Générer un tableau d'amortissement d'immobilisation avec SQL DB2

- Les articles -

Date de mise en ligne : vendredi 6 février 2015

Description :

Quand on pense à SQL, on pense généralement à l'utilisation qu'il est possible d'en faire pour des opérations de consultation et/ou de mise à jour unitaire, et bien évidemment on pense à ses puissantes fonctions d'agrégation, dans le cadre d'analyses statistiques. Mais il est un domaine auquel on pense moins, c'est l'utilisation de SQL pour le développement de processus métiers. On n'imagine pas, par exemple, qu'il soit possible de générer un tableau d'amortissement d'immobilisation au moyen d'une seule requête SQL. C'est pourtant ce que nous allons voir dans cet article.

Environnement iSeries

Dans les années 90, j'avais travaillé sur le développement d'un logiciel de gestion de société d'HLM, et j'avais notamment développé l'intégralité du module de gestion des immobilisations (module incluant la gestion des amortissements). Je m'étais régalé sur le développement de ce module (écrit en ADELIA), qui incluait différents types d'amortissements (linéaire, dégressif, linéaire-progressif). J'avoue que j'ai un peu oublié comment se calcule l'amortissement dégressif (légèrement plus complexe sur la fin du tableau que le linéaire), ainsi que le linéaire progressif (mode de calcul que je n'ai vu pratiquer que dans le secteur HLM). Je n'ai en revanche pas eu trop de mal pour me rappeler du principe de calcul de l'amortissement linéaire, mais je n'ai pas vraiment de mérite car il est relativement simple. C'est celui-ci que nous étudierons dans cet article, et surtout que nous allons implémenter en SQL DB2.

Pour ne rien vous cacher, cela faisait un moment que je me triturais le cerveau pour trouver un moyen de générer un tableau d'amortissement au moyen d'une requête SQL. Je me disais que cela devait être possible, mais je butais sur un problème, et c'est la découverte des RCTE (Recursive Common Table Expression) qui m'a apporté la solution. Pour rappel, j'avais présenté la technique des RCTE dans un précédent article de XDocs400 et je vous redonne juste un petit bout de code SQL pour rappeler le principe. Voici donc un exemple de RCTE qui permet de générer un jeu de données de 10 lignes :

```
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 ;
```

Pour ceux qui ne connaîtraient pas le principe d'un tableau d'amortissement, voici un exemple :

- produit acheté pour une valeur de 100.000 euros
- durée de l'amortissement : 10 ans
- taux de l'amortissement : 10%
- date d'acquisition du bien : 1er juillet 2001

Le bien étant acquis en cours d'année (en l'occurrence le 1er juillet), la valeur de la première annuité d'amortissement sera divisée par 2, et la moitié résiduelle sera répercutée sur la 11ème année. On aura donc un amortissement sur 11 ans au lieu des 10 ans prévus initialement. Cette subtilité impacte bien évidemment l'algorithme de calcul du tableau.

A partir des données indiquées ci-dessus, voici le tableau que l'on souhaite obtenir :

```
ANNEE VNC_DEBUT_EXERCICE    ANNUITE    VNC_FIN_EXERCICE
 ----- ------------------ ------------- ----------------
 2001      100000,000000    5000,000000     95000,000000
 2002       95000,000000   10000,000000     85000,000000
 2003       85000,000000   10000,000000     75000,000000
 2004       75000,000000   10000,000000     65000,000000
 2005       65000,000000   10000,000000     55000,000000
 2006       55000,000000   10000,000000     45000,000000
 2007       45000,000000   10000,000000     35000,000000
 2008       35000,000000   10000,000000     25000,000000
 2009       25000,000000   10000,000000     15000,000000
 2010       15000,000000   10000,000000      5000,000000
 2011        5000,000000    5000,000000         0,000000
```

Pour pouvoir réaliser cette performance, nous allons utiliser une cascade de CTE (Common Table Expression), et une RCTE (CTE récursive).

Pour démarrer dans de bonnes conditions, je vous propose de créer une première CTE qui consistera à préparer les données utilisées pour la génération de notre tableau. On se contentera d'afficher le contenu de cette première CTE pour vérifier qu'elle fonctionne.

```
TMP_VALINITIALES (CAPITAL, TAUX, ANNEE_DEPART, NB_MOIS_AN1, NB_ANNUITES) as  (
 SELECT CAST(100000 AS DEC(11, 0)), -- capital initial
 CAST((10+0.0) / 100 AS DEC(5, 2)), -- taux d'amortissment
 CAST(2001 AS INTEGER), -- année de départ
 CAST(6 AS INTEGER), -- nbre mois pour calcul 1ère annuitée
 CAST(10 AS INTEGER) -- nombre d'années d'amortissement
 FROM SYSIBM.SYSDUMMY1
 )
 SELECT * FROM TMP_VALINITIALES ;

 CAPITAL       TAUX    ANNEE_DEPART NB_MOIS_AN1 NB_ANNUITES
 ------------- ------- ------------ ----------- -----------
 100000,    0,10         2001           6          10
```

OK, ça marche, on a notre capital initial, notre taux, notre année de départ, le nombre de mois d'amortissement pour l'année de départ, et le nombre d'annuités. Si l'on souhaite par la suite travailler avec d'autres valeurs, il sera facile de les modifier étant donné qu'elles sont regroupées dans cette première CTE.

On peut maintenant enlever le "SELECT * FROM TMP_VALINITIALES" et ajouter une seconde CTE qui est la suivante :

```
-- Conversion de certaines valeurs initiales en format décimal et précalcul de certaines données
 TMP_VALDEPART (CAPITAL, TAUX, ANNEE_DEPART, NB_MOIS_AN1, PRORATA, NB_ANNUITES) AS (
 SELECT CAPITAL, TAUX, ANNEE_DEPART, NB_MOIS_AN1,
 CAST( CAST(NB_MOIS_AN1 AS DEC(2, 0)) / 12.0 AS DEC(5, 4)) as PRORATA, -- prorata de la première annuité au
format décimal
 CASE WHEN NB_MOIS_AN1 = 12 THEN NB_ANNUITES ELSE NB_ANNUITES + 1 END as NB_ANNUITES -- nombre d'années
d'amortissement de type entier
 FROM TMP_VALINITIALES
 )
 select * from TMP_VALDEPART;

 CAPITAL       TAUX    ANNEE_DEPART NB_MOIS_AN1 PRORATA NB_ANNUITES
 ------------- ------- ------------ ----------- ------- -----------
 100000,    0,10         2001           6  0,5000          11
```

L'objectif de cette seconde CTE était d'ajouter les colonnes PRORATA et NB_ANNUITES, qui vont être nécessaires pour la suite du calcul. Après avoir vérifié que cette seconde CTE fonctionne bien, nous pouvons supprimer la dernière ligne contenant le "SELECT * FROM TMP_VALDEPART", et le remplacer par une troisième CTE, ou plutôt RCTE, et dans la foulée une 4ème CTE, histoire d'avancer un peu plus vite :

```
-- Génération des lignes du tableau
 GEN_IDS(NX) AS (
 SELECT 1 as N1 FROM SYSIBM.SYSDUMMY1
 UNION ALL
```

```
 SELECT NX+1 as N2 FROM GEN_IDS WHERE NX < (SELECT NB_ANNUITES FROM  TMP_VALDEPART)
 ) ,
 GEN_LIGNES AS (
 SELECT cast(NX as integer) as VAL_INC FROM GEN_IDS
 )
 SELECT * FROM GEN_LIGNES ;
```

L'objectif du code ci-dessus est simplement de générer 11 lignes qui nous serviront de point d'appui pour le calcul de nos 11 annuités d'amortissement (je rappelle que notre bien a été acquis en cours d'année, donc il est normal que l'amortissement se fasse sur 11 années au lieu de 10).

Je ne vous donne pas le détail des 11 lignes générées ici, je vous invite plutôt à vérifier par vous même que cela fonctionne. Ensuite supprimez la dernière ligne "SELECT * FROM GEN_LIGNES" et remplacez-la par la CTE ci-dessous :

```
-- Calcul d'un premier tableau d'annuités théoriques
 TMP_TABLEAU1 AS (
 SELECT VAL_INC, (SELECT TAUX FROM TMP_VALDEPART) as taux,
 (SELECT CAPITAL FROM TMP_VALDEPART) AS CAPITAL_INITIAL,
 CASE WHEN VAL_INC = 1 THEN
 -- la première année n'est pas forcément une année pleine, d'où application d'un prorata temporis sur la
mensualité
 (SELECT CAPITAL FROM TMP_VALDEPART) * (SELECT TAUX FROM TMP_VALDEPART) * (SELECT PRORATA FROM
TMP_VALDEPART)
 ELSE
 -- mensualité théorique pour une année pleine
 (SELECT CAPITAL FROM TMP_VALDEPART) * (SELECT TAUX FROM TMP_VALDEPART)
 END AS ANNUITES
 FROM GEN_LIGNES
 )
 select * from TMP_TABLEAU1;
```

Si vous exécutez la requête en l'état, vous devez obtenir un premier tableau qui a belle allure, même si ce n'est pas encore un véritable tableau d'amortissement :

```
VAL_INC  TAUX   CAP_INITIAL ANNUITES
 ------- ------ ------------ -------------
 1      0,10     100000,     5000,000000
 2      0,10     100000,    10000,000000
 3      0,10     100000,    10000,000000
 4      0,10     100000,    10000,000000
 5      0,10     100000,    10000,000000
 6      0,10     100000,    10000,000000
 7      0,10     100000,    10000,000000
 8      0,10     100000,    10000,000000
 9      0,10     100000,    10000,000000
 10      0,10     100000,    10000,000000
 11      0,10     100000,    10000,000000
```

Il nous reste une avant dernière CTE destinée à mettre en forme notre tableau d'amortissement :

```
-- Second tableau théorique incluant le calcul du CRD
 TMP_TABLEAU2 AS (
```

```
 SELECT A.VAL_INC, A.CAPITAL_INITIAL, A.ANNUITES,
 A.CAPITAL_INITIAL - (SELECT SUM(ANNUITES) FROM TMP_TABLEAU1 X WHERE X.VAL_INC <= A.VAL_INC) AS CRD
 FROM TMP_TABLEAU1 A
 ),
```

Et une dernière CTE destinée à rattraper la dernière annuité, dans le cas des immobilisations acquises en cours d'exercice, comme c'est le cas dans notre exemple :

```
-- Rattrapage de la dernière annuité si CRD négatif sur la dernière année
 TMP_TABLEAU3 AS (
 SELECT A.VAL_INC + (SELECT ANNEE_DEPART FROM TMP_VALDEPART) - 1 AS ANNEE,
 CASE WHEN CRD < 0 THEN
 A.ANNUITES + CRD
 ELSE
 A.ANNUITES
 END AS ANNUITE,
 CASE WHEN CRD < 0 THEN
 0
 ELSE
 A.CRD
 END AS VNC_FIN_EXERCICE
 FROM TMP_TABLEAU2 A
 )
 SELECT ANNEE, VNC_FIN_EXERCICE + ANNUITE AS VNC_DEBUT_EXERCICE, ANNUITE, VNC_FIN_EXERCICE
 FROM TMP_TABLEAU3
```

Si tout s'est bien passé, votre tableau d'amortissement final devrait ressembler à ceci :

```
ANNEE VNC_DEBUT_EXERCICE    ANNUITE    VNC_FIN_EXERCICE
 ----- ------------------ ------------- ----------------
 2001      100000,000000    5000,000000     95000,000000
 2002       95000,000000   10000,000000     85000,000000
 2003       85000,000000   10000,000000     75000,000000
 2004       75000,000000   10000,000000     65000,000000
 2005       65000,000000   10000,000000     55000,000000
 2006       55000,000000   10000,000000     45000,000000
 2007       45000,000000   10000,000000     35000,000000
 2008       35000,000000   10000,000000     25000,000000
 2009       25000,000000   10000,000000     15000,000000
 2010       15000,000000   10000,000000      5000,000000
 2011        5000,000000    5000,000000         0,000000
```

Vous trouverez, attaché à cet article, un fichier texte contenant l'intégralité de la requête SQL présentée dans cet article. Ce code a été testé sur DB2 Express C (en version 9.7) et DB2 for i (en V7).

Il est bien évident que ce type d'approche est quelque peu radical, et peut heurter quand on n'y est pas habitué. Il aurait sans doute été plus simple d'écrire ce même code métier en PL/SQL selon une algorithmie plus "procédurale". C'est une affaire de style, et honnêtement je n'ai aucune préférence.

J'espère surtout, au travers de cet article, vous avoir fait toucher du doigt toute la puissance que la base de données DB2 est en mesure de mettre à votre disposition, dans le cadre du développement de code "métier". Peut être cet article donnera-t-il à certains lecteurs des idées pour implémenter leur propre code métier, si c'est le cas j'en serai vraiment ravi.

Tiens, j'essaierai un de ces jours de m'atteler au calcul de l'amortissement dégressif, j'ai dans l'idée que le calcul des dernières lignes du tableau sera gratiné :)