Extrait du Environnement iSeries

http://www.xdocs400.com/spip.php?article455

Générer des jeux de données à partir de rien avec SQL DB2 et la récursivité

- Les articles -

Date de mise en ligne : samedi 24 mai 2014

Date de parution : 24 mai 2014

Description :

Il peut être parfois utile, sous SQL DB2, de générer une série de valeurs (par exemple de 1 à 10), valeurs qui serviront de point d'appui pour effectuer des jointures et récupérer - ou calculer - des données secondaires.

Environnement iSeries

Il peut être parfois utile, sous SQL DB2, de générer une série de valeurs (par exemple de 1 à 10), valeurs qui serviront de point d'appui pour effectuer des jointures et récupérer - ou calculer - des données secondaires.

Vous vous demandez peut être à quoi cela peut servir. Eh bien, imaginez que vous ayiez besoin d'afficher une statistique de chiffre d'affaires (CA) pour un produit, et cela sur 52 semaines. Supposons que vous n'ayiez pas du tout vendu ce produit une dizaine de semaines dans l'année. Il y a dans ce cas toutes les chances pour que vous n'ayiez aucune ligne dans votre base de données pour ce produit et les semaines en question. Vous aurez donc un trou dans votre stat, ce qui peut être gênant si vous souhaitez voir apparaître toutes les semaines dans votre tableau statistique, y compris celles pour lesquelles vous n'avez pas de CA. Vous me rétorquerez peut être que vous pouvez gérer ça avec une boucle dans un programme RPG ou PHP. Oui certes, mais ce serait tellement plus confortable de générer vos 52 lignes - correspondant à vos 52 semaines - directement via SQL, sans passer par l'utilisation d'un langage complémentaire.

Eh bien, c'est possible, et nous allons voir 2 manières de procéder :

- via une UDTF (User Data Table Function)
- via l'utilisation d'une requête SQL récursive

Voyons tout de suite la 1ère solution, avec l'UDTF :

```
CREATE OR REPLACE FUNCTION MYLIBRARY.GETINCAUTO(VAL_MAX INTEGER)
 RETURNS TABLE (VAL_INC INTEGER)
 LANGUAGE SQL
 MODIFIES SQL DATA
 BEGIN
 DECLARE V_NUM INTEGER DEFAULT 1;
 DECLARE GLOBAL TEMPORARY TABLE TMPINCAUTO
 ( VAL_INC INTEGER )
 WITH REPLACE ;
 WHILE V_NUM <= VAL_MAX DO
 INSERT INTO SESSION.TMPINCAUTO (VAL_INC)
 SELECT V_NUM FROM  SYSIBM.SYSDUMMY1 ;
 SET V_NUM = V_NUM + 1;
 END WHILE ;
 RETURN
 SELECT VAL_INC FROM SESSION.TMPINCAUTO ;
 END ;
```

Si on veut obtenir une liste de valeurs allant de 1 à 10, on appellera l'UDTF de la façon suivante :

```
SELECT * FROM TABLE (MYLIBRARY.GETINCAUTO(10) ) MYUDTF ;
```

Cette approche consistant à utiliser une UDTF est sans aucun doute la plus facile à appréhender. Après tout, il s'agit ni plus ni moins d'alimenter une table temporaire, et de la renvoyer sous forme de "result set" à la requête appelante.

Attention, il est impératif de donner un nom à votre UDTF dans la requête appelante, comme je l'ai fait dans l'exemple ci-dessus en la nommant "MYUDTF". Mettez le nom que vous voulez, mais si vous n'en mettez pas, ça ne fonctionnera pas.

Voyons maintenant la seconde approche consistant à utiliser une requête récursive.

```
WITH GEN_IDS(NX) AS (
 SELECT 1 as N1 FROM SYSIBM.SYSDUMMY1
 UNION ALL
 SELECT NX+1 as N2 FROM GEN_IDS WHERE NX < 10
 )
 SELECT NX as N FROM GEN_IDS ;
```

Avec la récursivité, la technique est toujours la même : elle consiste à déclarer une CTE (Common Table Expression), qui va utiliser une clause "UNION ALL" entre une première requête qui sert de point de départ, et une seconde requête qui fait appel à la CTE en cours d'exécution (d'où la récursivité). Dans cette seconde requête, il ne faut pas oublier de définir une condition de terminaison (ici : NX < 10), sinon vous risquez de partir dans une boucle sans fin.

Vous pouvez bien sûr encapsuler cette technique récursive dans une UDTF, si vous le jugez utile.

Voici deux exemples d'utilisation que je vous laisse le soin de décortiquer :

```
-- génération d'un calendrier (sur 30 jours) :
 WITH
 GEN_IDS(NX) AS (
 SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
 UNION ALL
 SELECT NX+1 AS N2 FROM GEN_IDS WHERE NX < 30
 )
 SELECT (CURRENT DATE + NX DAYS) AS N,
 DAYOFWEEK_ISO(CURRENT DATE + NX DAYS) AS DAY_OF_WEEK,
 DAYOFYEAR( CURRENT DATE + NX DAYS ) AS DAY_OF_YEAR,
 WEEK_ISO(CURRENT DATE + NX DAYS) AS WEEK
 FROM GEN_IDS
 ;

 -- génération d'identifiants aléatoires (pour constituer un jeu d'essai par exemple)
 WITH
 GEN_IDS(NX) AS (
 SELECT 1 AS N1 FROM SYSIBM.SYSDUMMY1
 UNION ALL
 SELECT NX+1 AS N2 FROM GEN_IDS WHERE NX < 10
 )
 SELECT NX AS NUMERO,
 1000+INT(RAND() * 1000) AS RANDOMID,
 CAST(RAND() * 100 AS DEC(5, 2)) AS RANDOM_PRICE,
 GENERATE_UNIQUE() AS RANDOM_UNIQ_ID ,
 CURRENT_TIMESTAMP + (NX*1000) MICROSECOND AS RANDOM_TSTAMP
 FROM GEN_IDS ;
```

Qu'on utilise la technique récursive, ou la technique plus classique faisant appel à l'UDTF présentée au début de l'article, je trouve extrêmement intéressante cette possibilité de générer à la volée, via SQL, des séries de valeurs n'existant pas initialement dans la base de données. Je n'ai pas trouvé dans la littérature SQL de terme définissant explicitement cette technique. Cela ressemble un peu au principe de la table pivot, mais une table pivot contenant plusieurs lignes, est-ce encore une table pivot ? Pour finir sur une note humoristique, je vous propose de désigner

cette technique sous l'acronyme de TPAIS pour "table pivot à incrément séquentiel". Oui je sais, ce n'est pas terrible, oubliez cette proposition ;)

Post-scriptum :

La technique récursive présentée brièvement dans cet article peut aussi être utilisée pour travailler sur des références croisées produites par un DSPPGMREF, ou extraites des tables systèmes SYSVIEWDEP et SYSROUTINEDEP. On peut ainsi afficher la hiérarchie des dépendances entre différents objets d'une base de données. J'y reviendrai dans un prochain article.