Adaptadores y conversores

SQLite soporta de forma nativa los siguientes tipos: NULL, INTEGER, REAL, TEXT, BLOB.

Los siguientes tipos de Python pueden ser enviados a SQLite sin ningún problema:

Tipo de Python	Tipo de SQLite
None	NULL
int	INTEGER
float	REAL
str	TEXT
bytes	BLOB

Por defecto los tipos de SQLite son convertidos a los tipos de Python de la siguiente manera:

Tipo de SQLite	Tipo de Python
NULL	None
INTEGER	int
REAL	float
TEXT	Depende del parámetro text_factory, por default se convierte a str
BLOB	bytes

El sistema de tipos del módulo *sqilte3* es extensible de dos maneras:

1. Se puede almacenar tipos de Python adicionales en una base de datos SQLite por medio de adaptadores.
2. Se puede dejar que el módulo *sqlite3* convierta tipos de SQLite a diferentes tipos de Python por medio de conversores.

Adaptadores

Para utilizar otros tipos de Python con SQLite, debes adaptarlos a uno de los tipos soportados por el módulo *sqlite3*: None, int, float, str, bytes.

Hay dos formas de habilitar el módulo *sqlite3* para adaptar un tipo de Python personalizado a uno de los tipos soportados por la librería.

Dejando que el objeto se adapte

Este es un buen enfoque si estás escribiendo tu propia clase. Supongamos que tenemos definida la clase Point con las coordenadas x e y. Y queremos guardar el punto en una solo columna de SQLite. Primero deberemos elegir uno de los tipos soportados para representar el punto. Por ejemplo, usemos un *string* separando las coordenadas por un punto y coma. Para esto tenemos que definir el método *__conform__* que debe devolver el valor convertido.

```
import sqlite3

class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __conform__(self, protocol):
```

```
        if protocol is sqlite3.PrepareProtocol:
            return "%f;%f" % (self.x, self.y)

con = sqlite3.connect(":memory:")
cur = con.cursor()

p = Point(4.0, -3.2)
cur.execute("select ?", (p,))
print(cur.fetchone()[0])

con.close()
```

Registrando una función adaptadora

La otra posibilidad es crear una función que convierta el tipo a la representación en *string* y registrar la función con *register_adapter()*. Por ejemplo:

```
import sqlite3

class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

def adapt_point(point):
    return "%f;%f" % (point.x, point.y)

sqlite3.register_adapter(Point, adapt_point)

con = sqlite3.connect(":memory:")
cur = con.cursor()
```

```
p = Point(4.0, -3.2)
cur.execute("select ?", (p,))
print(cur.fetchone()[0])

con.close()
```

Convertir valores SQLite a tipos personalizados de Python

Escribiendo un adaptador podemos enviar tipos de Python personalizados a SQLite. Pero para que sea realmente útil necesitamos también convertir el tipo SQLite en un tipo de Python. Para esto creamos una función que convierta el tipo SQLite en un tipo de Python y la registramos en el módulo *sqlite3*.

```
import sqlite3

class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __repr__(self):
        return "(%f;%f)" % (self.x, self.y)

def adapt_point(point):
    return ("%f;%f" % (point.x, point.y)).encode('ascii')

def convert_point(s):
    x, y = list(map(float, s.split(b";")))
    return Point(x, y)

sqlite3.register_adapter(Point, adapt_point)
```

```
sqlite3.register_converter("point", convert_point)

con = sqlite3.connect(":memory:", detect_types=sqlite3.PARSE_DECLTYPES)
cur = con.cursor()
cur.execute("create table test(p point)")

cur.execute("insert into test(p) values (?)", (p,))
cur.execute("select p from test")
print("with declared types:", cur.fetchone()[0])
cur.close()
con.close()
```

Adaptadores y conversores por defecto

Existen unos adaptadores por defecto para los tipos *fecha* y *fecha y hora.* Los mismo envían la fecha o la fecha y hora en formato ISO a SQLite.

El conversor por defecto para las fechas se llama “date” y el de fecha y hora se llama “timestamp”.

A continuación, se muestra un ejemplo de cómo utilizar los conversores de fecha y fecha y hora.

```
import sqlite3
import datetime

con = sqlite3.connect(":memory:",
detect_types=sqlite3.PARSE_DECLTYPES|sqlite3.PARSE_COLNAMES)
cur = con.cursor()
cur.execute("create table test(d date, ts timestamp)")
```

```
today = datetime.date.today()
now = datetime.datetime.now()

cur.execute("insert into test(d, ts) values (?, ?)", (today, now))
cur.execute("select d, ts from test")
row = cur.fetchone()
print(today, "=>", row[0], type(row[0]))
print(now, "=>", row[1], type(row[1]))

cur.execute('select current_date as "d [date]", current_timestamp as "ts [timestamp]"')
row = cur.fetchone()
print("current_date", row[0], type(row[0]))
print("current_timestamp", row[1], type(row[1]))

con.close()
```