Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 27 de agosto de 2016

Base de datos SQLite con APSW

APSW (*Another Python SQLite Wrapper*) es una capa software del motor de base de datos relacional SQLite para el desarrollo de aplicaciones Python. Permite realizar las operaciones que se pueden efectuar con la API de SQLite nativa.

Para conocer cómo funciona el gestor de base de datos SQLite recomendamos la lectura de la Guía Rápida de SQLite que hemos incluido como anexo en este blog.

Instalar el módulo APSW

Para instalar el módulo **APSW** con el instalador **PIP** se recomienda hacerlo desde **GitHub**. Para ello, ejecutar el siguiente comando:

pip install https://github.com/rogerbinns/apsw/releases/download/3.13.0-r1/apsw-3.13.0-r1.zip --global-option=fetch --global-option=--version --global-option=3.13.0 --global-option=--all --global-option=build --global-option=--enable-all-extensions

La opción *--user* se puede agregar al comando para instalar el módulo en el repositorio de paquetes del perfil del usuario.

Consultar otras opciones para descargar e instalar el módulo.

Comprobar versión instalada

Después de realizar la instalación, para comprobar las versiones instaladas del módulo **ASPW** y del gestor de base de datos SQLite, ejecutar el siguiente código:

```
import apsw

print("Archivo del modulo APSW..:",apsw.__file__)
print("Version APSW.............:",apsw.apswversion())
print("Version biblioteca SQLite:",apsw.sqlitelibversion())
print("Version cabecera SQLite..:",apsw.SQLITE_VERSION_NUMBER)
```

Crear y/o abrir una base datos SQLite

Para crear una base de datos o abrir una existente; y declarar un cursor para realizar operaciones:

```
import apsw

conexion=apsw.Connection("deportistas.db")
cursor=conexion.cursor()
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

Ejecutar comandos SQL

El método **execute()** ejecuta cualquier comando SQL admitido por SQLite. Si el comando no es válido se producirá una excepción del tipo **SQLError**.

cursor.execute("comando_SQL")

Con el método **execute()** también se pueden ejecutar varios comandos SQL separando los comandos con un punto y coma (;).

cursor.execute("comando_SQL1; comando_SQL2; ...; comando_SQLn")

A continuación, varios ejemplos que muestran el modo de trabajar con la base de datos SQLite creada con anterioridad:

Crear tablas

```
import apsw

tabla_usuarios = """CREATE TABLE usuarios (
 id_usu INTEGER PRIMARY KEY AUTOINCREMENT,
 cta_usu TEXT UNIQUE,
 nombre TEXT NOT NULL,
 id_dpte INTEGER,
 ecorreo TEXT NOT NULL,
 FOREIGN KEY (id_dpte) REFERENCES dptes(id_dpte)
);"""

tabla_deportes = """CREATE TABLE dptes (
 id_dpte INTEGER PRIMARY KEY AUTOINCREMENT,
 denom TEXT NOT NULL UNIQUE
);"""

cursor.execute(tabla_usuarios)
cursor.execute(tabla_deportes)
```

Listar campos de una tabla

```
for campos_usuarios in cursor.execute("PRAGMA table_info('usuarios');"):
    print(campos_usuarios)

# (0, 'id_usu', 'INTEGER', 0, None, 1)
# (1, 'cta_usu', 'TEXT', 0, None, 0)
# (2, 'nombre', 'TEXT', 1, None, 0)
# (3, 'id_dpte', 'INTEGER', 0, None, 0)
# (4, 'ecorreo', 'TEXT', 1, None, 0)
```

Insertar registros

```
deportes = ["Atletismo", "Baloncesto", "Tenis"]

for deporte in deportes:
    cursor.execute("insert into dptes values(?,?)", (None, deporte))

deportistas = [("rnadal", "Rafa Nadal", 3, "rnadal@o.es"),
               ("rbeitia", "Ruth Beitia", 1, "rbeitia@o.es"),
               ("pgasol", "Pau Gasol", 2, "pgasol@o.es")]

for dpta in deportistas:
    cursor.execute("insert into usuarios values(?,?,?,?,?)",
                   (None, dpta[0], dpta[1], dpta[2], dpta[3]))
```

Consultar registros

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

agosto 2016 (1)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
consulta_dptes = "SELECT * FROM dptes;"
for fila in cursor.execute(consulta_dptes):
    print(fila)

# (1, 'Atletismo')
# (2, 'Baloncesto')
# (3, 'Tenis')

consulta_usuarios = "SELECT nombre, id_dpte, ecorreo FROM usuarios;"
for fila in cursor.execute(consulta_usuarios):
    print(fila[0], fila[1], fila[2])

# Rafa Nadal 3 rnadal@o.es
# Ruth Beitia 1 rbeitia@o.es
# Pau Gasol 2 pgasol@o.es
```

Encadenar varias consultas

```
consulta_dptes = """SELECT id_dpte, denom FROM dptes WHERE id_dpte=1;
SELECT id_dpte, denom FROM dptes WHERE id_dpte=3;"""
for campo1, campo2 in cursor.execute(consulta_dptes):
    print(campo1, campo2)

# (1, 'Atletismo')
# (2, 'Baloncesto')
# (3, 'Tenis')
```

Obtener la descripción de los campos

```
# El método getdescription() devuelve una
# tupla de tuplas con los nombres y el tipo
# de los campos obtenidos en una consulta:

iterador = cursor.execute("SELECT * FROM dptes;")
print(iterador.getdescription())
for fila in iterador:
    print(fila)

# (('id_dpte', 'INTEGER'), ('denom', 'TEXT'))
# (1, 'Atletismo')
# (2, 'Baloncesto')
# (3, 'Tenis')
```

Habilitar restricciones de integridad referencial

```
# Para habilitar las restricciones de integridad
# referencial de la base de datos:

cursor.execute("PRAGMA foreign_keys=ON;")

# Después de activar las restricciones de
# integridad referencial, en este caso, al
# intentar insertar un nuevo registro (como
# la clave externa no existe) producirá una
# excepción de tipo ConstraintError:

cursor.execute("INSERT INTO usuarios values(?,?,?,?,?)",
               (None, "mbelmonte", "Mireia Belmonte", 4, "mb@o.es"))

# Para que no se produzca la excepción es
# imprescindible que exista un "deporte"
# identificado con el número entero 4 en la
# tabla auxiliar. Para insertar dicho registro:

cursor.execute("INSERT INTO dptes values(?,?)", (None, "Natacion"))

# Consultamos ahora todos los deportes existentes:

consulta_dptes = "SELECT * FROM dptes;"
for fila in cursor.execute(consulta_dptes):
```

```
    print(fila)

# (1, 'Atletismo')
# (2, 'Baloncesto')
# (3, 'Tenis')
# (4, 'Natacion')

# Por último, insertamos el registro que producía
# la excepción, ahora sin problemas:

cursor.execute("INSERT INTO usuarios values(?,?,?,?,?)",
               (None, "mbelmonte", "Mireia Belmonte", 4, "mb@o.es"))
```

Borrar registros

```
# Como en el apartado anterior se activaron
# las restricciones de integridad referencial
# al intentar borrar una fila referenciada en
# otra tabla el sistema generará una excepción
# de tipo ConstraintError:

cursor.execute("DELETE FROM dptes WHERE id_dpte=?", (4,))
```

Crear una vista

```
# Para crear una vista que cruza los datos de la
# tabla "usuarios" con los de la tabla "dptes":

vista = """CREATE VIEW vista_dptas AS SELECT nombre, denom, \
ecorreo FROM usuarios INNER JOIN dptes ON \
usuarios.id_dpte = dptes.id_dpte;"""
cursor.execute(vista)

# Para consultar los datos de la vista creada:

consulta = "SELECT * FROM vista_dptas;"
for fila in cursor.execute(consulta):
    print(fila)

# ('Rafa Nadal', 'Tenis', 'rnadal@o.es')
# ('Ruth Beitia', 'Atletismo', 'rbeitia@o.es')
# ('Pau Gasol', 'Baloncesto', 'pgasol@o.es')
# ('Mireia Belmonte', 'Natacion', 'mb@o.es')
```

Insertar registros utilizando bindings (ataduras)

```
# -Por la posición en una secuencia:

# Indicando después del signo de
# interrogación "?" el número de posición
# que ocupa un valor en una secuencia.

# En el ejemplo siguiente el primer campo
# de la tabla tomará el valor "c", el segundo
# el valor "b" y el tercero el valor "a".

cursor.execute("CREATE TABLE tabla(campo1,campo2,campo3)")
cursor.execute("INSERT INTO tabla values(?3,?2,?1)", ('a','b','c'))

# -Por la clave de un diccionario:

# En la posición de cada campo se indica la
# clave de un diccionario para acceder a
# su valor:

cursor.execute("INSERT INTO tabla values(:a,:b,:c)",
               {'a':1,'b':2,'c':3})
```

Trazar la ejecución de comandos

Trazar comandos

```
# El siguiente ejemplo ejecuta varios
# comandos SQL y muestra, uno a uno,
# dichos comandos (con sus opciones) a medida
# que se ejecutan

def rastreando_comandos(cursor, statement, bindings):
    print("SQL:", statement)
    if bindings:
        print("Bindings:", bindings)
    return True

cursor.setexectrace(rastreando_comandos)
comandos = """DROP TABLE IF EXISTS tabla;
CREATE TABLE tabla(campo);
INSERT INTO tabla(campo) VALUES(1), (2), (3), (2), (1), (1);
SELECT * FROM tabla WHERE campo=?"""
cursor.execute(comandos, (1,))

# SQL: DROP TABLE IF EXISTS tabla;
# SQL: CREATE TABLE tabla(campo);
# SQL: INSERT INTO tabla(campo) VALUES(1), (2), (3), (2), (1), (1);
# SQL: SELECT * FROM tabla WHERE campo=?
# Bindings: (1,)
```

Trazar resultados

```
# El siguiente ejemplo ejecuta varios
# comandos SQL y muestra, uno a uno,
# dichos comandos y el resultado que obtienen
# de la base de datos

def rastreando_rtdos(cursor, row):
    print("Row:", row)
    return(row)

cursor.setrowtrace(rastreando_rtdos)
comandos = """SELECT * FROM tabla WHERE campo=1;
SELECT * FROM tabla WHERE campo=3"""
for row in cursor.execute(comandos):
     pass

# SQL: SELECT * FROM tabla WHERE campo=1;
# Row: (1,)
# Row: (1,)
# Row: (1,)
# SQL: SELECT * FROM tabla WHERE campo=3
# Row: (3,)
```

Ejecutar varias sentencias

```
# Inicializar trazadores (tracers)

cursor.setrowtrace(None)
cursor.setexectrace(None)

# Ejecutar varias veces el comando INSERT
# Inserta tres registros en la tabla

cursor.executemany("insert into tabla (campo) values(?)",
                   ( [1], [2], [3] ) )

# Ejecutar varias consultas (SELECT)
# Lanza tres consultas sobre la tabla

for row in cursor.executemany("select * from tabla where campo=?",
                              ( [1], [2], [3] ) ):
    print(row)
```

Relacionado:

- Base de Datos SQLite3
- Guía Rápida de SQLite3

Ir al índice del tutorial de Python

Publicado por Pherkad en 15:41

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua