Stocker et récupérer de l'Unicode dans PostgreSQL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 30 mai 2006. Dernière mise à jour le 9 novembre 2007

https://www.bortzmeyer.org/postgresql-unicode.html

Ayant récemment eu à stocker et récupérer de l'Unicode dans une base de données PostgreSQL, j'en profite pour expliquer brièvement comment cela marche.

PostgreSQL est un excellent logiciel libre de gestion de base de données. Il permet notamment de stocker les données de type texte sous différents encodages. Tout le processus est bien documenté <http://www.postgresql.org/docs/current/interactive/multibyte.html> sur leur site Web.

Au moment où on crée une nouvelle base, on peut spécifier l'encodage de celle-ci. Je recommande toujours l'encodage UNICODE (qui est en fait de l'UTF-8, celui qui a choisi le terme UNICODE n'avait pas forcément saisi les subtilités d'Unicode mais, depuis la version 8.1, PostgreSQL permet le terme correct, "UTF-8"). Cela permet de stocker tous les caractères possibles et imaginables.

Pour manipuler des caractères, il vaut mieux suivre deux règles simples :

— Choisir un encodage canonique et s'y tenir (ici, UTF-8, que PostgreSQL nommait UNICODE),
— Ne faire les éventuelles conversions qu'une fois (que ce soit en entrée ou en sortie) de façon à pouvoir se souvenir de quel est l'encodage utilisé.

Ici, je vais supposer que les applications préfèrent du Latin-1. Si on n'a pas besoin de conversion, c'est encore plus simple.

Un avertissement préalable (merci à Daniel Verite pour ses explications) : il vaut mieux que la structure des bases de données aie été créée avec le même encodage que les bases. Comme le dit la documentation de PostgreSQL <http://www.postgresql.org/docs/current/interactive/app-initdb.html>, "*For these reasons it is important to choose the right locale when running initdb.*". Donc, si vous prévoyez des bases en UTF-8, il vaut mieux avoir fait tourner initdb avec cet encodage. Comme par défaut, initdb utilise la locale courante, on peut facilement se tromper. On installe PostgreSQL, on lance (ou bien le système de paquetages lance pour vous) initdb et il utilise la mauvaise locale. Si vous obtenez ce genre de bogues :

```
postgres=# create database u8 encoding 'UTF8';
CREATE DATABASE
postgres=# \c u8
...
postgres=# set client_encoding=latin1;
SET
u8=# select lower('à')
ERROR:  invalid byte sequence for encoding "UTF8": 0xe3a0
HINT:  This error can also happen if the byte sequence does not match the encoding expected by the server, w
```

c'est probablement à cause de cette incohérence.

Pour l'éviter, je suggère que **tout** sur le serveur soit en UTF-8. Donc, (re)lancer initdb ainsi (la commande convient pour une Debian, d'autres systèmes d'exploitation peuvent mettre `initdb` ou bien le répertoire contenant les bases ailleurs) :

```
% /usr/lib/postgresql/$PG_VERSION/bin/initdb --encoding=UTF8 --locale=fr_FR.UTF8 --pgdata=/var/lib/pos
```

et s'assurer que le serveur a bien une locale UTF-8. Par exemple, dans le paquetage Debian, je mets :

```
LC_CTYPE=fr_FR.UTF8
```

dans `/etc/postgresql/$PG_VERSION/main/environment`.

Une fois passé ce piège, je crée la base (si on a suivi les conseils précédents, l'option `--unicode` est facultative) :

```
% createdb --encoding UNICODE ${DB}
% psql -f ./create.sql ${DB}
```

où le fichier `create.sql` contient :

```
CREATE TABLE Adresses (
   id SERIAL UNIQUE NOT NULL,
   prenom TEXT,
   nom TEXT
);
```

Je peux ensuite remplir la base. D'abord, utilisons `psql`, l'interface ligne de commandes de PostgreSQL (je me méfie des cliquodromes) :

```
% psql -f ./insert-utf8.sql ${DB}
```

où `insert-utf8.sql` contient du bel UTF-8 :

```
INSERT INTO Adresses (Prenom, Nom)
   VALUES ('Stéphane', 'Bortzmeyer');
INSERT INTO Adresses (Prenom, Nom)
   VALUES ('Pierre', 'Louÿs');
```

Si on préfère que le fichier soit en Latin-1, parce qu'on édite plus facilement du Latin-1, rien de plus simple, mais il faut l'indiquer à PostgreSQL :

```
SET CLIENT_ENCODING='iso-8859-1';

INSERT INTO Adresses (Prenom, Nom)
   VALUES ('Stéphane', 'Bortzmeyer');
INSERT INTO Adresses (Prenom, Nom)
   VALUES ('Pierre', 'Louÿs');
```

et

```
% psql -f ./insert-latin1.sql ${DB}
```

produira le résultat attendu.

L'interêt de gérer des fichiers UTF-8 est que cela permet de représenter des caractères qui n'existent pas dans Latin-1. Si vous n'avez affaire qu'à des noms français, ce n'est pas un problème mais supposons que vous vouliez enregistrer le nom d'une célèbre physicienne d'origine polonaise, dont le nom comporte le caractère Unicode U+0142 ("*LATIN SMALL LETTER L WITH STROKE*"), qui existe en Latin-2 mais pas en Latin-1, vous pouvez alors :

```
INSERT INTO Adresses (Prenom, Nom)
   VALUES ('Marii', 'Curie-Sk␣lodowskiej');
```

et ces caractères coexisteront sans problèmes avec ceux utilisés en français. C'est bien pour cela qu'Unicode a été inventé.

Le support d'Unicode dans PostgreSQL n'est pas sans faille. La recherche avec les expressions rationnelles <http://www.postgresql.org/docs/current/interactive/functions-matching.html#FUNCTIONS-POSIX-REGEXP>, par exemple, ne fonctionne que partiellement :

```
carnet=> SELECT * FROM adresses WHERE prenom ~ 'St.phane';
 id |  prenom  |    nom
----+----------+------------
 19 | Stéphane | Bortzmeyer
(2 rows)

carnet=> SELECT * FROM adresses WHERE prenom ~ 'St\wphane';
 id | prenom | nom
----+--------+-----
(0 rows)
```

Le point (qui remplace n'importe quel caractère) a bien fonctionné (PostgreSQL comprend bien la différence entre un caractère et un octet, le caractère é s'écrivant avec deux octets en UTF-8) mais le \w qui veut normalement dire "un caractère alphanumérique" n'a pas été compris.

De même, je n'ai pas pu faire fonctionner la capitalisation pour l'opérateur ILIKE (comme LIKE, mais insensible à la casse) :

```
carnet=> SELECT * FROM adresses WHERE prenom ILIKE '%stéphane%';
 id |  prenom  |    nom
----+----------+------------
  5 | Stéphane | Bortzmeyer
(1 rows)

carnet=> SELECT * FROM adresses WHERE prenom ILIKE '%STÉPHANE%';
 id | prenom | nom
----+--------+-----
(0 rows)
```

C'est le genre de petits problèmes agaçants qu'on rencontre souvent <https://www.bortzmeyer.org/pas-encore-utf8.html> avec Unicode.

Affichons maintenant ce qu'on a enregistré. `psql -c "SELECT * FROM Adresses" ${DB}` suffit mais, si la console utilisée est en Latin-1, il faut le demander :

```
PGCLIENTENCODING=iso-8859-1 psql -c "SELECT * FROM Adresses" ${DB}
```

Et on voit bien les caractères, qu'ils aient été entrée en UTF-8 ou bien en Latin-1 :

```
 id |  prenom  |    nom
----+----------+------------
  5 | Stéphane | Bortzmeyer
  6 | Pierre   | Louÿs
  7 | Stéphane | Bortzmeyer
  8 | Pierre   | Louÿs
(4 rows)
```

(Naturellement, les noms qui ne sont pas représentables en Latin-1 seront malmenés.)

Si maintenant, on veut traiter ces données dans un programme, c'est également possible, ici en Python :

```
#!/usr/bin/python

import psycopg
import sys

db_encoding = "UTF-8"

if len(sys.argv) <= 1:
    sys.stderr.write("Usage: %s database [encoding]\n" % sys.argv[0])
    sys.exit(1)
```

```
db = sys.argv[1]
if len(sys.argv) > 2:
    def tr(text):
        return text.decode(db_encoding).encode(sys.argv[2], 'replace')
else:
    def tr(text):
        return text

connection = psycopg.connect("dbname=%s" % db)
cursor = connection.cursor()

cursor.execute("""
   SELECT prenom,nom FROM Adresses
   """)
for mytuple in cursor.fetchall():
    sys.stdout.write("%s %s\n" % (tr(mytuple[0]), tr(mytuple[1])))

cursor.close()
connection.close()
```

Ce programme va lire les données de la base, il peut les convertir en Unicode (Python manipule nativement l'Unicode), pour traitement (`text.decode(db_encoding)`) et les mettre dans l'encodage souhaité pour l'affichage (`unicode_string.encode(myencoding, 'replace')`).

Si on souhaite publier sur le Web, on a tout intérêt à publier en UTF-8, puisque tous les outils Web savent le gérer. Il faut juste penser à indiquer cet encodage au client Web (les détails dépendent du serveur HTTP utilisé).

Tous ces programmes ont été testés avec PostgreSQL 7.4 et 8.1.