Adaptation des types Python à PostgreSQL pour psycopg

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 septembre 2008

https://www.bortzmeyer.org/psycopg2-adaptation.html

---

psycopg <http://initd.org/pub/software/psycopg/> est l'interface avec PostgreSQL la plus répandue chez les programmeurs Python. Parmi toutes ses qualités, psycopg adapte automatiquement les types Python aux types PostgreSQL. Mais cette **adaptation** ne fonctionne pas toujours toute seule et a parfois besoin d'un coup de main.

Avec psycopg <http://initd.org/pub/software/psycopg/>, si j'ai créé une table en SQL avec :

```
CREATE TABLE Foobar (t TEXT, i INTEGER);
```

je peux écrire dans mon programme Python :

```
cursor.execute("INSERT INTO Foobar (t, i) VALUES (%s, %s)",
               ["I like Python", 42])
```

et psycopg trouve tout seul que "I like Python" est une chaîne de caractères alors que 42 est un entier. Il "*quote*" (met entre apostrophes) donc le premier et pas le second. PostgreSQL recevra (instruction SQL affichée grâce à `log_statement = 'all'`) :

```
2008-09-16 13:40:19 CEST STATEMENT:  INSERT INTO Foobar (t, i) VALUES ('I like Python', 42)
```

Cela marche bien dans ce cas car Python connait les types utilisés et peut donc le dire à psycopg2 :

```
>>> type(42)
<type 'int'>
>>> type("I like Python")
<type 'str'>
```

Mais si on a affaire à des types qui existent dans PostgreSQL et pas dans Python ? Le cas est fréquent car PostgreSQL dispose de nombreux types <http://www.postgresql.org/docs/current/interactive/datatype.html> comme par exemple les points <http://www.postgresql.org/docs/current/interactive/datatype-geometric.html#AEN5480>, les adresses IP <http://www.postgresql.org/docs/current/interactive/datatype-net-types.html#DATATYPE-INET> ou encore les UUID <http://www.postgresql.org/docs/current/interactive/datatype-uuid.html> (RFC 4122 [1]). Essayons avec les adresses IP. Créons la table :

```
CREATE TABLE Foobar (addr INET);
```

puis remplissons-la :

```
cursor.execute("INSERT INTO Foobar (addr) VALUES (%s)",
                           ["2001:DB8::CAFE:1"])
```

Ça marche, l'adresse IP `2001:db8::cafe:1` a bien été enregistrée. Python ne connait pas ce type, donc l'a traité comme une chaîne de caractères, que PostgreSQL a su analyser.

Mais si on veut mettre un tableau <http://www.postgresql.org/docs/current/interactive/arrays.html> de telles adresses (petit détour par le DNS : il est tout à fait légal, et même fréquent, qu'à un nom de machine donnée correspondent plusieurs adresses IP) ? Là, Python, psycopg et PostgreSQL ne sont pas assez intelligents. On crée la table :

```
CREATE TABLE Foobar (addr INET[]);
```

et on tente de la peupler :

```
>>> cursor.execute("INSERT INTO Foobar (addr) VALUES (%s)", [["2001:DB8::CAFE:1", "192.168.25.34"]])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
psycopg2.ProgrammingError: column "addr" is of type inet[] but expression is of type text[]
LINE 1: INSERT INTO Foobar (addr) VALUES (ARRAY['2001:DB8::CAFE:1', ...
                           ^
HINT:  You will need to rewrite or cast the expression.
```

En effet, PostgreSQL aura reçu :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4122.txt`

```
2008-09-16 13:53:01 CEST LOG:  statement: INSERT INTO Foobar (addr) VALUES (ARRAY['2001:DB8::CAFE:1', '192.168.2
```

alors qu'il aurait fallu envoyer `INSERT INTO Foobar (addr) VALUES ('{2001:DB8::CAFE:1, 192.168.25.34}')`. (Le problème serait le même pour le type UUID.)

Quelle est la bonne solution ? psycopg pourrait être plus malin et mieux connaître les types de PostgreSQL mais c'est plus compliqué que ça en a l'air. Heureusement, il existe une solution assez simple. Depuis sa version 2, psycopg permet de définir des **adaptateurs** soi-même, c'est-à-dire le code qui va faire la conversion d'un objet Python, au moment de l'appel à PostgreSQL. Cette technique est documentée (sommairement) dans le fichier `doc/extensions.rst` qui est distribué avec psycopg. (On le trouve aussi sur le Web en <http://www.initd.org/svn/psycopg/psycopg2/trunk/doc/extensions.rst> mais, servi avec une mauvaise indication de son type, il est difficile à lire.) Voici l'exemple que donne la documentation. Il utilise le type POINT <http://www.postgresql.org/docs/current/interactive/datatype-geometric.html#AEN5480> de PostgreSQL, qui représente un point dans un espace cartésien à deux dimensions. On crée la table avec :

```
CREATE TABLE Atable (apoint POINT);
```

et on peut la peupler en SQL ainsi :

```
INSERT INTO Atable (apoint) VALUES ('(1, 3.14159)');
```

Pour que cela soit fait automatiquement depuis le programme Python utilisant psycopg2, le code est :

```
    from psycopg2.extensions import adapt, register_adapter, AsIs

    class Point(object):
        def __init__(self, x=0.0, y=0.0):
            self.x = x
            self.y = y

    def adapt_point(point):
        return AsIs("'(%s,%s)'" % (adapt(point.x), adapt(point.y)))

    register_adapter(Point, adapt_point)

    curs.execute("INSERT INTO atable (apoint) VALUES (%s)",
                 (Point(1.23, 4.56),))
```

Et voici enfin le (tout petit) adaptateur que j'ai écrit pour le type PostgreSQL INET <http://www.postgresql.org/docs/current/interactive/datatype-net-types.html#DATATYPE-INET>, qui permet de représenter des adresses IP :

```
from psycopg2.extensions import adapt, register_adapter, AsIs

class Inet(object):
    """ Classe pour stocker les adresses IP. Pour que psycopg puisse
    faire correctement la traduction en SQL, il faut que les objets
    soient typés. Les chaînes de caractères ne conviennent donc pas. """

    def __init__(self, addr):
```

```
        """ Never call it with unchecked data, there is no protection
against injection """
        self.addr = addr

def adapt_inet(address):
    """ Adaptateur du type Python Inet vers le type PostgreSQL du même
    nom. On utilise un "cast" PostgreSQL, représenté par '::inet' """
    return AsIs("'%s'::inet" % address.addr)

register_adapter(Inet, adapt_inet)
```

Et on l'utilise ainsi (notez qu'il faut appeler le constructeur `Inet()` pour « typer » les valeurs) :

```
cursor.execute("INSERT INTO Foobar (addr) VALUES (%s)",
                [[Inet("2001:DB8::CAFE:1"), Inet("192.168.25.34")]])
```

Désormais, cela fonctionne. PostgreSQL reçoit :

```
2008-09-16 17:43:48 CEST LOG:  statement: INSERT INTO Foobar (addr) VALUES (ARRAY['2001:DB8::CAFE:1'::inet,
```

et exécute bien la requête.

Merci à Chris Cogdon et Federico Di Gregorio (l'auteur de psycopg) pour leur aide sur ce sujet.