SQLAlchemy

Set a database URL

```
from sqlalchemy.engine.url import URL

postgres_db = {'drivername': 'postgres',
               'username': 'postgres',
               'password': 'postgres',
               'host': '192.168.99.100',
               'port': 5432}
print(URL(**postgres_db))

sqlite_db = {'drivername': 'sqlite', 'database': 'db.sqlite'}
print(URL(**sqlite_db))
```

output:

```
$ python sqlalchemy_url.py
postgres://postgres:postgres@192.168.99.100:5432
sqlite:///db.sqlite
```

Sqlalchemy Support DBAPI - PEP249

```
from sqlalchemy import create_engine

db_uri = "sqlite:///db.sqlite"
engine = create_engine(db_uri)

# DBAPI - PEP249
# create table
engine.execute('CREATE TABLE "EX1" ('
               'id INTEGER NOT NULL,'
               'name VARCHAR, '
               'PRIMARY KEY (id));')
# insert a raw
engine.execute('INSERT INTO "EX1" '
               '(id, name) '
               'VALUES (1,"raw1")')

# select *
result = engine.execute('SELECT * FROM '
                        '"EX1"')
for _r in result:
   print(_r)

# delete *
engine.execute('DELETE from "EX1" where id=1;')
result = engine.execute('SELECT * FROM "EX1"')
print(result.fetchall())
```

Transaction and Connect Object

```
from sqlalchemy import create_engine

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create connection
conn = engine.connect()
# Begin transaction
trans = conn.begin()
conn.execute('INSERT INTO "EX1" (name) '
             'VALUES ("Hello")')
trans.commit()
# Close connection
conn.close()
```

Metadata - Generating Database Schema

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create a metadata instance
metadata = MetaData(engine)
# Declare a table
table = Table('Example',metadata,
              Column('id',Integer, primary_key=True),
              Column('name',String))
# Create all tables
metadata.create_all()
for _t in metadata.tables:
   print("Table: ", _t)
```

Inspect - Get Database Information

```
from sqlalchemy import create_engine
from sqlalchemy import inspect

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

inspector = inspect(engine)
```

```
# Get table information
print(inspector.get_table_names())

# Get column information
print(inspector.get_columns('EX1'))
```

Reflection - Loading Table from Existing Database

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create a MetaData instance
metadata = MetaData()
print(metadata.tables)

# reflect db schema to MetaData
metadata.reflect(bind=engine)
print(metadata.tables)
```

Print Create Table Statement with Indexes (SQL DDL)

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String

def metadata_dump(sql, *multiparams, **params):
    print(sql.compile(dialect=engine.dialect))

meta = MetaData()
example_table = Table('Example',meta,
                      Column('id', Integer, primary_key=True),
                      Column('name', String(10), index=True))

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri, strategy='mock', executor=metadata_dump)

meta.create_all(bind=engine, tables=[example_table])
```

output:

```
CREATE TABLE "Example" (
    id INTEGER NOT NULL,
```

```
    name VARCHAR(10),
    PRIMARY KEY (id)
)

CREATE INDEX "ix_Example_name" ON "Example" (name)
```

Get Table from MetaData

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create MetaData instance
metadata = MetaData(engine).reflect()
print(metadata.tables)

# Get Table
ex_table = metadata.tables['Example']
print(ex_table)
```

Create all Tables Store in “MetaData”

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
meta = MetaData(engine)

# Register t1, t2 to metadata
t1 = Table('EX1', meta,
           Column('id',Integer, primary_key=True),
           Column('name',String))

t2 = Table('EX2', meta,
           Column('id',Integer, primary_key=True),
           Column('val',Integer))
# Create all tables in meta
meta.create_all()
```

Create Specific Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
```

```
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

meta = MetaData(engine)
t1 = Table('Table_1', meta,
           Column('id', Integer, primary_key=True),
           Column('name',String))
t2 = Table('Table_2', meta,
           Column('id', Integer, primary_key=True),
           Column('val',Integer))
t1.create()
```

Create table with same columns

```
from sqlalchemy import (
    create_engine,
    inspect,
    Column,
    String,
    Integer)

from sqlalchemy.ext.declarative import declarative_base

db_url = "sqlite://"
engine = create_engine(db_url)

Base = declarative_base()

class TemplateTable(object):
    id   = Column(Integer, primary_key=True)
    name = Column(String)
    age  = Column(Integer)

class DowntownAPeople(TemplateTable, Base):
    __tablename__ = "downtown_a_people"

class DowntownBPeople(TemplateTable, Base):
    __tablename__ = "downtown_b_people"

Base.metadata.create_all(bind=engine)

# check table exists
ins = inspect(engine)
for _t in ins.get_table_names():
    print(_t)
```

Drop a Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import inspect
from sqlalchemy import Table
from sqlalchemy import Column, Integer, String
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
m = MetaData()
table = Table('Test', m,
              Column('id', Integer, primary_key=True),
              Column('key', String, nullable=True),
              Column('val', String))

table.create(engine)
inspector = inspect(engine)
print('Test' in inspector.get_table_names())

table.drop(engine)
inspector = inspect(engine)
print('Test' in inspector.get_table_names())
```

output:

```
$ python sqlalchemy_drop.py
$ True
$ False
```

Some Table Object Operation

```
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

meta = MetaData()
t = Table('ex_table', meta,
          Column('id', Integer, primary_key=True),
          Column('key', String),
          Column('val', Integer))
# Get Table Name
print(t.name)

# Get Columns
print(t.columns.keys())

# Get Column
c = t.c.key
```

```
print(c.name)
# Or
c = t.columns.key
print(c.name)

# Get Table from Column
print(c.table)
```

SQL Expression Language

```
# Think Column as "ColumnElement"
# Implement via overwrite special function
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String
from sqlalchemy import or_

meta = MetaData()
table = Table('example', meta,
              Column('id', Integer, primary_key=True),
              Column('l_name', String),
              Column('f_name', String))
# sql expression binary object
print(repr(table.c.l_name == 'ed'))
# exhbit sql expression
print(str(table.c.l_name == 'ed'))

print(repr(table.c.f_name != 'ed'))

# comparison operator
print(repr(table.c.id > 3))

# or expression
print((table.c.id > 5) | (table.c.id < 2))
# Equal to
print(or_(table.c.id > 5, table.c.id < 2))

# compare to None produce IS NULL
print(table.c.l_name == None)
# Equal to
print(table.c.l_name.is_(None))

# + means "addition"
print(table.c.id + 5)
# or means "string concatenation"
print(table.c.l_name + "some name")

# in expression
print(table.c.l_name.in_(['a','b']))
```

insert() - Create an “INSERT” Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# create table
meta = MetaData(engine)
table = Table('user', meta,
   Column('id', Integer, primary_key=True),
   Column('l_name', String),
   Column('f_name', String))
meta.create_all()

# insert data via insert() construct
ins = table.insert().values(
      l_name='Hello',
      f_name='World')
conn = engine.connect()
conn.execute(ins)

# insert multiple data
conn.execute(table.insert(),[
   {'l_name':'Hi','f_name':'bob'},
   {'l_name':'yo','f_name':'alice'}])
```

select() - Create a “SELECT” Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import select
from sqlalchemy import or_

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
conn = engine.connect()

meta = MetaData(engine).reflect()
table = meta.tables['user']

# select * from 'user'
select_st = select([table]).where(
   table.c.l_name == 'Hello')
res = conn.execute(select_st)
for _row in res:
    print(_row)

# or equal to
select_st = table.select().where(
```

```
    table.c.l_name == 'Hello')
res = conn.execute(select_st)
for _row in res:
    print(_row)

# combine with "OR"
select_st = select([
    table.c.l_name,
    table.c.f_name]).where(or_(
        table.c.l_name == 'Hello',
        table.c.l_name == 'Hi'))
res = conn.execute(select_st)
for _row in res:
    print(_row)

# combine with "ORDER_BY"
select_st = select([table]).where(or_(
        table.c.l_name == 'Hello',
        table.c.l_name == 'Hi')).order_by(table.c.f_name)
res = conn.execute(select_st)
for _row in res:
    print(_row)
```

join() - Joined Two Tables via “JOIN” Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy import select

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

meta = MetaData(engine).reflect()
email_t = Table('email_addr', meta,
        Column('id', Integer, primary_key=True),
        Column('email',String),
        Column('name',String))
meta.create_all()

# get user table
user_t = meta.tables['user']

# insert
conn = engine.connect()
conn.execute(email_t.insert(),[
    {'email':'ker@test','name':'Hi'},
    {'email':'yo@test','name':'Hello'}])
# join statement
join_obj = user_t.join(email_t,
            email_t.c.name == user_t.c.l_name)
# using select_from
```

```
sel_st = select(
   [user_t.c.l_name, email_t.c.email]).select_from(join_obj)
res = conn.execute(sel_st)
for _row in res:
    print(_row)
```

Fastest Bulk Insert in PostgreSQL via “COPY” Statement

```
# This method found here: https://gist.github.com/jsheedy/efa9a69926a754bebf0e9078
import io
from datetime import date

from sqlalchemy.engine.url import URL
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy import Date


db_url = {'drivername': 'postgres',
        'username': 'postgres',
        'password': 'postgres',
        'host': '192.168.99.100',
        'port': 5432}
engine = create_engine(URL(**db_url))

# create table
meta = MetaData(engine)
table = Table('userinfo', meta,
    Column('id', Integer, primary_key=True),
    Column('first_name', String),
    Column('age', Integer),
    Column('birth_day', Date),
)
meta.create_all()

# file-like object (tsv format)
datafile = io.StringIO()

# generate rows
for i in range(100):
    line = '\t'.join(
        [
            f'Name {i}',    # first_name
            str(18 + i),    # age
            str(date.today()),   # birth_day
        ]
    )
    datafile.write(line + '\n')
```

```
    # reset file to start
    datafile.seek(0)

    # bulk insert via `COPY` statement
    conn = engine.raw_connection()
    with conn.cursor() as cur:
        # https://www.psycopg.org/docs/cursor.html#cursor.copy_from
        cur.copy_from(
            datafile,
            table.name,  # table name
            sep='\t',
            columns=('first_name', 'age', 'birth_day'),
        )
    conn.commit()
```

Bulk PostgreSQL Insert and Return Inserted IDs

```
    from sqlalchemy.engine.url import URL
    from sqlalchemy import create_engine
    from sqlalchemy import MetaData
    from sqlalchemy import Table
    from sqlalchemy import Column
    from sqlalchemy import Integer
    from sqlalchemy import String

    db_url = {'drivername': 'postgres',
            'username': 'postgres',
            'password': 'postgres',
            'host': '192.168.99.100',
            'port': 5432}
    engine = create_engine(URL(**db_url))

    # create table
    meta = MetaData(engine)
    table = Table('userinfo', meta,
        Column('id', Integer, primary_key=True),
        Column('first_name', String),
        Column('age', Integer),
    )
    meta.create_all()

    # generate rows
    data = [{'first_name': f'Name {i}', 'age': 18+i} for i in range(10)]

    stmt = table.insert().values(data).returning(table.c.id)
    # converted into SQL:
    # INSERT INTO userinfo (first_name, age) VALUES
    #  (%(first_name_m0)s, %(age_m0)s), (%(first_name_m1)s, %(age_m1)s),
    #  (%(first_name_m2)s, %(age_m2)s), (%(first_name_m3)s, %(age_m3)s),
    #  (%(first_name_m4)s, %(age_m4)s), (%(first_name_m5)s, %(age_m5)s),
    #  (%(first_name_m6)s, %(age_m6)s), (%(first_name_m7)s, %(age_m7)s),
    #  (%(first_name_m8)s, %(age_m8)s), (%(first_name_m9)s, %(age_m9)s)
    # RETURNING userinfo.id
```

```
for rowid in engine.execute(stmt).fetchall():
    print(rowid['id'])
```

output:

```
$ python sqlalchemy_bulk.py
1
2
3
4
5
6
7
8
9
10
```

Update Multiple Rows

```
from sqlalchemy.engine.url import URL
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy.sql.expression import bindparam

db_url = {'drivername': 'postgres',
        'username': 'postgres',
        'password': 'postgres',
        'host': '192.168.99.100',
        'port': 5432}
engine = create_engine(URL(**db_url))

# create table
meta = MetaData(engine)
table = Table('userinfo', meta,
    Column('id', Integer, primary_key=True),
    Column('first_name', String),
    Column('birth_year', Integer),
)
meta.create_all()

# update data
data = [
    {'_id': 1, 'first_name': 'Johnny', 'birth_year': 1975},
    {'_id': 2, 'first_name': 'Jim', 'birth_year': 1973},
    {'_id': 3, 'first_name': 'Kaley', 'birth_year': 1985},
    {'_id': 4, 'first_name': 'Simon', 'birth_year': 1980},
    {'_id': 5, 'first_name': 'Kunal', 'birth_year': 1981},
    {'_id': 6, 'first_name': 'Mayim', 'birth_year': 1975},
    {'_id': 7, 'first_name': 'Melissa', 'birth_year': 1980},
]
```

```
stmt = table.update().where(table.c.id == bindparam('_id')).\
       values({
           'first_name': bindparam('first_name'),
           'birth_year': bindparam('birth_year'),
       })
# conveted to SQL:
# UPDATE userinfo SET first_name=%(first_name)s, birth_year=%(birth_year)s WHERE u

engine.execute(stmt, data)
```

Delete Rows from Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
conn = engine.connect()

meta = MetaData(engine).reflect()
user_t = meta.tables['user']

# select * from user_t
sel_st = user_t.select()
res = conn.execute(sel_st)
for _row in res:
    print(_row)

# delete l_name == 'Hello'
del_st = user_t.delete().where(
      user_t.c.l_name == 'Hello')
print('----- delete -----')
res = conn.execute(del_st)

# check rows has been delete
sel_st = user_t.select()
res = conn.execute(sel_st)
for _row in res:
    print(_row)
```

Check Table Existing

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Column
from sqlalchemy import Integer, String
from sqlalchemy import inspect
from sqlalchemy.ext.declarative import declarative_base

Modal = declarative_base()
```

```
class Example(Modal):
    __tablename__ = "ex_t"
    id = Column(Integer, primary_key=True)
    name = Column(String(20))

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
Modal.metadata.create_all(engine)

# check register table exist to Modal
for _t in Modal.metadata.tables:
    print(_t)

# check all table in database
meta = MetaData(engine).reflect()
for _t in meta.tables:
    print(_t)

# check table names exists via inspect
ins = inspect(engine)
for _t in ins.get_table_names():
    print(_t)
```

Create multiple tables at once

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import inspect
from sqlalchemy import Column, String, Integer
from sqlalchemy.engine.url import URL

db = {'drivername': 'postgres',
      'username': 'postgres',
      'password': 'postgres',
      'host': '192.168.99.100',
      'port': 5432}

url = URL(**db)
engine = create_engine(url)

metadata = MetaData()
metadata.reflect(bind=engine)

def create_table(name, metadata):
    tables = metadata.tables.keys()
    if name not in tables:
        table = Table(name, metadata,
                      Column('id', Integer, primary_key=True),
                      Column('key', String),
                      Column('val', Integer))
        table.create(engine)

tables = ['table1', 'table2', 'table3']
for _t in tables: create_table(_t, metadata)
```

```
inspector = inspect(engine)
print(inspector.get_table_names())
```

output:

```
$ python sqlalchemy_create.py
[u'table1', u'table2', u'table3']
```

Create tables with dynamic columns (Table)

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String
from sqlalchemy import Table
from sqlalchemy import MetaData
from sqlalchemy import inspect
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

engine = create_engine(URL(**db_url))

def create_table(name, *cols):
    meta = MetaData()
    meta.reflect(bind=engine)
    if name in meta.tables: return

    table = Table(name, meta, *cols)
    table.create(engine)

create_table('Table1',
             Column('id', Integer, primary_key=True),
             Column('name', String))
create_table('Table2',
             Column('id', Integer, primary_key=True),
             Column('key', String),
             Column('val', String))

inspector = inspect(engine)
for _t in inspector.get_table_names():
    print(_t)
```

output:

```
$ python sqlalchemy_dynamic.py
Table1
Table2
```

Object Relational add data

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))

Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

data = {'a': 5566, 'b': 9527, 'c': 183}
try:
    for _key, _val in data.items():
        row = TestTable(key=_key, val=_val)
        session.add(row)
    session.commit()
except SQLAlchemyError as e:
    print(e)
finally:
    session.close()
```

Object Relational update data

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
```

```
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

try:
    # add row to database
    row = TestTable(key="hello", val="world")
    session.add(row)
    session.commit()

    # update row to database
    row = session.query(TestTable).filter(
          TestTable.key == 'hello').first()
    print('original:', row.key, row.val)
    row.key = "Hello"
    row.val = "World"
    session.commit()

    # check update correct
    row = session.query(TestTable).filter(
          TestTable.key == 'Hello').first()
    print('update:', row.key, row.val)
except SQLAlchemyError as e:
    print(e)
finally:
    session.close()
```

output:

```
$ python sqlalchemy_update.py
original: hello world
update: Hello World
```

Object Relational delete row

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL


db_url = {'drivername': 'postgres',
        'username': 'postgres',
        'password': 'postgres',
        'host': '192.168.99.100',
        'port': 5432}
engine = create_engine(URL(**db_url))
Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

row = TestTable(key='hello', val='world')
session.add(row)
query = session.query(TestTable).filter(
        TestTable.key=='hello')
print(query.first())
query.delete()
query = session.query(TestTable).filter(
        TestTable.key=='hello')
print(query.all())
```

output:

```
$ python sqlalchemy_delete.py
<__main__.TestTable object at 0x104eb8f50>
[]
```

Object Relational relationship

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

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    addresses = relationship("Address", backref="user")

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    email = Column(String)
    user_id = Column(Integer, ForeignKey('user.id'))

u1 = User()
a1 = Address()
print(u1.addresses)
print(a1.user)

u1.addresses.append(a1)
print(u1.addresses)
print(a1.user)
```

output:

```
$ python sqlalchemy_relationship.py
[]
None
[<__main__.Address object at 0x10c4edb50>]
<__main__.User object at 0x10c4ed810>
```

Object Relational self association

```
import json

from sqlalchemy import (
    Column,
    Integer,
    String,
    ForeignKey,
    Table)

from sqlalchemy.orm import (
    sessionmaker,
    relationship)

from sqlalchemy.ext.declarative import declarative_base

base = declarative_base()
```

```
association = Table("Association", base.metadata,
    Column('left', Integer, ForeignKey('node.id'), primary_key=True),
    Column('right', Integer, ForeignKey('node.id'), primary_key=True))

class Node(base):
    __tablename__ = 'node'
    id = Column(Integer, primary_key=True)
    label = Column(String)
    friends = relationship('Node',
                           secondary=association,
                           primaryjoin=id==association.c.left,
                           secondaryjoin=id==association.c.right,
                           backref='left')
    def to_json(self):
        return dict(id=self.id,
                    friends=[_.label for _ in self.friends])

nodes = [Node(label='node_{}'.format(_)) for _ in range(0, 3)]
nodes[0].friends.extend([nodes[1], nodes[2]])
nodes[1].friends.append(nodes[2])

print('----> right')
print(json.dumps([_.to_json() for _ in nodes], indent=2))

print('----> left')
print(json.dumps([_n.to_json() for _n in nodes[1].left], indent=2))
```

output:

```
----> right
[
  {
    "friends": [
      "node_1",
      "node_2"
    ],
    "id": null
  },
  {
    "friends": [
      "node_2"
    ],
    "id": null
  },
  {
    "friends": [],
    "id": null
  }
]
----> left
[
  {
    "friends": [
      "node_1",
      "node_2"
```

```
    ],
    "id": null
  }
]
```

Object Relational basic query

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, String, Integer, DateTime
from sqlalchemy import or_
from sqlalchemy import desc
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

Base = declarative_base()

class User(Base):
    __tablename__ = 'User'
    id       = Column(Integer, primary_key=True)
    name     = Column(String, nullable=False)
    fullname = Column(String, nullable=False)
    birth    = Column(DateTime)

# create tables
engine = create_engine(URL(**db_url))
Base.metadata.create_all(bind=engine)

users = [
    User(name='ed',
         fullname='Ed Jones',
         birth=datetime(1989,7,1)),
    User(name='wendy',
         fullname='Wendy Williams',
         birth=datetime(1983,4,1)),
    User(name='mary',
         fullname='Mary Contrary',
         birth=datetime(1990,1,30)),
    User(name='fred',
         fullname='Fred Flinstone',
         birth=datetime(1977,3,12)),
    User(name='justin',
         fullname="Justin Bieber")]

# create session
Session = sessionmaker()
```

```
Session.configure(bind=engine)
session = Session()

# add_all
session.add_all(users)
session.commit()

print("----> order_by(id):")
query = session.query(User).order_by(User.id)
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> order_by(desc(id)):")
query = session.query(User).order_by(desc(User.id))
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> order_by(date):")
query = session.query(User).order_by(User.birth)
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> EQUAL:")
query = session.query(User).filter(User.id == 2)
_row = query.first()
print(_row.name, _row.fullname, _row.birth)

print("\n----> NOT EQUAL:")
query = session.query(User).filter(User.id != 2)
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> IN:")
query = session.query(User).filter(User.name.in_(['ed', 'wendy']))
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> NOT IN:")
query = session.query(User).filter(~User.name.in_(['ed', 'wendy']))
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> AND:")
query = session.query(User).filter(
        User.name=='ed', User.fullname=='Ed Jones')
_row = query.first()
print(_row.name, _row.fullname, _row.birth)

print("\n----> OR:")
query = session.query(User).filter(
        or_(User.name=='ed', User.name=='wendy'))
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> NULL:")
query = session.query(User).filter(User.birth == None)
for _row in query.all():
```

```
    print(_row.name, _row.fullname)

print("\n----> NOT NULL:")
query = session.query(User).filter(User.birth != None)
for _row in query.all():
    print(_row.name, _row.fullname)

print("\n----> LIKE")
query = session.query(User).filter(User.name.like('%ed%'))
for _row in query.all():
    print(_row.name, _row.fullname)
```

output:

```
----> order_by(id):
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> order_by(desc(id)):
justin Justin Bieber None
fred Fred Flinstone 1977-03-12 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
ed Ed Jones 1989-07-01 00:00:00

----> order_by(date):
fred Fred Flinstone 1977-03-12 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
ed Ed Jones 1989-07-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
justin Justin Bieber None

----> EQUAL:
wendy Wendy Williams 1983-04-01 00:00:00

----> NOT EQUAL:
ed Ed Jones 1989-07-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> IN:
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00

----> NOT IN:
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> AND:
ed Ed Jones 1989-07-01 00:00:00
```

```
----> OR:
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00

----> NULL:
justin Justin Bieber

----> NOT NULL:
ed Ed Jones
wendy Wendy Williams
mary Mary Contrary
fred Fred Flinstone

----> LIKE
ed Ed Jones
fred Fred Flinstone
```

mapper: Map `Table` to `class`

```
from sqlalchemy import (
    create_engine,
    Table,
    MetaData,
    Column,
    Integer,
    String,
    ForeignKey)

from sqlalchemy.orm import (
    mapper,
    relationship,
    sessionmaker)

# classical mapping: map "table" to "class"
db_url = 'sqlite://'
engine = create_engine(db_url)

meta = MetaData(bind=engine)

user = Table('User', meta,
             Column('id', Integer, primary_key=True),
             Column('name', String),
             Column('fullname', String),
             Column('password', String))

addr = Table('Address', meta,
             Column('id', Integer, primary_key=True),
             Column('email', String),
             Column('user_id', Integer, ForeignKey('User.id')))

# map table to class
class User(object):
    def __init__(self, name, fullname, password):
        self.name = name
        self.fullname = fullname
```

```
        self.password = password

class Address(object):
    def __init__(self, email):
        self.email = email

mapper(User, user, properties={
       'addresses': relationship(Address, backref='user')})
mapper(Address, addr)

# create table
meta.create_all()

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

u = User(name='Hello', fullname='HelloWorld', password='ker')
a = Address(email='hello@hello.com')
u.addresses.append(a)
try:
    session.add(u)
    session.commit()

    # query result
    u = session.query(User).filter(User.name == 'Hello').first()
    print(u.name, u.fullname, u.password)

finally:
    session.close()
```

output:

```
$ python map_table_class.py
Hello HelloWorld ker
```

Get table dynamically

```
from sqlalchemy import (
    create_engine,
    MetaData,
    Table,
    inspect,
    Column,
    String,
    Integer)

from sqlalchemy.orm import (
    mapper,
    scoped_session,
    sessionmaker)

db_url = "sqlite://"
```

```
engine = create_engine(db_url)
metadata = MetaData(engine)

class TableTemp(object):
    def __init__(self, name):
        self.name = name

def get_table(name):
    if name in metadata.tables:
        table = metadata.tables[name]
    else:
        table = Table(name, metadata,
                Column('id', Integer, primary_key=True),
                Column('name', String))
        table.create(engine)

    cls = type(name.title(), (TableTemp,), {})
    mapper(cls, table)
    return cls

# get table first times
t = get_table('Hello')

# get table secone times
t = get_table('Hello')

Session = scoped_session(sessionmaker(bind=engine))
try:
    Session.add(t(name='foo'))
    Session.add(t(name='bar'))
    for _ in Session.query(t).all():
        print(_.name)
except Exception as e:
    Session.rollback()
finally:
    Session.close()
```

output:

```
$ python get_table.py
foo
bar
```

Object Relational join two tables

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.engine.url import URL
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
```

```
class User(Base):
    __tablename__ = 'user'
    id    = Column(Integer, primary_key=True)
    name  = Column(String)
    addresses = relationship("Address", backref="user")

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    email = Column(String)
    user_id = Column(Integer, ForeignKey('user.id'))

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

# create engine
engine = create_engine(URL(**db_url))

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

user = User(name='user1')
mail1 = Address(email='user1@foo.com')
mail2 = Address(email='user1@bar.com')
user.addresses.extend([mail1, mail2])

session.add(user)
session.add_all([mail1, mail2])
session.commit()

query = session.query(Address, User).join(User)
for _a, _u in query.all():
    print(_u.name, _a.email)
```

output:

```
$ python sqlalchemy_join.py
user1 user1@foo.com
user1 user1@bar.com
```

join on relationship and group_by count

```
from sqlalchemy import (
    create_engine,
    Column,
    String,
```

```
    Integer,
    ForeignKey,
    func)

from sqlalchemy.orm import (
    relationship,
    sessionmaker,
    scoped_session)

from sqlalchemy.ext.declarative import declarative_base

db_url = 'sqlite://'
engine = create_engine(db_url)

Base = declarative_base()

class Parent(Base):
    __tablename__ = 'parent'
    id       = Column(Integer, primary_key=True)
    name     = Column(String)
    children = relationship('Child', back_populates='parent')

class Child(Base):
    __tablename__ = 'child'
    id        = Column(Integer, primary_key=True)
    name      = Column(String)
    parent_id = Column(Integer, ForeignKey('parent.id'))
    parent    = relationship('Parent', back_populates='children')

Base.metadata.create_all(bind=engine)
Session = scoped_session(sessionmaker(bind=engine))

p1 = Parent(name="Alice")
p2 = Parent(name="Bob")

c1 = Child(name="foo")
c2 = Child(name="bar")
c3 = Child(name="ker")
c4 = Child(name="cat")

p1.children.extend([c1, c2, c3])
p2.children.append(c4)

try:
    Session.add(p1)
    Session.add(p2)
    Session.commit()

    # count number of children
    q = Session.query(Parent, func.count(Child.id))\
               .join(Child)\
               .group_by(Parent.id)

    # print result
    for _p, _c in q.all():
        print('parent: {}, num_child: {}'.format(_p.name, _c))
```

```
    finally:
        Session.remove()
```

output:

```
$ python join_group_by.py
parent: Alice, num_child: 3
parent: Bob, num_child: 1
```

Create tables with dynamic columns (ORM)

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String
from sqlalchemy import inspect
from sqlalchemy.engine.url import URL
from sqlalchemy.ext.declarative import declarative_base

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

engine = create_engine(URL(**db_url))
Base = declarative_base()

def create_table(name, cols):
    Base.metadata.reflect(engine)
    if name in Base.metadata.tables: return

    table = type(name, (Base,), cols)
    table.__table__.create(bind=engine)

create_table('Table1', {
             '__tablename__': 'Table1',
             'id': Column(Integer, primary_key=True),
             'name': Column(String)})

create_table('Table2', {
             '__tablename__': 'Table2',
             'id': Column(Integer, primary_key=True),
             'key': Column(String),
             'val': Column(String)})

inspector = inspect(engine)
for _t in inspector.get_table_names():
    print(_t)
```

output:

```
$ python sqlalchemy_dynamic_orm.py
Table1
```

```
Table2
```

Close database connection

```
from sqlalchemy import (
    create_engine,
    event,
    Column,
    Integer)

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

engine = create_engine('sqlite://')
base = declarative_base()

@event.listens_for(engine, 'engine_disposed')
def receive_engine_disposed(engine):
    print("engine dispose")

class Table(base):
    __tablename__ = 'example table'
    id = Column(Integer, primary_key=True)

base.metadata.create_all(bind=engine)
session = sessionmaker(bind=engine)()

try:
    try:
        row = Table()
        session.add(row)
    except Exception as e:
        session.rollback()
        raise
    finally:
        session.close()
finally:
    engine.dispose()
```

output:

```
$ python db_dispose.py
engine dispose
```

Warning:

Be careful. Close *session* does not mean close database connection. SQLAlchemy *session* generally represents the *transactions*, not connections.

Cannot use the object after close the session

```
from __future__ import print_function

from sqlalchemy import (
    create_engine,
    Column,
    String,
    Integer)

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


url = 'sqlite://'
engine = create_engine(url)
base = declarative_base()

class Table(base):
    __tablename__ = 'table'
    id  = Column(Integer, primary_key=True)
    key = Column(String)
    val = Column(String)

base.metadata.create_all(bind=engine)
session = sessionmaker(bind=engine)()

try:
    t = Table(key="key", val="val")
    try:
        print(t.key, t.val)
        session.add(t)
        session.commit()
    except Exception as e:
        print(e)
        session.rollback()
    finally:
        session.close()

    print(t.key, t.val) # exception raise from here
except Exception as e:
    print("Cannot use the object after close the session")
finally:
    engine.dispose()
```

output:

```
$ python sql.py
key val
Cannot use the object after close the session
```

Hooks

```
from sqlalchemy import Column, String, Integer
from sqlalchemy import create_engine
from sqlalchemy import event
from sqlalchemy.orm import sessionmaker
from sqlalchemy.orm import scoped_session
from sqlalchemy.ext.declarative import declarative_base


Base = declarative_base()


class User(Base):
    __tablename__ = "user"
    id = Column(Integer, primary_key=True)
    name = Column(String)
    age = Column(Integer)


url = "sqlite:///:memory:"
engine = create_engine(url)
Base.metadata.create_all(bind=engine)
Session = sessionmaker(bind=engine)


@event.listens_for(User, "before_insert")
def before_insert(mapper, connection, user):
    print(f"before insert: {user.name}")


@event.listens_for(User, "after_insert")
def after_insert(mapper, connection, user):
    print(f"after insert: {user.name}")


try:
    session = scoped_session(Session)
    user = User(name="bob", age=18)
    session.add(user)
    session.commit()
except SQLAlchemyError as e:
    session.rollback()
finally:
    session.close()
```

This project tries to provide many snippets of Python code that make life easier.

Useful Links

pysheeet website
pysheeet @ GitHub
Issue Tracker

pysheeet as a PDF

Cheat Sheets

C/C++ cheat sheet

Table of Contents

SQLAlchemy
- Set a database URL
- Sqlalchemy Support DBAPI - PEP249
- Transaction and Connect Object
- Metadata - Generating Database Schema
- Inspect - Get Database Information
- Reflection - Loading Table from Existing Database
- Print Create Table Statement with Indexes (SQL DDL)
- Get Table from MetaData
- Create all Tables Store in “MetaData”
- Create Specific Table
- Create table with same columns
- Drop a Table
- Some Table Object Operation
- SQL Expression Language
- insert() - Create an “INSERT” Statement
- select() - Create a “SELECT” Statement
- join() - Joined Two Tables via “JOIN” Statement
- Fastest Bulk Insert in PostgreSQL via “COPY” Statement
- Bulk PostgreSQL Insert and Return Inserted IDs
- Update Multiple Rows
- Delete Rows from Table
- Check Table Existing
- Create multiple tables at once
- Create tables with dynamic columns (Table)
- Object Relational add data
- Object Relational update data
- Object Relational delete row
- Object Relational relationship
- Object Relational self association
- Object Relational basic query
- mapper: Map `Table` to `class`
- Get table dynamically
- Object Relational join two tables
- join on relationship and group_by count
- Create tables with dynamic columns (ORM)
- Close database connection
- Cannot use the object after close the session
- Hooks

Quick search