Créer un agrégat en PostgreSQL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 janvier 2007

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

Devant gérer une base de données où un champ pouvait prendre plusieurs valeurs, j'ai découvert que PostgreSQL permettait de créer ses propres agrégats <http://www.postgresql.org/docs/current/interactive/functions-aggregate.html>, ces fonctions qui sont à SQL ce que `reduce` est aux listes.

Le problème était d'exporter en SQL les données du "*language subtag registry*", le registre des sous-étiquettes de langues, ces identificateurs décrits par le RFC 4646 [1]. Dans le registre originel <https://www.iana.org/assignments/language-subtag-registry>, chaque sous-étiquette peut avoir un champ Description ou plusieurs (le record, la langue zazaki en a six, le slavon en a cinq). Dans le modèle relationnel pur, cela ne permet pas de mettre la description dans la table Langues car les valeurs des champs doivent être scalaires (PostgreSQL autorise les tableaux <http://www.postgresql.org/docs/current/interactive/arrays.html> mais c'est une extension pas toujours bien acceptée du modèle relationnel). On pourrait concaténer toutes les descriptions en une seule, mais on perdait alors la possibilité de chercher facilement une description particulière.

La solution casher en SQL est donc de créer une table Descriptions et une autre table faisant le lien entre la table Langues et la table Description. Cela donne :

```
CREATE TABLE Languages (
    id SERIAL PRIMARY KEY NOT NULL,
    code TEXT UNIQUE NOT NULL, -- Typically an ISO 639 code
    added DATE,
    comments TEXT);

CREATE TABLE Descriptions (
     id SERIAL PRIMARY KEY NOT NULL,
     description TEXT NOT NULL);

CREATE TABLE Descriptions_Languages (
    description INTEGER NOT NULL REFERENCES Descriptions(id),
    lang TEXT NOT NULL REFERENCES Languages(code));
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4646.txt`

Certes, c'est un peu pénible pour afficher toutes les langues et leur(s) descriptions mais les vues ont été inventées pour cela :

```
CREATE VIEW Languages_with_all_descr AS
         SELECT l.code, l.added, d.description
     FROM languages l,descriptions d,descriptions_languages WHERE
        descriptions_languages.lang = l.code AND
        descriptions_languages.description = d.id;
```

On peut alors écrire :

```
SELECT code, added, description FROM Languages_with_all_descr;
```

et on a bien :

```
lsr=> SELECT * FROM Languages_with_all_descr WHERE  code='es';
 code |   added    | description
------+------------+-------------
 es   | 2005-10-16 | Spanish
 es   | 2005-10-16 | Castilian
(2 rows)
```

Maintenant, si on veut afficher un seul tuple par langue, c'est un peu plus compliqué en SQL Il faut **réduire** les descriptions à un scalaire. Traditionnellement, en SQL, c'est le travail des fonctions agrégats comme `avg` (*"average"*, la moyenne) ou bien `count` (le nombre de tuples). Mais, à ma surprise, il ne semble pas y avoir de fonction agrégat de concaténation de chaînes de caractères, que ce soit en SQL standard ou même avec les innombrables extensions de PostgreSQL. Heureusement, PostgreSQL permet de créer ses propres fonctions agrégat `<http://www.postgresql.org/docs/current/interactive/xaggr.html>`.

On ajoute donc une fonction auxiliaire qui concatène intelligemment (c'est-à-dire en ne mettant pas le séparateur si l'une des chaînes est vide ou non définie) **deux** chaînes :

```
CREATE OR REPLACE FUNCTION concat2(text, text) RETURNS text AS '
    SELECT CASE WHEN $1 IS NULL OR $1 = \'\' THEN $2
                WHEN $2 IS NULL OR $2 = \'\' THEN $1
                ELSE $1 || \' / \' || $2
                END;
 '
 LANGUAGE SQL;
```

(`||` est l'opérateur de concaténation de deux chaînes pour PostgreSQL.) On définit alors la fonction agrégat, ce qui se fait en indiquant la valeur de départ (une chaîne vide) et la fonction qui prend un agrégat et un élément et les combine en un nouvel agrégat, ici `concat2` :

```
CREATE AGGREGATE concatenate (
    sfunc = concat2,
    basetype = text,
    stype = text,
    initcond = ''
);
```

On peut alors définir une vue qui ne présente qu'un seul tuple par langue :

```
CREATE VIEW Languages_with_descr AS
         SELECT l.code, l.added,
                concatenate(d.description) FROM languages l, descriptions d,
                                               descriptions_languages WHERE
     descriptions_languages.lang = l.code AND
     descriptions_languages.description = d.id
          GROUP BY l.code, l.added, l;
```

Et le résultat est bien celui attendu :

```
lsr=> SELECT * FROM Languages_with_descr WHERE  code='es';
 code |   added    |     concatenate
------+------------+---------------------
 es   | 2005-10-16 | Spanish / Castilian
(1 row)

lsr=> SELECT * FROM Languages_with_descr WHERE  code='zza';
 code |   added    |                     concatenate
------+------------+-----------------------------------------------------
 zza  | 2006-08-24 | Zaza / Dimili / Dimli / Kirdki / Kirmanjki / Zazaki
(1 row)
```

Pour aller plus loin, avec le séparateur en argument, voir l'article de Sébastien Lardière <http://sebastien.lardiere.net/blog/index.php/post/2011/01/21/Agr%C3%A9gats-avec-arguments>.