

# GeoAlchemy2 Documentation

***Release 0.12.0***

**Eric Lemoine**

**Jun 07, 2022**

# Contents


**1 Requirements** **3**

**2 Installation** **5**

**3 What's New in GeoAlchemy 2** **7**
3.1 Migrate to GeoAlchemy 2 . . . 7

**4 Tutorials** **9**
4.1 ORM Tutorial . . . 9
4.2 Core Tutorial . . . 15
4.3 SpatiaLite Tutorial . . . 20

**5 Gallery** **25**
5.1 Gallery . . . 25

**6 Use with Alembic** **41**
6.1 Use Alembic with GeoAlchemy 2 . . . 41

**7 Reference Documentation** **47**
7.1 Types . . . 47
7.2 Elements . . . 51
7.3 Spatial Functions . . . 52
7.4 Spatial Operators . . . 102
7.5 Shapely Integration . . . 105
7.6 Alembic helpers . . . 105

**8 Development** **111**

**9 Indices and tables** **113**

**Python Module Index** **115**

**Index** **117**

*Using SQLAlchemy with Spatial Databases.*

GeoAlchemy 2 provides extensions to SQLAlchemy for working with spatial databases.

GeoAlchemy 2 focuses on PostGIS. PostGIS 1.5, PostGIS 2 and PostGIS 3 are supported.

SpatiaLite is also supported, but using GeoAlchemy 2 with SpatiaLite requires some specific configuration on the application side. GeoAlchemy 2 works with SpatiaLite 4.3.0 and higher (except for alembic helpers that need SpatiaLite >= 5).

GeoAlchemy 2 aims to be simpler than its predecessor, GeoAlchemy. Simpler to use, and simpler to maintain.

The current version of this documentation applies to the version 0.12.0 of GeoAlchemy 2.

CHAPTER 1

# Requirements

GeoAlchemy 2 requires SQLAlchemy 1.4.

CHAPTER 2

# Installation

GeoAlchemy 2 is available on the Python Package Index. So it can be installed with the standard pip or easy_install tools.

CHAPTER 3

# What's New in GeoAlchemy 2

- GeoAlchemy 2 supports PostGIS' `geometry` type, as well as the `geography` and `raster` types.
- The first series had its own namespace for spatial functions. With GeoAlchemy 2, spatial functions are called like any other SQLAlchemy function, using `func`, which is SQLAlchemy's standard way of calling SQL functions.
- GeoAlchemy 2 works with SQLAlchemy's ORM, as well as with SQLAlchemy's *SQL Expression Language* (a.k.a the SQLAlchemy Core). (This is thanks to SQLAlchemy's new type-level comparator system.)
- GeoAlchemy 2 supports reflection of geometry and geography columns.
- GeoAlchemy 2 adds `to_shape`, `from_shape` functions for a better integration with Shapely.

## 3.1 Migrate to GeoAlchemy 2

This section describes how to migrate an application from the first series of GeoAlchemy to GeoAlchemy 2.

### 3.1.1 Defining Geometry Columns

The first series has specific types like `Point`, `LineString` and `Polygon`. These are gone, the *geoalchemy2.types.Geometry* type should be used instead, and a `geometry_type` can be passed to it.

So, for example, a `polygon` column that used to be defined like this:

```
geom = Column(Polygon)
```

is now defined like this:

```
geom = Column(Geometry('POLYGON'))
```

This change is related to GeoAlchemy 2 supporting the *geoalchemy2.types.Geography* type.

### 3.1.2 Calling Spatial Functions

The first series has its own namespace/object for calling spatial functions, namely `geoalchemy.functions`. With GeoAlchemy 2, SQLAlchemy's `func` object should be used.

For example, the expression

```
functions.buffer(functions.centroid(box), 10, 2)
```

would be rewritten to this with GeoAlchemy 2:

```
func.ST_Buffer(func.ST_Centroid(box), 10, 2)
```

Also, as the previous example hinted it, the names of spatial functions are now all prefixed with `ST_`. (This is to be consistent with PostGIS and the `SQL-MM` standard.) The `ST_` prefix should be used even when applying spatial functions to columns, *geoalchemy2.elements.WKTElement*, or *geoalchemy2.elements.WKTElement* objects:

```
Lake.geom.ST_Buffer(10, 2)
lake_table.c.geom.ST_Buffer(10, 2)
lake.geom.ST_Buffer(10, 2)
```

### 3.1.3 WKB and WKT Elements

The first series has classes like `PersistentSpatialElement`, `PGPersistentSpatialElement`, `WKTSpatialElement`.

They're all gone, and replaced by two classes only: *geoalchemy2.elements.WKTElement* and *geoalchemy2.elements.WKBElement*.

*geoalchemy2.elements.WKTElement* is to be used in expressions where a geometry with a specific SRID should be specified. For example:

```
Lake.geom.ST_Touches(WKTElement('POINT(1 1)', srid=4326))
```

If no SRID need be specified, a string can used directly:

```
Lake.geom.ST_Touches('POINT(1 1)')
```

- *geoalchemy2.elements.WKTElement* literally replaces the first series' `WKTSpatialElement`.
- *geoalchemy2.elements.WKBElement* is the type into which GeoAlchemy 2 converts geometry values read from the database.

  For example, the `geom` attributes of `Lake` objects loaded from the database would be references to *geoalchemy2.elements.WKBElement* objects. This class replaces the first series' `PersistentSpatialElement` classes.

See the *Migrate to GeoAlchemy 2* page for details on how to migrate a GeoAlchemy application to GeoAlchemy 2.

CHAPTER 4

# Tutorials

GeoAlchemy 2 works with both SQLAlchemy's *Object Relational Mapping* (ORM) and *SQL Expression Language*. This documentation provides a tutorial for each system. If you're new to GeoAlchemy 2 start with this.

## 4.1 ORM Tutorial

(This tutorial is greatly inspired by the SQLAlchemy ORM Tutorial, which is recommended reading, eventually.)

GeoAlchemy does not provide an Object Relational Mapper (ORM), but works well with the SQLAlchemy ORM. This tutorial shows how to use the SQLAlchemy ORM with spatial tables, using GeoAlchemy.

### 4.1.1 Connect to the DB

For this tutorial we will use a PostGIS 2 database. To connect we use SQLAlchemy's `create_engine()` function:

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('postgresql://gis:gis@localhost/gis', echo=True)
```

In this example the name of the database, the database user, and the database password, is `gis`.

The `echo` flag is a shortcut to setting up SQLAlchemy logging, which is accomplished via Python's standard logging module. With it is enabled, we'll see all the generated SQL produced.

The return value of `create_engine` is an `Engine` object, which represents the core interface to the database.

### 4.1.2 Declare a Mapping

When using the ORM, the configurational process starts by describing the database tables we'll be dealing with, and then by defining our own classes which will be mapped to those tables. In modern SQLAlchemy, these two tasks are usually performed together, using a system known as `Declarative`, which allows us to create classes that include directives to describe the actual database table they will be mapped to.

```
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column, Integer, String
>>> from geoalchemy2 import Geometry
>>>
>>> Base = declarative_base()
>>>
>>> class Lake(Base):
...     __tablename__ = 'lake'
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     geom = Column(Geometry('POLYGON'))
```

The `Lake` class establishes details about the table being mapped, including the name of the table denoted by `__tablename__`, and three columns `id`, `name`, and `geom`. The `id` column will be the primary key of the table. The `geom` column is a *`geoalchemy2.types.Geometry`* column whose `geometry_type` is `POLYGON`.

### 4.1.3 Create the Table in the Database

The `Lake` class has a corresponding `Table` object representing the database table. This `Table` object was created automatically by SQLAlchemy, it is referenced to by the `Lake.__table__` property:

```
>>> Lake.__table__
Table('lake', MetaData(bind=None), Column('id', Integer(), table=<lake>,
primary_key=True, nullable=False), Column('name', String(), table=<lake>),
Column('geom', Polygon(srid=4326), table=<lake>), schema=None)
```

To create the `lake` table in the database:

```
>>> Lake.__table__.create(engine)
```

If we wanted to drop the table we'd use:

```
>>> Lake.__table__.drop(engine)
```

### 4.1.4 Create an Instance of the Mapped Class

With the mapping declared, we can create a `Lake` object:

```
>>> lake = Lake(name='Majeur', geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
>>> lake.geom
'POLYGON((0 0,1 0,1 1,0 1,0 0))'
>>> str(lake.id)
'None'
```

A WKT is passed to the `Lake` constructor for its geometry. This WKT represents the shape of our lake. Since we have not yet told SQLAlchemy to persist the `lake` object, its `id` is `None`.

The EWKT (Extended WKT) format is also supported. So, for example, if the spatial reference system for the geometry column were `4326`, the string `SRID=4326;POLYGON((0 0,1 0,1,0 1,0 0))` could be used as the geometry representation.

### 4.1.5 Create a Session

The ORM interacts with the database through a `Session`. Let's create a `Session` class:

```
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker(bind=engine)
```

This custom-made `Session` class will create new `Session` objects which are bound to our database. Then, whenever we need to have a conversation with the database, we instantiate a `Session`:

```
>>> session = Session()
```

The above `Session` is associated with our PostgreSQL `Engine`, but it hasn't opened any connection yet.

### 4.1.6 Add New Objects

To persist our `Lake` object, we `add()` it to the `Session`:

```
>>> session.add(lake)
```

At this point the `lake` object has been added to the `Session`, but no SQL has been issued to the database. The object is in a *pending* state. To persist the object a *flush* or *commit* operation must occur (commit implies flush):

```
>>> session.commit()
```

We can now query the database for `Majeur`:

```
>>> our_lake = session.query(Lake).filter_by(name='Majeur').first()
>>> our_lake.name
u'Majeur'
>>> our_lake.geom
<WKBElement at 0x9af594c;
↪'0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000f03f000000000000000000000000000000
↪'>
>>> our_lake.id
1
```

`our_lake.geom` is a *geoalchemy2.elements.WKBElement*, which a type provided by GeoAlchemy. *geoalchemy2.elements.WKBElement* wraps a WKB value returned by the database.

Let's add more lakes:

```
>>> session.add_all([
...     Lake(name='Garde', geom='POLYGON((1 0,3 0,3 2,1 2,1 0))'),
...     Lake(name='Orta', geom='POLYGON((3 0,6 0,6 3,3 3,3 0))')
... ])
>>> session.commit()
```

### 4.1.7 Query

A `Query` object is created using the `query()` function on `Session`. For example here's a `Query` that loads `Lake` instances ordered by their names:

```
>>> query = session.query(Lake).order_by(Lake.name)
```

Any `Query` is iterable:

```
>>> for lake in query:
...     print lake.name
...
Garde
Majeur
Orta
```

Another way to execute the query and get a list of `Lake` objects involves calling `all()` on the `Query`:

```
>>> lakes = session.query(Lake).order_by(Lake.name).all()
```

The SQLAlchemy ORM Tutorial's Querying section provides more examples of queries.

### 4.1.8 Make Spatial Queries

Using spatial filters in SQL SELECT queries is very common. Such queries are performed by using spatial relationship functions, or operators, in the `WHERE` clause of the SQL query.

For example, to find the `Lake` s that contain the point `POINT(4 1)`, we can use this `Query`:

```
>>> from sqlalchemy import func
>>> query = session.query(Lake).filter(
...             func.ST_Contains(Lake.geom, 'POINT(4 1)'))
...
>>> for lake in query:
...     print lake.name
...
Orta
```

GeoAlchemy allows rewriting this `Query` more concisely:

```
>>> query = session.query(Lake).filter(Lake.geom.ST_Contains('POINT(4 1)'))
>>> for lake in query:
...     print lake.name
...
Orta
```

Here the `ST_Contains` function is applied to the `Lake.geom` column property. In that case the column property is actually passed to the function, as its first argument.

Here's another spatial filtering query, based on `ST_Intersects`:

```
>>> query = session.query(Lake).filter(
...             Lake.geom.ST_Intersects('LINESTRING(2 1,4 1)'))
...
>>> for lake in query:
...     print lake.name
...
Garde
Orta
```

We can also apply relationship functions to *`geoalchemy2.elements.WKBElement`*. For example:

```
>>> lake = session.query(Lake).filter_by(name='Garde').one()
>>> print session.scalar(lake.geom.ST_Intersects('LINESTRING(2 1,4 1)'))
True
```

`session.scalar` allows executing a clause and returning a scalar value (a boolean value in this case).

The GeoAlchemy functions all start with `ST_`. Operators are also called as functions, but the function names don't include the `ST_` prefix. As an example let's use PostGIS' `&&` operator, which allows testing whether the bounding boxes of geometries intersect. GeoAlchemy provides the `intersects` function for that:

```
>>> query = session.query
>>> query = session.query(Lake).filter(
...             Lake.geom.intersects('LINESTRING(2 1,4 1)'))
...
>>> for lake in query:
...     print lake.name
...
Garde
Orta
```

### 4.1.9 Set Spatial Relationships in the Model

Let's assume that in addition to `lake` we have another table, `treasure`, that includes treasure locations. And let's say that we are interested in discovering the treasures hidden at the bottom of lakes.

The `Treasure` class is the following:

```
>>> class Treasure(Base):
...      __tablename__ = 'treasure'
...      id = Column(Integer, primary_key=True)
...      geom = Column(Geometry('POINT'))
```

We can now add a `relationship` to the `Lake` table to automatically load the treasures contained by each lake:

```
>>> from sqlalchemy.orm import relationship, backref
>>> class Lake(Base):
...     __tablename__ = 'lake'
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     geom = Column(Geometry('POLYGON'))
...     treasures = relationship(
...         'Treasure',
...         primaryjoin='func.ST_Contains(foreign(Lake.geom), Treasure.geom).as_
↪comparison(1, 2)',
...         backref=backref('lake', uselist=False),
...         viewonly=True,
...         uselist=True,
...     )
```

Note the use of the `as_comparison` function. It is required for using an SQL function (`ST_Contains` here) in a `primaryjoin` condition. This only works with SQLAlchemy 1.3, as the `as_comparison` function did not exist before that version. See the Custom operators based on SQL function section of the SQLAlchemy documentation for more information.

Some information on the parameters used for configuring this `relationship`:

- `backref` is used to provide the name of property to be placed on the class that handles this relationship in the other direction, namely `Treasure`;
- `viewonly=True` specifies that the relationship is used only for loading objects, and not for persistence operations;

- `uselist=True` indicates that the property should be loaded as a list, as opposed to a scalar.

Also, note that the `treasures` property on `lake` objects (and the `lake` property on `treasure` objects) is loaded "lazily" when the property is first accessed. Another loading strategy may be configured in the `relationship`. For example you'd use `lazy='joined'` for related items to be loaded "eagerly" in the same query as that of the parent, using a `JOIN` or `LEFT OUTER JOIN`.

See the Relationships API section of the SQLAlchemy documentation for more detail on the `relationship` function, and all the parameters that can be used to configure it.

### 4.1.10 Use Other Spatial Functions

Here's a `Query` that calculates the areas of buffers for our lakes:

```
>>> from sqlalchemy import func
>>> query = session.query(Lake.name,
...                       func.ST_Area(func.ST_Buffer(Lake.geom, 2)) \
...                           .label('bufferarea'))
>>> for row in query:
...     print '%s: %f' % (row.name, row.bufferarea)
...
Majeur: 21.485781
Garde: 32.485781
Orta: 45.485781
```

This `Query` applies the PostGIS `ST_Buffer` function to the geometry column of every row of the `lake` table. The return value is a list of rows, where each row is actually a tuple of two values: the lake name, and the area of a buffer of the lake. Each tuple is actually an SQLAlchemy `KeyedTuple` object, which provides property type accessors.

Again, the `Query` can written more concisely:

```
>>> query = session.query(Lake.name,
...                       Lake.geom.ST_Buffer(2).ST_Area().label('bufferarea'))
>>> for row in query:
...     print '%s: %f' % (row.name, row.bufferarea)
...
Majeur: 21.485781
Garde: 32.485781
Orta: 45.485781
```

Obviously, processing and measurement functions can also be used in `WHERE` clauses. For example:

```
>>> lake = session.query(Lake).filter(
...             Lake.geom.ST_Buffer(2).ST_Area() > 33).one()
...
>>> print lake.name
Orta
```

And, like any other functions supported by GeoAlchemy, processing and measurement functions can be applied to *`geoalchemy2.elements.WKBElement`*. For example:

```
>>> lake = session.query(Lake).filter_by(name='Majeur').one()
>>> bufferarea = session.scalar(lake.geom.ST_Buffer(2).ST_Area())
>>> print '%s: %f' % (lake.name, bufferarea)
Majeur: 21.485781
Majeur: 21.485781
```

### 4.1.11 Use Raster functions

A few functions (like *ST_Transform()*, *ST_Union()*, *ST_SnapToGrid()*, ...) can be used on both *geoalchemy2.types.Geometry* and *geoalchemy2.types.Raster* types. In GeoAlchemy2, these functions are only defined for `Geometry` as it can not be defined for several types at the same time. Thus using these functions on `Raster` requires minor tweaking to enforce the type by passing the *type_=Raster* argument to the function:

```
>>> query = session.query(Lake.raster.ST_Transform(2154, type_=Raster))
```

### 4.1.12 Further Reference

- Spatial Functions Reference: *Spatial Functions*
- Spatial Operators Reference: *Spatial Operators*
- Elements Reference: *Elements*

## 4.2 Core Tutorial

(This tutorial is greatly inspired from the SQLAlchemy SQL Expression Language Tutorial, which is recommended reading, eventually.)

This tutorial shows how to use the SQLAlchemy Expression Language (a.k.a. SQLAlchemy Core) with GeoAlchemy. As defined by the SQLAlchemy documentation itself, in contrast to the ORM's domain-centric mode of usage, the SQL Expression Language provides a schema-centric usage paradigm.

### 4.2.1 Connect to the DB

For this tutorial we will use a PostGIS 2 database. To connect we use SQLAlchemy's `create_engine()` function:

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('postgresql://gis:gis@localhost/gis', echo=True)
```

In this example the name of the database, the database user, and the database password, is `gis`.

The `echo` flag is a shortcut to setting up SQLAlchemy logging, which is accomplished via Python's standard logging module. With it is enabled, we'll see all the generated SQL produced.

The return value of `create_engine` is an `Engine` object, which respresents the core interface to the database.

### 4.2.2 Define a Table

The very first object that we need to create is a `Table`. Here we create a `lake_table` object, which will correspond to the `lake` table in the database:

```
>>> from sqlalchemy import Table, Column, Integer, String, MetaData
>>> from geoalchemy2 import Geometry
>>>
>>> metadata = MetaData()
>>> lake_table = Table('lake', metadata,
...     Column('id', Integer, primary_key=True),
...     Column('name', String),
```

(continues on next page)

(continued from previous page)

```
...     Column('geom', Geometry('POLYGON'))
... )
```

This table is composed of three columns, `id`, `name` and `geom`. The `geom` column is a *geoalchemy2.types.Geometry* column whose `geometry_type` is `POLYGON`.

Any `Table` object is added to a `MetaData` object, which is a catalog of `Table` objects (and other related objects).

### 4.2.3 Create the Table

With our `Table` being defined we're ready (to have SQLAlchemy) create it in the database:

```
>>> lake_table.create(engine)
```

Calling `create_all()` on `metadata` would have worked equally well:

```
>>> metadata.create_all(engine)
```

In that case every `Table` that's referenced to by `metadata` would be created in the database. The `metadata` object includes one `Table` here, our now well-known `lake_table` object.

### 4.2.4 Reflecting tables

The reflection system of SQLAlchemy can be used on tables containing *geoalchemy2.types.Geometry* or *geoalchemy2.types.Geography* columns. In this case, the type must be imported to be registered into SQLAlchemy, even if it is not used explicitly.

```
>>> from geoalchemy2 import Geometry  # <= not used but must be imported
>>> from sqlalchemy import create_engine, MetaData
>>> engine = create_engine("postgresql://myuser:mypass@mydb.host.tld/mydbname")
>>> meta = MetaData()
>>> meta.reflect(bind=engine)
```

### 4.2.5 Insertions

We want to insert records into the `lake` table. For that we need to create an `Insert` object. SQLAlchemy provides multiple constructs for creating an `Insert` object, here's one:

```
>>> ins = lake_table.insert()
>>> str(ins)
INSERT INTO lake (id, name, geom) VALUES (:id, :name, ST_GeomFromEWKT(:geom))
```

The `geom` column being a `Geometry` column, the `:geom` bind value is wrapped in a `ST_GeomFromEWKT` call.

To limit the columns named in the `INSERT` query the `values()` method can be used:

```
>>> ins = lake_table.insert().values(name='Majeur',
...                                  geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
...
>>> str(ins)
INSERT INTO lake (name, geom) VALUES (:name, ST_GeomFromEWKT(:geom))
```

**Tip:** The string representation of the SQL expression does not include the data placed in `values`. We got named bind parameters instead. To view the data we can get a compiled form of the expression, and ask for its `params`:

```
>>> ins.compile.params()
{'geom': 'POLYGON((0 0,1 0,1 1,0 1,0 0))', 'name': 'Majeur'}
```

Up to now we've created an `INSERT` query but we haven't sent this query to the database yet. Before being able to send it to the database we need a database `Connection`. We can get a `Connection` from the `Engine` object we created earlier:

```
>>> conn = engine.connect()
```

We're now ready to execute our `INSERT` statement:

```
>>> result = conn.execute(ins)
```

This is what the logging system should output:

```
INSERT INTO lake (name, geom) VALUES (%(name)s, ST_GeomFromEWKT(%(geom)s)) RETURNING␣
↪lake.id
{'geom': 'POLYGON((0 0,1 0,1 1,0 1,0 0))', 'name': 'Majeur'}
COMMIT
```

The value returned by `conn.execute()`, stored in `result`, is a `sqlalchemy.engine.ResultProxy` object. In the case of an `INSERT` we can get the primary key value which was generated from our statement:

```
>>> result.inserted_primary_key
[1]
```

Instead of using `values()` to specify our `INSERT` data, we can send the data to the `execute()` method on `Connection`. So we could rewrite things as follows:

```
>>> conn.execute(lake_table.insert(),
...              name='Majeur', geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
```

Now let's use another form, allowing to insert multiple rows at once:

```
>>> conn.execute(lake_table.insert(), [
...     {'name': 'Garde', 'geom': 'POLYGON((1 0,3 0,3 2,1 2,1 0))'},
...     {'name': 'Orta', 'geom': 'POLYGON((3 0,6 0,6 3,3 3,3 0))'}
...     ])
...
```

**Tip:** In the above examples the geometries are specified as WKT strings. Specifying them as EWKT strings is also supported.

### 4.2.6 Selections

Inserting involved creating an `Insert` object, so it'd come to no surprise that Selecting involves creating a `Select` object. The primary construct to generate `SELECT` statements is SQLAlchemy's `select()` function:

```
>>> from sqlalchemy.sql import select
>>> s = select([lake_table])
>>> str(s)
SELECT lake.id, lake.name, ST_AsEWKB(lake.geom) AS geom FROM lake
```

The `geom` column being a `Geometry` it is wrapped in a `ST_AsEWKB` call when specified as a column in a `SELECT` statement.

We can now execute the statement and look at the results:

```
>>> result = conn.execute(s)
>>> for row in result:
...     print 'name:', row['name'], '; geom:', row['geom'].desc
...
name: Majeur ; geom: 0103...
name: Garde ; geom: 0103...
name: Orta ; geom: 0103...
```

`row['geom']` is a `geoalchemy2.types.WKBElement` instance. In this example we just get an hexadecimal representation of the geometry's WKB value using the `desc` property.

### 4.2.7 Spatial Query

As spatial database users executing spatial queries is of a great interest to us. There comes GeoAlchemy!

#### Spatial relationship

Using spatial filters in SQL SELECT queries is very common. Such queries are performed by using spatial relationship functions, or operators, in the `WHERE` clause of the SQL query.

For example, to find lakes that contain the point `POINT(4 1)`, we can use this:

```
>>> from sqlalchemy import func
>>> s = select([lake_table],
               func.ST_Contains(lake_table.c.geom, 'POINT(4 1)'))
>>> str(s)
SELECT lake.id, lake.name, ST_AsEWKB(lake.geom) AS geom FROM lake WHERE ST_
↪Contains(lake.geom, :param_1)
>>> result = conn.execute(s)
>>> for row in result:
...     print 'name:', row['name'], '; geom:', row['geom'].desc
...
name: Orta ; geom: 0103...
```

GeoAlchemy allows rewriting this more concisely:

```
>>> s = select([lake_table], lake_table.c.geom.ST_Contains('POINT(4 1)'))
>>> str(s)
SELECT lake.id, lake.name, ST_AsEWKB(lake.geom) AS geom FROM lake WHERE ST_
↪Contains(lake.geom, :param_1)
```

Here the `ST_Contains` function is applied to `lake.c.geom`. And the generated SQL the `lake.geom` column is actually passed to the `ST_Contains` function as the first argument.

Here's another spatial query, based on `ST_Intersects`:

```
   >>> s = select([lake_table],
   ...            lake_table.c.geom.ST_Intersects('LINESTRING(2 1,4 1)'))
   >>> result = conn.execute(s)
   >>> for row in result:
   ...     print 'name:', row['name'], '; geom:', row['geom'].desc
   ...
   name: Garde ; geom: 0103...
   name: Orta ; geom: 0103...

This query selects lakes whose geometries intersect ``LINESTRING(2 1,4 1)``.
```

The GeoAlchemy functions all start with `ST_`. Operators are also called as functions, but the names of operator functions don't include the `ST_` prefix.

As an example let's use PostGIS' `&&` operator, which allows testing whether the bounding boxes of geometries intersect. GeoAlchemy provides the `intersects` function for that:

```
>>> s = select([lake_table],
...            lake_table.c.geom.intersects('LINESTRING(2 1,4 1)'))
>>> result = conn.execute(s)
>>> for row in result:
...     print 'name:', row['name'], '; geom:', row['geom'].desc
...
name: Garde ; geom: 0103...
name: Orta ; geom: 0103...
```

### Processing and Measurement

Here's a `Select` that calculates the areas of buffers for our lakes:

```
>>> s = select([lake_table.c.name,
                func.ST_Area(
                    lake_table.c.geom.ST_Buffer(2)).label('bufferarea')])
>>> str(s)
SELECT lake.name, ST_Area(ST_Buffer(lake.geom, %(param_1)s)) AS bufferarea FROM lake
>>> result = conn.execute(s)
>>> for row in result:
...     print '%s: %f' % (row['name'], row['bufferarea'])
Majeur: 21.485781
Garde: 32.485781
Orta: 45.485781
```

Obviously, processing and measurement functions can also be used in `WHERE` clauses. For example:

```
>>> s = select([lake_table.c.name],
               lake_table.c.geom.ST_Buffer(2).ST_Area() > 33)
>>> str(s)
SELECT lake.name FROM lake WHERE ST_Area(ST_Buffer(lake.geom, :param_1)) > :ST_Area_1
>>> result = conn.execute(s)
>>> for row in result:
...     print row['name']
Orta
```

And, like any other functions supported by GeoAlchemy, processing and measurement functions can be applied to *`geoalchemy2.elements.WKBElement`*. For example:

```
>>> s = select([lake_table], lake_table.c.name == 'Majeur')
>>> result = conn.execute(s)
>>> lake = result.fetchone()
>>> bufferarea = conn.scalar(lake[lake_table.c.geom].ST_Buffer(2).ST_Area())
>>> print '%s: %f' % (lake['name'], bufferarea)
Majeur: 21.485781
```

### 4.2.8 Use Raster functions

A few functions (like *ST_Transform()*, *ST_Union()*, *ST_SnapToGrid()*, ...) can be used on both *geoalchemy2.types.Geometry* and *geoalchemy2.types.Raster* types. In GeoAlchemy2, these functions are only defined for `Geometry` as it can not be defined for several types at the same time. Thus using these functions on `Raster` requires minor tweaking to enforce the type by passing the *type_=Raster* argument to the function:

```
>>> s = select([func.ST_Transform(
                    lake_table.c.raster,
                    2154,
                    type_=Raster)
                .label('transformed_raster')])
```

### 4.2.9 Further Reference

- Spatial Functions Reference: *Spatial Functions*
- Spatial Operators Reference: *Spatial Operators*
- Elements Reference: *Elements*

## 4.3 SpatiaLite Tutorial

GeoAlchemy 2's main target is PostGIS. But GeoAlchemy 2 also supports SpatiaLite, the spatial extension to SQLite. This tutorial describes how to use GeoAlchemy 2 with SpatiaLite. It's based on the *ORM Tutorial*, which you may want to read first.

### 4.3.1 Connect to the DB

Just like when using PostGIS connecting to a SpatiaLite database requires an `Engine`. This is how you create one for SpatiaLite:

```
>>> from sqlalchemy import create_engine
>>> from sqlalchemy.event import listen
>>>
>>> def load_spatialite(dbapi_conn, connection_record):
...     dbapi_conn.enable_load_extension(True)
...     dbapi_conn.load_extension('/usr/lib/x86_64-linux-gnu/mod_spatialite.so')
...
>>>
>>> engine = create_engine('sqlite:///gis.db', echo=True)
>>> listen(engine, 'connect', load_spatialite)
```

The call to `create_engine` creates an engine bound to the database file `gis.db`. After that a `connect` listener is registered on the engine. The listener is responsible for loading the SpatiaLite extension, which is a necessary operation for using SpatiaLite through SQL.

At this point you can test that you are able to connect to the database:

```
>> conn = engine.connect()
2018-05-30 17:12:02,675 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain␣
↪returns' AS VARCHAR(60)) AS anon_1
2018-05-30 17:12:02,676 INFO sqlalchemy.engine.base.Engine ()
2018-05-30 17:12:02,676 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode␣
↪returns' AS VARCHAR(60)) AS anon_1
2018-05-30 17:12:02,676 INFO sqlalchemy.engine.base.Engine ()
```

You can also check that the `gis.db` SQLite database file was created on the file system.

One additional step is required for using SpatiaLite: create the `geometry_columns` and `spatial_ref_sys` metadata tables. This is done by calling SpatiaLite's `InitSpatialMetaData` function:

```
>>> from sqlalchemy.sql import select, func
>>>
>>> conn.execute(select([func.InitSpatialMetaData()]))
```

Note that this operation may take some time the first time it is executed for a database. When `InitSpatialMetaData` is executed again it will report an error:

```
InitSpatiaMetaData() error:"table spatial_ref_sys already exists"
```

You can safely ignore that error.

Before going further we can close the current connection:

```
>>> conn.close()
```

### 4.3.2 Declare a Mapping

Now that we have a working connection we can go ahead and create a mapping between a Python class and a database table.

```
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column, Integer, String
>>> from geoalchemy2 import Geometry
>>>
>>> Base = declarative_base()
>>>
>>> class Lake(Base):
...     __tablename__ = 'lake'
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     geom = Column(Geometry(geometry_type='POLYGON', management=True))
```

This basically works in the way as with PostGIS. The difference is the `management` argument that must be set to `True`.

Setting `management` to `True` indicates that the `AddGeometryColumn` and `DiscardGeometryColumn` management functions will be used for the creation and removal of the geometry column. This is required with SpatiaLite.

### 4.3.3 Create the Table in the Database

We can now create the `lake` table in the `gis.db` database:

```
>>> Lake.__table__.create(engine)
```

If we wanted to drop the table we'd use:

```
>>> Lake.__table__.drop(engine)
```

There's nothing specific to SpatiaLite here.

### 4.3.4 Create a Session

When using the SQLAlchemy ORM the ORM interacts with the database through a `Session`.

```
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker(bind=engine)
>>> session = Session()
```

The session is associated with our SpatiaLite `Engine`. Again, there's nothing specific to SpatiaLite here.

### 4.3.5 Add New Objects

We can now create and insert new `Lake` objects into the database, the same way we'd do it using GeoAlchemy 2 with PostGIS.

```
>>> lake = Lake(name='Majeur', geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
>>> session.add(lake)
>>> session.commit()
```

We can now query the database for `Majeur`:

```
>>> our_lake = session.query(Lake).filter_by(name='Majeur').first()
>>> our_lake.name
u'Majeur'
>>> our_lake.geom
<WKBElement at 0x9af594c;
↪'0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000f03f000000000000000000000000000000
↪'>
>>> our_lake.id
1
```

Let's add more lakes:

```
>>> session.add_all([
...     Lake(name='Garde', geom='POLYGON((1 0,3 0,3 2,1 2,1 0))'),
...     Lake(name='Orta', geom='POLYGON((3 0,6 0,6 3,3 3,3 0))')
... ])
>>> session.commit()
```

### 4.3.6 Query

Let's make a simple, non-spatial, query:

```
>>> query = session.query(Lake).order_by(Lake.name)
>>> for lake in query:
...     print(lake.name)
...
Garde
Majeur
Orta
```

Now a spatial query:

```
>>> from geolachemy2 import WKTElement
>>> query = session.query(Lake).filter(
...             func.ST_Contains(Lake.geom, WKTElement('POINT(4 1)')))
...
>>> for lake in query:
...     print(lake.name)
...
Orta
```

Here's another spatial query, using `ST_Intersects` this time:

```
>>> query = session.query(Lake).filter(
...             Lake.geom.ST_Intersects(WKTElement('LINESTRING(2 1,4 1)')))
...
>>> for lake in query:
...     print(lake.name)
...
Garde
Orta
```

We can also apply relationship functions to *`geoalchemy2.elements.WKBElement`*. For example:

```
>>> lake = session.query(Lake).filter_by(name='Garde').one()
>>> print(session.scalar(lake.geom.ST_Intersects(WKTElement('LINESTRING(2 1,4 1)'))))
1
```

`session.scalar` allows executing a clause and returning a scalar value (an integer value in this case).

The value `1` indicates that the lake "Garde" does intersects the `LINESTRING(2 1,4 1)` geometry. See the SpatiaLite SQL functions reference list for more information.

### 4.3.7 Function mapping

Several functions have different names in SpatiaLite than in PostGIS. The GeoAlchemy 2 package is based on the PostGIS syntax but it is possible to automatically translate the queries into SpatiaLite ones. For example, the function *ST_GeomFromEWKT* is automatically translated into *GeomFromEWKT*. Unfortunately, only a few functions are automatically mapped (the ones internally used by GeoAlchemy 2). Nevertheless, it is possible to define new mappings in order to translate the queries automatically. Here is an example to register a mapping for the *ST_Buffer* function:

```
>>> geoalchemy2.functions.register_sqlite_mapping(
...     {'ST_Buffer': 'Buffer'}
... )
```

After this command, all *ST_Buffer* calls in the queries will be translated to *Buffer* calls when the query is executed on a SQLite DB.

A more complex example is provided for when the *PostGIS* function should be mapped depending on the given parameters. For example, the *ST_Buffer* function can actually be translate into either the *Buffer* function or the *SingleSidedBuffer* function (only when *side=right* or *side=left* is passed). See the *Function translation for specific dialect* example in the gallery.

### 4.3.8 Further Reference

- GeoAlchemy 2 ORM Tutotial: *ORM Tutorial*
- GeoAlchemy 2 Spatial Functions Reference: *Spatial Functions*
- GeoAlchemy 2 Spatial Operators Reference: *Spatial Operators*
- GeoAlchemy 2 Elements Reference: *Elements*
- SpatiaLite 4.3.0 SQL functions reference list

CHAPTER 5

# Gallery

## 5.1 Gallery

### 5.1.1 Automatically use a function at insert or select

Sometimes the application wants to apply a function in an insert or in a select. For example, the application might need the geometry with lat/lon coordinates while they are projected in the DB. To avoid having to always tweak the query with a `ST_Transform()`, it is possible to define a TypeDecorator

```
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import MetaData
from sqlalchemy import func
from sqlalchemy import text
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.types import TypeDecorator

from geoalchemy2 import Geometry
from geoalchemy2 import shape

# Tests imports
from tests import test_only_with_dialects

metadata = MetaData()

Base = declarative_base(metadata=metadata)


class TransformedGeometry(TypeDecorator):
    """This class is used to insert a ST_Transform() in each insert or select."""
    impl = Geometry

    cache_ok = True
```

(continues on next page)

(continued from previous page)

```
35
36      def __init__(self, db_srid, app_srid, **kwargs):
37          kwargs["srid"] = db_srid
38          super().__init__(**kwargs)
39          self.app_srid = app_srid
40          self.db_srid = db_srid
41
42      def column_expression(self, col):
43          """The column_expression() method is overridden to set the correct type.
44
45          This is needed so that the returned element will also be decorated. In this␣
    ↪case we don't
46          want to transform it again afterwards so we set the same SRID to both the␣
    ↪``db_srid`` and
47          ``app_srid`` arguments.
48          Without this the SRID of the WKBElement would be wrong.
49          """
50          return getattr(func, self.impl.as_binary)(
51              func.ST_Transform(col, self.app_srid),
52              type_=self.__class__(db_srid=self.app_srid, app_srid=self.app_srid)
53          )
54
55      def bind_expression(self, bindvalue):
56          return func.ST_Transform(
57              self.impl.bind_expression(bindvalue), self.db_srid,
58              type_=self,
59          )
60
61
62  class ThreeDGeometry(TypeDecorator):
63      """This class is used to insert a ST_Force3D() in each insert."""
64      impl = Geometry
65
66      cache_ok = True
67
68      def column_expression(self, col):
69          """The column_expression() method is overridden to set the correct type.
70
71          This is not needed in this example but it is needed if one wants to override␣
    ↪other methods
72          of the TypeDecorator class, like ``process_result_value()`` for example.
73          """
74          return getattr(func, self.impl.as_binary)(col, type_=self)
75
76      def bind_expression(self, bindvalue):
77          return func.ST_Force3D(
78              self.impl.bind_expression(bindvalue),
79              type=self,
80          )
81
82
83  class Point(Base):
84      __tablename__ = "point"
85      id = Column(Integer, primary_key=True)
86      raw_geom = Column(Geometry(srid=4326, geometry_type="POINT"))
87      geom = Column(
88          TransformedGeometry(
```

(continues on next page)

(continued from previous page)

```
            db_srid=2154, app_srid=4326, geometry_type="POINT"))
    three_d_geom = Column(
        ThreeDGeometry(srid=4326, geometry_type="POINTZ", dimension=3))


def check_wkb(wkb, x, y):
    pt = shape.to_shape(wkb)
    assert round(pt.x, 5) == x
    assert round(pt.y, 5) == y


@test_only_with_dialects("postgresql")
class TestTypeDecorator():

    def _create_one_point(self, session, conn):
        metadata.drop_all(conn, checkfirst=True)
        metadata.create_all(conn)

        # Create new point instance
        p = Point()
        p.raw_geom = "SRID=4326;POINT(5 45)"
        p.geom = "SRID=4326;POINT(5 45)"
        p.three_d_geom = "SRID=4326;POINT(5 45)"  # Insert 2D geometry into 3D column

        # Insert point
        session.add(p)
        session.flush()
        session.expire(p)

        return p.id

    def test_transform(self, session, conn):
        self._create_one_point(session, conn)

        # Query the point and check the result
        pt = session.query(Point).one()
        assert pt.id == 1
        assert pt.raw_geom.srid == 4326
        check_wkb(pt.raw_geom, 5, 45)

        assert pt.geom.srid == 4326
        check_wkb(pt.geom, 5, 45)

        # Check that the data is correct in DB using raw query
        q = text("SELECT id, ST_AsEWKT(geom) AS geom FROM point;")
        res_q = session.execute(q).fetchone()
        assert res_q.id == 1
        assert res_q.geom == "SRID=2154;POINT(857581.899319668 6435414.7478354)"

        # Compare geom, raw_geom with auto transform and explicit transform
        pt_trans = session.query(
            Point,
            Point.raw_geom,
            func.ST_Transform(Point.raw_geom, 2154).label("trans"),
        ).one()

        assert pt_trans[0].id == 1
```

(continues on next page)

(continued from previous page)

```
        assert pt_trans[0].geom.srid == 4326
        check_wkb(pt_trans[0].geom, 5, 45)

        assert pt_trans[0].raw_geom.srid == 4326
        check_wkb(pt_trans[0].raw_geom, 5, 45)

        assert pt_trans[1].srid == 4326
        check_wkb(pt_trans[1], 5, 45)

        assert pt_trans[2].srid == 2154
        check_wkb(pt_trans[2], 857581.89932, 6435414.74784)

    def test_force_3d(self, session, conn):
        self._create_one_point(session, conn)

        # Query the point and check the result
        pt = session.query(Point).one()

        assert pt.id == 1
        assert pt.three_d_geom.srid == 4326
        assert pt.three_d_geom.desc.lower() == (
            '01010000a0e6100000000000000000144000000000000804640000000000000000000')
```

### 5.1.2 Compute length on insert

It is possible to insert a geometry and ask PostgreSQL to compute its length at the same time. This example uses SQLAlchemy core queries.

```
from sqlalchemy import Column
from sqlalchemy import Float
from sqlalchemy import Integer
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import bindparam
from sqlalchemy import func

from geoalchemy2 import Geometry
from geoalchemy2.shape import to_shape

# Tests imports
from tests import select
from tests import test_only_with_dialects

metadata = MetaData()

table = Table(
    "inserts",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("geom", Geometry("LINESTRING", 4326)),
    Column("distance", Float),
)


```

(continues on next page)

(continued from previous page)

```
@test_only_with_dialects("postgresql")
class TestLengthAtInsert():

    def test_query(self, conn):
        metadata.drop_all(conn, checkfirst=True)
        metadata.create_all(conn)

        # Define geometries to insert
        values = [
            {"ewkt": "SRID=4326;LINESTRING(0 0, 1 0)"},
            {"ewkt": "SRID=4326;LINESTRING(0 0, 0 1)"}
        ]

        # Define the query to compute distance (without spheroid)
        distance = func.ST_Length(func.ST_GeomFromText(bindparam("ewkt")), False)

        i = table.insert()
        i = i.values(geom=bindparam("ewkt"), distance=distance)

        # Execute the query with values as parameters
        conn.execute(i, values)

        # Check the result
        q = select([table])
        res = conn.execute(q).fetchall()

        # Check results
        assert len(res) == 2

        r1 = res[0]
        assert r1[0] == 1
        assert r1[1].srid == 4326
        assert to_shape(r1[1]).wkt == "LINESTRING (0 0, 1 0)"
        assert round(r1[2]) == 111195

        r2 = res[1]
        assert r2[0] == 2
        assert r2[1].srid == 4326
        assert to_shape(r2[1]).wkt == "LINESTRING (0 0, 0 1)"
        assert round(r2[2]) == 111195
```

### 5.1.3 Decipher Raster

The *RasterElement* objects store the Raster data in WKB form. When using rasters it is usually better to convert them into TIFF, PNG, JPEG or whatever. Nevertheless, it is possible to decipher the WKB to get a 2D list of values. This example uses SQLAlchemy ORM queries.

```
import binascii
import struct

import pytest
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import MetaData
from sqlalchemy.ext.declarative import declarative_base
```

(continues on next page)

(continued from previous page)

```

from geoalchemy2 import Raster
from geoalchemy2 import WKTElement

# Tests imports
from tests import test_only_with_dialects

metadata = MetaData()
Base = declarative_base(metadata=metadata)


class Ocean(Base):
    __tablename__ = 'ocean'
    id = Column(Integer, primary_key=True)
    rast = Column(Raster)

    def __init__(self, rast):
        self.rast = rast


def _format_e(endianness, struct_format):
    return _ENDIANNESS[endianness] + struct_format


def wkbHeader(raw):
    # Function to decipher the WKB header
    # See http://trac.osgeo.org/postgis/browser/trunk/raster/doc/RFC2-
↪WellKnownBinaryFormat

    header = {}

    header['endianness'] = struct.unpack('b', raw[0:1])[0]

    e = header['endianness']
    header['version'] = struct.unpack(_format_e(e, 'H'), raw[1:3])[0]
    header['nbands'] = struct.unpack(_format_e(e, 'H'), raw[3:5])[0]
    header['scaleX'] = struct.unpack(_format_e(e, 'd'), raw[5:13])[0]
    header['scaleY'] = struct.unpack(_format_e(e, 'd'), raw[13:21])[0]
    header['ipX'] = struct.unpack(_format_e(e, 'd'), raw[21:29])[0]
    header['ipY'] = struct.unpack(_format_e(e, 'd'), raw[29:37])[0]
    header['skewX'] = struct.unpack(_format_e(e, 'd'), raw[37:45])[0]
    header['skewY'] = struct.unpack(_format_e(e, 'd'), raw[45:53])[0]
    header['srid'] = struct.unpack(_format_e(e, 'i'), raw[53:57])[0]
    header['width'] = struct.unpack(_format_e(e, 'H'), raw[57:59])[0]
    header['height'] = struct.unpack(_format_e(e, 'H'), raw[59:61])[0]

    return header


def read_band(data, offset, pixtype, height, width, endianness=1):
    ptype, _, psize = _PTYPE[pixtype]
    pix_data = data[offset + 1: offset + 1 + width * height * psize]
    band = [
        [
            struct.unpack(_format_e(endianness, ptype), pix_data[
                (i * width + j) * psize: (i * width + j + 1) * psize
            ])[0]
```

(continues on next page)

(continued from previous page)

```
            for j in range(width)
        ]
        for i in range(height)
    ]
    return band


def read_band_numpy(data, offset, pixtype, height, width, endianness=1):
    import numpy as np  # noqa
    _, dtype, psize = _PTYPE[pixtype]
    dt = np.dtype(dtype)
    dt = dt.newbyteorder(_ENDIANNESS[endianness])
    band = np.frombuffer(data, dtype=dtype,
                         count=height * width, offset=offset + 1)
    band = (np.reshape(band, ((height, width))))
    return band


_PTYPE = {
    0: ['?', '?', 1],
    1: ['B', 'B', 1],
    2: ['B', 'B', 1],
    3: ['b', 'b', 1],
    4: ['B', 'B', 1],
    5: ['h', 'i2', 2],
    6: ['H', 'u2', 2],
    7: ['i', 'i4', 4],
    8: ['I', 'u4', 4],
    10: ['f', 'f4', 4],
    11: ['d', 'f8', 8],
}

_ENDIANNESS = {
    0: '>',
    1: '<',
}


def wkbImage(raster_data, use_numpy=False):
    """Function to decipher the WKB raster data"""

    # Get binary data
    raw = binascii.unhexlify(raster_data)

    # Read header
    h = wkbHeader(bytes(raw))
    e = h["endianness"]

    img = []  # array to store image bands
    offset = 61  # header raw length in bytes
    band_size = h['width'] * h['height']  # number of pixels in each band

    for i in range(h['nbands']):
        # Determine pixtype for this band
        pixtype = struct.unpack(_format_e(e, 'b'), raw[offset: offset + 1])[0] - 64

        # Read data with either pure Python or Numpy
```

(continues on next page)

(continued from previous page)

```
        if use_numpy:
            band = read_band_numpy(
                raw, offset, pixtype, h['height'], h['width'])
        else:
            band = read_band(
                raw, offset, pixtype, h['height'], h['width'])

        # Store the result
        img.append(band)
        offset = offset + 2 + band_size

    return img


@test_only_with_dialects("postgresql")
class TestDecipherRaster():

    @pytest.mark.parametrize("pixel_type", [
        '1BB',
        '2BUI',
        '4BUI',
        '8BSI',
        '8BUI',
        '16BSI',
        '16BUI',
        '32BSI',
        '32BUI',
        '32BF',
        '64BF'
    ])
    def test_decipher_raster(self, pixel_type, session, conn):
        """Create a raster and decipher it"""
        metadata.drop_all(conn, checkfirst=True)
        metadata.create_all(conn)

        # Create a new raster
        polygon = WKTElement('POLYGON((0 0,1 1,0 1,0 0))', srid=4326)
        o = Ocean(polygon.ST_AsRaster(5, 6, pixel_type))
        session.add(o)
        session.flush()

        # Decipher data from each raster
        image = wkbImage(o.rast.data)

        # Define expected result
        expected = [
            [0, 1, 1, 1, 1],
            [1, 1, 1, 1, 1],
            [0, 1, 1, 1, 0],
            [0, 1, 1, 0, 0],
            [0, 1, 0, 0, 0],
            [0, 0, 0, 0, 0]
        ]

        # Check results
        band = image[0]
        assert band == expected
```

### 5.1.4 Disable wrapping in select

If the application wants to build queries with GeoAlchemy 2 and gets them as strings, the wrapping of geometry columns with a *ST_AsEWKB()* function might be annoying. In this case it is possible to disable this wrapping. This example uses SQLAlchemy ORM queries.

```
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import func
from sqlalchemy.ext.declarative import declarative_base

from geoalchemy2 import Geometry

# Tests imports
from tests import select

Base = declarative_base()


class RawGeometry(Geometry):
    """This class is used to remove the 'ST_AsEWKB()'' function from select queries""
↪"

    def column_expression(self, col):
        return col


class Point(Base):
    __tablename__ = "point"
    id = Column(Integer, primary_key=True)
    geom = Column(Geometry(srid=4326, geometry_type="POINT"))
    raw_geom = Column(
        RawGeometry(srid=4326, geometry_type="POINT"))


def test_no_wrapping():
    # Select all columns
    select_query = select([Point])

    # Check that the 'geom' column is wrapped by 'ST_AsEWKB()' and that the column
    # 'raw_geom' is not.
    assert str(select_query) == (
        "SELECT point.id, ST_AsEWKB(point.geom) AS geom, point.raw_geom \n"
        "FROM point"
    )


def test_func_no_wrapping():
    # Select query with function
    select_query = select([
        func.ST_Buffer(Point.geom),  # with wrapping (default behavior)
        func.ST_Buffer(Point.geom, type_=Geometry),  # with wrapping
        func.ST_Buffer(Point.geom, type_=RawGeometry)  # without wrapping
    ])

    # Check the query
    assert str(select_query) == (
        "SELECT "
```

(continues on next page)

(continued from previous page)

```
        "ST_AsEWKB(ST_Buffer(point.geom)) AS \"ST_Buffer_1\", "
        "ST_AsEWKB(ST_Buffer(point.geom)) AS \"ST_Buffer_2\", "
        "ST_Buffer(point.geom) AS \"ST_Buffer_3\" \n"
        "FROM point"
    )
```

### 5.1.5 Function translation for specific dialect

Some functions have different names depending on the dialect. But sometimes one function in one dialect can be mapped to several other functions in another dialect, depending on the arguments passed. For example, the *ST_Buffer* function in PostgreSQL can translate into 2 functions in SQLite:

1. if the buffer is two-sided (symmetric), the PostgreSQL function:

   ```
   ST_Buffer(the_table.geom, 10)
   ```

   should become in SQLite:

   ```
   Buffer(the_table.geom, 10)
   ```

2. if the buffer is one-sided, the PostgreSQL function:

   ```
   ST_Buffer(the_table.geom, 10, 'side=right')
   ```

   should become in SQLite:

   ```
   SingleSidedBuffer(the_table.geom, 10, 0)
   ```

This case is much more complicated than just mapping a function name and we show here how to deal with it.

This example uses SQLAlchemy core queries.

```
from sqlalchemy import MetaData
from sqlalchemy import func
from sqlalchemy.ext.compiler import compiles
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.sql.expression import BindParameter

from geoalchemy2 import WKTElement
from geoalchemy2 import functions

# Tests imports
from tests import format_wkt
from tests import select

metadata = MetaData()
Base = declarative_base(metadata=metadata)


def _compile_buffer_default(element, compiler, **kw):
    """Compile the element in the default case (no specific dialect).

    This function should not be needed for SQLAlchemy >= 1.1.
    """
    return '{}({})'.format('ST_Buffer', compiler.process(element.clauses, **kw))
```

(continues on next page)

(continued from previous page)

```


def _compile_buffer_sqlite(element, compiler, **kw):
    """Compile the element for the SQLite dialect."""
    # Get the side parameters
    compiled = compiler.process(element.clauses, **kw)
    side_params = [
        i for i in element.clauses
        if isinstance(i, BindParameter) and 'side' in str(i.value)
    ]

    if side_params:
        side_param = side_params[0]
        if 'right' in side_param.value:
            # If the given side is 'right', we translate the value into 0 and switch to the sided
            # function
            side_param.value = 0
            element.identifier = 'SingleSidedBuffer'
        elif 'left' in side_param.value:
            # If the given side is 'left', we translate the value into 1 and switch to the sided
            # function
            side_param.value = 1
            element.identifier = 'SingleSidedBuffer'

    if element.identifier == 'ST_Buffer':
        # If the identifier is still the default ST_Buffer we switch to the SpatiaLite function
        element.identifier = 'Buffer'

    # If there is no side parameter or if the side value is 'both', we use the default function
    return '{}({})'.format(element.identifier, compiled)


# Register the specific compilation rules
compiles(functions.ST_Buffer)(_compile_buffer_default)
compiles(functions.ST_Buffer, 'sqlite')(_compile_buffer_sqlite)


def test_specific_compilation(conn):
    # Build a query with a sided buffer
    query = select([
        func.ST_AsText(
            func.ST_Buffer(WKTElement('LINESTRING(0 0, 1 0)', srid=4326), 1, 'side=left')
        )
    ])

    # Check the compiled query: the sided buffer should appear only in the SQLite query
    compiled_query = str(query.compile(dialect=conn.dialect))
    if conn.dialect.name == 'sqlite':
        assert 'SingleSidedBuffer' in compiled_query
        assert 'ST_Buffer' not in compiled_query
    else:
```

(continues on next page)

(continued from previous page)

```
        assert 'SingleSidedBuffer' not in compiled_query
        assert 'ST_Buffer' in compiled_query

    # Check the actual result of the query
    res = conn.execute(query).scalar()
    assert format_wkt(res) == 'POLYGON((1 0,0 0,0 1,1 1,1 0))'

    # Build a query with symmetric buffer to check nothing was broken
    query = select([
        func.ST_AsText(
            func.ST_Buffer(WKTElement('LINESTRING(0 0, 1 0)', srid=4326), 1)
        )
    ])

    # Check the compiled query: the sided buffer should never appear in the query
    compiled_query = str(query.compile(dialect=conn.dialect))
    assert 'SingleSidedBuffer' not in compiled_query
    if conn.dialect.name == 'sqlite':
        assert 'ST_Buffer' not in compiled_query
        assert 'Buffer' in compiled_query
    else:
        assert 'ST_Buffer' in compiled_query

    # Check the actual result of the query
    res = conn.execute(query).scalar()
    assert format_wkt(res) != 'POLYGON((1 0,0 0,0 1,1 1,1 0))'
    assert format_wkt(res).startswith('POLYGON((1 1,1')
```

### 5.1.6 Reproject a Raster using ST_Transform

The *ST_Transform()* function (and a few others like *ST_SnapToGrid()*) can be used on both *Geometry* and *Raster* types. In *GeoAlchemy2*, this function is only defined for *Geometry* as it can not be defined for several types at the same time. Thus using this function on *Raster* requires minor tweaking.

This example uses both SQLAlchemy core and ORM queries.

```
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import func
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import Query

from geoalchemy2 import Geometry
from geoalchemy2 import Raster

# Tests imports
from tests import select

metadata = MetaData()
Base = declarative_base(metadata=metadata)

table = Table(
    "raster_table",
```

(continues on next page)

(continued from previous page)

```
    metadata,
    Column("id", Integer, primary_key=True),
    Column("geom", Geometry("POLYGON", 4326)),
    Column("rast", Raster()),
)


class RasterTable(Base):
    __tablename__ = 'raster_table_orm'
    id = Column(Integer, primary_key=True)
    geom = Column(Geometry("POLYGON", 4326))
    rast = Column(Raster())

    def __init__(self, rast):
        self.rast = rast


def test_transform_core():
    # Define the transform query for both the geometry and the raster in a naive way
    wrong_query = select([
        func.ST_Transform(table.c.geom, 2154),
        func.ST_Transform(table.c.rast, 2154)
    ])

    # Check the query
    assert str(wrong_query) == (
        "SELECT "
        "ST_AsEWKB("
        "ST_Transform(raster_table.geom, :ST_Transform_2)) AS \"ST_Transform_1\", "
        "ST_AsEWKB("  # <= Note that the raster is processed as a Geometry here
        "ST_Transform(raster_table.rast, :ST_Transform_4)) AS \"ST_Transform_3\" \n"
        "FROM raster_table"
    )

    # Define the transform query for both the geometry and the raster in the correct way
    correct_query = select([
        func.ST_Transform(table.c.geom, 2154),
        func.ST_Transform(table.c.rast, 2154, type_=Raster)
    ])

    # Check the query
    assert str(correct_query) == (
        "SELECT "
        "ST_AsEWKB("
        "ST_Transform(raster_table.geom, :ST_Transform_2)) AS \"ST_Transform_1\", "
        "raster("  # <= This time the raster is correctly processed as a Raster
        "ST_Transform(raster_table.rast, :ST_Transform_4)) AS \"ST_Transform_3\" \n"
        "FROM raster_table"
    )


def test_transform_ORM():
    # Define the transform query for both the geometry and the raster in a naive way
    wrong_query = Query([
        RasterTable.geom.ST_Transform(2154),
        RasterTable.rast.ST_Transform(2154)
```

(continues on next page)

(continued from previous page)

```
    ])

    # Check the query
    assert str(wrong_query) == (
        "SELECT "
        "ST_AsEWKB("
        "ST_Transform(raster_table_orm.geom, :ST_Transform_2)) AS \"ST_Transform_1\",
↪ "
        "ST_AsEWKB("  # <= Note that the raster is processed as a Geometry here
        "ST_Transform(raster_table_orm.rast, :ST_Transform_4)) AS \"ST_Transform_3\"␣
↪\n"
        "FROM raster_table_orm"
    )

    # Define the transform query for both the geometry and the raster in the correct␣
↪way
    correct_query = Query([
        RasterTable.geom.ST_Transform(2154),
        RasterTable.rast.ST_Transform(2154, type_=Raster)
    ])

    # Check the query
    assert str(correct_query) == (
        "SELECT "
        "ST_AsEWKB("
        "ST_Transform(raster_table_orm.geom, :ST_Transform_2)) AS \"ST_Transform_1\",
↪ "
        "raster("  # <= This time the raster is correctly processed as a Raster
        "ST_Transform(raster_table_orm.rast, :ST_Transform_4)) AS \"ST_Transform_3\"␣
↪\n"
        "FROM raster_table_orm"
    )
```

### 5.1.7 Use CompositeType

Some functions return composite types. This example shows how to deal with this kind of functions.

```
import pytest
from pkg_resources import parse_version
from sqlalchemy import Column
from sqlalchemy import Float
from sqlalchemy import Integer
from sqlalchemy import MetaData
from sqlalchemy import __version__ as SA_VERSION
from sqlalchemy.ext.declarative import declarative_base

from geoalchemy2 import Raster
from geoalchemy2 import WKTElement
from geoalchemy2.functions import GenericFunction
from geoalchemy2.types import CompositeType

# Tests imports
from tests import select
from tests import test_only_with_dialects

```

(continues on next page)

(continued from previous page)

```
class SummaryStatsCustomType(CompositeType):
    """Define the composite type returned by the function ST_SummaryStatsAgg."""
    typemap = {
        'count': Integer,
        'sum': Float,
        'mean': Float,
        'stddev': Float,
        'min': Float,
        'max': Float,
    }

    cache_ok = True


class ST_SummaryStatsAgg(GenericFunction):
    type = SummaryStatsCustomType
    # Set a specific identifier to not override the actual ST_SummaryStatsAgg function
    identifier = "ST_SummaryStatsAgg_custom"

    inherit_cache = True


metadata = MetaData()
Base = declarative_base(metadata=metadata)


class Ocean(Base):
    __tablename__ = 'ocean'
    id = Column(Integer, primary_key=True)
    rast = Column(Raster)

    def __init__(self, rast):
        self.rast = rast


@test_only_with_dialects("postgresql")
class TestSTSummaryStatsAgg():

    @pytest.mark.skipif(
        parse_version(SA_VERSION) < parse_version("1.4"),
        reason="requires SQLAlchely>1.4",
    )
    def test_st_summary_stats_agg(self, session, conn):
        metadata.drop_all(conn, checkfirst=True)
        metadata.create_all(conn)

        # Create a new raster
        polygon = WKTElement('POLYGON((0 0,1 1,0 1,0 0))', srid=4326)
        o = Ocean(polygon.ST_AsRaster(5, 6))
        session.add(o)
        session.flush()

        # Define the query to compute stats
        stats_agg = select([
            Ocean.rast.ST_SummaryStatsAgg_custom(1, True, 1).label("stats")
```

(continues on next page)

(continued from previous page)

```
        ])
        stats_agg_alias = stats_agg.alias("stats_agg")

        # Use these stats
        query = select([
            stats_agg_alias.c.stats.count.label("count"),
            stats_agg_alias.c.stats.sum.label("sum"),
            stats_agg_alias.c.stats.mean.label("mean"),
            stats_agg_alias.c.stats.stddev.label("stddev"),
            stats_agg_alias.c.stats.min.label("min"),
            stats_agg_alias.c.stats.max.label("max")
        ])

        # Check the query
        assert str(query.compile(dialect=session.bind.dialect)) == (
            "SELECT "
            "(stats_agg.stats).count AS count, "
            "(stats_agg.stats).sum AS sum, "
            "(stats_agg.stats).mean AS mean, "
            "(stats_agg.stats).stddev AS stddev, "
            "(stats_agg.stats).min AS min, "
            "(stats_agg.stats).max AS max \n"
            "FROM ("
            "SELECT "
            "ST_SummaryStatsAgg("
            "ocean.rast, "
            "%(ST_SummaryStatsAgg_1)s, %(ST_SummaryStatsAgg_2)s, %(ST_
↪SummaryStatsAgg_3)s"
            ") AS stats \n"
            "FROM ocean) AS stats_agg"
        )

        # Execute the query
        res = session.execute(query).fetchall()

        # Check the result
        assert res == [(15, 15.0, 1.0, 0.0, 1.0, 1.0)]
```

The *Gallery* page shows examples of the GeoAlchemy 2's functionalities.

CHAPTER 6

# Use with Alembic

## 6.1 Use Alembic with GeoAlchemy 2

The Alembic package is a lightweight database migration tool which is able to automatically detect the table column types.

### 6.1.1 Interactions between Alembic and GeoAlchemy 2

Interactions between some features of Alembic and GeoAlchemy 2 may lead to errors in migration scripts, especially when using the `--autogenerate` feature of Alembic with the `spatial_index=True` feature of GeoAlchemy 2. In this case, the following errors occur:

1. the migration script miss the relevant imports from `geoalchemy2`.
2. the migration script will create the indexes of the spatial columns after the table is created, but these indexes are already automatically created during table creation, which will lead to an error.

For example, suppose the following table is defined:

```
class Lake(Base):
    __tablename__ = 'lake'
    id = Column(Integer, primary_key=True)
    geom = Column(
        Geometry(
            geometry_type='LINESTRING',
            srid=4326,
            spatial_index=True,
        )
    )
```

Then the command `alembic revision --autogenerate -m "Create new table"` will create the following migration script:

```
"""Create new table

Revision ID: <rev_id>
Revises: <down_rev_id>
Create Date: <date>

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = "<rev_id>"
down_revision = "<down_rev_id>"
branch_labels = None
depends_on = None


def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table(
        "lake",
        sa.Column("id", sa.Integer(), nullable=False),
        sa.Column(
            "geom",
            geoalchemy2.types.Geometry(
                geometry_type="LINESTRING",
                srid=4326,
                from_text="ST_GeomFromEWKT",
                name="geometry",
            ),
            nullable=True,
        ),
        sa.PrimaryKeyConstraint("id"),
    )
    op.create_index(
        "idx_lake_geom",
        "lake",
        ["geom"],
        unique=False,
        postgresql_using="gist",
        postgresql_ops={},
    )
    # ### end Alembic commands ###


def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_index(
        "idx_lake_geom",
        table_name="lake",
        postgresql_using="gist",
        postgresql_ops={},
    )
    op.drop_table("lake")
    # ### end Alembic commands ###
```

In this case, we have to do the following changes to make it work:

1. add the missing import `from geoalchemy2 import Geometry`.
2. remove the `create_index` statement in the `upgrade()` function.
3. remove the `drop_index` statement in the `downgrade()` function.

### 6.1.2 Helpers

In order to make the use of Alembic easier, a few helpers are provided in *geoalchemy2.alembic_helpers*. These helpers can be used in the `env.py` file used by Alembic, like in the following example:

```
# ...
from geoalchemy2.alembic_helpers import include_object
from geoalchemy2.alembic_helpers import render_item
# ...

def run_migrations_offline():
    # ...
    context.configure(
        # ...
        process_revision_directives=alembic_helpers.writer,
        render_item=alembic_helpers.render_item,
    )
    # ...


def run_migrations_online():
    # ...
    context.configure(
        # ...
        process_revision_directives=alembic_helpers.writer,
        render_item=alembic_helpers.render_item,
    )
    # ...
```

After running the `alembic` command, the migration script should be properly generated and should not need to be manually edited.

### 6.1.3 Dealing with custom types

With `SQLAlchemy`, users are able to define custom types, as shown in *Automatically use a function at insert or select*. In this case, you can refer to the dedicated page of Alembic's documentation for the details.

A simple solution for this case is to create a new `render_item` function to add specific imports for these custom types. For example, if your custom type is called `TheCustomType` and is defined in `my_package.custom_types`, you just have to edit the `env.py` file like the following:

```
# ...
from geoalchemy2.alembic_helpers import include_object
from geoalchemy2.alembic_helpers import render_item as spatial_render_item
from my_package.custom_types import TheCustomType
# ...


def render_item(obj_type, obj, autogen_context):
    """Apply custom rendering for selected items."""
```

(continues on next page)

(continued from previous page)

```
    spatial_type = spatial_render_item(obj_type, obj, autogen_context)
    if spatial_type:
        return spatial_type

    # For the custom type
    if obj_type == 'type' and isinstance(obj, TheCustomType):
        import_name = obj.__class__.__name__
        autogen_context.imports.add(f"from my_package.custom_types import {import_
↪name}")
        return "%r" % obj

    # default rendering for other objects
    return False



def run_migrations_offline():
    # ...
    context.configure(
        # ...
        process_revision_directives=alembic_helpers.writer,
        render_item=render_item,
    )
    # ...


def run_migrations_online():
    # ...
    context.configure(
        # ...
        process_revision_directives=alembic_helpers.writer,
        render_item=render_item,
    )
    # ...
```

Then the proper imports will be automatically added in the migration scripts.

### 6.1.4 Dialects

Some dialects (like SQLite) require some specific management to alter columns or tables. In this case, other dedicated helpers are provided to handle this. For example, if one wants to add and drop columns in a SQLite database, the SpatiaLite extension should be loaded when the engine connects, thus the `env.py` file should look like the following:

```
from geoalchemy2.alembic_helpers import load_spatialite
from geoalchemy2.alembic_helpers import writer


def run_migrations_offline():
    # ...
    context.configure(
        # ...
        process_revision_directives=writer,
        render_item=alembic_helpers.render_item,
    )
    # ...
```

(continues on next page)

(continued from previous page)

```
def run_migrations_online():
    # ...
    if connectable.dialect.name == "sqlite":
        # Load the SpatiaLite extension when the engine connects to the DB
        listen(connectable, 'connect', load_spatialite)

    with connectable.connect() as connection:
        # ...
        context.configure(
            # ...
            process_revision_directives=writer,
            render_item=alembic_helpers.render_item,
        )
        # ...
```

The GeoAlchemy 2 package is compatible with the migration tool Alembic. The *Use Alembic with GeoAlchemy 2* page provides more details on this topic.

CHAPTER 7

# Reference Documentation

## 7.1 Types

This module defines the *geoalchemy2.types.Geometry*, *geoalchemy2.types.Geography*, and *geoalchemy2.types.Raster* classes, that are used when defining geometry, geography and raster columns/properties in models.

### 7.1.1 Reference

**class** geoalchemy2.types.**CompositeType**

Bases: sqlalchemy.sql.type_api.UserDefinedType

A wrapper for *geoalchemy2.elements.CompositeElement*, that can be used as the return type in PostgreSQL functions that return composite values.

This is used as the base class of *geoalchemy2.types.GeometryDump*.

**class comparator_factory**(*expr*)

Bases: sqlalchemy.sql.type_api.Comparator

**typemap = {}**

Dictionary used for defining the content types and their corresponding keys. Set in subclasses.

**class** geoalchemy2.types.**Geography**(*geometry_type='GEOMETRY'*, *srid=-1*, *dimension=2*, *spatial_index=True*, *use_N_D_index=False*, *management=False*, *use_typmod=None*, *from_text=None*, *name=None*, *nullable=True*, *_spatial_index_reflected=None*)

Bases: *geoalchemy2.types._GISType*

The Geography type.

Creating a geography column is done like this:

```
Column(Geography(geometry_type='POINT', srid=4326))
```

See *geoalchemy2.types._GISType* for the list of arguments that can be passed to the constructor.

**ElementType**
alias of *geoalchemy2.elements.WKBElement*

**as_binary = 'ST_AsBinary'**
The "as binary" function to use. Used by the parent class' `column_expression` method.

**cache_ok = False**
Disable cache for this type.

**from_text = 'ST_GeogFromText'**
The `FromText` geography constructor. Used by the parent class' `bind_expression` method.

**name = 'geography'**
Type name used for defining geography columns in `CREATE TABLE`.

**class** geoalchemy2.types.**Geometry**(*geometry_type='GEOMETRY'*, *srid=-1*, *dimension=2*, *spatial_index=True*, *use_N_D_index=False*, *management=False*, *use_typmod=None*, *from_text=None*, *name=None*, *nullable=True*, *_spatial_index_reflected=None*)

Bases: *geoalchemy2.types._GISType*

The Geometry type.

Creating a geometry column is done like this:

```
Column(Geometry(geometry_type='POINT', srid=4326))
```

See *geoalchemy2.types._GISType* for the list of arguments that can be passed to the constructor.

If `srid` is set then the `WKBElement` objects resulting from queries will have that SRID, and, when constructing the `WKBElement` objects, the SRID won't be read from the data returned by the database. If `srid` is not set (meaning it's `-1`) then the SRID set in `WKBElement` objects will be read from the data returned by the database.

**ElementType**
alias of *geoalchemy2.elements.WKBElement*

**as_binary = 'ST_AsEWKB'**
The "as binary" function to use. Used by the parent class' `column_expression` method.

**cache_ok = False**
Disable cache for this type.

**from_text = 'ST_GeomFromEWKT'**
The "from text" geometry constructor. Used by the parent class' `bind_expression` method.

**name = 'geometry'**
Type name used for defining geometry columns in `CREATE TABLE`.

**class** geoalchemy2.types.**GeometryDump**

Bases: *geoalchemy2.types.CompositeType*

The return type for functions like `ST_Dump`, consisting of a path and a geom field. You should normally never use this class directly.

**cache_ok = True**
Enable cache for this type.

**typemap = {'geom': <class 'geoalchemy2.types.Geometry'>, 'path': ARRAY(Integer())}**
Dictionary defining the contents of a `geometry_dump`.

**class** geoalchemy2.types.**Raster**(*spatial_index=True*, *from_text=None*, *name=None*, *nullable=True*)
Bases: *geoalchemy2.types._GISType*

The Raster column type.

Creating a raster column is done like this:

```
Column(Raster)
```

This class defines the `result_processor` method, so that raster values received from the database are converted to *geoalchemy2.elements.RasterElement* objects.

**Parameters** **spatial_index** – Indicate if a spatial index should be created. Default is `True`.

**ElementType**
alias of *geoalchemy2.elements.RasterElement*

**as_binary = 'raster'**
The "as binary" function to use. Used by the parent class' `column_expression` method.

**cache_ok = False**
Disable cache for this type.

**comparator_factory**
alias of *geoalchemy2.comparator.BaseComparator*

**from_text = 'raster'**
The "from text" raster constructor. Used by the parent class' `bind_expression` method.

**name = 'raster'**
Type name used for defining raster columns in `CREATE TABLE`.

**class** geoalchemy2.types.**SummaryStats**
Bases: *geoalchemy2.types.CompositeType*

Define the composite type returned by the function ST_SummaryStatsAgg

**cache_ok = True**
Enable cache for this type.

**class** geoalchemy2.types.**_DummyGeometry**(*geometry_type='GEOMETRY'*, *srid=-1*, *dimension=2*, *spatial_index=True*, *use_N_D_index=False*, *management=False*, *use_typmod=None*, *from_text=None*, *name=None*, *nullable=True*, *_spatial_index_reflected=None*)
Bases: *geoalchemy2.types.Geometry*

A dummy type only used with SQLite.

**class** geoalchemy2.types.**_GISType**(*geometry_type='GEOMETRY'*, *srid=-1*, *dimension=2*, *spatial_index=True*, *use_N_D_index=False*, *management=False*, *use_typmod=None*, *from_text=None*, *name=None*, *nullable=True*, *_spatial_index_reflected=None*)
Bases: `sqlalchemy.sql.type_api.UserDefinedType`

The base class for *geoalchemy2.types.Geometry* and *geoalchemy2.types.Geography*.

This class defines `bind_expression` and `column_expression` methods that wrap column expressions in `ST_GeomFromEWKT`, `ST_GeogFromText`, or `ST_AsEWKB` calls.

This class also defines `result_processor` and `bind_processor` methods. The function returned by `result_processor` converts WKB values received from the database to *geoalchemy2.elements.WKBElement* objects. The function returned by `bind_processor` converts *geoalchemy2.elements.WKTElement* objects to EWKT strings.

**Parameters**

- **geometry_type** – The geometry type.

  Possible values are:

  - `"GEOMETRY"`,
  - `"POINT"`,
  - `"LINESTRING"`,
  - `"POLYGON"`,
  - `"MULTIPOINT"`,
  - `"MULTILINESTRING"`,
  - `"MULTIPOLYGON"`,
  - `"GEOMETRYCOLLECTION"`,
  - `"CURVE"`,
  - `None`.

  The latter is actually not supported with *geoalchemy2.types.Geography*.

  When set to `None` then no "geometry type" constraints will be attached to the geometry type declaration. Using `None` here is not compatible with setting `management` to `True`.

  Default is `"GEOMETRY"`.

- **srid** – The SRID for this column. E.g. 4326. Default is `-1`.

- **dimension** – The dimension of the geometry. Default is `2`.

  With `management` set to `True`, that is when `AddGeometryColumn` is used to add the geometry column, there are two constraints:

  - The `geometry_type` must not end with `"ZM"`. This is due to PostGIS' `AddGeometryColumn` failing with ZM geometry types. Instead the "simple" geometry type (e.g. POINT rather POINTZM) should be used with `dimension` set to `4`.
  - When the `geometry_type` ends with `"Z"` or `"M"` then `dimension` must be set to `3`.

  With `management` set to `False` (the default) `dimension` is not taken into account, and the actual dimension is fully defined with the `geometry_type`.

- **spatial_index** – Indicate if a spatial index should be created. Default is `True`.

- **use_N_D_index** – Use the N-D index instead of the standard 2-D index.

- **management** – Indicate if the `AddGeometryColumn` and `DropGeometryColumn` managements functions should be called when adding and dropping the geometry column. Should be set to `True` for PostGIS 1.x and SQLite. Default is `False`. Note that this option has no effect for *geoalchemy2.types.Geography*.

- **use_typmod** – By default PostgreSQL type modifiers are used to create the geometry column. To use check constraints instead set `use_typmod` to `False`. By default this option is not included in the call to `AddGeometryColumn`. Note that this option is only taken into account if `management` is set to `True` and is only available for PostGIS 2.x.

**as_binary = None**
The name of the "as binary" function for this type. Set in subclasses.

**bind_expression**(*bindvalue*)
Specific bind_expression that automatically adds a conversion function

**bind_processor**(*dialect*)
Specific bind_processor that automatically process spatial elements

**cache_ok = False**
Disable cache for this type.

**column_expression**(*col*)
Specific column_expression that automatically adds a conversion function

**comparator_factory**
alias of *geoalchemy2.comparator.Comparator*

**from_text = None**
The name of "from text" function for this type. Set in subclasses.

**name = None**
Name used for defining the main geo type (geometry or geography) in CREATE TABLE statements. Set in subclasses.

**result_processor**(*dialect*, *coltype*)
Specific result_processor that automatically process spatial elements

## 7.2 Elements

**class** geoalchemy2.elements.**HasFunction**
Bases: object

Base class used as a marker to know if a given element has a 'geom_from' function.

**class** geoalchemy2.elements.**_SpatialElement**(*data*, *srid=-1*, *extended=False*)
Bases: *geoalchemy2.elements.HasFunction*

The base class for public spatial elements.

**Parameters**

- **data** – The first argument passed to the constructor is the data wrapped by the _SpatialElement object being constructed.
- **srid** – An integer representing the spatial reference system. E.g. 4326. Default value is -1, which means no/unknown reference system.
- **extended** – A boolean indicating whether the extended format (EWKT or EWKB) is used. Default is False.

**class** geoalchemy2.elements.**WKTElement**(*data*, *srid=-1*, *extended=False*)
Bases: *geoalchemy2.elements._SpatialElement*

Instances of this class wrap a WKT or EWKT value.

Usage examples:

```
wkt_element_1 = WKTElement('POINT(5 45)')
wkt_element_2 = WKTElement('POINT(5 45)', srid=4326)
wkt_element_3 = WKTElement('SRID=4326;POINT(5 45)', extended=True)
```

**desc**
This element's description string.

**geom_from = 'ST_GeomFromText'**

**geom_from_extended_version = 'ST_GeomFromEWKT'**

**class** geoalchemy2.elements.**WKBElement**(*data*, *srid=-1*, *extended=False*)
Bases: *geoalchemy2.elements._SpatialElement*

Instances of this class wrap a WKB or EWKB value.

Geometry values read from the database are converted to instances of this type. In most cases you won't need to create `WKBElement` instances yourself.

If `extended` is `True` and `srid` is `-1` at construction time then the SRID will be read from the EWKB data.

Note: you can create `WKBElement` objects from Shapely geometries using the *geoalchemy2.shape.from_shape()* function.

**desc**
This element's description string.

**geom_from = 'ST_GeomFromWKB'**

**geom_from_extended_version = 'ST_GeomFromEWKB'**

**class** geoalchemy2.elements.**RasterElement**(*data*)
Bases: *geoalchemy2.elements._SpatialElement*

Instances of this class wrap a `raster` value. Raster values read from the database are converted to instances of this type. In most cases you won't need to create `RasterElement` instances yourself.

**desc**
This element's description string.

**class** geoalchemy2.elements.**CompositeElement**(*base*, *field*, *type_*)
Bases: `sqlalchemy.sql.functions.FunctionElement`

Instances of this class wrap a Postgres composite type.

## 7.3 Spatial Functions

This module defines the *GenericFunction* class, which is the base for the implementation of spatial functions in GeoAlchemy. This module is also where actual spatial functions are defined. Spatial functions supported by GeoAlchemy are defined in this module. See *GenericFunction* to know how to create new spatial functions.

---

**Note:** By convention the names of spatial functions are prefixed by `ST_`. This is to be consistent with PostGIS', which itself is based on the `SQL-MM` standard.

---

Functions created by subclassing *GenericFunction* can be called in several ways:

- By using the `func` object, which is the SQLAlchemy standard way of calling a function. For example, without the ORM:

```
select([func.ST_Area(lake_table.c.geom)])
```

and with the ORM:

```
Session.query(func.ST_Area(Lake.geom))
```

- By applying the function to a geometry column. For example, without the ORM:

```
select([lake_table.c.geom.ST_Area()])
```

and with the ORM:

```
Session.query(Lake.geom.ST_Area())
```

- By applying the function to a *geoalchemy2.elements.WKBElement* object (*geoalchemy2.elements.WKBElement* is the type into which GeoAlchemy converts geometry values read from the database), or to a *geoalchemy2.elements.WKTElement* object. For example, without the ORM:

```
conn.scalar(lake['geom'].ST_Area())
```

and with the ORM:

```
session.scalar(lake.geom.ST_Area())
```

**Warning:** A few functions (like *ST_Transform()*, *ST_Union()*, *ST_SnapToGrid()*, ...) can be used on several spatial types (*geoalchemy2.types.Geometry*, *geoalchemy2.types.Geography* and / or *geoalchemy2.types.Raster* types). In GeoAlchemy2, these functions are only defined for the *geoalchemy2.types.Geometry* type, as it can not be defined for several types at the same time. Therefore, using these functions on *geoalchemy2.types.Geography* or *geoalchemy2.types.Raster* requires minor tweaking to enforce the type by passing the *type_=Geography* or *type_=Raster* argument to the function:

```
s = select([func.ST_Transform(
                    lake_table.c.raster,
                    2154,
                    type_=Raster)
                .label('transformed_raster')])
```

### 7.3.1 Reference

**class** geoalchemy2.functions.**AddAuth**(**args*, ***kwargs*)
Adds an authorization token to be used in the current transaction.

see http://postgis.net/docs/AddAuth.html

**class** geoalchemy2.functions.**AddGeometryColumn**(**args*, ***kwargs*)
Adds a geometry column to an existing table.

see http://postgis.net/docs/AddGeometryColumn.html

**class** geoalchemy2.functions.**Box2D**(**args*, ***kwargs*)
Returns a BOX2D representing the 2D extent of the geometry.

see http://postgis.net/docs/Box2D_type.html

**class** geoalchemy2.functions.**Box3D**(**args*, ***kwargs*)
[geometry] Returns a BOX3D representing the 3D extent of the geometry. OR [raster] Returns the box 3d representation of the enclosing box of the raster.

see http://postgis.net/docs/Box3D_type.html

**class** geoalchemy2.functions.**CheckAuth**(**args*, ***kwargs*)
Creates a trigger on a table to prevent/allow updates and deletes of rows based on authorization token.

see http://postgis.net/docs/CheckAuth.html

**class** geoalchemy2.functions.**DisableLongTransactions**(**args*, ***kwargs*)
Disables long transaction support.

see http://postgis.net/docs/DisableLongTransactions.html

**class** geoalchemy2.functions.**DropGeometryColumn**(**args*, ***kwargs*)
Removes a geometry column from a spatial table.

see http://postgis.net/docs/DropGeometryColumn.html

**class** geoalchemy2.functions.**DropGeometryTable**(**args*, ***kwargs*)
Drops a table and all its references in geometry_columns.

see http://postgis.net/docs/DropGeometryTable.html

**class** geoalchemy2.functions.**EnableLongTransactions**(**args*, ***kwargs*)
Enables long transaction support.

see http://postgis.net/docs/EnableLongTransactions.html

**class** geoalchemy2.functions.**Find_SRID**(**args*, ***kwargs*)
Returns the SRID defined for a geometry column.

see http://postgis.net/docs/Find_SRID.html

**class** geoalchemy2.functions.**GenericFunction**(**args*, ***kwargs*)
The base class for GeoAlchemy functions.

This class inherits from `sqlalchemy.sql.functions.GenericFunction`, so functions defined by subclassing this class can be given a fixed return type. For example, functions like *`ST_Buffer`* and *`ST_Envelope`* have their `type` attributes set to *`geoalchemy2.types.Geometry`*.

This class allows constructs like `Lake.geom.ST_Buffer(2)`. In that case the `Function` instance is bound to an expression (`Lake.geom` here), and that expression is passed to the function when the function is actually called.

If you need to use a function that GeoAlchemy does not provide you will certainly want to subclass this class. For example, if you need the `ST_TransScale` spatial function, which isn't (currently) natively supported by GeoAlchemy, you will write this:

```
from geoalchemy2 import Geometry
from geoalchemy2.functions import GenericFunction

class ST_TransScale(GenericFunction):
    name = 'ST_TransScale'
    type = Geometry
```

**class** geoalchemy2.functions.**GeometryType**(**args*, ***kwargs*)
Returns the type of a geometry as text.

see http://postgis.net/docs/GeometryType.html

**class** geoalchemy2.functions.**LockRow**(**args*, ***kwargs*)
Sets lock/authorization for a row in a table.

see http://postgis.net/docs/LockRow.html

**class** geoalchemy2.functions.**Populate_Geometry_Columns**(**args*, ***kwargs*)
Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.

see http://postgis.net/docs/Populate_Geometry_Columns.html

**class** geoalchemy2.functions.**PostGIS_AddBBox**(**args*, ***kwargs*)
Add bounding box to the geometry.

see http://postgis.net/docs/PostGIS_AddBBox.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**PostGIS_DropBBox**(**args*, ***kwargs*)
Drop the bounding box cache from the geometry.

see http://postgis.net/docs/PostGIS_DropBBox.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**PostGIS_Extensions_Upgrade**(**args*, ***kwargs*)
Packages and upgrades postgis extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to latest available version.

see http://postgis.net/docs/PostGIS_Extensions_Upgrade.html

**class** geoalchemy2.functions.**PostGIS_Full_Version**(**args*, ***kwargs*)
Reports full postgis version and build configuration infos.

see http://postgis.net/docs/PostGIS_Full_Version.html

**class** geoalchemy2.functions.**PostGIS_GEOS_Version**(**args*, ***kwargs*)
Returns the version number of the GEOS library.

see http://postgis.net/docs/PostGIS_GEOS_Version.html

**class** geoalchemy2.functions.**PostGIS_HasBBox**(**args*, ***kwargs*)
Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.

see http://postgis.net/docs/PostGIS_HasBBox.html

**class** geoalchemy2.functions.**PostGIS_LibXML_Version**(**args*, ***kwargs*)
Returns the version number of the libxml2 library.

see http://postgis.net/docs/PostGIS_LibXML_Version.html

**class** geoalchemy2.functions.**PostGIS_Lib_Build_Date**(**args*, ***kwargs*)
Returns build date of the PostGIS library.

see http://postgis.net/docs/PostGIS_Lib_Build_Date.html

**class** geoalchemy2.functions.**PostGIS_Lib_Version**(**args*, ***kwargs*)
Returns the version number of the PostGIS library.

see http://postgis.net/docs/PostGIS_Lib_Version.html

**class** geoalchemy2.functions.**PostGIS_Liblwgeom_Version**(**args*, ***kwargs*)
Returns the version number of the liblwgeom library. This should match the version of PostGIS.

see http://postgis.net/docs/PostGIS_Liblwgeom_Version.html

**class** geoalchemy2.functions.**PostGIS_PROJ_Version**(**args*, ***kwargs*)
Returns the version number of the PROJ4 library.

see http://postgis.net/docs/PostGIS_PROJ_Version.html

**class** geoalchemy2.functions.**PostGIS_Scripts_Build_Date**(**args*, ***kwargs*)
Returns build date of the PostGIS scripts.

see http://postgis.net/docs/PostGIS_Scripts_Build_Date.html

**class** geoalchemy2.functions.**PostGIS_Scripts_Installed**(**args*, ***kwargs*)
Returns version of the postgis scripts installed in this database.

see http://postgis.net/docs/PostGIS_Scripts_Installed.html

**class** geoalchemy2.functions.**PostGIS_Scripts_Released**(**args*, ***kwargs*)
Returns the version number of the postgis.sql script released with the installed postgis lib.

see http://postgis.net/docs/PostGIS_Scripts_Released.html

**class** geoalchemy2.functions.**PostGIS_Version**(**args*, ***kwargs*)
Returns PostGIS version number and compile-time options.

see http://postgis.net/docs/PostGIS_Version.html

**class** geoalchemy2.functions.**PostGIS_Wagyu_Version**(**args*, ***kwargs*)
Returns the version number of the internal Wagyu library.

see http://postgis.net/docs/PostGIS_Wagyu_Version.html

**class** geoalchemy2.functions.**ST_3DArea**(**args*, ***kwargs*)
Computes area of 3D surface geometries. Will return 0 for solids.

see http://postgis.net/docs/ST_3DArea.html

**class** geoalchemy2.functions.**ST_3DClosestPoint**(**args*, ***kwargs*)
Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.

see http://postgis.net/docs/ST_3DClosestPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_3DDFullyWithin**(**args*, ***kwargs*)
Returns true if all of the 3D geometries are within the specified distance of one another.

see http://postgis.net/docs/ST_3DDFullyWithin.html

**class** geoalchemy2.functions.**ST_3DDWithin**(**args*, ***kwargs*)
For 3d (z) geometry type Returns true if two geometries 3d distance is within number of units.

see http://postgis.net/docs/ST_3DDWithin.html

**class** geoalchemy2.functions.**ST_3DDifference**(**args*, ***kwargs*)
Perform 3D difference

see http://postgis.net/docs/ST_3DDifference.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_3DDistance**(**args*, ***kwargs*)
Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.

see http://postgis.net/docs/ST_3DDistance.html

**class** geoalchemy2.functions.**ST_3DExtent**(**args*, ***kwargs*)
an aggregate function that returns the 3D bounding box that bounds rows of geometries.

see http://postgis.net/docs/ST_3DExtent.html

**class** geoalchemy2.functions.**ST_3DIntersection**(**args*, ***kwargs*)
Perform 3D intersection

see http://postgis.net/docs/ST_3DIntersection.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_3DIntersects**(**args*, ***kwargs*)
Returns TRUE if the Geometries "spatially intersect" in 3D - only for points, linestrings, polygons, polyhedral surface (area).

see http://postgis.net/docs/ST_3DIntersects.html

**class** geoalchemy2.functions.**ST_3DLength**(**args*, ***kwargs*)
Returns the 3D length of a linear geometry.

see http://postgis.net/docs/ST_3DLength.html

**class** geoalchemy2.functions.**ST_3DLineInterpolatePoint**(**args*, ***kwargs*)
Returns a point interpolated along a line in 3D. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.

see http://postgis.net/docs/ST_3DLineInterpolatePoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_3DLongestLine**(**args*, ***kwargs*)
Returns the 3D longest line between two geometries

see http://postgis.net/docs/ST_3DLongestLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_3DMakeBox**(**args*, ***kwargs*)
Creates a BOX3D defined by two 3D point geometries.

see http://postgis.net/docs/ST_3DMakeBox.html

**class** geoalchemy2.functions.**ST_3DMaxDistance**(**args*, ***kwargs*)
Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.

see http://postgis.net/docs/ST_3DMaxDistance.html

**class** geoalchemy2.functions.**ST_3DPerimeter**(**args*, ***kwargs*)
Returns the 3D perimeter of a polygonal geometry.

see http://postgis.net/docs/ST_3DPerimeter.html

**class** geoalchemy2.functions.**ST_3DShortestLine**(**args*, ***kwargs*)
Returns the 3D shortest line between two geometries

see http://postgis.net/docs/ST_3DShortestLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_3DUnion**(**args*, ***kwargs*)
Perform 3D union

see http://postgis.net/docs/ST_3DUnion.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_AddBand**(**args*, ***kwargs*)
Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.

see http://postgis.net/docs/RT_ST_AddBand.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_AddMeasure**(**args*, ***kwargs*)
Return a derived geometry with measure elements linearly interpolated between the start and end points.

see http://postgis.net/docs/ST_AddMeasure.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_AddPoint**(**args*, ***kwargs*)
Add a point to a LineString.

see http://postgis.net/docs/ST_AddPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Affine**(**args*, ***kwargs*)
Apply a 3D affine transformation to a geometry.

see http://postgis.net/docs/ST_Affine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Angle**(**args*, ***kwargs*)
Returns the angle between 3 points, or between 2 vectors (4 points or 2 lines).

see http://postgis.net/docs/ST_Angle.html

**class** geoalchemy2.functions.**ST_ApproximateMedialAxis**(**args*, ***kwargs*)
Compute the approximate medial axis of an areal geometry.

see http://postgis.net/docs/ST_ApproximateMedialAxis.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Area**(**args*, ***kwargs*)
Returns the area of a polygonal geometry.

see http://postgis.net/docs/ST_Area.html

**class** geoalchemy2.functions.**ST_AsBinary**(**args*, ***kwargs*)
[gometry] Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. OR [raster] Return the Well-Known Binary (WKB) representation of the raster.

see http://postgis.net/docs/ST_AsBinary.html

**class** geoalchemy2.functions.**ST_AsEWKB**(**args*, ***kwargs*)
Return the Well-Known Binary (WKB) representation of the geometry with SRID meta data.

see http://postgis.net/docs/ST_AsEWKB.html

**class** geoalchemy2.functions.**ST_AsEWKT**(**args*, ***kwargs*)
Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.

see http://postgis.net/docs/ST_AsEWKT.html

**class** geoalchemy2.functions.**ST_AsEncodedPolyline**(**args*, ***kwargs*)
Returns an Encoded Polyline from a LineString geometry.

see http://postgis.net/docs/ST_AsEncodedPolyline.html

**class** geoalchemy2.functions.**ST_AsGDALRaster**(**args*, ***kwargs*)
Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.

see http://postgis.net/docs/RT_ST_AsGDALRaster.html

**class** geoalchemy2.functions.**ST_AsGML**(**args*, ***kwargs*)
Return the geometry as a GML version 2 or 3 element.

see http://postgis.net/docs/ST_AsGML.html

**class** geoalchemy2.functions.**ST_AsGeoJSON**(**args*, ***kwargs*)
Return the geometry as a GeoJSON "geometry" object, or the row as a GeoJSON feature" object (PostGIS 3 only). (Cf GeoJSON specifications RFC 7946). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry types (no curve support for example). See https://postgis.net/docs/ST_AsGeoJSON.html

**class** geoalchemy2.functions.**ST_AsGeobuf**(**args*, ***kwargs*)
Return a Geobuf representation of a set of rows.

see http://postgis.net/docs/ST_AsGeobuf.html

**class** geoalchemy2.functions.**ST_AsHEXEWKB**(**args*, ***kwargs*)
Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.

see http://postgis.net/docs/ST_AsHEXEWKB.html

**class** geoalchemy2.functions.**ST_AsHexWKB**(**args*, ***kwargs*)
Return the Well-Known Binary (WKB) in Hex representation of the raster.

see http://postgis.net/docs/RT_ST_AsHexWKB.html

**class** geoalchemy2.functions.**ST_AsJPEG**(**args*, ***kwargs*)
Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If

no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.

see http://postgis.net/docs/RT_ST_AsJPEG.html

**class** geoalchemy2.functions.**ST_AsKML**(**args*, ***kwargs*)
Return the geometry as a KML element. Several variants. Default version=2, default maxdecimaldigits=15

see http://postgis.net/docs/ST_AsKML.html

**class** geoalchemy2.functions.**ST_AsLatLonText**(**args*, ***kwargs*)
Return the Degrees, Minutes, Seconds representation of the given point.

see http://postgis.net/docs/ST_AsLatLonText.html

**class** geoalchemy2.functions.**ST_AsMVT**(**args*, ***kwargs*)
Aggregate function returning a Mapbox Vector Tile representation of a set of rows.

see http://postgis.net/docs/ST_AsMVT.html

**class** geoalchemy2.functions.**ST_AsMVTGeom**(**args*, ***kwargs*)
Transform a geometry into the coordinate space of a Mapbox Vector Tile.

see http://postgis.net/docs/ST_AsMVTGeom.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_AsPNG**(**args*, ***kwargs*)
Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.

see http://postgis.net/docs/RT_ST_AsPNG.html

**class** geoalchemy2.functions.**ST_AsRaster**(**args*, ***kwargs*)
Converts a PostGIS geometry to a PostGIS raster.

see http://postgis.net/docs/RT_ST_AsRaster.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_AsSVG**(**args*, ***kwargs*)
Returns SVG path data for a geometry.

see http://postgis.net/docs/ST_AsSVG.html

**class** geoalchemy2.functions.**ST_AsTIFF**(**args*, ***kwargs*)
Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.

see http://postgis.net/docs/RT_ST_AsTIFF.html

**class** geoalchemy2.functions.**ST_AsTWKB**(**args*, ***kwargs*)
Returns the geometry as TWKB, aka "Tiny Well-Known Binary"

see http://postgis.net/docs/ST_AsTWKB.html

**class** geoalchemy2.functions.**ST_AsText**(**args*, ***kwargs*)
Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.

see http://postgis.net/docs/ST_AsText.html

**class** geoalchemy2.functions.**ST_AsWKB**(**args*, ***kwargs*)
Return the Well-Known Binary (WKB) representation of the raster.

see http://postgis.net/docs/RT_ST_AsBinary.html

**class** geoalchemy2.functions.**ST_AsX3D**(**args*, ***kwargs*)
Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML

see http://postgis.net/docs/ST_AsX3D.html

**class** geoalchemy2.functions.**ST_Aspect**(**args*, ***kwargs*)
Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

see http://postgis.net/docs/RT_ST_Aspect.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Azimuth**(**args*, ***kwargs*)
Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on pointA to pointB.

see http://postgis.net/docs/ST_Azimuth.html

**class** geoalchemy2.functions.**ST_Band**(**args*, ***kwargs*)
Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters.

see http://postgis.net/docs/RT_ST_Band.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_BandFileSize**(**args*, ***kwargs*)
Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.

see http://postgis.net/docs/RT_ST_BandFileSize.html

**class** geoalchemy2.functions.**ST_BandFileTimestamp**(**args*, ***kwargs*)
Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

see http://postgis.net/docs/RT_ST_BandFileTimestamp.html

**class** geoalchemy2.functions.**ST_BandIsNoData**(**args*, ***kwargs*)
Returns true if the band is filled with only nodata values.

see http://postgis.net/docs/RT_ST_BandIsNoData.html

**class** geoalchemy2.functions.**ST_BandMetaData**(**args*, ***kwargs*)
Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.

see http://postgis.net/docs/RT_ST_BandMetaData.html

**class** geoalchemy2.functions.**ST_BandNoDataValue**(**args*, ***kwargs*)
Returns the value in a given band that represents no data. If no band num 1 is assumed.

see http://postgis.net/docs/RT_ST_BandNoDataValue.html

**class** geoalchemy2.functions.**ST_BandPath**(**args*, ***kwargs*)
Returns system file path to a band stored in file system. If no bandnum specified, 1 is assumed.

see http://postgis.net/docs/RT_ST_BandPath.html

**class** geoalchemy2.functions.**ST_BandPixelType**(**args*, ***kwargs*)
Returns the type of pixel for given band. If no bandnum specified, 1 is assumed.

see http://postgis.net/docs/RT_ST_BandPixelType.html

**class** geoalchemy2.functions.**ST_BdMPolyFromText**(**args*, ***kwargs*)
Construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString text representation Well-Known text representation.

see http://postgis.net/docs/ST_BdMPolyFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_BdPolyFromText**(**args*, ***kwargs*)
Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.

see http://postgis.net/docs/ST_BdPolyFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Boundary**(**args*, ***kwargs*)
Returns the boundary of a geometry.

see http://postgis.net/docs/ST_Boundary.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_BoundingDiagonal**(**args*, ***kwargs*)
Returns the diagonal of a geometry's bounding box.

see http://postgis.net/docs/ST_BoundingDiagonal.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Box2dFromGeoHash**(**args*, ***kwargs*)
Return a BOX2D from a GeoHash string.

see http://postgis.net/docs/ST_Box2dFromGeoHash.html

**class** geoalchemy2.functions.**ST_Buffer**(**args*, ***kwargs*)

(T) Returns a geometry covering all points within a given distance from the input geometry.

see http://postgis.net/docs/ST_Buffer.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_BuildArea**(**args*, ***kwargs*)
Creates an areal geometry formed by the constituent linework of given geometry

see http://postgis.net/docs/ST_BuildArea.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_CPAWithin**(**args*, ***kwargs*)
Returns true if the closest point of approach of two trajectories is within the specified distance.

see http://postgis.net/docs/ST_CPAWithin.html

**class** geoalchemy2.functions.**ST_Centroid**(**args*, ***kwargs*)
Returns the geometric center of a geometry.

see http://postgis.net/docs/ST_Centroid.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ChaikinSmoothing**(**args*, ***kwargs*)
Returns a "smoothed" version of the given geometry using the Chaikin algorithm

see http://postgis.net/docs/ST_ChaikinSmoothing.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Clip**(**args*, ***kwargs*)
Returns the raster clipped by the input geometry. If band number not is specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped.

see http://postgis.net/docs/RT_ST_Clip.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_ClipByBox2D**(**args*, ***kwargs*)
Returns the portion of a geometry falling within a rectangle.

see http://postgis.net/docs/ST_ClipByBox2D.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ClosestPoint**(**args*, ***kwargs*)
Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line.

see http://postgis.net/docs/ST_ClosestPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ClosestPointOfApproach**(**args*, ***kwargs*)
Returns the measure at which points interpolated along two trajectories are closest.

see http://postgis.net/docs/ST_ClosestPointOfApproach.html

**class** geoalchemy2.functions.**ST_ClusterDBSCAN**(**args*, ***kwargs*)
Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

see http://postgis.net/docs/ST_ClusterDBSCAN.html

**class** geoalchemy2.functions.**ST_ClusterIntersecting**(**args*, ***kwargs*)
Aggregate function that clusters the input geometries into connected sets.

see http://postgis.net/docs/ST_ClusterIntersecting.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ClusterKMeans**(**args*, ***kwargs*)
Window function that returns a cluster id for each input geometry using the K-means algorithm.

see http://postgis.net/docs/ST_ClusterKMeans.html

**class** geoalchemy2.functions.**ST_ClusterWithin**(**args*, ***kwargs*)
Aggregate function that clusters the input geometries by separation distance.

see http://postgis.net/docs/ST_ClusterWithin.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Collect**(**args*, ***kwargs*)
Creates a GeometryCollection or Multi* geometry from a set of geometries.

see http://postgis.net/docs/ST_Collect.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_CollectionExtract**(**args*, ***kwargs*)
Given a (multi)geometry, return a (multi)geometry consisting only of elements of the specified type.

see http://postgis.net/docs/ST_CollectionExtract.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_CollectionHomogenize**(**args*, ***kwargs*)
Given a geometry collection, return the "simplest" representation of the contents.

see http://postgis.net/docs/ST_CollectionHomogenize.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ColorMap**(**args*, ***kwargs*)
Creates a new raster of up to four 8BUI bands (grayscale, RGB, RGBA) from the source raster and a specified band. Band 1 is assumed if not specified.

see http://postgis.net/docs/RT_ST_ColorMap.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_ConcaveHull**(**args*, ***kwargs*)
The concave hull of a geometry represents a possibly concave geometry that encloses all geometries within the set. You can think of it as shrink wrapping.

see http://postgis.net/docs/ST_ConcaveHull.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ConstrainedDelaunayTriangles**(**args*, ***kwargs*)
Return a constrained Delaunay triangulation around the given input geometry.

see http://postgis.net/docs/ST_ConstrainedDelaunayTriangles.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Contains**(**args*, ***kwargs*)
[geometry] Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A. OR [raster] Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.

see http://postgis.net/docs/ST_Contains.html

**class** geoalchemy2.functions.**ST_ContainsProperly**(**args*, ***kwargs*)
[geometry] Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself. OR [raster] Return true if rastB intersects the interior of rastA but not the boundary or exterior of rastA.

see http://postgis.net/docs/ST_ContainsProperly.html

**class** geoalchemy2.functions.**ST_ConvexHull**(**args*, ***kwargs*)
[geometry] Computes the convex hull of a geometry. OR [raster] Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular shaped and non-skewed rasters, this gives the same result as ST_Envelope so only useful for irregularly shaped or skewed rasters.

see http://postgis.net/docs/ST_ConvexHull.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_CoordDim**(**args*, ***kwargs*)
Return the coordinate dimension of a geometry.

see http://postgis.net/docs/ST_CoordDim.html

**class** geoalchemy2.functions.**ST_Count**(**args*, ***kwargs*)
Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the nodata value.

see http://postgis.net/docs/RT_ST_Count.html

**class** geoalchemy2.functions.**ST_CountAgg**(**args*, ***kwargs*)
Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If exclude_nodata_value is set to true, will only count pixels that are not equal to the NODATA value.

see http://postgis.net/docs/RT_ST_CountAgg.html

**class** geoalchemy2.functions.**ST_CoveredBy**(**args*, ***kwargs*)
[geometry] Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B OR [raster] Return true if no points of raster rastA lie outside raster rastB.

see http://postgis.net/docs/ST_CoveredBy.html

**class** geoalchemy2.functions.**ST_Covers**(**args*, ***kwargs*)
[geometry] Returns 1 (TRUE) if no point in Geometry B is outside Geometry A OR [raster] Return true if no points of raster rastB lie outside raster rastA.

see http://postgis.net/docs/ST_Covers.html

**class** geoalchemy2.functions.**ST_Crosses**(**args*, ***kwargs*)
Returns TRUE if the supplied geometries have some, but not all, interior points in common.

see http://postgis.net/docs/ST_Crosses.html

**class** geoalchemy2.functions.**ST_CurveToLine**(**args*, ***kwargs*)
Converts a CIRCULARSTRING/CURVEPOLYGON/MULTISURFACE to a LINESTRING/POLYGON/MULTIPOLYGON

see http://postgis.net/docs/ST_CurveToLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_DFullyWithin**(**args*, ***kwargs*)
[geometry] Returns true if all of the geometries are within the specified distance of one another OR [raster] Return true if rasters rastA and rastB are fully within the specified distance of each other.

see http://postgis.net/docs/ST_DFullyWithin.html

**class** geoalchemy2.functions.**ST_DWithin**(**args*, ***kwargs*)
[geometry] Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and for geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere. OR [raster] Return true if rasters rastA and rastB are within the specified distance of each other.

see http://postgis.net/docs/ST_DWithin.html

**class** geoalchemy2.functions.**ST_DelaunayTriangles**(**args*, ***kwargs*)
Return a Delaunay triangulation around the given input points.

see http://postgis.net/docs/ST_DelaunayTriangles.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Difference**(**args*, ***kwargs*)
Returns a geometry that represents that part of geometry A that does not intersect with geometry B.

see http://postgis.net/docs/ST_Difference.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Dimension**(**args*, ***kwargs*)
Returns the topological dimension of a geometry.

see http://postgis.net/docs/ST_Dimension.html

**class** geoalchemy2.functions.**ST_Disjoint**(**args*, ***kwargs*)
[geometry] Returns TRUE if the Geometries do not "spatially intersect" - if they do not share any space together. OR [raster] Return true if raster rastA does not spatially intersect rastB.

see http://postgis.net/docs/ST_Disjoint.html

**class** geoalchemy2.functions.**ST_Distance**(**args*, ***kwargs*)
Returns the distance between two geometry or geography values.

see http://postgis.net/docs/ST_Distance.html

**class** geoalchemy2.functions.**ST_DistanceCPA**(**args*, ***kwargs*)
Returns the distance between the closest point of approach of two trajectories.

see http://postgis.net/docs/ST_DistanceCPA.html

**class** geoalchemy2.functions.**ST_DistanceSphere**(**args*, ***kwargs*)
Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.

see http://postgis.net/docs/ST_DistanceSphere.html

**class** geoalchemy2.functions.**ST_DistanceSpheroid**(**args*, ***kwargs*)
Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.

see http://postgis.net/docs/ST_DistanceSpheroid.html

**class** geoalchemy2.functions.**ST_Distance_Sphere**(**args*, ***kwargs*)
Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius of 6370986 meters. Faster than `ST_Distance_Spheroid`, but less accurate. PostGIS versions prior to 1.5 only implemented for points.

see http://postgis.net/docs/ST_Distance_Sphere.html

**class** geoalchemy2.functions.**ST_Distinct4ma**(**args*, ***kwargs*)
Raster processing function that calculates the number of unique pixel values in a neighborhood.

see http://postgis.net/docs/RT_ST_Distinct4ma.html

**class** geoalchemy2.functions.**ST_Dump**(**args*, ***kwargs*)
Returns a set of geometry_dump rows for the components of a geometry.

see http://postgis.net/docs/ST_Dump.html

Return type: *geoalchemy2.types.GeometryDump*.

**type**
alias of *geoalchemy2.types.GeometryDump*

**class** geoalchemy2.functions.**ST_DumpAsPolygons**(**args*, ***kwargs*)
Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.

see http://postgis.net/docs/RT_ST_DumpAsPolygons.html

**class** geoalchemy2.functions.**ST_DumpPoints**(**args*, ***kwargs*)
Returns a set of geometry_dump rows for the points in a geometry.

see http://postgis.net/docs/ST_DumpPoints.html

Return type: *geoalchemy2.types.GeometryDump*.

**type**
alias of *geoalchemy2.types.GeometryDump*

**class** geoalchemy2.functions.**ST_DumpRings**(**args*, ***kwargs*)
Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.

see http://postgis.net/docs/ST_DumpRings.html

Return type: *geoalchemy2.types.GeometryDump*.

**type**
alias of *geoalchemy2.types.GeometryDump*

**class** geoalchemy2.functions.**ST_DumpValues**(**args*, ***kwargs*)
Get the values of the specified band as a 2-dimension array.

see http://postgis.net/docs/RT_ST_DumpValues.html

**class** geoalchemy2.functions.**ST_EndPoint**(**args*, ***kwargs*)
Returns the last point of a LineString or CircularLineString.

see http://postgis.net/docs/ST_EndPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Envelope**(**args*, ***kwargs*)
[geometry] Returns a geometry representing the bounding box of a geometry. OR [raster] Returns the polygon representation of the extent of the raster.

see http://postgis.net/docs/ST_Envelope.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Equals**(**args*, ***kwargs*)
Returns true if the given geometries represent the same geometry. Directionality is ignored.

see http://postgis.net/docs/ST_Equals.html

**class** geoalchemy2.functions.**ST_EstimatedExtent**(**args*, ***kwargs*)
Return the 'estimated' extent of a spatial table.

see http://postgis.net/docs/ST_EstimatedExtent.html

**class** geoalchemy2.functions.**ST_Expand**(**args*, ***kwargs*)
Returns a bounding box expanded from another bounding box or a geometry.

see http://postgis.net/docs/ST_Expand.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Extent**(**args*, ***kwargs*)
an aggregate function that returns the bounding box that bounds rows of geometries.

see http://postgis.net/docs/ST_Extent.html

**class** geoalchemy2.functions.**ST_ExteriorRing**(**args*, ***kwargs*)
Returns a LineString representing the exterior ring of a Polygon.

see http://postgis.net/docs/ST_ExteriorRing.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Extrude**(**args*, ***kwargs*)
Extrude a surface to a related volume

see http://postgis.net/docs/ST_Extrude.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_FilterByM**(**args*, ***kwargs*)
Filters vertex points based on their m-value

see http://postgis.net/docs/ST_FilterByM.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_FlipCoordinates**(**args*, ***kwargs*)
Returns a version of the given geometry with X and Y axis flipped. Useful for people who have built latitude/longitude features and need to fix them.

see http://postgis.net/docs/ST_FlipCoordinates.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Force2D**(**args*, ***kwargs*)
Force the geometries into a "2-dimensional mode".

see http://postgis.net/docs/ST_Force2D.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Force3D**(**args*, ***kwargs*)
Force the geometries into XYZ mode. This is an alias for ST_Force3DZ.

see http://postgis.net/docs/ST_Force_3D.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Force3DM**(**args*, ***kwargs*)
Force the geometries into XYM mode.

see http://postgis.net/docs/ST_Force_3DZ.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Force3DZ**(**args*, ***kwargs*)
Force the geometries into XYZ mode.

see http://postgis.net/docs/ST_Force_3DZ.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Force4D**(**args*, ***kwargs*)
Force the geometries into XYZM mode.

see http://postgis.net/docs/ST_Force_4D.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForceCollection**(**args*, ***kwargs*)
Convert the geometry into a GEOMETRYCOLLECTION.

see http://postgis.net/docs/ST_Force_Collection.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForceCurve**(**args*, ***kwargs*)
Upcast a geometry into its curved type, if applicable.

see http://postgis.net/docs/ST_ForceCurve.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForceLHR**(**args*, ***kwargs*)
Force LHR orientation

see http://postgis.net/docs/ST_ForceLHR.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForcePolygonCCW**(**args*, ***kwargs*)
Orients all exterior rings counter-clockwise and all interior rings clockwise.

see http://postgis.net/docs/ST_ForcePolygonCCW.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForcePolygonCW**(**args*, ***kwargs*)
Orients all exterior rings clockwise and all interior rings counter-clockwise.

see http://postgis.net/docs/ST_ForcePolygonCW.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForceRHR**(**args*, ***kwargs*)
Force the orientation of the vertices in a polygon to follow the Right-Hand-Rule.

see http://postgis.net/docs/ST_ForceRHR.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ForceSFS**(**args*, ***kwargs*)
Force the geometries to use SFS 1.1 geometry types only.

see http://postgis.net/docs/ST_ForceSFS.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_FrechetDistance**(**args*, ***kwargs*)
Returns the Fréchet distance between two geometries.

see http://postgis.net/docs/ST_FrechetDistance.html

**class** geoalchemy2.functions.**ST_FromGDALRaster**(**args*, ***kwargs*)
Returns a raster from a supported GDAL raster file.

see http://postgis.net/docs/RT_ST_FromGDALRaster.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_GMLToSQL**(**args*, ***kwargs*)
Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML

see http://postgis.net/docs/ST_GMLToSQL.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeneratePoints**(**args*, ***kwargs*)
Converts a polygon or multi-polygon into a multi-point composed of randomly location points within the original areas.

see http://postgis.net/docs/ST_GeneratePoints.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeoHash**(**args*, ***kwargs*)
Return a GeoHash representation of the geometry.

see http://postgis.net/docs/ST_GeoHash.html

**class** geoalchemy2.functions.**ST_GeoReference**(**args*, ***kwargs*)
Returns the georeference meta data in GDAL or ESRI format as commonly seen in a world file. Default is GDAL.

see http://postgis.net/docs/RT_ST_GeoReference.html

**class** geoalchemy2.functions.**ST_GeogFromText**(**args*, ***kwargs*)
Return a specified geography value from Well-Known Text representation or extended (WKT).

see http://postgis.net/docs/ST_GeogFromText.html

Return type: *geoalchemy2.types.Geography*.

**type**
alias of *geoalchemy2.types.Geography*

**class** geoalchemy2.functions.**ST_GeogFromWKB**(**args*, ***kwargs*)
Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).

see http://postgis.net/docs/ST_GeogFromWKB.html

Return type: *geoalchemy2.types.Geography*.

**type**
alias of *geoalchemy2.types.Geography*

**class** geoalchemy2.functions.**ST_GeographyFromText**(**args*, ***kwargs*)
Return a specified geography value from Well-Known Text representation or extended (WKT).

see http://postgis.net/docs/ST_GeographyFromText.html

Return type: *geoalchemy2.types.Geography*.

**type**
alias of *geoalchemy2.types.Geography*

**class** geoalchemy2.functions.**ST_GeomCollFromText**(**args*, ***kwargs*)
Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.

see http://postgis.net/docs/ST_GeomCollFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromEWKB**(**args*, ***kwargs*)
Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).

see http://postgis.net/docs/ST_GeomFromEWKB.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromEWKT**(**args*, ***kwargs*)
Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).

see http://postgis.net/docs/ST_GeomFromEWKT.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromGML**(**args*, ***kwargs*)
Takes as input GML representation of geometry and outputs a PostGIS geometry object

see http://postgis.net/docs/ST_GeomFromGML.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromGeoHash**(**args*, ***kwargs*)
Return a geometry from a GeoHash string.

see http://postgis.net/docs/ST_GeomFromGeoHash.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromGeoJSON**(**args*, ***kwargs*)
Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object

see http://postgis.net/docs/ST_GeomFromGeoJSON.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromKML**(**args*, ***kwargs*)
Takes as input KML representation of geometry and outputs a PostGIS geometry object

see http://postgis.net/docs/ST_GeomFromKML.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromTWKB**(**args*, ***kwargs*)
Creates a geometry instance from a TWKB ("Tiny Well-Known Binary") geometry representation.

see http://postgis.net/docs/ST_GeomFromTWKB.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromText**(**args*, ***kwargs*)
Return a specified ST_Geometry value from Well-Known Text representation (WKT).

see http://postgis.net/docs/ST_GeomFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeomFromWKB**(**args*, ***kwargs*)
Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.

see http://postgis.net/docs/ST_GeomFromWKB.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeometricMedian**(**args*, ***kwargs*)
Returns the geometric median of a MultiPoint.

see http://postgis.net/docs/ST_GeometricMedian.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeometryFromText**(**args*, ***kwargs*)
Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

see http://postgis.net/docs/ST_GeometryFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeometryN**(**args*, ***kwargs*)
Return the Nth geometry element of a geometry collection.

see http://postgis.net/docs/ST_GeometryN.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_GeometryType**(**args*, ***kwargs*)
Returns the SQL-MM type of a geometry as text.

see http://postgis.net/docs/ST_GeometryType.html

**class** geoalchemy2.functions.**ST_Grayscale**(**args*, ***kwargs*)
Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

see http://postgis.net/docs/RT_ST_Grayscale.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_HasArc**(**args*, ***kwargs*)
Tests if a geometry contains a circular arc

see http://postgis.net/docs/ST_HasArc.html

**class** geoalchemy2.functions.**ST_HasNoBand**(**args*, ***kwargs*)
Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.

see http://postgis.net/docs/RT_ST_HasNoBand.html

**class** geoalchemy2.functions.**ST_HausdorffDistance**(**args*, ***kwargs*)
Returns the Hausdorff distance between two geometries.

see http://postgis.net/docs/ST_HausdorffDistance.html

**class** geoalchemy2.functions.**ST_Height**(**args*, ***kwargs*)
Returns the height of the raster in pixels.

see http://postgis.net/docs/RT_ST_Height.html

**class** geoalchemy2.functions.**ST_HillShade**(**args*, ***kwargs*)
Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.

see http://postgis.net/docs/RT_ST_HillShade.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Histogram**(**args*, ***kwargs*)
Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.

see http://postgis.net/docs/RT_ST_Histogram.html

**class** geoalchemy2.functions.**ST_InteriorRingN**(**args*, ***kwargs*)
Returns the Nth interior ring (hole) of a Polygon.

see http://postgis.net/docs/ST_InteriorRingN.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_InterpolatePoint**(**args*, ***kwargs*)
Return the value of the measure dimension of a geometry at the point closed to the provided point.

see http://postgis.net/docs/ST_InterpolatePoint.html

**class** geoalchemy2.functions.**ST_Intersection**(**args*, ***kwargs*)
[geometry] (T) Returns a geometry that represents the shared portion of geomA and geomB. OR [raster] Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.

see http://postgis.net/docs/ST_Intersection.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Intersects**(**args*, ***kwargs*)
[geometry] Returns TRUE if the Geometries/Geography "spatially intersect in 2D" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography tolerance is 0.00001 meters (so any points that close are considered to intersect) OR [raster] Return true if raster rastA spatially intersects raster rastB.

see http://postgis.net/docs/ST_Intersects.html

**class** geoalchemy2.functions.**ST_InvDistWeight4ma**(**args*, ***kwargs*)
Raster processing function that interpolates a pixel's value from the pixel's neighborhood.

see http://postgis.net/docs/RT_ST_InvDistWeight4ma.html

**class** geoalchemy2.functions.**ST_IsClosed**(**args*, ***kwargs*)
Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).

see http://postgis.net/docs/ST_IsClosed.html

**class** geoalchemy2.functions.**ST_IsCollection**(**args*, ***kwargs*)
Tests if a geometry is a geometry collection type.

see http://postgis.net/docs/ST_IsCollection.html

**class** geoalchemy2.functions.**ST_IsEmpty**(**args*, ***kwargs*)
[geometry] Tests if a geometry is empty. OR [raster] Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.

see http://postgis.net/docs/ST_IsEmpty.html

**class** geoalchemy2.functions.**ST_IsPlanar**(**args*, ***kwargs*)
Check if a surface is or not planar

see http://postgis.net/docs/ST_IsPlanar.html

**class** geoalchemy2.functions.**ST_IsPolygonCCW**(**args*, ***kwargs*)
Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.

see http://postgis.net/docs/ST_IsPolygonCCW.html

**class** geoalchemy2.functions.**ST_IsPolygonCW**(**args*, ***kwargs*)
Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

see http://postgis.net/docs/ST_IsPolygonCW.html

**class** geoalchemy2.functions.**ST_IsRing**(**args*, ***kwargs*)
Tests if a LineString is closed and simple.

see http://postgis.net/docs/ST_IsRing.html

**class** geoalchemy2.functions.**ST_IsSimple**(**args*, ***kwargs*)
Tests if a geometry has no points of self-intersection or self-tangency.

see http://postgis.net/docs/ST_IsSimple.html

**class** geoalchemy2.functions.**ST_IsSolid**(**args*, ***kwargs*)
Test if the geometry is a solid. No validity check is performed.

see http://postgis.net/docs/ST_IsSolid.html

**class** geoalchemy2.functions.**ST_IsValid**(**args*, ***kwargs*)
Tests if a geometry is well-formed in 2D.

see http://postgis.net/docs/ST_IsValid.html

**class** geoalchemy2.functions.**ST_IsValidDetail**(**args*, ***kwargs*)
Returns a valid_detail row stating if a geometry is valid, and if not a reason why and a location.

see http://postgis.net/docs/ST_IsValidDetail.html

**class** geoalchemy2.functions.**ST_IsValidReason**(**args*, ***kwargs*)
Returns text stating if a geometry is valid, or a reason for invalidity.

see http://postgis.net/docs/ST_IsValidReason.html

**class** geoalchemy2.functions.**ST_IsValidTrajectory**(**args*, ***kwargs*)
Returns true if the geometry is a valid trajectory.

see http://postgis.net/docs/ST_IsValidTrajectory.html

**class** geoalchemy2.functions.**ST_Length**(**args*, ***kwargs*)
Returns the 2D length of a linear geometry.

see http://postgis.net/docs/ST_Length.html

**class** geoalchemy2.functions.**ST_Length2D**(**args*, ***kwargs*)
Returns the 2D length of a linear geometry. Alias for ST_Length

see http://postgis.net/docs/ST_Length2D.html

**class** geoalchemy2.functions.**ST_LengthSpheroid**(**args*, ***kwargs*)
Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.

see http://postgis.net/docs/ST_LengthSpheroid.html

**class** geoalchemy2.functions.**ST_LineCrossingDirection**(**args*, ***kwargs*)
Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing.

see http://postgis.net/docs/ST_LineCrossingDirection.html

**class** geoalchemy2.functions.**ST_LineFromEncodedPolyline**(**args*, ***kwargs*)
Creates a LineString from an Encoded Polyline.

see http://postgis.net/docs/ST_LineFromEncodedPolyline.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineFromMultiPoint**(**args*, ***kwargs*)
Creates a LineString from a MultiPoint geometry.

see http://postgis.net/docs/ST_LineFromMultiPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineFromText**(**args*, ***kwargs*)
Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to 0.

see http://postgis.net/docs/ST_LineFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineFromWKB**(**args*, ***kwargs*)
Makes a LINESTRING from WKB with the given SRID

see http://postgis.net/docs/ST_LineFromWKB.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineInterpolatePoint**(**args*, ***kwargs*)
Returns a point interpolated along a line. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.

see http://postgis.net/docs/ST_LineInterpolatePoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineInterpolatePoints**(**args*, ***kwargs*)
Returns one or more points interpolated along a line.

see http://postgis.net/docs/ST_LineInterpolatePoints.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineLocatePoint**(**args*, ***kwargs*)
Returns a float between 0 and 1 representing the location of the closest point on LineString to the given Point, as a fraction of total 2d line length.

see http://postgis.net/docs/ST_LineLocatePoint.html

**class** geoalchemy2.functions.**ST_LineMerge**(**args*, ***kwargs*)
Return a (set of) LineString(s) formed by sewing together a MULTILINESTRING.

see http://postgis.net/docs/ST_LineMerge.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineSubstring**(**args*, ***kwargs*)
Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1.

see http://postgis.net/docs/ST_LineSubstring.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LineToCurve**(**args*, ***kwargs*)
Converts a LINESTRING/POLYGON to a CIRCULARSTRING, CURVEPOLYGON

see http://postgis.net/docs/ST_LineToCurve.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LinestringFromWKB**(**args*, ***kwargs*)
Makes a geometry from WKB with the given SRID.

see http://postgis.net/docs/ST_LinestringFromWKB.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LocateAlong**(**args*, ***kwargs*)
Return a derived geometry collection value with elements that match the specified measure. Polygonal elements are not supported.

see http://postgis.net/docs/ST_LocateAlong.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LocateBetween**(**args*, ***kwargs*)
Return a derived geometry collection value with elements that match the specified range of measures inclusively.

see http://postgis.net/docs/ST_LocateBetween.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LocateBetweenElevations**(**args*, ***kwargs*)
Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively.

see http://postgis.net/docs/ST_LocateBetweenElevations.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_LongestLine**(**args*, ***kwargs*)
Returns the 2D longest line between two geometries.

see http://postgis.net/docs/ST_LongestLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_M**(**args*, ***kwargs*)
Returns the M coordinate of a Point.

see http://postgis.net/docs/ST_M.html

**class** geoalchemy2.functions.**ST_MLineFromText**(**args*, ***kwargs*)
Return a specified ST_MultiLineString value from WKT representation.

see http://postgis.net/docs/ST_MLineFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MPointFromText**(**args*, ***kwargs*)
Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

see http://postgis.net/docs/ST_MPointFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MPolyFromText**(**args*, ***kwargs*)
Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

see http://postgis.net/docs/ST_MPolyFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakeBox2D**(**args*, ***kwargs*)
Creates a BOX2D defined by two 2D point geometries.

see http://postgis.net/docs/ST_MakeBox2D.html

**class** geoalchemy2.functions.**ST_MakeEmptyCoverage**(**args*, ***kwargs*)
Cover georeferenced area with a grid of empty raster tiles.

see http://postgis.net/docs/RT_ST_MakeEmptyCoverage.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_MakeEmptyRaster**(**args*, ***kwargs*)
Returns an empty raster (having no bands) of given dimensions (width & height), upperleft X and Y, pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid). If a raster is passed in, returns a new raster with the same size, alignment and SRID. If srid is left out, the spatial ref is set to unknown (0).

see http://postgis.net/docs/RT_ST_MakeEmptyRaster.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_MakeEnvelope**(**args*, ***kwargs*)
Creates a rectangular Polygon from minimum and maximum coordinates.

see http://postgis.net/docs/ST_MakeEnvelope.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakeLine**(**args*, ***kwargs*)
Creates a Linestring from Point, MultiPoint, or LineString geometries.

see http://postgis.net/docs/ST_MakeLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakePoint**(**args*, ***kwargs*)
Creates a 2D, 3DZ or 4D Point.

see http://postgis.net/docs/ST_MakePoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakePointM**(**args*, ***kwargs*)
Creates a Point from X, Y and M values.

see http://postgis.net/docs/ST_MakePointM.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakePolygon**(**args*, ***kwargs*)
Creates a Polygon from a shell and optional list of holes.

see http://postgis.net/docs/ST_MakePolygon.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakeSolid**(**args*, ***kwargs*)
Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.

see http://postgis.net/docs/ST_MakeSolid.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MakeValid**(**args*, ***kwargs*)
Attempts to make an invalid geometry valid without losing vertices.

see http://postgis.net/docs/ST_MakeValid.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MapAlgebra**(**args*, ***kwargs*)
[raster] Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function. OR [raster] Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

see http://postgis.net/docs/RT_ST_MapAlgebra.html

**class** geoalchemy2.functions.**ST_MapAlgebraExpr**(**args*, ***kwargs*)
[raster] 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified. OR [raster] 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.

see http://postgis.net/docs/RT_ST_MapAlgebraExpr.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_MapAlgebraFct**(**args*, ***kwargs*)
[raster] 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype prodived. Band 1 is assumed if no band is specified. OR [raster] 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype prodived. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.

see http://postgis.net/docs/RT_ST_MapAlgebraFct.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_MapAlgebraFctNgb**(**args*, ***kwargs*)
1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.

see http://postgis.net/docs/RT_ST_MapAlgebraFctNgb.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Max4ma**(**args*, ***kwargs*)
Raster processing function that calculates the maximum pixel value in a neighborhood.

see http://postgis.net/docs/RT_ST_Max4ma.html

**class** geoalchemy2.functions.**ST_MaxDistance**(**args*, ***kwargs*)
Returns the 2D largest distance between two geometries in projected units.

see http://postgis.net/docs/ST_MaxDistance.html

**class** geoalchemy2.functions.**ST_Mean4ma**(**args*, ***kwargs*)
Raster processing function that calculates the mean pixel value in a neighborhood.

see http://postgis.net/docs/RT_ST_Mean4ma.html

**class** geoalchemy2.functions.**ST_MemSize**(**args*, ***kwargs*)
[geometry] Returns the amount of memory space a geometry takes. OR [raster] Returns the amount of space (in bytes) the raster takes.

see http://postgis.net/docs/ST_MemSize.html

**class** geoalchemy2.functions.**ST_MemUnion**(**args*, ***kwargs*)
Same as ST_Union, only memory-friendly (uses less memory and more processor time).

see http://postgis.net/docs/ST_MemUnion.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MetaData**(**args*, ***kwargs*)
Returns basic meta data about a raster object such as pixel size, rotation (skew), upper, lower left, etc.

see http://postgis.net/docs/RT_ST_MetaData.html

**class** geoalchemy2.functions.**ST_Min4ma**(**args*, ***kwargs*)
Raster processing function that calculates the minimum pixel value in a neighborhood.

see http://postgis.net/docs/RT_ST_Min4ma.html

**class** geoalchemy2.functions.**ST_MinConvexHull**(**args*, ***kwargs*)
Return the convex hull geometry of the raster excluding NODATA pixels.

see http://postgis.net/docs/RT_ST_MinConvexHull.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MinDist4ma**(**args*, ***kwargs*)
Raster processing function that returns the minimum distance (in number of pixels) between the pixel of interest and a neighboring pixel with value.

see http://postgis.net/docs/RT_ST_MinDist4ma.html

**class** geoalchemy2.functions.**ST_MinPossibleValue**(**args*, ***kwargs*)
Returns the minimum value this pixeltype can store.

see http://postgis.net/docs/ST_MinPossibleValue.html

**class** geoalchemy2.functions.**ST_MinimumBoundingCircle**(**args*, ***kwargs*)
Returns the smallest circle polygon that can fully contain a geometry. Default uses 48 segments per quarter circle.

see http://postgis.net/docs/ST_MinimumBoundingCircle.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MinimumBoundingRadius**(**args*, ***kwargs*)
Returns the center point and radius of the smallest circle that can fully contain a geometry.

see http://postgis.net/docs/ST_MinimumBoundingRadius.html

**class** geoalchemy2.functions.**ST_MinimumClearance**(**args*, ***kwargs*)
Returns the minimum clearance of a geometry, a measure of a geometry's robustness.

see http://postgis.net/docs/ST_MinimumClearance.html

**class** geoalchemy2.functions.**ST_MinimumClearanceLine**(**args*, ***kwargs*)
Returns the two-point LineString spanning a geometry's minimum clearance.

see http://postgis.net/docs/ST_MinimumClearanceLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_MinkowskiSum**(**args*, ***kwargs*)
Performs Minkowski sum

see http://postgis.net/docs/ST_MinkowskiSum.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Multi**(**args*, ***kwargs*)
Return the geometry as a MULTI* geometry.

see http://postgis.net/docs/ST_Multi.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_NDims**(**args*, ***kwargs*)
Returns the coordinate dimension of a geometry.

see http://postgis.net/docs/ST_NDims.html

**class** geoalchemy2.functions.**ST_NPoints**(**args*, ***kwargs*)
Returns the number of points (vertices) in a geometry.

see http://postgis.net/docs/ST_NPoints.html

**class** geoalchemy2.functions.**ST_NRings**(**args*, ***kwargs*)
Returns the number of rings in a polygonal geometry.

see http://postgis.net/docs/ST_NRings.html

**class** geoalchemy2.functions.**ST_NearestValue**(**args*, ***kwargs*)
Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.

see http://postgis.net/docs/RT_ST_NearestValue.html

**class** geoalchemy2.functions.**ST_Neighborhood**(**args*, ***kwargs*)
Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.

see http://postgis.net/docs/RT_ST_Neighborhood.html

**class** geoalchemy2.functions.**ST_Node**(**args*, ***kwargs*)
Node a set of linestrings.

see http://postgis.net/docs/ST_Node.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Normalize**(**args*, ***kwargs*)
Return the geometry in its canonical form.

see http://postgis.net/docs/ST_Normalize.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_NotSameAlignmentReason**(**args*, ***kwargs*)
Returns text stating if rasters are aligned and if not aligned, a reason why.

see http://postgis.net/docs/RT_ST_NotSameAlignmentReason.html

**class** geoalchemy2.functions.**ST_NumBands**(**args*, ***kwargs*)
Returns the number of bands in the raster object.

see http://postgis.net/docs/RT_ST_NumBands.html

**class** geoalchemy2.functions.**ST_NumGeometries**(**args*, ***kwargs*)
Returns the number of elements in a geometry collection.

see http://postgis.net/docs/ST_NumGeometries.html

**class** geoalchemy2.functions.**ST_NumInteriorRing**(**args*, ***kwargs*)
Returns the number of interior rings (holes) of a Polygon. Aias for ST_NumInteriorRings

see http://postgis.net/docs/ST_NumInteriorRing.html

**class** geoalchemy2.functions.**ST_NumInteriorRings**(**args*, ***kwargs*)
Returns the number of interior rings (holes) of a Polygon.

see http://postgis.net/docs/ST_NumInteriorRings.html

**class** geoalchemy2.functions.**ST_NumPatches**(**args*, ***kwargs*)
Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.

see http://postgis.net/docs/ST_NumPatches.html

**class** geoalchemy2.functions.**ST_NumPoints**(**args*, ***kwargs*)
Returns the number of points in a LineString or CircularString.

see http://postgis.net/docs/ST_NumPoints.html

**class** geoalchemy2.functions.**ST_OffsetCurve**(**args*, ***kwargs*)
Return an offset line at a given distance and side from an input line. Useful for computing parallel lines about a center line

see http://postgis.net/docs/ST_OffsetCurve.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_OrderingEquals**(**args*, ***kwargs*)
Returns true if the given geometries represent the same geometry and points are in the same directional order.

see http://postgis.net/docs/ST_OrderingEquals.html

**class** geoalchemy2.functions.**ST_Orientation**(**args*, ***kwargs*)
Determine surface orientation

see http://postgis.net/docs/ST_Orientation.html

**class** geoalchemy2.functions.**ST_OrientedEnvelope**(**args*, ***kwargs*)
Returns a minimum rotated rectangle enclosing a geometry.

see http://postgis.net/docs/ST_OrientedEnvelope.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Overlaps**(**args*, ***kwargs*)
[geometry] Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other. OR [raster] Return true if raster rastA and rastB intersect but one does not completely contain the other.

see http://postgis.net/docs/ST_Overlaps.html

**class** geoalchemy2.functions.**ST_PatchN**(**args*, ***kwargs*)
Returns the Nth geometry (face) of a PolyhedralSurface.

see http://postgis.net/docs/ST_PatchN.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Perimeter**(**args*, ***kwargs*)
Returns the length of the boundary of a polygonal geometry or geography.

see http://postgis.net/docs/ST_Perimeter.html

**class** geoalchemy2.functions.**ST_Perimeter2D**(**args*, ***kwargs*)
Returns the 2D perimeter of a polygonal geometry. Alias for ST_Perimeter.

see http://postgis.net/docs/ST_Perimeter2D.html

**class** geoalchemy2.functions.**ST_PixelAsCentroid**(**args*, ***kwargs*)
Returns the centroid (point geometry) of the area represented by a pixel.

see http://postgis.net/docs/RT_ST_PixelAsCentroid.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PixelAsCentroids**(**args*, ***kwargs*)
Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.

see http://postgis.net/docs/RT_ST_PixelAsCentroids.html

**class** geoalchemy2.functions.**ST_PixelAsPoint**(**args*, ***kwargs*)
Returns a point geometry of the pixel's upper-left corner.

see http://postgis.net/docs/RT_ST_PixelAsPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PixelAsPoints**(**args*, ***kwargs*)
Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.

see http://postgis.net/docs/RT_ST_PixelAsPoints.html

**class** geoalchemy2.functions.**ST_PixelAsPolygon**(**args*, ***kwargs*)
Returns the polygon geometry that bounds the pixel for a particular row and column.

see http://postgis.net/docs/RT_ST_PixelAsPolygon.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PixelAsPolygons**(**args*, ***kwargs*)
Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.

see http://postgis.net/docs/RT_ST_PixelAsPolygons.html

**class** geoalchemy2.functions.**ST_PixelHeight**(**args*, ***kwargs*)
Returns the pixel height in geometric units of the spatial reference system.

see http://postgis.net/docs/RT_ST_PixelHeight.html

**class** geoalchemy2.functions.**ST_PixelOfValue**(**args*, ***kwargs*)
Get the columnx, rowy coordinates of the pixel whose value equals the search value.

see http://postgis.net/docs/RT_ST_PixelOfValue.html

**class** geoalchemy2.functions.**ST_PixelWidth**(**args*, ***kwargs*)
Returns the pixel width in geometric units of the spatial reference system.

see http://postgis.net/docs/RT_ST_PixelWidth.html

**class** geoalchemy2.functions.**ST_Point**(**args*, ***kwargs*)
Creates a Point with the given coordinate values. Alias for ST_MakePoint.

see http://postgis.net/docs/ST_Point.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PointFromGeoHash**(**args*, ***kwargs*)
Return a point from a GeoHash string.

see http://postgis.net/docs/ST_PointFromGeoHash.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PointFromText**(**args*, ***kwargs*)
Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown.

see http://postgis.net/docs/ST_PointFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PointFromWKB**(**args*, ***kwargs*)
Makes a geometry from WKB with the given SRID

see http://postgis.net/docs/ST_PointFromWKB.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PointInsideCircle**(**args*, ***kwargs*)
Is the point geometry inside the circle defined by center_x, center_y, radius

see http://postgis.net/docs/ST_PointInsideCircle.html

**class** geoalchemy2.functions.**ST_PointN**(**args*, ***kwargs*)
Returns the Nth point in the first LineString or circular LineString in a geometry.

see http://postgis.net/docs/ST_PointN.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PointOnSurface**(**args*, ***kwargs*)
Returns a POINT guaranteed to lie on the surface.

see http://postgis.net/docs/ST_PointOnSurface.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Points**(**args*, ***kwargs*)
Returns a MultiPoint containing all the coordinates of a geometry.

see http://postgis.net/docs/ST_Points.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Polygon**(**args*, ***kwargs*)
[geometry] Creates a Polygon from a LineString with a specified SRID. OR [raster] Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.

see http://postgis.net/docs/ST_Polygon.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_PolygonFromText**(**args*, ***kwargs*)
Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

see http://postgis.net/docs/ST_PolygonFromText.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Polygonize**(**args*, ***kwargs*)
Aggregate. Creates a GeometryCollection containing possible polygons formed from the constituent linework of a set of geometries.

see http://postgis.net/docs/ST_Polygonize.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Project**(**args*, ***kwargs*)
Returns a point projected from a start point by a distance and bearing (azimuth).

see http://postgis.net/docs/ST_Project.html

Return type: *geoalchemy2.types.Geography*.

**type**
alias of *geoalchemy2.types.Geography*

**class** geoalchemy2.functions.**ST_Quantile**(**args*, ***kwargs*)
Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.

see http://postgis.net/docs/RT_ST_Quantile.html

**class** geoalchemy2.functions.**ST_QuantizeCoordinates**(**args*, ***kwargs*)
Sets least significant bits of coordinates to zero

see http://postgis.net/docs/ST_QuantizeCoordinates.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Range4ma**(**args*, ***kwargs*)
Raster processing function that calculates the range of pixel values in a neighborhood.

see http://postgis.net/docs/RT_ST_Range4ma.html

**class** geoalchemy2.functions.**ST_RastFromHexWKB**(**args*, ***kwargs*)
Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

see http://postgis.net/docs/RT_ST_RastFromHexWKB.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_RastFromWKB**(**args*, ***kwargs*)
Return a raster value from a Well-Known Binary (WKB) raster.

see http://postgis.net/docs/RT_ST_RastFromWKB.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_RasterToWorldCoord**(**args*, ***kwargs*)
Returns the raster's upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.

see http://postgis.net/docs/RT_ST_RasterToWorldCoord.html

**class** geoalchemy2.functions.**ST_RasterToWorldCoordX**(**args*, ***kwargs*)
Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.

see http://postgis.net/docs/RT_ST_RasterToWorldCoordX.html

**class** geoalchemy2.functions.**ST_RasterToWorldCoordY**(**args*, ***kwargs*)
Returns the geometric Y coordinate upper left corner of a raster, column and rows. Numbering of columns and rows starts at 1.

see http://postgis.net/docs/RT_ST_RasterToWorldCoordY.html

**class** geoalchemy2.functions.**ST_Reclass**(**args*, ***kwargs*)
Creates a new raster composed of band types reclassified from original. The nband is the band to be changed.

If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.

see http://postgis.net/docs/RT_ST_Reclass.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Relate**(**args*, ***kwargs*)
Returns true if this Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern. If no intersectionMatrixPattern is passed in, then returns the maximum intersectionMatrixPattern that relates the 2 geometries.

see http://postgis.net/docs/ST_Relate.html

**class** geoalchemy2.functions.**ST_RelateMatch**(**args*, ***kwargs*)
Returns true if intersectionMattrixPattern1 implies intersectionMatrixPattern2

see http://postgis.net/docs/ST_RelateMatch.html

**class** geoalchemy2.functions.**ST_RemovePoint**(**args*, ***kwargs*)
Remove point from a linestring.

see http://postgis.net/docs/ST_RemovePoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_RemoveRepeatedPoints**(**args*, ***kwargs*)
Returns a version of the given geometry with duplicated points removed.

see http://postgis.net/docs/ST_RemoveRepeatedPoints.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Resample**(**args*, ***kwargs*)
Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.

see http://postgis.net/docs/RT_ST_Resample.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Rescale**(**args*, ***kwargs*)
Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

see http://postgis.net/docs/RT_ST_Rescale.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Resize**(**args*, ***kwargs*)
Resize a raster to a new width/height

see http://postgis.net/docs/RT_ST_Resize.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Reskew**(**args*, ***kwargs*)
Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

see http://postgis.net/docs/RT_ST_Reskew.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Retile**(**args*, ***kwargs*)
Return a set of configured tiles from an arbitrarily tiled raster coverage.

see http://postgis.net/docs/RT_ST_Retile.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Reverse**(**args*, ***kwargs*)
Return the geometry with vertex order reversed.

see http://postgis.net/docs/ST_Reverse.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Rotate**(**args*, ***kwargs*)
Rotates a geometry about an origin point.

see http://postgis.net/docs/ST_Rotate.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_RotateX**(**args*, ***kwargs*)
Rotates a geometry about the X axis.

see http://postgis.net/docs/ST_RotateX.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_RotateY**(**args*, ***kwargs*)
Rotates a geometry about the Y axis.

see http://postgis.net/docs/ST_RotateY.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_RotateZ**(**args*, ***kwargs*)
Rotates a geometry about the Z axis.

see http://postgis.net/docs/ST_RotateZ.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Rotation**(**args*, ***kwargs*)
Returns the rotation of the raster in radian.

see http://postgis.net/docs/RT_ST_Rotation.html

**class** geoalchemy2.functions.**ST_Roughness**(**args*, ***kwargs*)
Returns a raster with the calculated "roughness" of a DEM.

see http://postgis.net/docs/RT_ST_Roughness.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SRID**(**args*, ***kwargs*)
[geometry] Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. OR [raster] Returns the spatial reference identifier of the raster as defined in spatial_ref_sys table.

see http://postgis.net/docs/ST_SRID.html

**class** geoalchemy2.functions.**ST_SameAlignment**(**args*, ***kwargs*)
Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.

see http://postgis.net/docs/RT_ST_SameAlignment.html

**class** geoalchemy2.functions.**ST_Scale**(**args*, ***kwargs*)
Scales a geometry by given factors.

see http://postgis.net/docs/ST_Scale.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ScaleX**(**args*, ***kwargs*)
Returns the X component of the pixel width in units of coordinate reference system.

see http://postgis.net/docs/RT_ST_ScaleX.html

**class** geoalchemy2.functions.**ST_ScaleY**(**args*, ***kwargs*)
Returns the Y component of the pixel height in units of coordinate reference system.

see http://postgis.net/docs/RT_ST_ScaleY.html

**class** geoalchemy2.functions.**ST_Segmentize**(**args*, ***kwargs*)
Return a modified geometry/geography having no segment longer than the given distance.

see http://postgis.net/docs/ST_Segmentize.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SetBandIndex**(**args*, ***kwargs*)
Update the external band number of an out-db band

see http://postgis.net/docs/RT_ST_SetBandIndex.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetBandIsNoData**(**args*, ***kwargs*)
Sets the isnodata flag of the band to TRUE.

see http://postgis.net/docs/RT_ST_SetBandIsNoData.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetBandNoDataValue**(**args*, ***kwargs*)
Sets the value for the given band that represents no data. Band 1 is assumed if no band is specified. To mark a band as having no nodata value, set the nodata value = NULL.

see http://postgis.net/docs/RT_ST_SetBandNoDataValue.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetBandPath**(**args*, ***kwargs*)
Update the external path and band number of an out-db band

see http://postgis.net/docs/RT_ST_SetBandPath.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetEffectiveArea**(**args*, ***kwargs*)
Sets the effective area for each vertex, storing the value in the M ordinate. A simplified geometry can then be generated by filtering on the M ordinate.

see http://postgis.net/docs/ST_SetEffectiveArea.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SetGeoReference**(**args*, ***kwargs*)
Set Georeference 6 georeference parameters in a single call. Numbers should be separated by white space. Accepts inputs in GDAL or ESRI format. Default is GDAL.

see http://postgis.net/docs/RT_ST_SetGeoReference.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetPoint**(**args*, ***kwargs*)
Replace point of a linestring with a given point.

see http://postgis.net/docs/ST_SetPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SetRotation**(**args*, ***kwargs*)
Set the rotation of the raster in radian.

see http://postgis.net/docs/RT_ST_SetRotation.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetSRID**(**args*, ***kwargs*)
[geometry] Set the SRID on a geometry to a particular integer value. OR [raster] Sets the SRID of a raster to a particular integer srid defined in the spatial_ref_sys table.

see http://postgis.net/docs/ST_SetSRID.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SetScale**(**args*, ***kwargs*)
Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height.

see http://postgis.net/docs/RT_ST_SetScale.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetSkew**(**args*, ***kwargs*)
Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value.

see http://postgis.net/docs/RT_ST_SetSkew.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetUpperLeft**(**args*, ***kwargs*)
Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

see http://postgis.net/docs/RT_ST_SetUpperLeft.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetValue**(**args*, ***kwargs*)
Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.

see http://postgis.net/docs/RT_ST_SetValue.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SetValues**(**args*, ***kwargs*)
Returns modified raster resulting from setting the values of a given band.

see http://postgis.net/docs/RT_ST_SetValues.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_SharedPaths**(**args*, ***kwargs*)
Returns a collection containing paths shared by the two input linestrings/multilinestrings.

see http://postgis.net/docs/ST_SharedPaths.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ShiftLongitude**(**args*, ***kwargs*)
Toggle geometry coordinates between -180..180 and 0..360 ranges.

see http://postgis.net/docs/ST_Shift_Longitude.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_ShortestLine**(**args*, ***kwargs*)
Returns the 2D shortest line between two geometries

see http://postgis.net/docs/ST_ShortestLine.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Simplify**(**args*, ***kwargs*)
Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm.

see http://postgis.net/docs/ST_Simplify.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SimplifyPreserveTopology**(**args*, ***kwargs*)
Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will avoid creating derived geometries (polygons in particular) that are invalid.

see http://postgis.net/docs/ST_SimplifyPreserveTopology.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SimplifyVW**(**args*, ***kwargs*)
Returns a "simplified" version of the given geometry using the Visvalingam-Whyatt algorithm

see http://postgis.net/docs/ST_SimplifyVW.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SkewX**(**args*, ***kwargs*)
Returns the georeference X skew (or rotation parameter).

see http://postgis.net/docs/RT_ST_SkewX.html

**class** geoalchemy2.functions.**ST_SkewY**(**args*, ***kwargs*)
Returns the georeference Y skew (or rotation parameter).

see http://postgis.net/docs/RT_ST_SkewY.html

**class** geoalchemy2.functions.**ST_Slope**(**args*, ***kwargs*)
Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

see http://postgis.net/docs/RT_ST_Slope.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Snap**(**args*, ***kwargs*)
Snap segments and vertices of input geometry to vertices of a reference geometry.

see http://postgis.net/docs/ST_Snap.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SnapToGrid**(**args*, ***kwargs*)
[geometry] Snap all points of the input geometry to a regular grid. OR [raster] Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

see http://postgis.net/docs/ST_SnapToGrid.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Split**(**args*, ***kwargs*)
Returns a collection of geometries resulting by splitting a geometry.

see http://postgis.net/docs/ST_Split.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_StartPoint**(**args*, ***kwargs*)
Returns the first point of a LineString.

see http://postgis.net/docs/ST_StartPoint.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_StdDev4ma**(**args*, ***kwargs*)
Raster processing function that calculates the standard deviation of pixel values in a neighborhood.

see http://postgis.net/docs/RT_ST_StdDev4ma.html

**class** geoalchemy2.functions.**ST_StraightSkeleton**(**args*, ***kwargs*)
Compute a straight skeleton from a geometry

see http://postgis.net/docs/ST_StraightSkeleton.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Subdivide**(**args*, ***kwargs*)
Returns a set of geometry where no geometry in the set has more than the specified number of vertices.

see http://postgis.net/docs/ST_Subdivide.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Sum4ma**(**args*, ***kwargs*)
Raster processing function that calculates the sum of all pixel values in a neighborhood.

see http://postgis.net/docs/RT_ST_Sum4ma.html

**class** geoalchemy2.functions.**ST_Summary**(**args*, ***kwargs*)
[geometry] Returns a text summary of the contents of a geometry. OR [raster] Returns a text summary of the contents of the raster.

see http://postgis.net/docs/ST_Summary.html

**class** geoalchemy2.functions.**ST_SummaryStats**(**args*, ***kwargs*)
Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.

see http://postgis.net/docs/RT_ST_SummaryStats.html

**class** geoalchemy2.functions.**ST_SummaryStatsAgg**(**args*, ***kwargs*)
Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.

see http://postgis.net/docs/RT_ST_SummaryStatsAgg.html

Return type: *geoalchemy2.types.SummaryStats*.

**type**
alias of *geoalchemy2.types.SummaryStats*

**class** geoalchemy2.functions.**ST_SwapOrdinates**(**args*, ***kwargs*)
Returns a version of the given geometry with given ordinate values swapped.

see http://postgis.net/docs/ST_SwapOrdinates.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_SymDifference**(**args*, ***kwargs*)
Returns a geometry that represents the portions of A and B that do not intersect. It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A).

see http://postgis.net/docs/ST_SymDifference.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_TPI**(**args*, ***kwargs*)
Returns a raster with the calculated Topographic Position Index.

see http://postgis.net/docs/RT_ST_TPI.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_TRI**(**args*, ***kwargs*)
Returns a raster with the calculated Terrain Ruggedness Index.

see http://postgis.net/docs/RT_ST_TRI.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_Tesselate**(**args*, ***kwargs*)
Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

see http://postgis.net/docs/ST_Tesselate.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Tile**(**args*, ***kwargs*)
Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

see http://postgis.net/docs/RT_ST_Tile.html

Return type: *geoalchemy2.types.Raster*.

**type**
alias of *geoalchemy2.types.Raster*

**class** geoalchemy2.functions.**ST_TileEnvelope**(**args*, ***kwargs*)
Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

see http://postgis.net/docs/ST_TileEnvelope.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Touches**(**args*, ***kwargs*)
[geometry] Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect. OR [raster] Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.

see http://postgis.net/docs/ST_Touches.html

**class** geoalchemy2.functions.**ST_TransScale**(**args*, ***kwargs*)
Translates and scales a geometry by given offsets and factors.

see http://postgis.net/docs/ST_TransScale.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Transform**(**args*, ***kwargs*)
[geometry] Return a new geometry with its coordinates transformed to a different spatial reference system. OR [raster] Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.

see http://postgis.net/docs/ST_Transform.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Translate**(**args*, ***kwargs*)
Translates a geometry by given offsets.

see http://postgis.net/docs/ST_Translate.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_UnaryUnion**(**args*, ***kwargs*)
Like ST_Union, but working at the geometry component level.

see http://postgis.net/docs/ST_UnaryUnion.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Union**(**args*, ***kwargs*)
[geometry] Returns a geometry that represents the point set union of the Geometries. OR [raster] Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.

see http://postgis.net/docs/ST_Union.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_UpperLeftX**(**args*, ***kwargs*)
Returns the upper left X coordinate of raster in projected spatial ref.

see http://postgis.net/docs/RT_ST_UpperLeftX.html

**class** geoalchemy2.functions.**ST_UpperLeftY**(**args*, ***kwargs*)
Returns the upper left Y coordinate of raster in projected spatial ref.

see http://postgis.net/docs/RT_ST_UpperLeftY.html

**class** geoalchemy2.functions.**ST_Value**(**args*, ***kwargs*)
Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If exclude_nodata_value is set to false, then all pixels include nodata pixels are considered to intersect and return value. If exclude_nodata_value is not passed in then reads it from metadata of raster.

see http://postgis.net/docs/RT_ST_Value.html

**class** geoalchemy2.functions.**ST_ValueCount**(**args*, ***kwargs*)
Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted. and all other values in the pixel are output and pixel band values are rounded to the nearest integer.

see http://postgis.net/docs/RT_ST_ValueCount.html

**class** geoalchemy2.functions.**ST_Volume**(**args*, ***kwargs*)
Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.

see http://postgis.net/docs/ST_Volume.html

**class** geoalchemy2.functions.**ST_VoronoiLines**(**args*, ***kwargs*)
Returns the boundaries between the cells of the Voronoi diagram constructed from the vertices of a geometry.

see http://postgis.net/docs/ST_VoronoiLines.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_VoronoiPolygons**(**args*, ***kwargs*)
Returns the cells of the Voronoi diagram constructed from the vertices of a geometry.

see http://postgis.net/docs/ST_VoronoiPolygons.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_WKBToSQL**(**args*, ***kwargs*)
Return a specified ST_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST_GeomFromWKB that takes no srid

see http://postgis.net/docs/ST_WKBToSQL.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_WKTToSQL**(**args*, ***kwargs*)
Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

see http://postgis.net/docs/ST_WKTToSQL.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_Width**(**args*, ***kwargs*)
Returns the width of the raster in pixels.

see http://postgis.net/docs/RT_ST_Width.html

**class** geoalchemy2.functions.**ST_Within**(**args*, ***kwargs*)
[geometry] Returns true if the geometry A is completely inside geometry B OR [raster] Return true if no points of raster rastA lie in the exterior of raster rastB and at least one point of the interior of rastA lies in the interior of rastB.

see http://postgis.net/docs/ST_Within.html

**class** geoalchemy2.functions.**ST_WorldToRasterCoord**(**args*, ***kwargs*)
Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry expressed in the spatial reference coordinate system of the raster.

see http://postgis.net/docs/RT_ST_WorldToRasterCoord.html

**class** geoalchemy2.functions.**ST_WorldToRasterCoordX**(**args*, ***kwargs*)
Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

see http://postgis.net/docs/RT_ST_WorldToRasterCoordX.html

**class** geoalchemy2.functions.**ST_WorldToRasterCoordY**(**args*, ***kwargs*)
Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

see http://postgis.net/docs/RT_ST_WorldToRasterCoordY.html

**class** geoalchemy2.functions.**ST_WrapX**(**args*, ***kwargs*)
Wrap a geometry around an X value.

see http://postgis.net/docs/ST_WrapX.html

Return type: *geoalchemy2.types.Geometry*.

**type**
alias of *geoalchemy2.types.Geometry*

**class** geoalchemy2.functions.**ST_X**(**args*, ***kwargs*)
Returns the X coordinate of a Point.

see http://postgis.net/docs/ST_X.html

**class** geoalchemy2.functions.**ST_XMax**(**args*, ***kwargs*)
Returns the X maxima of a 2D or 3D bounding box or a geometry.

see http://postgis.net/docs/ST_XMax.html

**class** geoalchemy2.functions.**ST_XMin**(**args*, ***kwargs*)
Returns the X minima of a 2D or 3D bounding box or a geometry.

see http://postgis.net/docs/ST_XMin.html

**class** geoalchemy2.functions.**ST_Y**(**args*, ***kwargs*)
Returns the Y coordinate of a Point.

see http://postgis.net/docs/ST_Y.html

**class** geoalchemy2.functions.**ST_YMax**(**args*, ***kwargs*)
Returns the Y maxima of a 2D or 3D bounding box or a geometry.

see http://postgis.net/docs/ST_YMax.html

**class** geoalchemy2.functions.**ST_YMin**(**args*, ***kwargs*)
Returns the Y minima of a 2D or 3D bounding box or a geometry.

see http://postgis.net/docs/ST_YMin.html

**class** geoalchemy2.functions.**ST_Z**(**args*, ***kwargs*)
Returns the Z coordinate of a Point.

see http://postgis.net/docs/ST_Z.html

**class** geoalchemy2.functions.**ST_ZMax**(**args*, ***kwargs*)
Returns the Z maxima of a 2D or 3D bounding box or a geometry.

see http://postgis.net/docs/ST_ZMax.html

**class** geoalchemy2.functions.**ST_ZMin**(**args*, ***kwargs*)
Returns the Z minima of a 2D or 3D bounding box or a geometry.

see http://postgis.net/docs/ST_ZMin.html

**class** geoalchemy2.functions.**ST_Zmflag**(**args*, ***kwargs*)
Returns a code indicating the ZM coordinate dimension of a geometry.

see http://postgis.net/docs/ST_Zmflag.html

**class** geoalchemy2.functions.**TableRowElement**(*selectable*)

**class** geoalchemy2.functions.**UnlockRows**(**args*, ***kwargs*)
Removes all locks held by an authorization token.

see http://postgis.net/docs/UnlockRows.html

**class** geoalchemy2.functions.**UpdateGeometrySRID**(**args*, ***kwargs*)
Updates the SRID of all features in a geometry column, and the table metadata.

see http://postgis.net/docs/UpdateGeometrySRID.html

**class** geoalchemy2.functions.**postgis_sfcgal_version**(**args*, ***kwargs*)
Returns the version of SFCGAL in use

see http://postgis.net/docs/postgis_sfcgal_version.html

geoalchemy2.functions.**register_sqlite_mapping**(*mapping*)
Register compilation mappings for the given functions.

**Parameters** **mapping** – Should have the following form:

```
{
    "function_name_1": "sqlite_function_name_1",
    "function_name_2": "sqlite_function_name_2",
    ...
}
```

## 7.4 Spatial Operators

This module defines a `Comparator` class for use with geometry and geography objects. This is where spatial operators, like `&&`, `&<`, are defined. Spatial operators very often apply to the bounding boxes of geometries. For

example, `geom1 && geom2` indicates if geom1's bounding box intersects geom2's.

#### Examples

Select the objects whose bounding boxes are to the left of the bounding box of `POLYGON((-5 45,5 45,5 -45, -5 -45,-5 45))`:

```
select([table]).where(table.c.geom.to_left(
    'POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))'))
```

The << and >> operators are a bit specific, because they have corresponding Python operator (`__lshift__` and `__rshift__`). The above `SELECT` expression can thus be rewritten like this:

```
select([table]).where(
    table.c.geom << 'POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))')
```

Operators can also be used when using the ORM. For example:

```
Session.query(Cls).filter(
    Cls.geom << 'POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))')
```

Now some other examples with the <#> operator.

Select the ten objects that are the closest to `POINT(0 0)` (typical closed neighbors problem):

```
select([table]).order_by(table.c.geom.distance_box('POINT(0 0)')).limit(10)
```

Using the ORM:

```
Session.query(Cls).order_by(Cls.geom.distance_box('POINT(0 0)')).limit(10)
```

### 7.4.1 Reference

**class** geoalchemy2.comparator.**BaseComparator**(*expr*)

Bases: sqlalchemy.sql.type_api.Comparator

A custom comparator base class. It adds the ability to call spatial functions on columns that use this kind of comparator. It also defines functions that map to operators supported by `Geometry`, `Geography` and `Raster` columns.

This comparator is used by the *geoalchemy2.types.Raster*.

**__weakref__**

list of weak references to the object (if defined)

**intersects**(*other*)

The `&&` operator. A's BBOX intersects B's.

**overlaps_or_to_left**(*other*)

The `&<` operator. A's BBOX overlaps or is to the left of B's.

**overlaps_or_to_right**(*other*)

The `&>` operator. A's BBOX overlaps or is to the right of B's.

**class** geoalchemy2.comparator.**Comparator**(*expr*)

Bases: *geoalchemy2.comparator.BaseComparator*

A custom comparator class. Used in *geoalchemy2.types.Geometry* and *geoalchemy2.types.Geography*.

This is where spatial operators like << and <-> are defined.

**__lshift__**(*other*)
The << operator. A's BBOX is strictly to the left of B's. Same as `to_left`, so:

```
table.c.geom << 'POINT(1 2)'
```

is the same as:

```
table.c.geom.to_left('POINT(1 2)')
```

**__rshift__**(*other*)
The >> operator. A's BBOX is strictly to the left of B's. Same as *to_'right'*, so:

```
table.c.geom >> 'POINT(1 2)'
```

is the same as:

```
table.c.geom.to_right('POINT(1 2)')
```

**above**(*other*)
The |>> operator. A's BBOX is strictly above B's.

**below**(*other*)
The <<| operator. A's BBOX is strictly below B's.

**contained**(*other*)
The @ operator. A's BBOX is contained by B's.

**contains**(*other*, ***kw*)
The ~ operator. A's BBOX contains B's.

**distance_box**(*other*)
The <#> operator. The distance between bounding box of two geometries.

**distance_centroid**(*other*)
The <-> operator. The distance between two points.

**intersects_nd**(*other*)
The &&& operator returns TRUE if the n-D bounding box of geometry A intersects the n-D bounding box of geometry B.

**overlaps_or_above**(*other*)
The |&> operator. A's BBOX overlaps or is above B's.

**overlaps_or_below**(*other*)
The &<| operator. A's BBOX overlaps or is below B's.

**same**(*other*)
The ~= operator. A's BBOX is the same as B's.

**to_left**(*other*)
The << operator. A's BBOX is strictly to the left of B's.

**to_right**(*other*)
The >> operator. A's BBOX is strictly to the right of B's.

## 7.5 Shapely Integration

This module provides utility functions for integrating with Shapely.

**Note:** As GeoAlchemy 2 itself has no dependency on *Shapely*, applications using functions of this module have to ensure that *Shapely* is available.

geoalchemy2.shape.**from_shape**(*shape*, *srid=-1*, *extended=False*)

Function to convert a Shapely geometry to a `geoalchemy2.types.WKBElement`.

**Parameters**

- **srid** – An integer representing the spatial reference system. E.g. `4326`. Default value is `-1`, which means no/unknown reference system.
- **extended** – A boolean to switch between WKB and EWKB. Default value is False.

Example:

```
from shapely.geometry import Point
wkb_element = from_shape(Point(5, 45), srid=4326)
ewkb_element = from_shape(Point(5, 45), srid=4326, extended=True)
```

geoalchemy2.shape.**to_shape**(*element*)

Function to convert a `geoalchemy2.types.SpatialElement` to a Shapely geometry.

**Parameters** **element** – The element to convert into a `Shapely` object.

Example:

```
lake = Session.query(Lake).get(1)
polygon = to_shape(lake.geom)
```

## 7.6 Alembic helpers

Some helpers to use with Alembic migration tool.

**class** geoalchemy2.alembic_helpers.**AddGeospatialColumnOp**(*table_name: str*, *column: Column*, *schema: Optional[str] = None*, ***kw*)

Bases: `alembic.operations.ops.AddColumnOp`

Add a Geospatial Column in an Alembic migration context. This methodology originates from: https://alembic.sqlalchemy.org/en/latest/api/operations.html#operation-plugins

**classmethod add_geospatial_column**(*operations*, *table_name*, *column*, *schema=None*)

This method is proxied on the `Operations` class, via the `Operations.add_geospatial_column()` method.

**classmethod batch_add_geospatial_column**(*operations*, *column*, *insert_before=None*, *insert_after=None*)

This method is proxied on the `BatchOperations` class, via the `BatchOperations.add_geospatial_column()` method.

**reverse**()

Used to autogenerate the downgrade function.

**class** geoalchemy2.alembic_helpers.**CreateGeospatialIndexOp**(*index_name: str, table_name: str, columns: Sequence[Union[str, TextClause, ColumnElement[Any]]], schema: Optional[str] = None, unique: bool = False, **kw*)

Bases: alembic.operations.ops.CreateIndexOp

**classmethod batch_create_geospatial_index**(*operations*, *index_name*, *columns*, ***kw*)

This method is proxied on the BatchOperations class, via the BatchOperations.create_geospatial_index() method.

**classmethod create_geospatial_index**(*operations*, *index_name*, *table_name*, *columns*, *schema=None*, *unique=False*, ***kw*)

This method is proxied on the Operations class, via the Operations.create_geospatial_index() method.

**reverse**()

Used to autogenerate the downgrade function.

**class** geoalchemy2.alembic_helpers.**CreateGeospatialTableOp**(*table_name: str, columns: Sequence[Union[Column, Constraint]], schema: Optional[str] = None, _namespace_metadata: Optional[MetaData] = None, _constraints_included: bool = False, **kw*)

Bases: alembic.operations.ops.CreateTableOp

Create a Geospatial Table in an Alembic migration context. This methodology originates from: https://alembic.sqlalchemy.org/en/latest/api/operations.html#operation-plugins

**classmethod create_geospatial_table**(*operations*, *table_name*, **columns*, ***kw*)

This method is proxied on the Operations class, via the Operations.create_geospatial_table() method.

**reverse**()

Used to autogenerate the downgrade function.

**class** geoalchemy2.alembic_helpers.**DropGeospatialColumnOp**(*table_name: str, column_name: str, schema: Optional[str] = None, _reverse: Optional[AddColumnOp] = None, **kw*)

Bases: alembic.operations.ops.DropColumnOp

Drop a Geospatial Column in an Alembic migration context.

**classmethod batch_drop_geospatial_column**(*operations*, *column_name*, ***kw*)

This method is proxied on the BatchOperations class, via the BatchOperations.drop_geospatial_column() method.

**classmethod drop_geospatial_column**(*operations*, *table_name*, *column_name*, *schema=None*, ***kw*)

This method is proxied on the `Operations` class, via the `Operations.drop_geospatial_column()` method.

**reverse**()
Used to autogenerate the downgrade function.

**class** geoalchemy2.alembic_helpers.**DropGeospatialIndexOp**(**args*, *column_name*, ***kwargs*)
Bases: `alembic.operations.ops.DropIndexOp`

**classmethod batch_drop_geospatial_index**(*operations*, *index_name*, ***kw*)
This method is proxied on the `BatchOperations` class, via the `BatchOperations.drop_geospatial_index()` method.

**classmethod drop_geospatial_index**(*operations*, *index_name*, *table_name*, *column_name*, *schema=None*, *unique=False*, ***kw*)
This method is proxied on the `Operations` class, via the `Operations.drop_geospatial_index()` method.

**reverse**()
Used to autogenerate the downgrade function.

**class** geoalchemy2.alembic_helpers.**DropGeospatialTableOp**(*table_name: str*, *schema: Optional[str] = None*, *table_kw: Optional[MutableMapping[Any, Any]] = None*, *_reverse: Optional[CreateTableOp] = None*)
Bases: `alembic.operations.ops.DropTableOp`

**classmethod drop_geospatial_table**(*operations*, *table_name*, *schema=None*, ***kw*)
This method is proxied on the `Operations` class, via the `Operations.drop_geospatial_table()` method.

**reverse**()
Used to autogenerate the downgrade function.

geoalchemy2.alembic_helpers.**add_geo_column**(*context*, *revision*, *op*)
Replace the default AddColumnOp by a geospatial-specific one.

geoalchemy2.alembic_helpers.**add_geospatial_column**(*operations*, *operation*)
Handle the actual column addition according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – AddGeospatialColumnOp call, with attributes for table_name, column_name, column_type, and optional keywords.

geoalchemy2.alembic_helpers.**create_geo_index**(*context*, *revision*, *op*)
Replace the default CreateIndexOp by a geospatial-specific one.

geoalchemy2.alembic_helpers.**create_geo_table**(*context*, *revision*, *op*)
Replace the default CreateTableOp by a geospatial-specific one.

geoalchemy2.alembic_helpers.**create_geospatial_index**(*operations*, *operation*)
Handle the actual index creation according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – CreateGeospatialIndexOp call, with attributes for table_name, column_name, column_type, and optional keywords.

geoalchemy2.alembic_helpers.**create_geospatial_table**(*operations*, *operation*)
Handle the actual table creation according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – CreateGeospatialTableOp call, with attributes for table_name, column_name, column_type, and optional keywords.

geoalchemy2.alembic_helpers.**drop_geo_column**(*context*, *revision*, *op*)
Replace the default DropColumnOp by a geospatial-specific one.

geoalchemy2.alembic_helpers.**drop_geo_index**(*context*, *revision*, *op*)
Replace the default DropIndexOp by a geospatial-specific one.

geoalchemy2.alembic_helpers.**drop_geo_table**(*context*, *revision*, *op*)
Replace the default DropTableOp by a geospatial-specific one.

geoalchemy2.alembic_helpers.**drop_geospatial_column**(*operations*, *operation*)
Handle the actual column removal according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – AddGeospatialColumnOp call, with attributes for table_name, column_name, column_type, and optional keywords.

geoalchemy2.alembic_helpers.**drop_geospatial_index**(*operations*, *operation*)
Handle the actual index drop according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – DropGeospatialIndexOp call, with attributes for table_name, column_name, column_type, and optional keywords.

geoalchemy2.alembic_helpers.**drop_geospatial_table**(*operations*, *operation*)
Handle the actual table removal according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – DropGeospatialTableOp call, with attributes for table_name, column_name, column_type, and optional keywords.

geoalchemy2.alembic_helpers.**render_add_geo_column**(*autogen_context*, *op*)
Render the add_geospatial_column operation in migration script.

geoalchemy2.alembic_helpers.**render_create_geo_index**(*autogen_context*, *op*)
Render the create_geospatial_index operation in migration script.

geoalchemy2.alembic_helpers.**render_create_geo_table**(*autogen_context*, *op*)
Render the create_geospatial_table operation in migration script.

geoalchemy2.alembic_helpers.**render_drop_geo_column**(*autogen_context*, *op*)
Render the drop_geospatial_column operation in migration script.

geoalchemy2.alembic_helpers.**render_drop_geo_index**(*autogen_context*, *op*)
Render the drop_geospatial_index operation in migration script.

geoalchemy2.alembic_helpers.**render_drop_geo_table**(*autogen_context*, *op*)
Render the drop_geospatial_table operation in migration script.

geoalchemy2.alembic_helpers.**render_item**(*obj_type*, *obj*, *autogen_context*)
Add proper imports for spatial types.

geoalchemy2.alembic_helpers.**visit_drop_geospatial_table**(*element*, *compiler*, ***kw*)
Specific compilation rule to drop spatial tables with SQLite dialect.

geoalchemy2.alembic_helpers.**visit_rename_geospatial_table**(*element*, *compiler*, ***kw*)
Specific compilation rule to rename spatial tables with SQLite dialect.

CHAPTER 8

# Development

The code is available on GitHub: https://github.com/geoalchemy/geoalchemy2.

Contributors:

- Adrien Berchet (https://github.com/adrien-berchet)
- Éric Lemoine (https://github.com/elemoine)
- Dolf Andringa (https://github.com/dolfandringa)
- Frédéric Junod, Camptocamp SA (https://github.com/fredj)
- ijl (https://github.com/ijl)
- Loïc Gasser (https://github.com/loicgasser)
- Marcel Radischat (https://github.com/quiqua)
- rapto (https://github.com/rapto)
- Serge Bouchut (https://github.com/SergeBouchut)
- Tobias Bieniek (https://github.com/Turbo87)
- Tom Payne (https://github.com/twpayne)

Many thanks to Mike Bayer for his guidance and support! He also fostered the birth of GeoAlchemy 2.

CHAPTER 9

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## g

geoalchemy2.alembic_helpers, 105
geoalchemy2.comparator, 102
geoalchemy2.functions, 52
geoalchemy2.shape, 105
geoalchemy2.types, 47

# Index

## Symbols


_DummyGeometry (*class in geoalchemy2.types*), 49
_GISType (*class in geoalchemy2.types*), 49
_SpatialElement (*class in geoalchemy2.elements*), 51
__lshift__() (*geoalchemy2.comparator.Comparator method*), 104
__rshift__() (*geoalchemy2.comparator.Comparator method*), 104
__weakref__ (*geoalchemy2.comparator.BaseComparator attribute*), 103


## A


above() (*geoalchemy2.comparator.Comparator method*), 104
add_geo_column() (*in module geoalchemy2.alembic_helpers*), 107
add_geospatial_column() (*geoalchemy2.alembic_helpers.AddGeospatialColumnOp class method*), 105
add_geospatial_column() (*in module geoalchemy2.alembic_helpers*), 107
AddAuth (*class in geoalchemy2.functions*), 53
AddGeometryColumn (*class in geoalchemy2.functions*), 53
AddGeospatialColumnOp (*class in geoalchemy2.alembic_helpers*), 105
as_binary (*geoalchemy2.types._GISType attribute*), 50
as_binary (*geoalchemy2.types.Geography attribute*), 48
as_binary (*geoalchemy2.types.Geometry attribute*), 48
as_binary (*geoalchemy2.types.Raster attribute*), 49


## B


BaseComparator (*class in geoalchemy2.comparator*), 103
batch_add_geospatial_column() (*geoalchemy2.alembic_helpers.AddGeospatialColumnOp class method*), 105
batch_create_geospatial_index() (*geoalchemy2.alembic_helpers.CreateGeospatialIndexOp class method*), 106
batch_drop_geospatial_column() (*geoalchemy2.alembic_helpers.DropGeospatialColumnOp class method*), 106
batch_drop_geospatial_index() (*geoalchemy2.alembic_helpers.DropGeospatialIndexOp class method*), 107
below() (*geoalchemy2.comparator.Comparator method*), 104
bind_expression() (*geoalchemy2.types._GISType method*), 51
bind_processor() (*geoalchemy2.types._GISType method*), 51
Box2D (*class in geoalchemy2.functions*), 53
Box3D (*class in geoalchemy2.functions*), 53


## C


cache_ok (*geoalchemy2.types._GISType attribute*), 51
cache_ok (*geoalchemy2.types.Geography attribute*), 48
cache_ok (*geoalchemy2.types.Geometry attribute*), 48
cache_ok (*geoalchemy2.types.GeometryDump attribute*), 48
cache_ok (*geoalchemy2.types.Raster attribute*), 49
cache_ok (*geoalchemy2.types.SummaryStats attribute*), 49
CheckAuth (*class in geoalchemy2.functions*), 53
column_expression() (*geoalchemy2.types._GISType method*), 51
Comparator (*class in geoalchemy2.comparator*), 103
comparator_factory (*geoalchemy2.types._GISType attribute*), 51
comparator_factory (*geoalchemy2.types.Raster attribute*), 49

CompositeElement (*class in geoalchemy2.elements*), 52
CompositeType (*class in geoalchemy2.types*), 47
CompositeType.comparator_factory (*class in geoalchemy2.types*), 47
contained() (*geoalchemy2.comparator.Comparator method*), 104
contains() (*geoalchemy2.comparator.Comparator method*), 104
create_geo_index() (*in module geoalchemy2.alembic_helpers*), 107
create_geo_table() (*in module geoalchemy2.alembic_helpers*), 107
create_geospatial_index() (*geoalchemy2.alembic_helpers.CreateGeospatialIndexOp class method*), 106
create_geospatial_index() (*in module geoalchemy2.alembic_helpers*), 107
create_geospatial_table() (*geoalchemy2.alembic_helpers.CreateGeospatialTableOp class method*), 106
create_geospatial_table() (*in module geoalchemy2.alembic_helpers*), 108
CreateGeospatialIndexOp (*class in geoalchemy2.alembic_helpers*), 105
CreateGeospatialTableOp (*class in geoalchemy2.alembic_helpers*), 106

## D

desc (*geoalchemy2.elements.RasterElement attribute*), 52
desc (*geoalchemy2.elements.WKBElement attribute*), 52
desc (*geoalchemy2.elements.WKTElement attribute*), 51
DisableLongTransactions (*class in geoalchemy2.functions*), 54
distance_box() (*geoalchemy2.comparator.Comparator method*), 104
distance_centroid() (*geoalchemy2.comparator.Comparator method*), 104
drop_geo_column() (*in module geoalchemy2.alembic_helpers*), 108
drop_geo_index() (*in module geoalchemy2.alembic_helpers*), 108
drop_geo_table() (*in module geoalchemy2.alembic_helpers*), 108
drop_geospatial_column() (*geoalchemy2.alembic_helpers.DropGeospatialColumnOp class method*), 106
drop_geospatial_column() (*in module geoalchemy2.alembic_helpers*), 108
drop_geospatial_index() (*geoalchemy2.alembic_helpers.DropGeospatialIndexOp class method*), 107
drop_geospatial_index() (*in module geoalchemy2.alembic_helpers*), 108
drop_geospatial_table() (*geoalchemy2.alembic_helpers.DropGeospatialTableOp class method*), 107
drop_geospatial_table() (*in module geoalchemy2.alembic_helpers*), 108
DropGeometryColumn (*class in geoalchemy2.functions*), 54
DropGeometryTable (*class in geoalchemy2.functions*), 54
DropGeospatialColumnOp (*class in geoalchemy2.alembic_helpers*), 106
DropGeospatialIndexOp (*class in geoalchemy2.alembic_helpers*), 107
DropGeospatialTableOp (*class in geoalchemy2.alembic_helpers*), 107

## E

ElementType (*geoalchemy2.types.Geography attribute*), 48
ElementType (*geoalchemy2.types.Geometry attribute*), 48
ElementType (*geoalchemy2.types.Raster attribute*), 49
EnableLongTransactions (*class in geoalchemy2.functions*), 54

## F

Find_SRID (*class in geoalchemy2.functions*), 54
from_shape() (*in module geoalchemy2.shape*), 105
from_text (*geoalchemy2.types._GISType attribute*), 51
from_text (*geoalchemy2.types.Geography attribute*), 48
from_text (*geoalchemy2.types.Geometry attribute*), 48
from_text (*geoalchemy2.types.Raster attribute*), 49

## G

GenericFunction (*class in geoalchemy2.functions*), 54
geoalchemy2.alembic_helpers (*module*), 105
geoalchemy2.comparator (*module*), 102
geoalchemy2.functions (*module*), 52
geoalchemy2.shape (*module*), 105
geoalchemy2.types (*module*), 47
Geography (*class in geoalchemy2.types*), 47
geom_from (*geoalchemy2.elements.WKBElement attribute*), 52
geom_from (*geoalchemy2.elements.WKTElement attribute*), 52
geom_from_extended_version (*geoalchemy2.elements.WKBElement attribute*), 52

geom_from_extended_version (*geoalchemy2.elements.WKTElement attribute*), 52
Geometry (*class in geoalchemy2.types*), 48
GeometryDump (*class in geoalchemy2.types*), 48
GeometryType (*class in geoalchemy2.functions*), 54

## H

HasFunction (*class in geoalchemy2.elements*), 51

## I

intersects() (*geoalchemy2.comparator.BaseComparator method*), 103
intersects_nd() (*geoalchemy2.comparator.Comparator method*), 104

## L

LockRow (*class in geoalchemy2.functions*), 54

## N

name (*geoalchemy2.types._GISType attribute*), 51
name (*geoalchemy2.types.Geography attribute*), 48
name (*geoalchemy2.types.Geometry attribute*), 48
name (*geoalchemy2.types.Raster attribute*), 49

## O

overlaps_or_above() (*geoalchemy2.comparator.Comparator method*), 104
overlaps_or_below() (*geoalchemy2.comparator.Comparator method*), 104
overlaps_or_to_left() (*geoalchemy2.comparator.BaseComparator method*), 103
overlaps_or_to_right() (*geoalchemy2.comparator.BaseComparator method*), 103

## P

Populate_Geometry_Columns (*class in geoalchemy2.functions*), 54
PostGIS_AddBBox (*class in geoalchemy2.functions*), 55
PostGIS_DropBBox (*class in geoalchemy2.functions*), 55
PostGIS_Extensions_Upgrade (*class in geoalchemy2.functions*), 55
PostGIS_Full_Version (*class in geoalchemy2.functions*), 55
PostGIS_GEOS_Version (*class in geoalchemy2.functions*), 55
PostGIS_HasBBox (*class in geoalchemy2.functions*), 55
PostGIS_Lib_Build_Date (*class in geoalchemy2.functions*), 55
PostGIS_Lib_Version (*class in geoalchemy2.functions*), 55
PostGIS_Liblwgeom_Version (*class in geoalchemy2.functions*), 55
PostGIS_LibXML_Version (*class in geoalchemy2.functions*), 55
PostGIS_PROJ_Version (*class in geoalchemy2.functions*), 55
PostGIS_Scripts_Build_Date (*class in geoalchemy2.functions*), 55
PostGIS_Scripts_Installed (*class in geoalchemy2.functions*), 56
PostGIS_Scripts_Released (*class in geoalchemy2.functions*), 56
postgis_sfcgal_version (*class in geoalchemy2.functions*), 102
PostGIS_Version (*class in geoalchemy2.functions*), 56
PostGIS_Wagyu_Version (*class in geoalchemy2.functions*), 56

## R

Raster (*class in geoalchemy2.types*), 49
RasterElement (*class in geoalchemy2.elements*), 52
register_sqlite_mapping() (*in module geoalchemy2.functions*), 102
render_add_geo_column() (*in module geoalchemy2.alembic_helpers*), 108
render_create_geo_index() (*in module geoalchemy2.alembic_helpers*), 108
render_create_geo_table() (*in module geoalchemy2.alembic_helpers*), 108
render_drop_geo_column() (*in module geoalchemy2.alembic_helpers*), 109
render_drop_geo_index() (*in module geoalchemy2.alembic_helpers*), 109
render_drop_geo_table() (*in module geoalchemy2.alembic_helpers*), 109
render_item() (*in module geoalchemy2.alembic_helpers*), 109
result_processor() (*geoalchemy2.types._GISType method*), 51
reverse() (*geoalchemy2.alembic_helpers.AddGeospatialColumnOp method*), 105
reverse() (*geoalchemy2.alembic_helpers.CreateGeospatialIndexOp method*), 106
reverse() (*geoalchemy2.alembic_helpers.CreateGeospatialTableOp method*), 106
reverse() (*geoalchemy2.alembic_helpers.DropGeospatialColumnOp method*), 107

reverse() (*geoalchemy2.alembic_helpers.DropGeospatialIndexOp method*), 107
reverse() (*geoalchemy2.alembic_helpers.DropGeospatialTableOp method*), 107

## S

same() (*geoalchemy2.comparator.Comparator method*), 104
ST_3DArea (*class in geoalchemy2.functions*), 56
ST_3DClosestPoint (*class in geoalchemy2.functions*), 56
ST_3DDFullyWithin (*class in geoalchemy2.functions*), 56
ST_3DDifference (*class in geoalchemy2.functions*), 56
ST_3DDistance (*class in geoalchemy2.functions*), 56
ST_3DDWithin (*class in geoalchemy2.functions*), 56
ST_3DExtent (*class in geoalchemy2.functions*), 56
ST_3DIntersection (*class in geoalchemy2.functions*), 57
ST_3DIntersects (*class in geoalchemy2.functions*), 57
ST_3DLength (*class in geoalchemy2.functions*), 57
ST_3DLineInterpolatePoint (*class in geoalchemy2.functions*), 57
ST_3DLongestLine (*class in geoalchemy2.functions*), 57
ST_3DMakeBox (*class in geoalchemy2.functions*), 57
ST_3DMaxDistance (*class in geoalchemy2.functions*), 57
ST_3DPerimeter (*class in geoalchemy2.functions*), 57
ST_3DShortestLine (*class in geoalchemy2.functions*), 57
ST_3DUnion (*class in geoalchemy2.functions*), 58
ST_AddBand (*class in geoalchemy2.functions*), 58
ST_AddMeasure (*class in geoalchemy2.functions*), 58
ST_AddPoint (*class in geoalchemy2.functions*), 58
ST_Affine (*class in geoalchemy2.functions*), 58
ST_Angle (*class in geoalchemy2.functions*), 58
ST_ApproximateMedialAxis (*class in geoalchemy2.functions*), 58
ST_Area (*class in geoalchemy2.functions*), 59
ST_AsBinary (*class in geoalchemy2.functions*), 59
ST_AsEncodedPolyline (*class in geoalchemy2.functions*), 59
ST_AsEWKB (*class in geoalchemy2.functions*), 59
ST_AsEWKT (*class in geoalchemy2.functions*), 59
ST_AsGDALRaster (*class in geoalchemy2.functions*), 59
ST_AsGeobuf (*class in geoalchemy2.functions*), 59
ST_AsGeoJSON (*class in geoalchemy2.functions*), 59
ST_AsGML (*class in geoalchemy2.functions*), 59
ST_AsHEXEWKB (*class in geoalchemy2.functions*), 59
ST_AsHexWKB (*class in geoalchemy2.functions*), 59
ST_AsJPEG (*class in geoalchemy2.functions*), 59
ST_AsKML (*class in geoalchemy2.functions*), 60
ST_AsLatLonText (*class in geoalchemy2.functions*), 60
ST_AsMVT (*class in geoalchemy2.functions*), 60
ST_AsMVTGeom (*class in geoalchemy2.functions*), 60
ST_Aspect (*class in geoalchemy2.functions*), 61
ST_AsPNG (*class in geoalchemy2.functions*), 60
ST_AsRaster (*class in geoalchemy2.functions*), 60
ST_AsSVG (*class in geoalchemy2.functions*), 60
ST_AsText (*class in geoalchemy2.functions*), 60
ST_AsTIFF (*class in geoalchemy2.functions*), 60
ST_AsTWKB (*class in geoalchemy2.functions*), 60
ST_AsWKB (*class in geoalchemy2.functions*), 61
ST_AsX3D (*class in geoalchemy2.functions*), 61
ST_Azimuth (*class in geoalchemy2.functions*), 61
ST_Band (*class in geoalchemy2.functions*), 61
ST_BandFileSize (*class in geoalchemy2.functions*), 61
ST_BandFileTimestamp (*class in geoalchemy2.functions*), 61
ST_BandIsNoData (*class in geoalchemy2.functions*), 61
ST_BandMetaData (*class in geoalchemy2.functions*), 61
ST_BandNoDataValue (*class in geoalchemy2.functions*), 61
ST_BandPath (*class in geoalchemy2.functions*), 61
ST_BandPixelType (*class in geoalchemy2.functions*), 62
ST_BdMPolyFromText (*class in geoalchemy2.functions*), 62
ST_BdPolyFromText (*class in geoalchemy2.functions*), 62
ST_Boundary (*class in geoalchemy2.functions*), 62
ST_BoundingDiagonal (*class in geoalchemy2.functions*), 62
ST_Box2dFromGeoHash (*class in geoalchemy2.functions*), 62
ST_Buffer (*class in geoalchemy2.functions*), 62
ST_BuildArea (*class in geoalchemy2.functions*), 62
ST_Centroid (*class in geoalchemy2.functions*), 63
ST_ChaikinSmoothing (*class in geoalchemy2.functions*), 63
ST_Clip (*class in geoalchemy2.functions*), 63
ST_ClipByBox2D (*class in geoalchemy2.functions*), 63
ST_ClosestPoint (*class in geoalchemy2.functions*), 63
ST_ClosestPointOfApproach (*class in geoalchemy2.functions*), 63
ST_ClusterDBSCAN (*class in geoalchemy2.functions*), 64

ST_ClusterIntersecting (*class in geoalchemy2.functions*), 64
ST_ClusterKMeans (*class in geoalchemy2.functions*), 64
ST_ClusterWithin (*class in geoalchemy2.functions*), 64
ST_Collect (*class in geoalchemy2.functions*), 64
ST_CollectionExtract (*class in geoalchemy2.functions*), 64
ST_CollectionHomogenize (*class in geoalchemy2.functions*), 64
ST_ColorMap (*class in geoalchemy2.functions*), 64
ST_ConcaveHull (*class in geoalchemy2.functions*), 65
ST_ConstrainedDelaunayTriangles (*class in geoalchemy2.functions*), 65
ST_Contains (*class in geoalchemy2.functions*), 65
ST_ContainsProperly (*class in geoalchemy2.functions*), 65
ST_ConvexHull (*class in geoalchemy2.functions*), 65
ST_CoordDim (*class in geoalchemy2.functions*), 65
ST_Count (*class in geoalchemy2.functions*), 65
ST_CountAgg (*class in geoalchemy2.functions*), 66
ST_CoveredBy (*class in geoalchemy2.functions*), 66
ST_Covers (*class in geoalchemy2.functions*), 66
ST_CPAWithin (*class in geoalchemy2.functions*), 63
ST_Crosses (*class in geoalchemy2.functions*), 66
ST_CurveToLine (*class in geoalchemy2.functions*), 66
ST_DelaunayTriangles (*class in geoalchemy2.functions*), 66
ST_DFullyWithin (*class in geoalchemy2.functions*), 66
ST_Difference (*class in geoalchemy2.functions*), 66
ST_Dimension (*class in geoalchemy2.functions*), 67
ST_Disjoint (*class in geoalchemy2.functions*), 67
ST_Distance (*class in geoalchemy2.functions*), 67
ST_Distance_Sphere (*class in geoalchemy2.functions*), 67
ST_DistanceCPA (*class in geoalchemy2.functions*), 67
ST_DistanceSphere (*class in geoalchemy2.functions*), 67
ST_DistanceSpheroid (*class in geoalchemy2.functions*), 67
ST_Distinct4ma (*class in geoalchemy2.functions*), 67
ST_Dump (*class in geoalchemy2.functions*), 67
ST_DumpAsPolygons (*class in geoalchemy2.functions*), 67
ST_DumpPoints (*class in geoalchemy2.functions*), 68
ST_DumpRings (*class in geoalchemy2.functions*), 68
ST_DumpValues (*class in geoalchemy2.functions*), 68
ST_DWithin (*class in geoalchemy2.functions*), 66
ST_EndPoint (*class in geoalchemy2.functions*), 68
ST_Envelope (*class in geoalchemy2.functions*), 68
ST_Equals (*class in geoalchemy2.functions*), 68
ST_EstimatedExtent (*class in geoalchemy2.functions*), 68
ST_Expand (*class in geoalchemy2.functions*), 68
ST_Extent (*class in geoalchemy2.functions*), 68
ST_ExteriorRing (*class in geoalchemy2.functions*), 69
ST_Extrude (*class in geoalchemy2.functions*), 69
ST_FilterByM (*class in geoalchemy2.functions*), 69
ST_FlipCoordinates (*class in geoalchemy2.functions*), 69
ST_Force2D (*class in geoalchemy2.functions*), 69
ST_Force3D (*class in geoalchemy2.functions*), 69
ST_Force3DM (*class in geoalchemy2.functions*), 69
ST_Force3DZ (*class in geoalchemy2.functions*), 70
ST_Force4D (*class in geoalchemy2.functions*), 70
ST_ForceCollection (*class in geoalchemy2.functions*), 70
ST_ForceCurve (*class in geoalchemy2.functions*), 70
ST_ForceLHR (*class in geoalchemy2.functions*), 70
ST_ForcePolygonCCW (*class in geoalchemy2.functions*), 70
ST_ForcePolygonCW (*class in geoalchemy2.functions*), 71
ST_ForceRHR (*class in geoalchemy2.functions*), 71
ST_ForceSFS (*class in geoalchemy2.functions*), 71
ST_FrechetDistance (*class in geoalchemy2.functions*), 71
ST_FromGDALRaster (*class in geoalchemy2.functions*), 71
ST_GeneratePoints (*class in geoalchemy2.functions*), 71
ST_GeogFromText (*class in geoalchemy2.functions*), 72
ST_GeogFromWKB (*class in geoalchemy2.functions*), 72
ST_GeographyFromText (*class in geoalchemy2.functions*), 72
ST_GeoHash (*class in geoalchemy2.functions*), 72
ST_GeomCollFromText (*class in geoalchemy2.functions*), 72
ST_GeometricMedian (*class in geoalchemy2.functions*), 74
ST_GeometryFromText (*class in geoalchemy2.functions*), 74
ST_GeometryN (*class in geoalchemy2.functions*), 74
ST_GeometryType (*class in geoalchemy2.functions*), 74
ST_GeomFromEWKB (*class in geoalchemy2.functions*), 72
ST_GeomFromEWKT (*class in geoalchemy2.functions*), 72

ST_GeomFromGeoHash (*class in geoalchemy2.functions*), 73
ST_GeomFromGeoJSON (*class in geoalchemy2.functions*), 73
ST_GeomFromGML (*class in geoalchemy2.functions*), 73
ST_GeomFromKML (*class in geoalchemy2.functions*), 73
ST_GeomFromText (*class in geoalchemy2.functions*), 73
ST_GeomFromTWKB (*class in geoalchemy2.functions*), 73
ST_GeomFromWKB (*class in geoalchemy2.functions*), 74
ST_GeoReference (*class in geoalchemy2.functions*), 72
ST_GMLToSQL (*class in geoalchemy2.functions*), 71
ST_Grayscale (*class in geoalchemy2.functions*), 74
ST_HasArc (*class in geoalchemy2.functions*), 74
ST_HasNoBand (*class in geoalchemy2.functions*), 74
ST_HausdorffDistance (*class in geoalchemy2.functions*), 75
ST_Height (*class in geoalchemy2.functions*), 75
ST_HillShade (*class in geoalchemy2.functions*), 75
ST_Histogram (*class in geoalchemy2.functions*), 75
ST_InteriorRingN (*class in geoalchemy2.functions*), 75
ST_InterpolatePoint (*class in geoalchemy2.functions*), 75
ST_Intersection (*class in geoalchemy2.functions*), 75
ST_Intersects (*class in geoalchemy2.functions*), 75
ST_InvDistWeight4ma (*class in geoalchemy2.functions*), 76
ST_IsClosed (*class in geoalchemy2.functions*), 76
ST_IsCollection (*class in geoalchemy2.functions*), 76
ST_IsEmpty (*class in geoalchemy2.functions*), 76
ST_IsPlanar (*class in geoalchemy2.functions*), 76
ST_IsPolygonCCW (*class in geoalchemy2.functions*), 76
ST_IsPolygonCW (*class in geoalchemy2.functions*), 76
ST_IsRing (*class in geoalchemy2.functions*), 76
ST_IsSimple (*class in geoalchemy2.functions*), 76
ST_IsSolid (*class in geoalchemy2.functions*), 76
ST_IsValid (*class in geoalchemy2.functions*), 76
ST_IsValidDetail (*class in geoalchemy2.functions*), 76
ST_IsValidReason (*class in geoalchemy2.functions*), 76
ST_IsValidTrajectory (*class in geoalchemy2.functions*), 77
ST_Length (*class in geoalchemy2.functions*), 77
ST_Length2D (*class in geoalchemy2.functions*), 77
ST_LengthSpheroid (*class in geoalchemy2.functions*), 77
ST_LineCrossingDirection (*class in geoalchemy2.functions*), 77
ST_LineFromEncodedPolyline (*class in geoalchemy2.functions*), 77
ST_LineFromMultiPoint (*class in geoalchemy2.functions*), 77
ST_LineFromText (*class in geoalchemy2.functions*), 77
ST_LineFromWKB (*class in geoalchemy2.functions*), 77
ST_LineInterpolatePoint (*class in geoalchemy2.functions*), 78
ST_LineInterpolatePoints (*class in geoalchemy2.functions*), 78
ST_LineLocatePoint (*class in geoalchemy2.functions*), 78
ST_LineMerge (*class in geoalchemy2.functions*), 78
ST_LinestringFromWKB (*class in geoalchemy2.functions*), 78
ST_LineSubstring (*class in geoalchemy2.functions*), 78
ST_LineToCurve (*class in geoalchemy2.functions*), 78
ST_LocateAlong (*class in geoalchemy2.functions*), 79
ST_LocateBetween (*class in geoalchemy2.functions*), 79
ST_LocateBetweenElevations (*class in geoalchemy2.functions*), 79
ST_LongestLine (*class in geoalchemy2.functions*), 79
ST_M (*class in geoalchemy2.functions*), 79
ST_MakeBox2D (*class in geoalchemy2.functions*), 80
ST_MakeEmptyCoverage (*class in geoalchemy2.functions*), 80
ST_MakeEmptyRaster (*class in geoalchemy2.functions*), 80
ST_MakeEnvelope (*class in geoalchemy2.functions*), 80
ST_MakeLine (*class in geoalchemy2.functions*), 80
ST_MakePoint (*class in geoalchemy2.functions*), 80
ST_MakePointM (*class in geoalchemy2.functions*), 81
ST_MakePolygon (*class in geoalchemy2.functions*), 81
ST_MakeSolid (*class in geoalchemy2.functions*), 81
ST_MakeValid (*class in geoalchemy2.functions*), 81
ST_MapAlgebra (*class in geoalchemy2.functions*), 81
ST_MapAlgebraExpr (*class in geoalchemy2.functions*), 81
ST_MapAlgebraFct (*class in geoalchemy2.functions*), 82

ST_MapAlgebraFctNgb (*class in geoalchemy2.functions*), 82
ST_Max4ma (*class in geoalchemy2.functions*), 82
ST_MaxDistance (*class in geoalchemy2.functions*), 82
ST_Mean4ma (*class in geoalchemy2.functions*), 82
ST_MemSize (*class in geoalchemy2.functions*), 82
ST_MemUnion (*class in geoalchemy2.functions*), 82
ST_MetaData (*class in geoalchemy2.functions*), 82
ST_Min4ma (*class in geoalchemy2.functions*), 82
ST_MinConvexHull (*class in geoalchemy2.functions*), 83
ST_MinDist4ma (*class in geoalchemy2.functions*), 83
ST_MinimumBoundingCircle (*class in geoalchemy2.functions*), 83
ST_MinimumBoundingRadius (*class in geoalchemy2.functions*), 83
ST_MinimumClearance (*class in geoalchemy2.functions*), 83
ST_MinimumClearanceLine (*class in geoalchemy2.functions*), 83
ST_MinkowskiSum (*class in geoalchemy2.functions*), 83
ST_MinPossibleValue (*class in geoalchemy2.functions*), 83
ST_MLineFromText (*class in geoalchemy2.functions*), 79
ST_MPointFromText (*class in geoalchemy2.functions*), 79
ST_MPolyFromText (*class in geoalchemy2.functions*), 80
ST_Multi (*class in geoalchemy2.functions*), 83
ST_NDims (*class in geoalchemy2.functions*), 84
ST_NearestValue (*class in geoalchemy2.functions*), 84
ST_Neighborhood (*class in geoalchemy2.functions*), 84
ST_Node (*class in geoalchemy2.functions*), 84
ST_Normalize (*class in geoalchemy2.functions*), 84
ST_NotSameAlignmentReason (*class in geoalchemy2.functions*), 84
ST_NPoints (*class in geoalchemy2.functions*), 84
ST_NRings (*class in geoalchemy2.functions*), 84
ST_NumBands (*class in geoalchemy2.functions*), 84
ST_NumGeometries (*class in geoalchemy2.functions*), 85
ST_NumInteriorRing (*class in geoalchemy2.functions*), 85
ST_NumInteriorRings (*class in geoalchemy2.functions*), 85
ST_NumPatches (*class in geoalchemy2.functions*), 85
ST_NumPoints (*class in geoalchemy2.functions*), 85
ST_OffsetCurve (*class in geoalchemy2.functions*), 85
ST_OrderingEquals (*class in geoalchemy2.functions*), 85
ST_Orientation (*class in geoalchemy2.functions*), 85
ST_OrientedEnvelope (*class in geoalchemy2.functions*), 85
ST_Overlaps (*class in geoalchemy2.functions*), 85
ST_PatchN (*class in geoalchemy2.functions*), 85
ST_Perimeter (*class in geoalchemy2.functions*), 86
ST_Perimeter2D (*class in geoalchemy2.functions*), 86
ST_PixelAsCentroid (*class in geoalchemy2.functions*), 86
ST_PixelAsCentroids (*class in geoalchemy2.functions*), 86
ST_PixelAsPoint (*class in geoalchemy2.functions*), 86
ST_PixelAsPoints (*class in geoalchemy2.functions*), 86
ST_PixelAsPolygon (*class in geoalchemy2.functions*), 86
ST_PixelAsPolygons (*class in geoalchemy2.functions*), 86
ST_PixelHeight (*class in geoalchemy2.functions*), 87
ST_PixelOfValue (*class in geoalchemy2.functions*), 87
ST_PixelWidth (*class in geoalchemy2.functions*), 87
ST_Point (*class in geoalchemy2.functions*), 87
ST_PointFromGeoHash (*class in geoalchemy2.functions*), 87
ST_PointFromText (*class in geoalchemy2.functions*), 87
ST_PointFromWKB (*class in geoalchemy2.functions*), 87
ST_PointInsideCircle (*class in geoalchemy2.functions*), 87
ST_PointN (*class in geoalchemy2.functions*), 87
ST_PointOnSurface (*class in geoalchemy2.functions*), 88
ST_Points (*class in geoalchemy2.functions*), 88
ST_Polygon (*class in geoalchemy2.functions*), 88
ST_PolygonFromText (*class in geoalchemy2.functions*), 88
ST_Polygonize (*class in geoalchemy2.functions*), 88
ST_Project (*class in geoalchemy2.functions*), 88
ST_Quantile (*class in geoalchemy2.functions*), 89
ST_QuantizeCoordinates (*class in geoalchemy2.functions*), 89
ST_Range4ma (*class in geoalchemy2.functions*), 89
ST_RasterToWorldCoord (*class in geoalchemy2.functions*), 89
ST_RasterToWorldCoordX (*class in geoalchemy2.functions*), 89

ST_RasterToWorldCoordY (*class in geoalchemy2.functions*), 89
ST_RastFromHexWKB (*class in geoalchemy2.functions*), 89
ST_RastFromWKB (*class in geoalchemy2.functions*), 89
ST_Reclass (*class in geoalchemy2.functions*), 89
ST_Relate (*class in geoalchemy2.functions*), 90
ST_RelateMatch (*class in geoalchemy2.functions*), 90
ST_RemovePoint (*class in geoalchemy2.functions*), 90
ST_RemoveRepeatedPoints (*class in geoalchemy2.functions*), 90
ST_Resample (*class in geoalchemy2.functions*), 90
ST_Rescale (*class in geoalchemy2.functions*), 90
ST_Resize (*class in geoalchemy2.functions*), 90
ST_Reskew (*class in geoalchemy2.functions*), 91
ST_Retile (*class in geoalchemy2.functions*), 91
ST_Reverse (*class in geoalchemy2.functions*), 91
ST_Rotate (*class in geoalchemy2.functions*), 91
ST_RotateX (*class in geoalchemy2.functions*), 91
ST_RotateY (*class in geoalchemy2.functions*), 91
ST_RotateZ (*class in geoalchemy2.functions*), 92
ST_Rotation (*class in geoalchemy2.functions*), 92
ST_Roughness (*class in geoalchemy2.functions*), 92
ST_SameAlignment (*class in geoalchemy2.functions*), 92
ST_Scale (*class in geoalchemy2.functions*), 92
ST_ScaleX (*class in geoalchemy2.functions*), 92
ST_ScaleY (*class in geoalchemy2.functions*), 92
ST_Segmentize (*class in geoalchemy2.functions*), 92
ST_SetBandIndex (*class in geoalchemy2.functions*), 93
ST_SetBandIsNoData (*class in geoalchemy2.functions*), 93
ST_SetBandNoDataValue (*class in geoalchemy2.functions*), 93
ST_SetBandPath (*class in geoalchemy2.functions*), 93
ST_SetEffectiveArea (*class in geoalchemy2.functions*), 93
ST_SetGeoReference (*class in geoalchemy2.functions*), 93
ST_SetPoint (*class in geoalchemy2.functions*), 94
ST_SetRotation (*class in geoalchemy2.functions*), 94
ST_SetScale (*class in geoalchemy2.functions*), 94
ST_SetSkew (*class in geoalchemy2.functions*), 94
ST_SetSRID (*class in geoalchemy2.functions*), 94
ST_SetUpperLeft (*class in geoalchemy2.functions*), 94
ST_SetValue (*class in geoalchemy2.functions*), 94
ST_SetValues (*class in geoalchemy2.functions*), 95
ST_SharedPaths (*class in geoalchemy2.functions*), 95
ST_ShiftLongitude (*class in geoalchemy2.functions*), 95
ST_ShortestLine (*class in geoalchemy2.functions*), 95
ST_Simplify (*class in geoalchemy2.functions*), 95
ST_SimplifyPreserveTopology (*class in geoalchemy2.functions*), 95
ST_SimplifyVW (*class in geoalchemy2.functions*), 96
ST_SkewX (*class in geoalchemy2.functions*), 96
ST_SkewY (*class in geoalchemy2.functions*), 96
ST_Slope (*class in geoalchemy2.functions*), 96
ST_Snap (*class in geoalchemy2.functions*), 96
ST_SnapToGrid (*class in geoalchemy2.functions*), 96
ST_Split (*class in geoalchemy2.functions*), 96
ST_SRID (*class in geoalchemy2.functions*), 92
ST_StartPoint (*class in geoalchemy2.functions*), 96
ST_StdDev4ma (*class in geoalchemy2.functions*), 97
ST_StraightSkeleton (*class in geoalchemy2.functions*), 97
ST_Subdivide (*class in geoalchemy2.functions*), 97
ST_Sum4ma (*class in geoalchemy2.functions*), 97
ST_Summary (*class in geoalchemy2.functions*), 97
ST_SummaryStats (*class in geoalchemy2.functions*), 97
ST_SummaryStatsAgg (*class in geoalchemy2.functions*), 97
ST_SwapOrdinates (*class in geoalchemy2.functions*), 97
ST_SymDifference (*class in geoalchemy2.functions*), 98
ST_Tesselate (*class in geoalchemy2.functions*), 98
ST_Tile (*class in geoalchemy2.functions*), 98
ST_TileEnvelope (*class in geoalchemy2.functions*), 98
ST_Touches (*class in geoalchemy2.functions*), 99
ST_TPI (*class in geoalchemy2.functions*), 98
ST_Transform (*class in geoalchemy2.functions*), 99
ST_Translate (*class in geoalchemy2.functions*), 99
ST_TransScale (*class in geoalchemy2.functions*), 99
ST_TRI (*class in geoalchemy2.functions*), 98
ST_UnaryUnion (*class in geoalchemy2.functions*), 99
ST_Union (*class in geoalchemy2.functions*), 99
ST_UpperLeftX (*class in geoalchemy2.functions*), 99
ST_UpperLeftY (*class in geoalchemy2.functions*), 100
ST_Value (*class in geoalchemy2.functions*), 100
ST_ValueCount (*class in geoalchemy2.functions*), 100
ST_Volume (*class in geoalchemy2.functions*), 100
ST_VoronoiLines (*class in geoalchemy2.functions*), 100
ST_VoronoiPolygons (*class in geoalchemy2.functions*), 100
ST_Width (*class in geoalchemy2.functions*), 101

ST_Within (*class in geoalchemy2.functions*), 101
ST_WKBToSQL (*class in geoalchemy2.functions*), 100
ST_WKTToSQL (*class in geoalchemy2.functions*), 100
ST_WorldToRasterCoord (*class in geoalchemy2.functions*), 101
ST_WorldToRasterCoordX (*class in geoalchemy2.functions*), 101
ST_WorldToRasterCoordY (*class in geoalchemy2.functions*), 101
ST_WrapX (*class in geoalchemy2.functions*), 101
ST_X (*class in geoalchemy2.functions*), 101
ST_XMax (*class in geoalchemy2.functions*), 101
ST_XMin (*class in geoalchemy2.functions*), 101
ST_Y (*class in geoalchemy2.functions*), 101
ST_YMax (*class in geoalchemy2.functions*), 101
ST_YMin (*class in geoalchemy2.functions*), 102
ST_Z (*class in geoalchemy2.functions*), 102
ST_ZMax (*class in geoalchemy2.functions*), 102
ST_Zmflag (*class in geoalchemy2.functions*), 102
ST_ZMin (*class in geoalchemy2.functions*), 102
SummaryStats (*class in geoalchemy2.types*), 49

## T

TableRowElement (*class in geoalchemy2.functions*), 102
to_left() (*geoalchemy2.comparator.Comparator method*), 104
to_right() (*geoalchemy2.comparator.Comparator method*), 104
to_shape() (*in module geoalchemy2.shape*), 105
type (*geoalchemy2.functions.PostGIS_AddBBox attribute*), 55
type (*geoalchemy2.functions.PostGIS_DropBBox attribute*), 55
type (*geoalchemy2.functions.ST_3DClosestPoint attribute*), 56
type (*geoalchemy2.functions.ST_3DDifference attribute*), 56
type (*geoalchemy2.functions.ST_3DIntersection attribute*), 57
type (*geoalchemy2.functions.ST_3DLineInterpolatePoint attribute*), 57
type (*geoalchemy2.functions.ST_3DLongestLine attribute*), 57
type (*geoalchemy2.functions.ST_3DShortestLine attribute*), 58
type (*geoalchemy2.functions.ST_3DUnion attribute*), 58
type (*geoalchemy2.functions.ST_AddBand attribute*), 58
type (*geoalchemy2.functions.ST_AddMeasure attribute*), 58
type (*geoalchemy2.functions.ST_AddPoint attribute*), 58
type (*geoalchemy2.functions.ST_Affine attribute*), 58
type (*geoalchemy2.functions.ST_ApproximateMedialAxis attribute*), 58
type (*geoalchemy2.functions.ST_AsMVTGeom attribute*), 60
type (*geoalchemy2.functions.ST_Aspect attribute*), 61
type (*geoalchemy2.functions.ST_AsRaster attribute*), 60
type (*geoalchemy2.functions.ST_Band attribute*), 61
type (*geoalchemy2.functions.ST_BdMPolyFromText attribute*), 62
type (*geoalchemy2.functions.ST_BdPolyFromText attribute*), 62
type (*geoalchemy2.functions.ST_Boundary attribute*), 62
type (*geoalchemy2.functions.ST_BoundingDiagonal attribute*), 62
type (*geoalchemy2.functions.ST_Buffer attribute*), 62
type (*geoalchemy2.functions.ST_BuildArea attribute*), 63
type (*geoalchemy2.functions.ST_Centroid attribute*), 63
type (*geoalchemy2.functions.ST_ChaikinSmoothing attribute*), 63
type (*geoalchemy2.functions.ST_Clip attribute*), 63
type (*geoalchemy2.functions.ST_ClipByBox2D attribute*), 63
type (*geoalchemy2.functions.ST_ClosestPoint attribute*), 63
type (*geoalchemy2.functions.ST_ClusterIntersecting attribute*), 64
type (*geoalchemy2.functions.ST_ClusterWithin attribute*), 64
type (*geoalchemy2.functions.ST_Collect attribute*), 64
type (*geoalchemy2.functions.ST_CollectionExtract attribute*), 64
type (*geoalchemy2.functions.ST_CollectionHomogenize attribute*), 64
type (*geoalchemy2.functions.ST_ColorMap attribute*), 65
type (*geoalchemy2.functions.ST_ConcaveHull attribute*), 65
type (*geoalchemy2.functions.ST_ConstrainedDelaunayTriangles attribute*), 65
type (*geoalchemy2.functions.ST_ConvexHull attribute*), 65
type (*geoalchemy2.functions.ST_CurveToLine attribute*), 66
type (*geoalchemy2.functions.ST_DelaunayTriangles attribute*), 66
type (*geoalchemy2.functions.ST_Difference attribute*), 67
type (*geoalchemy2.functions.ST_Dump attribute*), 67
type (*geoalchemy2.functions.ST_DumpPoints attribute*), 68
type (*geoalchemy2.functions.ST_DumpRings attribute*),

68
type (geoalchemy2.functions.ST_EndPoint attribute), 68
type (geoalchemy2.functions.ST_Envelope attribute), 68
type (geoalchemy2.functions.ST_Expand attribute), 68
type (geoalchemy2.functions.ST_ExteriorRing attribute), 69
type (geoalchemy2.functions.ST_Extrude attribute), 69
type (geoalchemy2.functions.ST_FilterByM attribute), 69
type (geoalchemy2.functions.ST_FlipCoordinates attribute), 69
type (geoalchemy2.functions.ST_Force2D attribute), 69
type (geoalchemy2.functions.ST_Force3D attribute), 69
type (geoalchemy2.functions.ST_Force3DM attribute), 70
type (geoalchemy2.functions.ST_Force3DZ attribute), 70
type (geoalchemy2.functions.ST_Force4D attribute), 70
type (geoalchemy2.functions.ST_ForceCollection attribute), 70
type (geoalchemy2.functions.ST_ForceCurve attribute), 70
type (geoalchemy2.functions.ST_ForceLHR attribute), 70
type (geoalchemy2.functions.ST_ForcePolygonCCW attribute), 70
type (geoalchemy2.functions.ST_ForcePolygonCW attribute), 71
type (geoalchemy2.functions.ST_ForceRHR attribute), 71
type (geoalchemy2.functions.ST_ForceSFS attribute), 71
type (geoalchemy2.functions.ST_FromGDALRaster attribute), 71
type (geoalchemy2.functions.ST_GeneratePoints attribute), 71
type (geoalchemy2.functions.ST_GeogFromText attribute), 72
type (geoalchemy2.functions.ST_GeogFromWKB attribute), 72
type (geoalchemy2.functions.ST_GeographyFromText attribute), 72
type (geoalchemy2.functions.ST_GeomCollFromText attribute), 72
type (geoalchemy2.functions.ST_GeometricMedian attribute), 74
type (geoalchemy2.functions.ST_GeometryFromText attribute), 74
type (geoalchemy2.functions.ST_GeometryN attribute), 74
type (geoalchemy2.functions.ST_GeomFromEWKB attribute), 72
type (geoalchemy2.functions.ST_GeomFromEWKT attribute), 73
type (geoalchemy2.functions.ST_GeomFromGeoHash attribute), 73
type (geoalchemy2.functions.ST_GeomFromGeoJSON attribute), 73
type (geoalchemy2.functions.ST_GeomFromGML attribute), 73
type (geoalchemy2.functions.ST_GeomFromKML attribute), 73
type (geoalchemy2.functions.ST_GeomFromText attribute), 73
type (geoalchemy2.functions.ST_GeomFromTWKB attribute), 73
type (geoalchemy2.functions.ST_GeomFromWKB attribute), 74
type (geoalchemy2.functions.ST_GMLToSQL attribute), 71
type (geoalchemy2.functions.ST_Grayscale attribute), 74
type (geoalchemy2.functions.ST_HillShade attribute), 75
type (geoalchemy2.functions.ST_InteriorRingN attribute), 75
type (geoalchemy2.functions.ST_Intersection attribute), 75
type (geoalchemy2.functions.ST_LineFromEncodedPolyline attribute), 77
type (geoalchemy2.functions.ST_LineFromMultiPoint attribute), 77
type (geoalchemy2.functions.ST_LineFromText attribute), 77
type (geoalchemy2.functions.ST_LineFromWKB attribute), 77
type (geoalchemy2.functions.ST_LineInterpolatePoint attribute), 78
type (geoalchemy2.functions.ST_LineInterpolatePoints attribute), 78
type (geoalchemy2.functions.ST_LineMerge attribute), 78
type (geoalchemy2.functions.ST_LinestringFromWKB attribute), 79
type (geoalchemy2.functions.ST_LineSubstring attribute), 78
type (geoalchemy2.functions.ST_LineToCurve attribute), 78
type (geoalchemy2.functions.ST_LocateAlong attribute), 79
type (geoalchemy2.functions.ST_LocateBetween attribute), 79
type (geoalchemy2.functions.ST_LocateBetweenElevations attribute), 79
type (geoalchemy2.functions.ST_LongestLine attribute), 79
type (geoalchemy2.functions.ST_MakeEmptyCoverage

*attribute*), 80
type (*geoalchemy2.functions.ST_MakeEmptyRaster attribute*), 80
type (*geoalchemy2.functions.ST_MakeEnvelope attribute*), 80
type (*geoalchemy2.functions.ST_MakeLine attribute*), 80
type (*geoalchemy2.functions.ST_MakePoint attribute*), 81
type (*geoalchemy2.functions.ST_MakePointM attribute*), 81
type (*geoalchemy2.functions.ST_MakePolygon attribute*), 81
type (*geoalchemy2.functions.ST_MakeSolid attribute*), 81
type (*geoalchemy2.functions.ST_MakeValid attribute*), 81
type (*geoalchemy2.functions.ST_MapAlgebraExpr attribute*), 81
type (*geoalchemy2.functions.ST_MapAlgebraFct attribute*), 82
type (*geoalchemy2.functions.ST_MapAlgebraFctNgb attribute*), 82
type (*geoalchemy2.functions.ST_MemUnion attribute*), 82
type (*geoalchemy2.functions.ST_MinConvexHull attribute*), 83
type (*geoalchemy2.functions.ST_MinimumBoundingCircle attribute*), 83
type (*geoalchemy2.functions.ST_MinimumClearanceLine attribute*), 83
type (*geoalchemy2.functions.ST_MinkowskiSum attribute*), 83
type (*geoalchemy2.functions.ST_MLineFromText attribute*), 79
type (*geoalchemy2.functions.ST_MPointFromText attribute*), 80
type (*geoalchemy2.functions.ST_MPolyFromText attribute*), 80
type (*geoalchemy2.functions.ST_Multi attribute*), 84
type (*geoalchemy2.functions.ST_Node attribute*), 84
type (*geoalchemy2.functions.ST_Normalize attribute*), 84
type (*geoalchemy2.functions.ST_OffsetCurve attribute*), 85
type (*geoalchemy2.functions.ST_OrientedEnvelope attribute*), 85
type (*geoalchemy2.functions.ST_PatchN attribute*), 86
type (*geoalchemy2.functions.ST_PixelAsCentroid attribute*), 86
type (*geoalchemy2.functions.ST_PixelAsPoint attribute*), 86
type (*geoalchemy2.functions.ST_PixelAsPolygon attribute*), 86
type (*geoalchemy2.functions.ST_Point attribute*), 87
type (*geoalchemy2.functions.ST_PointFromGeoHash attribute*), 87
type (*geoalchemy2.functions.ST_PointFromText attribute*), 87
type (*geoalchemy2.functions.ST_PointFromWKB attribute*), 87
type (*geoalchemy2.functions.ST_PointN attribute*), 88
type (*geoalchemy2.functions.ST_PointOnSurface attribute*), 88
type (*geoalchemy2.functions.ST_Points attribute*), 88
type (*geoalchemy2.functions.ST_Polygon attribute*), 88
type (*geoalchemy2.functions.ST_PolygonFromText attribute*), 88
type (*geoalchemy2.functions.ST_Polygonize attribute*), 88
type (*geoalchemy2.functions.ST_Project attribute*), 88
type (*geoalchemy2.functions.ST_QuantizeCoordinates attribute*), 89
type (*geoalchemy2.functions.ST_RastFromHexWKB attribute*), 89
type (*geoalchemy2.functions.ST_RastFromWKB attribute*), 89
type (*geoalchemy2.functions.ST_Reclass attribute*), 90
type (*geoalchemy2.functions.ST_RemovePoint attribute*), 90
type (*geoalchemy2.functions.ST_RemoveRepeatedPoints attribute*), 90
type (*geoalchemy2.functions.ST_Resample attribute*), 90
type (*geoalchemy2.functions.ST_Rescale attribute*), 90
type (*geoalchemy2.functions.ST_Resize attribute*), 91
type (*geoalchemy2.functions.ST_Reskew attribute*), 91
type (*geoalchemy2.functions.ST_Retile attribute*), 91
type (*geoalchemy2.functions.ST_Reverse attribute*), 91
type (*geoalchemy2.functions.ST_Rotate attribute*), 91
type (*geoalchemy2.functions.ST_RotateX attribute*), 91
type (*geoalchemy2.functions.ST_RotateY attribute*), 92
type (*geoalchemy2.functions.ST_RotateZ attribute*), 92
type (*geoalchemy2.functions.ST_Roughness attribute*), 92
type (*geoalchemy2.functions.ST_Scale attribute*), 92
type (*geoalchemy2.functions.ST_Segmentize attribute*), 93
type (*geoalchemy2.functions.ST_SetBandIndex attribute*), 93
type (*geoalchemy2.functions.ST_SetBandIsNoData attribute*), 93
type (*geoalchemy2.functions.ST_SetBandNoDataValue attribute*), 93
type (*geoalchemy2.functions.ST_SetBandPath attribute*), 93
type (*geoalchemy2.functions.ST_SetEffectiveArea attribute*), 93

type (*geoalchemy2.functions.ST_SetGeoReference attribute*), 93
type (*geoalchemy2.functions.ST_SetPoint attribute*), 94
type (*geoalchemy2.functions.ST_SetRotation attribute*), 94
type (*geoalchemy2.functions.ST_SetScale attribute*), 94
type (*geoalchemy2.functions.ST_SetSkew attribute*), 94
type (*geoalchemy2.functions.ST_SetSRID attribute*), 94
type (*geoalchemy2.functions.ST_SetUpperLeft attribute*), 94
type (*geoalchemy2.functions.ST_SetValue attribute*), 95
type (*geoalchemy2.functions.ST_SetValues attribute*), 95
type (*geoalchemy2.functions.ST_SharedPaths attribute*), 95
type (*geoalchemy2.functions.ST_ShiftLongitude attribute*), 95
type (*geoalchemy2.functions.ST_ShortestLine attribute*), 95
type (*geoalchemy2.functions.ST_Simplify attribute*), 95
type (*geoalchemy2.functions.ST_SimplifyPreserveTopology attribute*), 96
type (*geoalchemy2.functions.ST_SimplifyVW attribute*), 96
type (*geoalchemy2.functions.ST_Slope attribute*), 96
type (*geoalchemy2.functions.ST_Snap attribute*), 96
type (*geoalchemy2.functions.ST_SnapToGrid attribute*), 96
type (*geoalchemy2.functions.ST_Split attribute*), 96
type (*geoalchemy2.functions.ST_StartPoint attribute*), 97
type (*geoalchemy2.functions.ST_StraightSkeleton attribute*), 97
type (*geoalchemy2.functions.ST_Subdivide attribute*), 97
type (*geoalchemy2.functions.ST_SummaryStatsAgg attribute*), 97
type (*geoalchemy2.functions.ST_SwapOrdinates attribute*), 98
type (*geoalchemy2.functions.ST_SymDifference attribute*), 98
type (*geoalchemy2.functions.ST_Tesselate attribute*), 98
type (*geoalchemy2.functions.ST_Tile attribute*), 98
type (*geoalchemy2.functions.ST_TileEnvelope attribute*), 98
type (*geoalchemy2.functions.ST_TPI attribute*), 98
type (*geoalchemy2.functions.ST_Transform attribute*), 99
type (*geoalchemy2.functions.ST_Translate attribute*), 99
type (*geoalchemy2.functions.ST_TransScale attribute*), 99
type (*geoalchemy2.functions.ST_TRI attribute*), 98
type (*geoalchemy2.functions.ST_UnaryUnion attribute*), 99
type (*geoalchemy2.functions.ST_Union attribute*), 99
type (*geoalchemy2.functions.ST_VoronoiLines attribute*), 100
type (*geoalchemy2.functions.ST_VoronoiPolygons attribute*), 100
type (*geoalchemy2.functions.ST_WKBToSQL attribute*), 100
type (*geoalchemy2.functions.ST_WKTToSQL attribute*), 101
type (*geoalchemy2.functions.ST_WrapX attribute*), 101
typemap (*geoalchemy2.types.CompositeType attribute*), 47
typemap (*geoalchemy2.types.GeometryDump attribute*), 48

## U

UnlockRows (*class in geoalchemy2.functions*), 102
UpdateGeometrySRID (*class in geoalchemy2.functions*), 102

## V

visit_drop_geospatial_table() (*in module geoalchemy2.alembic_helpers*), 109
visit_rename_geospatial_table() (*in module geoalchemy2.alembic_helpers*), 109

## W

WKBElement (*class in geoalchemy2.elements*), 52
WKTElement (*class in geoalchemy2.elements*), 51