物件關聯對映(ORM)與資料庫連接

David Chiu

資料庫連結

關聯式資料庫

- 安全存儲、管理資料
  - 有效管理磁碟上的資料
- 保持資料的一致性
  - ACID 四原則
- 可以透過標準模型整合資料
  - 使用SQL 操作資料

常見開源關聯式資料庫

SQLite連結

SQLite

- 特性
  - self-contained
  - serverless
  - zero-configuration
  - Transactional
- ACID 資料庫
- 支援 SQL 92 語法
- 開源

使用Python 連結資料庫

```
import sqlite3 as lite
con = lite.connect('test.sqlite')
cur = con.cursor()
cur.execute('SELECT SQLITE_VERSION()')
data = cur.fetchone()
print(data)
con.close()
```

可以搭配 with 語句

透過SQLite 做資料新增、查詢

```
import sqlite3 as lite
with lite.connect("test.sqlite") as con:
    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS PhoneAddress")
    cur.execute("CREATE TABLE PhoneAddress(phone CHAR(10) PRIMARY KEY, address TEXT, name TEXT unique, age INT
NOT NULL)")
    cur.execute("INSERT INTO PhoneAddress VALUES('0912173381','United State','Jhon Doe',53)")
    cur.execute("INSERT INTO PhoneAddress VALUES('0928375018','Tokyo Japan','MuMu Cat',6)")
    cur.execute("INSERT INTO PhoneAddress VALUES('0957209108','Taipei','Richard',29)")
    cur.execute("SELECT phone,address FROM PhoneAddress")
    data = cur.fetchall()
    for rec in data:
        print(rec[0], rec[1])
```

fetchone v.s. fetchall

- fetchone

```
data = cur.fetchone()
print(data[0], data[1])
```

- fetchall

```
rows = cur.fetchall()
for row in rows:
    print(row)
```

使用Pandas 儲存資料

■ 建立DataFrame

```
import sqlite3 as lite
import pandas
employee = [{'name':'Mary', 'age':23 , 'gender': 'F'},{'name':'John',
'age':33 , 'gender': 'M'}]
df = pandas.DataFrame(employee)
```

■ 使用Pandas 儲存資料

```
with lite.connect('test.sqlite') as db:
    df.to_sql(name='employee', index=False, con=db,
if_exists='replace')
```

MySQL連結

MySQL

- MySQL
  - 關聯型資料庫管理系統,廣泛應用在網路服務上(LAMP)
  - 體積小、速度快、總體擁有成本低、開放原始碼、效能快捷、優化SQL語言、容易使用、多執行緒和可靠性、多使用者支援、可移植性和開放原始碼、遵守國際標準和國際化支援、為多種程式語言提供API

- 2019 年，哪些資料庫最受歡迎？
  - MySQL 以 38.9% 的使用率高居榜首，其後依次是 MongoDB（24.6%）、PostgreSQL（17.4%）

使用Python 連結 MySQL

■ 連接MySQL

```
import pymysql

conn = pymysql.connect(host='127.0.0.1', \
    user='david',password='1qaz@WSX', \
    db='test',charset='utf8', \
    use_unicode=True)

cur = conn.cursor()
sql = "SELECT * FROM test"
cur.execute(sql)

data = cur.fetchall()
print(data)

conn.close()
```

MySQL
安裝PyMySQL

pip install PyMySQL

PostgreSQL連結

PostgreSQL

- PostgreSQL是一套功能強大、開源的物件關聯資料庫系統，已開發超過15年，在系統可靠性、資料完整性和正確性獲得極佳評價

- PostgreSQL 是開源企業級資料庫系統， 具備先進資料庫系統功能，如多版本並行控制(Multi-Version Concurrency Control, MVCC), 時間點資料還原, 表格空間(tablespaces), 非同步資料複製(Asynchronous Replication), 巢狀交易(儲存點), 線上/熱備份, 複雜的查詢規劃器/優化, 支援容錯能力的優先寫入日誌檔

- PostgreSQL 是一套高擴展性的資料庫系統, 可處理大量數據資料, 可負荷大量使用者同時存取

PostgreSQL

使用Python 連結 PostgreSQL

- 連接PostgreSQL

```
import psycopg2

conn = psycopg2.connect(database='test')

cur = conn.cursor()
sql = "SELECT * FROM test"
cur.execute(sql)

data = cur.fetchall()
print(data)

conn.close()
```

PostgreSQL
安裝psycopg2

pip install psycopg2

ORM

Object-Relational Mapping

- Object-Relational Mapping (ORM)一詞就是將關聯式資料庫映射至物件導向的資料抽象化技術
- 讓程式開發人員可以用操作物件的方式對資料庫進行操作，而不直接使用SQL語法對資料庫進行操作
- 程式設計師不用管底層的資料庫系統是哪種廠牌或哪個版本的資料庫, 僅須用同一套語法撰寫存取資料庫的邏輯

為什麼要使用ORM

- 操作SQL語法的重複性
  - 自動生成增刪改查語句
- 大量SQL語句影響程序的擴展性和靈活性
  - 降低了物件導向程式與資料庫之間的耦合關係
- ORM核心原則
  - 簡單性
  - 傳達性
  - 精確性

ORM的優缺點

- 優點
  - 隱藏了數據訪問細節
  - 構造數據結構變得簡單
  - ORM可以防止SQL-Injection
  - 不用編碼，就可以操作資料庫
  - 提高開發效率
  - 方便轉移資料庫
- 缺點
  - 犧牲效能
  - 無法做複雜查詢

Python 常見 ORM 工具

Peewee v.s. SQLAlchemy

peewee

- 優點：
  類似Django式的API，容易和任意web框架搭配使用
- 缺點：
  不支援自動化 schema 遷移與建立

SQLAlchemy

- 優點：
  豐富的API，相當有彈性
  能輕鬆寫複雜查詢
- 缺點：
  稍微比較複雜，學習曲線長

SQLAlchemy

連結資料庫

```
import sqlalchemy
# SQLite
from sqlalchemy import create_engine
connect_info = 'sqlite:///:memory:'

# MySQL
connect_info = 'mysql+pymysql://david:1qaz@WSX@localhost:3306/test?charset=utf8'

# PostgreSQL
connect_info = create_engine('postgresql+psycopg2://david:1qaz@127.0.0.1:5432/test',
echo=True,client_encoding='utf8')

engine = create_engine(connect_info)
```

建立表格 (一)

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()

from sqlalchemy import Column, Integer, String
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)

    # Defines to_string() representation
    def __repr__(self):
        return "<User(name='%s', fullname='%s', password='%s')>" % (
                self.name, self.fullname, self.password)

Base.metadata.create_all(engine)
```

產生對應SQL

```
CREATE TABLE users (
    id INTEGER NOT NULL, name VARCHAR,
    fullname VARCHAR,
    password VARCHAR,
    PRIMARY KEY (id)
```

建立表格 (二)

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

metadata = MetaData()
user = Table('users', metadata,
  Column('id', Integer, primary_key=True),
  Column('name', String),
  Column('fullname', String),
  Column('password', String),
)
```

建立Session

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()
```

建立資料

```
duser = User(name='david', fullname='david chiu', password='edspassword')
session.add(duser)
duser
```

```
INSERT INTO users (name, fullname, password) VALUES ('david', 'david chiu', 'edspassword')
```

查詢資料

```
our_user = session.query(User).filter_by(name='david').first()
```

```
SELECT users.id AS users_id,
        users.name AS users_name,
        users.fullname AS users_fullname,
        users.password AS users_password
FROM users
WHERE users.name = 'david'
LIMIT 1 OFFSET 0
```

```
# 檢查兩者是否相同
duser is our_user
```

建立多筆資料

```
session.add_all([
        User(name='wendy', fullname='Wendy Williams', password='foobar'),
        User(name='mary', fullname='Mary Contrary', password='xxg527'),
        User(name='fred', fullname='Fred Flinstone', password='blah')])

duser.password = 'test123'
```

```
UPDATE users SET password='test123' WHERE users.id = 1

INSERT INTO users (name, fullname, password) VALUES ('wendy', 'Wendy Williams', 'foobar')
INSERT INTO users (name, fullname, password) VALUES ('mary', 'Mary Contrary', 'xxg527')
INSERT INTO users (name, fullname, password) VALUES ('fred', 'Fred Flinstone', 'blah')
```

Commit & Rollback

- Commit

```
session.commit()
```

Rollback

```
duser.name = 'davidson'

fake_user = User(name='fakeuser', fullname='Invalid', password='12345')
session.add(fake_user)

session.query(User).filter(User.name.in_(['davidson', 'fakeuser'])).all()
session.rollback()
```

查詢語句

```
for instance in session.query(User).order_by(User.id):
    print(instance)


# Select
for name, fullname in session.query(User.name, User.fullname):
    print(name, fullname)
```

條件查詢

```
# equal
session.query(User).filter(User.name == 'david')
# not equal
session.query(User).filter(User.name != 'david')
# like
session.query(User).filter(User.name.like('%david%'))
# in
session.query(User).filter(User.name.in_(['david', 'wendy', 'jack']))
# not in
session.query(User).filter(~User.name.in_(['david', 'wendy', 'jack']))
# or
session.query(User).filter(or_(User.name == 'david', User.name == 'wendy'))
```

關聯資料表

```
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship

class Address(Base):
    __tablename__ = 'addresses'
    id = Column(Integer, primary_key=True)
    email_address = Column(String, nullable=False)
    user_id = Column(Integer, ForeignKey('users.id'))
    user = relationship("User", back_populates="addresses")

    def __repr__(self):
        return "<Address(email_address='%s')>" % self.email_address
User.addresses = relationship("Address", order_by=Address.id, back_populates="user")

Base.metadata.create_all(engine)
```

建立與查詢關聯資訊

```
#建立使用者
jack = User(name='jack', fullname='Jack Bean', password='gjffdd')
jack.addresses

#建立關聯資料
jack.addresses = [Address(email_address='jack@google.com'), Address(email_address='j25@yahoo.com')]
jack.addresses[1]
jack.addresses[1].user
session.add(jack)
session.commit()

# 查詢關聯資料
jack = session.query(User).filter_by(name='jack').one()
jack.addresses
```

RAW Query

```
from sqlalchemy.sql import text
with engine.connect() as con:

    data = ( { "id": 1, "name": "u1", "fullname": "user1", "password": "p1" },
            { "id": 2, "name": "u2", "fullname": "user2", "password": "p2" },
    )
    statement = text("""INSERT INTO users(id, name, fullname,password) VALUES(:id, :name, :fullname, :password)""")

    for line in data:
        con.execute(statement, **line)
```

RAW Query

```
with engine.connect() as con:

    rs = con.execute('SELECT * FROM users')

    for row in rs:
        print row
```

搭配Pandas 操作

使用Pandas 讀寫資料

```
import pandas
from sqlalchemy import create_engine


connect_info = 'mysql+pymysql://david:1qaz@WSX@localhost:3306/test?charset=utf8'
engine = create_engine(connect_info)
sql = "SELECT * FROM users"
df = pandas.read_sql(sql=sql, con=engine)
print(df)
```

THANK YOU