ORM ?????

Таблица это класс с требованиями:

- Потомок объекта, возвращаемого функцией declarative_base
- Включает __tablename__ с именем таблицы
- Включает 1+ атрибутов, являющихся объектом Column
- При определении не включает имя столбца в конструкторе Column, имя столбца = имя атрибута
- 1+ атрибутов определяют первичный ключ
- __table_args__ свойства таблицы (ограничения,...)

```
from sqlalchemy import Table, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
□□□
□Base = declarative_base()
□
□class Cookie(Base):
□□__tablename__ = 'cookies'
□□__table_args__ = (CheckConstraint('quantity >= 0', name='quantity_positive'),)
□□cookie_id = Column(Integer(), primary_key=True)
□□cookie_name = Column(String(50), index=True)
□□quantity = Column(Integer())
```

Создание таблиц

```
from sqlalchemy import create_engine
from dataclasses import Base

engine = create_engine(...)
Base.metadata.create_all(engine)
```

Ограничения

```
__table_args__ = (ForeignKeyConstraint(['id'], ['other_table.id']), CheckConstraint(unit_cost >= 0.00', name='unit_cost_positive'))
```

Внешние связи

- Определяется столбец с ForeignKey
- Определяется дополнительный атрибут с relationship и необязательным backref
- При определении backref, relationship будет определен в указанном классе с указанным именем.

Один к одному:

```
cookie = relationship("Cookie", uselist=False)
```

Один ко многим:

```
user = relationship("User", backref=backref('orders'))
```

На себя (дерево) - неоднозначное решение.

```
class Employee(Base):
    __tablename__ = 'employees'
    id = Column(Integer(), primary_key=True)
    manager_id = Column(Integer(), ForeignKey('employees.id'))
    name = Column(String(255), nullable=False)
    manager = relationship("Employee", backref=backref('reports'), remote_side=[id])
```

Сессии

В ORM режиме, сессия упаковывает

- соединение с БД через engine и предоставляет словарь объектов, загруженных через сессию или ассоциированных с сессией
- транзакции, которые открыты до коммита сессии

Это похожая на хэш-систему, состоящую из списка объектов, таблиц и ключей. Сессия создается через sessionmaker, один раз. Соединяется с базой в момент необходимости

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

engine = create_engine('sqlite:///:memory:')
Session = sessionmaker(bind=engine)
session = Session()
```

Состояния объекта в сессии:

Transient	Объект не в сессии и не в БД
Pending	Объект добавлен в сессию add(), но не flush или commit
Persistent	Объект в сессии имеет связанную запись в БД
Detached	Объект больше не в сессии, но в БД есть соответствующая запись
Modified	Объект изменен

Просмотр состояния:

```
from sqlalchemy import inspect
insp = inspect(cc_cookie)
```

Отключение объекта от сессии:

```
session.expunge(cc_cookie)
```

Просмотр списка атрибутов и выяснение, что было модифицировано

```
for attr, attr_state in insp.attrs.items():
    if attr_state.history.has_changes():
        print('{}: {}'.format(attr, attr_state.value))
        print('History: {}\n'.format(attr_state.history))
```

Добавление данных

Создаем объект класса, добавляем в сессию и коммитим. Множественное добавление данных:

```
dcc = Cookie(...)
mcc = Cookie(...)
session.add(dcc)
session.add(mcc)
session.flush()
```

flush: не выполняет коммит и не завершает транзакцию, но получает id. Потом нужно в пределах сессии сделать commit. Commit в пределах чужой сессии не влияет. Дальше можно использовать объект.

Если дальше не нужно выполнять операции с объектами:

```
session.bulk_save_objects([dcc,mcc])
session.commit()
```

Получение данных

all()	все
first()	возвращает одну запись если она единственная и закрывает соединение
one()	возвращает одну запись и оставляет соединение. Аккуратно!
scalar()	возвращает одно значение если результат запроса одна строка с одним столбцом

```
cookies = session.query(Cookie).all()
print(cookies)
```

Если использовать через итератор, то без all:

```
for cookie in session.query(Cookie):
    print(cookie)
```

Получение определенных столбцов:

```
cookies = session.query(Cookie.cookie_id).all()
```

Сортировка:

```
session.query(Cookie).order_by(Cookie.quantity).all()
.order_by(desc(Cookie.quantity))
```

Ограничения через срезы или .limit(2)

Встроенные функции:

```
from sqlalchemy import func
inv_count = session.query(func.sum(Cookie.quantity)).scalar()
print(inv_count)
rec_count = session.query(func.count(Cookie.cookie_name)).first()
```

Метки: можно добавить для дальнейшего обращения

```
rec_count = session.query(func.count(Cookie.cookie_name).label('inventory_count')).first()
print(rec_count.inventory_count)
```

Фильтрация

```
record = session.query(Cookie).filter(cookie_name=='chocolate chip').first()
record = session.query(Cookie).filter_by(cookie_name='chocolate chip').first()
query = session.query(Cookie).filter(Cookie.cookie_name.like('%chocolate%'))
query = session.query(Cookie).filter(Cookie.quantity > 23, Cookie.unit_cost < 0.40)
```

Изменение строк при выводе

```
results = session.query(Cookie.cookie_name, 'SKU-' + Cookie.cookie_sku).all()
query = session.query(Cookie.cookie_name,
      cast((Cookie.quantity * Cookie.unit_cost),
           Numeric(12,2)).label('inv_cost'))
```

Join:

```
query = session.query(Order.order_id, User.username)
results = query.join(User).all()

query = session.query(User.username, func.count(Order.order_id))
query = query.outerjoin(Order).group_by(User.username)
```

Group:

```
query = session.query(User.username, func.count(Order.order_id))
query = query.outerjoin(Order).group_by(User.username)
```

Сырые запросы

```
from sqlalchemy import text
query = session.query(User).filter(text("username='cookiemon'"))
```

Обновление данных

Через объект

```
query = session.query(Cookie)
cc_cookie = query.filter(Cookie.cookie_name == "chocolate chip").first()
cc_cookie.quantity = cc_cookie.quantity + 120
session.commit()
```

Через метод update

```
query = session.query(Cookie)
query = query.filter(Cookie.cookie_name == "chocolate chip")
query.update({Cookie.quantity: Cookie.quantity - 20})
```

Удаление

```
session.delete(dcc_cookie)
session.commit()
```

Исключения

```
from sqlalchemy.orm.exc import MultipleResultsFound
try:
    results = session.query(Cookie).one()
except MultipleResultsFound as error:
    print('We found too many cookies... is that even possible?')
```

Транзакции

В ORM транзакция создается автоматически до очередного коммита.

```
session.add(order)
try:
    session.commit()
except IntegrityError as error:
    session.rollback()
```

Пример структурирования

db.py

```
from datetime import datetime
from sqlalchemy import (MetaData, Table, Column, Integer, Numeric, String,
                        DateTime, ForeignKey, Boolean, create_engine)
class DataAccessLayer:
    connection = None
    engine = None
    conn_string = None
    metadata = MetaData()
    cookies = Table('cookies',
                    metadata,
```

```
                    Column('cookie_id', Integer(), primary_key=True),
                    Column('cookie_name', String(50), index=True),
                    Column('cookie_recipe_url', String(255)),
                    Column('cookie_sku', String(55)),
                    Column('quantity', Integer()),
                    Column('unit_cost', Numeric(12, 2))
              )
□□□
    def db_init(self, conn_string):
        self.engine = create_engine(conn_string or self.conn_string)
        self.metadata.create_all(self.engine)
        self.connection = self.engine.connect()
dal = DataAccessLayer()
```

app.py

```
from db import dal
from sqlalchemy.sql import select

def get_orders_by_customer(cust_name, shipped=None, details=False):
    columns = [dal.orders.c.order_id, dal.users.c.username, dal.users.c.phone]
    joins = dal.users.join(dal.orders)
    if details:
        columns.extend([dal.cookies.c.cookie_name,
                        dal.line_items.c.quantity,
                        dal.line_items.c.extended_cost])
        joins = joins.join(dal.line_items).join(dal.cookies)
    cust_orders = select(columns)
    cust_orders = cust_orders.select_from(joins).where(
        dal.users.c.username == cust_name)
    if shipped is not None:
        cust_orders = cust_orders.where(dal.orders.c.shipped == shipped)
    return dal.connection.execute(cust_orders).fetchall()
```

test.py

```
import unittest
class TestApp(unittest.TestCase):

□@classmethod
```

```
    def setUpClass(cls):
        dal.db_init('sqlite:///:memory:')

def test_one(self):
    res = get_orders_by_customer('', False)
    self.assert_equal(res, [])
```

Revision #5
Created 23 July 2024 08:34:22 by Admin
Updated 7 April 2025 06:52:16 by Admin