Sqlalchemy

Установка

Ядро

```
pip install sqlalchemy
```

Драйвер для postgres, mysql

```
pip install psycopg2
```

```
pip install psycopg2-binary
```

```
pip  install  pymysql
```

Подключение

```
from sqlalchemy import create_engine
engine = create_engine('postgresql+psycopg2://username:password@localhost:5432/mydb')
engine = create_engine('mysql+pymysql://cookiem:chip@mysql01.com/cookies', pool_recycle=3600)
engine = create_engine('sqlite:///cookies.db')
engine2 = create_engine('sqlite:///:memory:')
```

Доп. ключи через запятую:

echo	булево. Лог запросов. По-умолчанию False.
encoding	строка. По-умолчанию utf-8
isolation_level	уровень изоляции
pool_recycle	число секунд для переподключения, желательно выставлять 3600. При работе с mysql соединение может быть активным до 4 часов.

Создание engine не создает фактического соединения с БД.

```
connection = engine.connect()
```

Сырой запрос:

```
result = connection.execute("select * from orders").fetchall()
```

Структура фреймворка

Таблицы -> MetaData -> Engine -> Dialect -> DB

MetaData: объект, в котором таблицы, индексы,...

- Может получать информацию о существующих сейчас сущностях в БД
- Может хранить шаблоны именования индексов и ограничений (constraint). Т е перед началом проекта добавляется настройка именования, затем при создании/удалении все ок. Это словарь ограничение:шаблон

Ограничение или индекс	Описание
ix	обычный индекс
uq	уникальный индекс
ck	ограничение проверки
fk	foreign-ключ
pk	primary-ключ

```
from sqlalchemy import MetaData
convention = {
'all_column_names': lambda constraint, table: '_'.join([
column.name for column in constraint.columns.values()
]),
'ix': 'ix__%(table_name)s__%(all_column_names)s',
'uq': 'uq__%(table_name)s__%(all_column_names)s',
'ck': 'ck__%(table_name)s__%(all_column_names)s',
'fk': ('fk__%(table_name)s__%(all_column_names)s' '%(referred_table_name)s'),
'pk': 'pk__%(table_name)s'
}
metadata_obj = MetaData(naming_convention=convention)
```

- Должен быть инициализирован до обращения к нему в таблицах
- Изначально пустой объект, можно или вручную занести данные, или получить из базы.
- Получение информации об одной таблице по имени
  - Нельзя получить одновременно две таблицы. Либо одна, либо вся база
  - Нельзя (и одна таблица, и база) получить ограничения (CONSTRAINT), комментарии, триггеры, значения по умолчанию. Но можно вручную добавить данные. Но похоже проще импортировать описание.

```
from sqlalchemy import ForeignKeyConstraint
album.append_constraint(ForeignKeyConstraint(['ArtistId'], ['artist.ArtistId']))
```

  - при получении базы, имена таблиц с большой и маленькой буквы. Т е удваивается количество объектов.

```
engine = create_engine(...)
metadata = MetaData()
cookie_tbl = Table('cookies', metadata, autoload_with=engine)
s = select(cookie_tbl)
with engine.connect() as conn:
    m = conn.execute(s)
    print(m.keys())
```

- Получение информации обо всей базе

```
from sqlalchemy import create_engine, MetaData

engine = create_engine(...)
metadata = MetaData()
metadata.reflect(bind=engine)
for table in metadata.sorted_tables:
    print(table.name)
#Потом получить объект таблицы:
mytable = metadata.tables['mytable']
```

- Получение информации обо всей базе через ORM + Automap

```
from sqlalchemy.ext.automap import automap_base
from sqlalchemy import create_engine

Base = automap_base()
engine = create_engine('sqlite:///Chinook_Sqlite.sqlite')
Base.prepare(engine, reflect=True)
# данные о классах загружены.
#Например для получения списка классов:
Base.classes.keys()
Artist = Base.classes.Artist # создание классов
#Внешние связи - в свойстве <related_object>_collection
artist = session.query(Artist).first()
for album in artist.album_collection:
    print('{} - {}'.format(artist.Name, album.Title))
```

Engine: Скрывает пул подключений и диалект.
Dialect: Скрывает детали реализации в конкретной базе

Core: SQL в чистом виде
ORM: абстракции

Работа с данными

INSERT

Вариант 1:

```
ins = cookies.insert().values(
    cookie_name="chocolate chip",
    cookie_recipe_url="http://some.aweso.me/cookie/recipe.html",
    cookie_sku="CC01",
    quantity="12",
    unit_cost="0.50"
    )
    result = connection.execute(ins)
    print(result.inserted_primary_key)
```

Вариант 2:

```
from sqlalchemy import insert
ins = insert(cookies).values(
    cookie_name="chocolate chip",
    cookie_recipe_url="http://some.aweso.me/cookie/recipe.html",
    cookie_sku="CC01",
    quantity="12",
    unit_cost="0.50"
)
```

Вариант 3

```
ins = cookies.insert()
result = connection.execute(
    ins,
    cookie_name='dark chocolate chip',
    cookie_recipe_url='http://some.aweso.me/cookie/recipe_dark.html',
    cookie_sku='CC02',
    quantity='1',
    unit_cost='0.75'
)
result.inserted_primary_key
```

Вариант 4

```
inventory_list = [
    {
```

```
        'cookie_name': 'peanut butter',
        'cookie_recipe_url': 'http://some.aweso.me/cookie/peanut.html',
        'cookie_sku': 'PB01',
        'quantity': '24',
        'unit_cost': '0.25'
    },
    {
        'cookie_name': 'oatmeal raisin',
        'cookie_recipe_url': 'http://some.okay.me/cookie/raisin.html',
        'cookie_sku': 'EWW01',
        'quantity': '100',
        'unit_cost': '1.00'
    }
]
result = connection.execute(ins, inventory_list)
```

SELECT

Вариант 1

```
from sqlalchemy.sql import select
s = select([cookies])
rp = connection.execute(s)
results = rp.fetchall()
```

Вариант 2

```
s = cookies.select()
rp = connection.execute(s)
results = rp.fetchall()
```

Вариант 3

```
s = cookies.select()
rp = connection.execute(s)
for record in rp:
    print(record.cookie_name)
```

SELECT определенных столбцов

```
s = select([cookies.c.cookie_name, cookies.c.quantity])
```

ORDERING

```
s = select([cookies.c.cookie_name, cookies.c.quantity])
```

Прямая сортировка

```
s = s.order_by(cookies.c.quantity)
```

Обратная сортировка

```
s = s.order_by(desc(cookies.c.quantity))
```

```
rp = connection.execute(s)
```

LIMITING

```
s = select([cookies.c.cookie_name, cookies.c.quantity])
s = s.order_by(cookies.c.quantity)
s = s.limit(2)
rp = connection.execute(s)
```

Встроенные функции SQL

Сумма: func.sum

```
from sqlalchemy.sql import func
s = select([func.sum(cookies.c.quantity)])
rp = connection.execute(s)
print(rp.scalar())
```

Количество: func.count

```
s = select([func.count(cookies.c.cookie_name)])
rp = connection.execute(s)
record = rp.first()
print(record.keys()) # ключи могут быть разные.
print(record.count_1)
```

Название свойства: label

```
s = select([func.count(cookies.c.cookie_name).label('inventory_count')])
rp = connection.execute(s)
record = rp.first()
print(record.keys())
print(record.inventory_count)
```

WHERE

Их можно комбинировать, используется AND

```
s = select([cookies]).where(cookies.c.cookie_name == 'chocolate chip')
rp = connection.execute(s)
```

Варианты модификаторов:

- between(cleft, cright) Значение столбца между cleft и cright
- concat(column_two) Объединение column и column_two
- distinct() Находит только уникальные значения в столбце
- in_([list]) Только если значения столбца в списке
- is_(None) Проверка на пустые значения
- contains(string) Значение столбца содержит строку
- endswith(string) Оканчивается строкой, зависит от больших букв
- like(string) зависит от больших букв
- startswith(string) зависит от больших букв
- ilike(string) не зависит от больших букв
- Есть отрицательные модификаторы not<method>, исключение - метод isnot()

Пример для LIKE

```
s = select([cookies]).where(cookies.c.cookie_name.like('%chocolate%'))
rp = connection.execute(s)
for record in rp.fetchall():
    print(record.cookie_name)
```

Модификация полученных данных по шаблону
Вариант 1: к каждому значению столбца

```
s = select([cookies.c.cookie_name, 'SKU-' + cookies.c.cookie_sku]) - добавит строку 'SKU-'
```

Вариант 2: через функцию cast

```
from sqlalchemy import cast
s = select([cookies.c.cookie_name,
          cast((cookies.c.quantity * cookies.c.unit_cost),
               Numeric(12,2)).label('inv_cost')])
for row in connection.execute(s):
    print('{} - {}'.format(row.cookie_name, row.inv_cost))
```

Логические функции

```
from sqlalchemy import and_, or_, not_
s = select([cookies]).where(
    and_(
        cookies.c.quantity > 23,
        cookies.c.unit_cost < 0.40
    )
)
```

Извлечение данных

```
first_row = results[0]
first_row[1]
first_row.cookie_name
first_row[cookies.c.cookie_name]
```

Для rp есть следующие варианты, однако без limit() извлекаются все данные, затем одна строка:

- first() - Первая запись и закрытие соединения
- fetchone() - Одна запись и оставляет открытый курсор для последующих запросов
- scalar() - Одно значение если запрос возвращает одно значение в одной строке
- rp.keys() - список столбцов

Обновление данных

```
from sqlalchemy import update
u = update(cookies).where(cookies.c.cookie_name == "chocolate chip")
u = u.values(quantity=(cookies.c.quantity + 120))
result = connection.execute(u)
```

Удаление данных

```
from sqlalchemy import delete
u = delete(cookies).where(cookies.c.cookie_name == "dark chocolate chip")
result = connection.execute(u)
```

JOIN

```
columns = [orders.c.order_id, users.c.username, users.c.phone,
        cookies.c.cookie_name, line_items.c.quantity,
        line_items.c.extended_cost]
cookiemon_orders = select(columns)
cookiemon_orders = cookiemon_orders.select_from(orders.join(users).join(
                line_items).join(cookies)).where(users.c.username ==
                    'cookiemon')
result = connection.execute(cookiemon_orders).fetchall()
```

OUTER JOIN

Для получения обратной статистики

```
columns = [users.c.username, func.count(orders.c.order_id)]
all_orders = select(columns)
all_orders = all_orders.select_from(users.outerjoin(orders))
all_orders = all_orders.group_by(users.c.username)
result = connection.execute(all_orders).fetchall()
```

Есть поддержка ALIAS

Группировка данных

```
columns = [users.c.username, func.count(orders.c.order_id)]
all_orders = select(columns)
all_orders = all_orders.select_from(users.outerjoin(orders))
all_orders = all_orders.group_by(users.c.username)
result = connection.execute(all_orders).fetchall()
```

Объединение запросов по условию

```
def get_orders_by_customer(cust_name, shipped=None, details=False):
    columns = [orders.c.order_id, users.c.username, users.c.phone]
    joins = users.join(orders)
    if details:
        columns.extend([cookies.c.cookie_name, line_items.c.quantity,
                        line_items.c.extended_cost])
        joins = joins.join(line_items).join(cookies)
    cust_orders = select(columns)
    cust_orders = cust_orders.select_from(joins)



    cust_orders = cust_orders.where(users.c.username == cust_name)
    if shipped is not None:
        cust_orders = cust_orders.where(orders.c.shipped == shipped)
    result = connection.execute(cust_orders).fetchall()
    return result
get_orders_by_customer('cakeeater')
get_orders_by_customer('cakeeater', details=True)
get_orders_by_customer('cakeeater', shipped=True)
get_orders_by_customer('cakeeater', shipped=False)
get_orders_by_customer('cakeeater', shipped=False, details=True)
```

Revision #10
Created 22 July 2024 15:59:18 by Admin
Updated 22 October 2024 16:53:04 by Admin