Lección 20: Bases de datos con SQLAlchemy

Índice

Anterior

Introducción

Muchas aplicaciones utilizan bases de datos para almacenar y recumerar datos. En python hay muchas alternativas para trabajar con bases de datos, pero una de las mas extendidas es SQLAlchemy.

SQLAlchemy es un kit de herramientas SQL para Python, pero además es un ORM (Object Relational Mapper), que es precisamente donde radica su potencia. Permite trabajar con bases de datos desde python sin tener que escribir peticiones SQL.

ORM

Un ORM es una herramienta que permite trabajar con tablas de bases de datos como si fueran objetos de Python.

La relación habitual es que una tabla de base de datos se corresponde con una clase, las columnas de la tabla con los atributos de la clase, cada fila de la tabla con un objeto (instancia de la clase) y las *Foreign keys* con relaciones entre atributos de las clases.

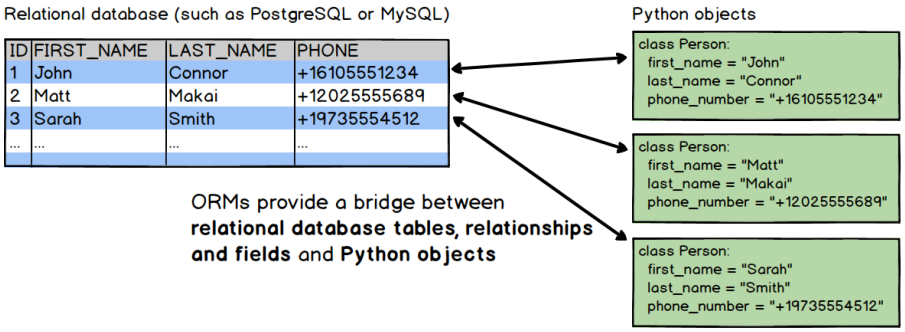

Figure 1: ORM mapping

¿Ventajas? - Acceder a tablas y datos de la base de datos como objetos y clases de python. - Casi no será necesario usar SQL, el ORM se hace cargo de eso. - Independencia de la base de datos. Se puede cambiar de base de datos con muy pocos cambios en el código.

SQLAlchemy

SQLAlchemy es una librería de Python que facilita el acceso a bases de datos realcionales, así como a las operaciones que se pueden llevar a cabo sobre ellas. Es independiente del motor de base de datos y es compatible con la mayoría de bases de datos conocidas: PostgreSQL, MySQL, Oracle, Microsoft SQL Server, Sqlite, ...

Con sQLAlchemy se puede trabajar utilizando peticiones SQL, pero su potencia radica en que también es un ORM. El ORM mapea las tablas a clases de Python y conviente automáticamente las llamadas a métodos de las clases en peticiones SQL.

SQLAlchemy proporciona una interfaz única para comunicarte con los diferentes drivers de bases de datos Python que implementan el estándar Python DBAPI. Al usar SQLAlchemy es necesario instalar también el driver/dialects que implemente la interfaz DBAPI para la base de datos que se vaya a utilizar.

Instalar SQLAlchemy

Se recomienda crear un entrono virtual e instalar SQLAlchemy dentro de ese entorno.

```
$ pip install sqlalchemy
```

Crear el engine

El primer paso es crear un *engine*, que viene a ser un motor de conexión con la base de datos. En este ejemplo creará un motor para conectar con una base de datos SQLite. Crear el engine no inicia la conexión, pero creo un Pool de conexiones *QeuePool*. En un modulo de python *db.py*.

```
# file: db.py
from sqlalchemy import create_engine

engine = create_engine('sqlite:///clientes.sqlite')
```

Sesiones

Con el *engine* creado, lo siguiente es crear una clase *Session* con el método *sessionmaker()* asociado a un engine e instanciar la clase.

```
# file: db.py
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

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

Modelos de base de datos (Tablas)

El el momento de crear los modelos, las clases en las que se mapearán las tablas de la base de datos. Las clases que representan a las tablas tienen que heredarse de una clase base especial, que se crea con otro método de factoría de clases *declarative_base()*:

```
# file: db.py
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///clientes.sqlite')
Session = sessionmaker(bind=engine)
session = Session()

Base = declarative_base() # Creamos la clase base de la que heredarán los modelos
```

La clase *Base* que se acaba de crear será la que hereden las clases que definen los modelos de la base de datos y que se traducirán automáticamente en tablas.

Los modelos se suelen crear en un módulo distinto, específico para definir los modelos. Por ejemplo: *models.py*.

```
# file: models.py
import db      # El módulo en el que se ha creado el engine, la sesión y la clase Base

from sqlalchemy import Column, Integer, String, Float

class Persona(db.Base):
    __tablename__ = 'Persona'
    pk = Column(Integer, primary_key=True)
    nombre = Column(String, nullable=False)
    apellido = Column(String, nullable=False)
    edad = Column(Integer,nullable=True)

    def __str__(self):
        return f"{self.nombre} {self.apellido}"

    def __repr__(self):
        return f"Persona: {self.nombre} {self.apellido}"
```

Se pueden añadir tantas tablas como sean necesarias para la base de datos.

Recrear la base de datos (crear las tablas)

Definidos los elementos que permiten la conexión en *db.py* y los modelos en *models.py*, se pueden crear las tablas correspondientes.

En un nuevo módulo de Python *main.py*:

```
#file: main.py
import db
from models import Persona

def cargar_datos():
    pass


if __name__ == '__main__':
    db.Base.metadata.create_all(db.engine) # Crea las tablas si no existen.
    cargar_datos()
```

Introducir datos en una tabla

```
import db
from models import Persona

def cargar_datos():
    """Documentar"""
    # TODO: Documentar la función
    with open('./datos.csv', encoding='utf-8') as f:
        lineas = f.readlines()
        for linea in lineas[1:]:
            nombre,apellido,edad = linea.split(sep=',')
            nuevo_cliente = Persona(nombre=nombre, apellido=apellido,edad=int(edad))
            print(nuevo_cliente.pk)
            db.session.add(nuevo_cliente)
            print(nuevo_cliente.pk)
            db.session.commit()
            print(nuevo_cliente.pk)
            print(nuevo_cliente)

if __name__ == '__main__':
    db.Base.metadata.create_all(db.engine) # Crea las tablas si no existen.
    cargar_datos()
```

Consultar datos de una tabla

```
import db
from models import Persona

def cargar_datos():
    """Documentar"""
    # TODO: Documentar la función
```

```
    with open('./datos.csv', encoding='utf-8') as f:
        lineas = f.readlines()
        for linea in lineas[1:]:
            nombre,apellido,edad = linea.split(sep=',')
            nuevo_cliente = Persona(nombre=nombre, apellido=apellido,edad=int(edad))
            print(nuevo_cliente.pk)
            db.session.add(nuevo_cliente)
            print(nuevo_cliente.pk)
            db.session.commit()
            print(nuevo_cliente.pk)
            print(nuevo_cliente)

def consultar_datos():
    """Documentar"""
    num_clientes = db.session.query(Persona).count()
    print(f"El total de clientes es {num_clientes}")
    clientes = db.session.query(Persona).all() # Obtener todos las filas de una tabla
    for cliente in clientes:
        print(cliente)
    #  Otras consultas
    print(db.session.query(Persona).filter_by(nombre='victor').first())
    print(db.session.query(Persona).filter(Persona.edad > 18).all())
    # Borrar una fila
    print(db.session.query(Persona).count())
    cliente = db.session.query(Persona).filter_by(nombre='victor').first()
    if cliente:
        db.session.delete(cliente)
    db.session.commit()
    print(db.session.query(Persona).count())



if __name__ == '__main__':
    db.Base.metadata.create_all(db.engine) # Crea las tablas si no existen.
    # cargar_datos()
    consultar_datos()
```

Siguiente