Bases de datos
DB-API
SQLite3

Antonio Espín Herranz

Bases de datos

- En Python, como en otros lenguajes como Java con JDBC, existe una propuesta de API estándar para el manejo de bases de datos, de forma que el código sea prácticamente igual independientemente de la base de datos que estemos utilizando por debajo.

- Esta especificación recibe el nombre de Python Database API o **DB-API**

- La especificación se puede consultar en:
  - https://www.python.org/dev/peps/pep-0249/

- **Para MySQL tenemos:**
  - **http://mysql-python.sourceforge.net/**

Sqlite3 en Linux

- Para instalar sqlite3 en un terminal tecleamos el comando:

  sudo apt-get install sqlite3 libsqlite3-dev

- Nos pide la password y empieza a descargar y a instalar los paquetes.

- Para comprobar la instalación teclear en un terminal el comando: **sqlite3**

```
antonio@antonio:~$ sqlite3
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Variables globales

- Todos los drivers compatibles con DB-API 2.0 deben tener 3 variables globales que los describen:
  - **apilevel**: Versión de DB API que utiliza.
    - Valores: 1.0 o 2.0
  - **threadsafety**: Se trata de un entero de 0 a 3 que describe lo seguro que es el módulo para el uso con threads.
  - **paramstyle**: informa sobre la sintaxis a utilizar para insertar valores en la consulta SQL de forma dinámica.

Variables Globales

- Valores para **SQLite3**:

```
import sqlite3 as dbapi
print (dbapi.apilevel)
    2.0
print (dbapi.threadsafety)
    1
print (dbapi.paramstyle)
    qmark
```

threadsafety

- **Valores**:
  - **0**: no se puede compartir el módulo entre threads sin utilizar algún tipo de mecanismo de sincronización.
  - **1**: pueden compartir el módulo pero no las conexiones.
  - **2**: módulos y conexiones pero no cursores.
  - **3**: es totalmente ***thread-safe***.

paramstyle

- **qmark**: interrogaciones.
  - sql = “select all from t where valor=**?**”
- **numeric**: un número indicando la posición.
  - sql = “select all from t where valor=**:1**”
- **named**: el nombre del valor.
  - sql = “select all from t where valor=:valor”
- **format**: especificadores de formato similares a los del printf de C.
  - sql = “select all from t where valor=%s”
- **pyformat**: similar al anterior, pero con las extensiones de Python.
  - sql = “select all from t where valor=%(valor)”

Excepciones

- Jerarquía de excepciones:

```
StandardError
|__Warning
|__Error
   |__InterfaceError
   |__DatabaseError
      |__DataError
      |__OperationalError
      |__IntegrityError
      |__InternalError
      |__ProgrammingError
      |__NotSupportedError
```

Excepciones I

- **StandardError**:
  - Super clase para todas las excepciones de DB API.
- **Warning**:
  - Excepción que se lanza para avisos importantes.
- **Error**:
  - Super clase de los errores.
- **InterfaceError**:
  - Errores relacionados con la interfaz de la base de datos, y no con la base de datos en sí.
- **DatabaseError**:
  - Errores relacionados con la base de datos.
- **DataError**:
  - Errores relacionados con los datos, como una división entre cero.

Excepciones II

- **OperationalError**:
  - Errores relacionados con el funcionamiento de la base de datos, como una desconexión inesperada.
- **IntegrityError**:
  - Errores relacionados con la integridad referencial.
- **InternalError**:
  - Error interno de la base de datos.
- **ProgrammingError**:
  - Errores de programación, como errores en el código SQL.
- **NotSupportedError**:
  - Excepción que se lanza cuando se solicita un método que no está soportado por la base de datos.

DB-API

- Para trabajar con la BD necesitamos crear una conexión.

- Dependiendo de la BD elegida se necesitarán más o menos parámetros.

- En el caso de **SQLite3** tenemos **dos opciones**:
  - Indicar un **fichero** donde guardar la BD.
  - O la cadena **:memory:** para utilizar la RAM y no grabarla en disco.

- En el caso de **MySQL** indicar:
  - Servidor, puerto, usuario, password y el nombre de la BD.

DB-API

- **Pasos**:
  - Crear la conexión.

    bbdd = dbapi.connect(“bbdd.dat”)
    - Con el método connect obtenemos la conexión e indicaremos la ruta a un fichero o :memory:.

  - Las operaciones contra la BD (básicamente enviar comandos de SQL DDL o DML). Se realizan a través de un cursor que se obtiene a partir de la conexión.
    - **c = bbdd.cursor()**

  - A partir del cursor podemos lanzar sentencias SQL a la BD.
    - **c.execute(“””sentencia SQL “””)**
    - Utilizar **docstring** para partir la sentencia.

DB-API

- **Transacciones**:
  - Si **auto-commit** está desactivada es necesario llamar al método **commit()** para hacer permanentes los cambios.
  - Posibilidad de deshacer con el método **rollback()**.

- Si nuestra base de datos soporta transacciones, si estas están activadas, y si la característica de *auto-commit está desactivada, será necesario llamar al método commit de la conexión para que se lleven a cabo las operaciones definidas en la transacción.*

Ejemplo

```
import sqlite3 as dbapi
bbdd = dbapi.connect(“bbdd.dat”)
cursor = bbdd.cursor()

cursor.execute(“””create table empleados (dni text,nombre text, departamento
    text)”””)
cursor.execute(“””insert into empleados values (‘12345678-A’, ‘Manuel Gil’,
    ‘Contabilidad’)”””)
bbdd.commit()

cursor.execute(“””select * from empleados where departamento= ’Contabilidad’”””)

for tupla in cursor.fetchall():
      print tupla

cursor.close()
bbdd.close()
```

DB-API

- Métodos de **cursor**:
  - **fetchone**
    - Devuelve la siguiente tupla del conjunto resultado o None cuando no existen más tuplas,
  - **fetchmany**
    - Devuelve el número de tuplas indicado por el entero pasado como parámetro o bien el número indicado por el atributo Cursor.arraysize si no se pasa ningún parámetro (Cursor.arraysize vale 1 por defecto)
  - **fetchall**
    - Devuelve un objeto iterable con todas las tuplas.

Ataques de inyección SQL

- Por ejemplo, suponiendo una URL:
  - http://www.mibanco.com/sucursales?ciudad=Madrid

  – Podríamos tener por detrás la siguiente consulta.
    - cursor.execute(“””select * from sucursales where ciudad=’” + ciudad + “’””””)
    - Podrían lanzarnos una query en la misma petición.
    - “http://www.mibanco.com/sucursales?ciudad=Madrid**’;SELECT * FROM contrasenyas**

- **La solución es utilizar consultas parametrizadas.**

Consultas parametrizadas

- Teniendo en cuenta que la variable **paramstyle** para SQLite3 era **qmark** por cada parámetro se añade una **?**

- cursor.execute(“””select * from sucursales where ciudad=?”””, (ciudad,))

- Cada **?** representa un parámetro que se rellenan de izquierda a derecha y los valores se indican en una tupla.

Row

- Clase sqlite3.**Row**
  - Una instancia que representa una fila de una consulta.
  - Está más optimizada que una tupla.
  - Admite el mapeo por nombre de columna e índice, iteración, representación, pruebas de igualdad y la función len().

Row: Ejemplo I

```
con = sqlite3.connect(":memory:")
cur = con.cursor()
cur.execute('''create table stocks
(date text, trans text, symbol text,
 qty real, price real)''')
cur.execute("""insert into stocks
          values ('2006-01-05','BUY','RHAT',100,35.14)""")
con.commit()
cur.close()
```

Row: Ejemplo II

```
>>> con.row_factory = sqlite3.Row
>>> cur = con.cursor()
>>> cur.execute('select * from stocks')
<sqlite3.Cursor object at 0x7f4e7dd8fa80>
>>> r = cur.fetchone()    → Leer un registro
>>> type(r)
<class 'sqlite3.Row'>
>>> tuple(r)              → Se puede convertir a tupla
('2006-01-05', 'BUY', 'RHAT', 100.0, 35.14)
>>> len(r)                → La longitud:
5
>>> r[2]                  → Acceso por posición
'RHAT'
>>> r.keys()              → Obtener los nombres de los campos
['date', 'trans', 'symbol', 'qty', 'price']
>>> r['qty']              → Acceso por el nombre del campo
100.0
>>> for member in r:      → Iterar por ítems de la fila
...     print(member)
2006-01-05
BUY
RHAT
100.0
35.14
```

Tipos SQL

- La API de bases de datos de Python incluye una serie de constructores a utilizar para crear estos tipos:

  - **Date(year, month, day)** Para almacenar fechas.
  - **Time(hour, minute, second)** Para almacenar horas.
  - **Timestamp(year, month, day, hour, minute, second)** Para almacenar timestamps (una fecha con su hora).
  - **DateFromTicks(ticks)** Para crear una fecha a partir de un número con los segundos transcurridos desde el *epoch (el 1 de Enero de 1970 a las 00:00:00 GMT).*
  - **TimeFromTicks(ticks)** Similar al anterior, para horas en lugar de fechas.
  - **TimestampFromTicks(ticks)** Similar al anterior, para timestamps.
  - **Binary(string)** Valor binario.

Enlaces

- http://www.pythondiario.com/2013/12/python-y-sqlite3-como-base-de-datos.html
- https://iqbalnaved.wordpress.com/2014/07/10/how-to-install-sqlite-3-8-2-on-ubuntu-14-04-and-commands-for-creating-database-and-tables/
- https://www.programcreek.com/python/example/3926/sqlite3.Row