying TypeEngine object, originally from the DBAPI cursor method fetchone() or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

CHAPTER 2

Python Module Dependencies

The `requirements.txt` file contains this project's module dependencies. You can install these dependencies using `pip`.

```
pip install -r requirements.txt
```

2.1 requirements.txt

```
addict>=2.1.3,<3
Flask>=1.0.2,<2
orderedset>=2.0.1,<3
parameterized>=0.6.1,<1
pip-check-reqs==2.0.1
pylint>=1.8.4,<2
pytest>=3.4.0,<4
pytest-cov>=2.5.1,<3
pytest-pythonpath>=0.7.2,<1
setuptools>=38.4.0
Sphinx>=1.7.1,<2
sphinx-rtd-theme>=0.2.4,<1
testing.postgresql>=1.3.0,<2
tox>=3.0.0,<4
twine>=1.11.0,<2
psycopg2-binary>=2.7.4,<3
GeoAlchemy2>=0.4.2,<1
SQLAlchemy>=1.2.6,<2
sqlparse>=0.2.4,<1
titlecase>=0.12.0,<1
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

R

S

T

U

W