Soporte a la Gestión de Datos con Programación Visual

SQL ALCHEMY

ORM OBJECT RELATIONAL MAPPING

- Se utilizan para manejar las bases de datos como colecciones de objetos
- Ayudan a eliminar trabajo repetitivo.
- SQLAlchemy es uno de los ORM más usados con Python
- Implementa los patrones
  - Data mapper
  - Unit of work
  - Identity map
- Versión actual: 1.2.7 (www.sqlalchemy.org)

ORM - MODELO DE DOMINIO

ORM: Object Relational Mapper

Presenta una forma de asociar:

- clases Python definidas por el usuario, con tablas de Bases de Datos,
- instancias de esas clases con filas de las correspondientes tablas.
- sincroniza cambios de estado en los objetos con las filas relacionadas. (Usa el patrón Unit of Work.)
- permite expresar querys en términos de clases definidas por el usuario.

PASOS A DAR

- Declarar el mapeo de la base de datos (Crea los metadatos)
- Crear clases para cada entidad
- Crear Engine para un determinado motor BD
- Relacionar Engine con metadatos
- Crear Session, relacionarla con Engine
- Usar la sesión para gestionar los datos

DECLARAR UN MAPEO

Primero hay que describir la Base de Datos en términos de objetos. Hay que configurar el ORM y por otra parte definir nuestras clases de mapeo. Tenemos que definir las tablas, fundamentalmente su nombre y nombres y tipos de datos de las columnas:

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

Base = declarative_base()              #--- clase padre para definir las tablas

class User(Base):
    __tablename__ = 'users‘                          #--- indispensable.
    id =            Column(Integer, primary_key=True)      #--- indispensable
    name =       Column(String)
    fullname =  Column(String)
    password = Column(String)
```

Este proceso genera Metadatos que relacionan los objetos con la base de datos

MAPEO CON TABLAS RELACIONADAS

```
Base = declarative_base()

class Persona(Base):
    __tablename__ = 'persona'          #--- nombre de la tabla
    id = Column(Integer, primary_key=True)
    nombre = Column(String(250), nullable=False)

class Domicilio(Base):
    __tablename__ = 'domicilio'
    id = Column(Integer, primary_key=True)
    calle = Column(String(250))
    calle_nro = Column(String(250))
    cod_postal = Column(String(250), nullable=False)
    persona_id = Column(Integer, ForeignKey('persona.id'))
    persona = relationship(Persona)        #--- Objeto persona relacionado con Domicilio


# Creamos una engine que almacena datos sqlite en la carpeta actual
engine = create_engine('sqlite:///sqlalchemy_ejemplo.db')

# Crea todas las tablas definidas. Es equivalente a "Create Table" en SQL
Base.metadata.create_all(engine)
```

TABLAS RELACIONADAS

Uno a muchos
Para obtener la lista de los registros hijos en children.

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child")

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
    parent_id = Column(Integer, ForeignKey('parent.id'))
```

Uno a muchos

Con relación bidireccional.
Se usa back_populates

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

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

ENGINE – RELACION CON B.DATOS

```
from sqlalchemy import create_engine
# se indica motor a emplear, nombre del archivo y modo de mostrar resultados:
engine = create_engine( 'sqlite:///:memory:', echo=True)
```

Engine es el objeto que actuará como interfaz con la base de datos. Es el corazón del ORM.

Usualmente no se usa el engine directamente, lo emplea el ORM internamente.

Nota: la conexión con la BD se intentará recién al recibir una orden concreta (lazy connecting).

CREATE_ENGINE - SINTAXIS

create_engine: (dialect+driver://username:password@host:port/database)

Ejemplos con mysql

```
# default
engine = create_engine('mysql://scott:tiger@localhost/foo')

# mysql-python
engine = create_engine('mysql+pymysql://root:1234@localhost/soporte')
```

Ejemplos con SQLite:

```
# sqlite://<nohostname>/<path>
# Si el  <path> es relativo:
engine = create_engine('sqlite:///foo.db')

#Para un path absoluto, las 3 barras  son seguidas por ese path:
engine = create_engine('sqlite:///C:\\path\\to\\foo.db')
#--- O bien, usando raw string:
engine = create_engine(r'sqlite:///C:\path\to\foo.db')

#Para usar SQLite en memoria, especificar una dirección vacía:
engine = create_engine('sqlite://')
```

EJEMPLO BÁSICO SIN ORM

```
import pymysql              # --- importar el conector correspondiente

#--- crear un objeto conexión con una base de datos:
cpar = “host='localhost', port=3306, user='root', passwd='1234', db='mydb‘ “
conn = pymysql.connect( cpar )

#--- crear un cursor:
cur = conn.cursor()

#--- usar el cursor creado para ejecutar distintos comandos SQL:
caux = “SELECT * FROM bancos"
cur.execute( caux )

obj = cur.fetchone()        #--- recuperar la información obtenida con una Select.   1 registro
print(cur.descripcion)      #--- campos de la tabla BANCOS:   descripcion,
print(cur.rowcount)

lista=cur.fetchall()        #--- recuperar la información obtenida con una Select.   Es una Lista con todos los registros.
for row in lista:
    print(row)

#--- cerrar todo …
cur.commit()
cur.close()
conn.close()
```

EJEMPLO BASICO CON ALCHEMY

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

Base = declarative_base()      # Metadatos

class Persona(Base):
    __tablename__ = 'persona'   # ----nombre de la tabla
    # Definimos las columnas de la tabla Persona
    id = Column(Integer, primary_key=True)
    nombre = Column(String(250), nullable=False)

# ---- datos sqlite en la carpeta actual
engine = create_engine('sqlite:///sqlalchemy_ejemplo0.db')
Base.metadata.bind = engine

#---- creamos una sesión para admin datos
DBSession = sessionmaker()
DBSession.bind = engine
session = DBSession()

def creaTabla():
    # Crea todas las tablas definidas en los metadatos
    Base.metadata.create_all(engine)
```

```
def insertaReg():
    oper = Persona()
    oper.nombre = 'Juan Carlos'
    session.add(oper)

    oper = Persona()
    oper.nombre = 'Miguel'
    session.add(oper)

    session.commit()      #---<<<<Graba

def consulta():
    lp = session.query(Persona).all()   #--- lista
    print('Lista de personas:')
    for p in lp:
        print('Persona: ', p.id, p.nombre)

if __name__ == '__main__':
    creaTabla()
    insertaReg()
    consulta()
```

Observar que se necesitan dos binds con Engine:

con los metadatos y con la factory de

CONSULTAS - SESSION.QUERY()

```
#--- Select de Personas
lp = session.query(Persona).all()
print('Lista de personas:')
for p in lp:
    print('Persona: ', p.id, p.nombre)

# select - Primera de la tabla
persona1 = session.query(Persona).first()
print('\nPrimera persona de la tabla:',  persona1.nombre)

# Buscamos todos los Domicilios de 1 persona
# select * from domicilio d, persona p where d.persona_id = p.id
sq = session.query(Domicilio).filter(Domicilio.persona == persona1).all()

print('\nTodos los cod. postales de los domicilios de la primer persona ')
for domic in sq:
    print('domicilio: ', domic.cod_postal, 'Persona:', domic.persona_id,
domic.persona.nombre)
```

SESSION.QUERY(...CLASE...)

***Algunos métodos**:*

.all() # lista: sq = session.query(Domicilio).all()

.first() # devuelve un solo objeto

.count() # n = session.query(Persona).count()

.filter(…condicion…)

ld = session.query(Domicilio).filter(Domicilio.persona == persona1).all()

.order_by()

lp = session.query(Persona).order_by(Persona.nombre).all()

.join()

ld =session.query(Domicilio).join(Persona).order_by(Persona.nombre).all()