


Intégrez un modèle de calcul à vos requêtes SQL : la clause MODEL d'Oracle

par Rob van Wijk ([About Oracle](#)) Antoine Dinimant (traducteur) ([antoun.developpez.com](#))

Date de publication : 7 août 2008

Dernière mise à jour :

La clause MODEL, spécifique à Oracle et introduite par la version 10g, permet de définir un modèle de calcul matriciel à l'intérieur d'une simple requête SQL. Pour ceux qui ne connaissent que la modélisation relationnelle normalisée, disons que MODEL permet de réaliser des calculs complexes, au besoin itératifs, sans programmation PL/SQL et dans un simple SELECT. Pour ceux qui ont déjà tâté des concepts décisionnels (*Business Intelligence*), disons que MODEL permet de définir dimensions et indicateurs par un mapping proche du ROLAP, à l'intérieur d'un simple SELECT. Ce tutoriel présente la clause MODEL en vous initiant à son utilisation.

L'auteur : Rob van Wijk est consultant senior sur Oracle pour Ciber Nederland, et tient un blog nommé  **About Oracle**.

La source : ce tutoriel est traduit du blog de Rob, et adapté pour Développez.com. Cette présentation et quelques notes en italiques ont été ajoutées par le traducteur. Les versions originales sont disponibles ici : *SQL Model Clause Tutorial*, **part one**, **part two**.

I - Quelques exemples pour débiter.....	3
II - Statistiques et simulation.....	7
II-A - Références multi-cellules.....	7
II-B - Modèles de référence.....	9
II-C - Itération.....	10

I - Quelques exemples pour débiter

La clause MODEL du SQL d'Oracle vous permet de construire un *modèle* composé d'une ou plusieurs matrices, avec un nombre variable de dimensions. Le modèle utilise une partie des colonnes disponibles dans la clause FROM. Il doit contenir au moins une dimension et un indicateur (*measure*), et éventuellement une ou plusieurs partitions. Le modèle peut être représenté comme un classeur de tableur, contenant des feuilles de calcul différentes pour chaque valeur calculée (indicateur). Une feuille de calcul présente un axe horizontal et un axe vertical (deux dimensions) ; vous pouvez séparer votre feuille en plusieurs zones identiques, chacune dédiée à un pays ou un département différent (partition).

La figure ci-dessous présente un modèle tiré de la classique table EMP, avec *deptno* comme partition, *empno* comme dimension, et les deux indicateurs *sal* et *comm*.

	indicateur sal	indicateur comm
Partition deptno 10	sal[7782] = 2450 sal[7839] = 5000 sal[7782] = 1300	comm[7782] = null comm[7839] = null comm[7782] = null
Partition deptno 20	sal[7369] = 800 sal[7566] = 2975 sal[7788] = 3000 sal[7876] = 1100 sal[7902] = 3000	comm[7369] = null comm[7566] = null comm[7788] = null comm[7876] = null comm[7902] = null
Partition deptno 30	sal[7499] = 1600 sal[7521] = 1250 sal[7654] = 1250	comm[7499] = 300 comm[7521] = 500 comm[7654] = 1400

Une fois le modèle mis en place, vous définissez les *règles* qui modifient la valeur des indicateurs. Ce sont ces règles qui sont au coeur de la clause MODEL. Avec quelques règles, vous pouvez faire des calculs complexes sur vos données, et même créer de nouvelles lignes. Dans le modèle, les colonnes d'indicateurs deviennent des tableaux indexés (*hash tables*) dont les clés sont les colonnes de dimension. Oracle y applique les règles de calcul à toutes les partitions. Une fois ces calculs effectués, le modèle est re-converti en lignes de données traditionnelles.

Selon mon expérience, le diagramme syntaxique de la clause MODEL présenté par la documentation Oracle est passablement complexe et tend à effrayer tout le monde. Pour éviter d'en arriver là, je vais tenter une autre approche, en utilisant un série de petits exemples sur la table EMP, en commençant par les plus simples, et en introduisant progressivement de nouveaux éléments. A la fin de cet article, vous trouverez un script que vous pouvez télécharger et exécuter sur votre propre base de données.

```

SQL> select empno
2         , ename
3         , sal
4 from emp
5 where deptno = 10
  
```

```
6 /
```

EMPNO	ENAME	SAL
7782	CLARK	2450
7839	KING	5000
7934	MILLER	1300

3 lignes sélectionnées


Voici le contenu (classique) de la table EMP pour le département 10. L'équivalent de cette requête SQL avec une clause MODEL (qui ne fait rien) est :

```
SQL> select empno
2      , ename
3      , sal
4  from emp
5  where deptno = 10
6  model
7      dimension by (empno)
8      measures (ename, sal)
9      rules
10     ()
11 /
```

EMPNO	ENAME	SAL
7782	CLARK	2450
7839	KING	5000
7934	MILLER	1300

3 lignes sélectionnées

Nous avons ici deux indicateurs, *ename* et *sal*, et une dimension, *empno*. La combinaison des colonnes de partitions et de dimensions doit permettre d'identifier chaque cellule de manière unique. Cette contrainte est vérifiée à l'exécution, et sa violation produira une erreur ORA-32638.

 *Autrement dit, cette combinaison des dimensions et partitions doit pouvoir servir de clef unique pour les lignes qui vont être traitées par la clause MODEL. Cette limite est la plus importante (elle peut imposer une agrégation préalable), mais aussi la plus révélatrice du fonctionnement du modèle. Une matrice de calcul est créée, où dimensions et partitions vont servir d'axes de coordonnées. Les habitués d'Hyperion Essbase peuvent considérer les partitions comme des dimensions sparses, tandis que les dimensions de la clause MODEL seraient les dimensions denses d'Essbase.*

Comme il y a deux indicateurs, Oracle crée deux tableaux à une dimension, indicés par *empno*. Sur la ligne 9, vous voyez le mot-clé RULES, sans contenu pour le moment. RULES est optionnel, mais par souci de clarté je l'utiliserai systématiquement.

```
SQL> select empno
2      , ename
3      , sal
4  from emp
5  where deptno = 10
6  model
7      dimension by (empno)
8      measures (sal)
9      rules
10     ()
11 /
```

Une fois le modèle de calcul appliqué, toutes les partitions, dimensions et indicateurs sont re-convertis en colonnes à l'intérieur de lignes classiques, ce qui veut dire que vous ne pouvez avoir dans le SELECT que des colonnes citées dans le modèle. Si, par exemple, je n'avais pas inclus la colonne *ename* comme indicateur, j'aurais eu ce message d'erreur :

```
ERREUR à la ligne 2 :  
ORA-32614: expression MODEL SELECT interdite
```


Avec l'exemple suivant, je crée une nouvelle ligne :

```
SQL> select empno  
2      , ename  
3      , sal  
4 from emp  
5 where deptno = 10  
6 model  
7   dimension by (empno)  
8   measures (ename,sal)  
9   rules  
10  ( ename[7777] = 'VAN WIJK'  
11  )  
12 /
```

EMPNO	ENAME	SAL
7782	CLARK	2450
7839	KING	5000
7934	MILLER	1300
7777	VAN WIJK	

4 lignes sélectionnées

La règle en ligne 10 indique que l'indicateur *ename* est croisé avec la dimension *empno* et prend la valeur « VAN WIJK ». Si la table EMP avait déjà un *empno* 7777, la règle aurait écrasé l'*ename* correspondant. Mais comme 7777 n'existe pas dans EMP, une nouvelle cellule a été créée, qui se traduit par une nouvelle ligne dans le résultat du SELECT. Notez bien que cette ligne n'est pas insérée dans une table, mais seulement affichée dans le résultat.

 La clause MODEL ne s'utilise que dans une requête SELECT. Cela inclut le INSERT... SELECT et le MERGE INTO... USING (SELECT...).

Avec une seconde règle, vous pourriez également renseigner la colonne *sal* :

```
SQL> select empno  
2      , ename  
3      , sal  
4 from emp  
5 where deptno = 10  
6 model  
7   dimension by (empno)  
8   measures (ename,sal)  
9   rules  
10  ( ename[7777] = 'VAN WIJK'  
11  , sal[7777] = 2500  
12  )  
13 /
```

EMPNO	ENAME	SAL
7782	CLARK	2450
7839	KING	5000
7934	MILLER	1300
7777	VAN WIJK	2500

4 lignes sélectionnée

La requête renvoie à la fois les lignes existantes et les nouvelles lignes. L'expression-clé RETURN UPDATED ROWS vous permet de ne renvoyer que les lignes modifiées ou créées :

```
SQL> select empno
2      , ename
3      , sal
4  from emp
5  where deptno = 10
6  model
7    return updated rows
8    dimension by (empno)
9    mesures (ename,sal)
10   rules
11   ( ename[7777] = 'VAN WIJK'
12     , sal[7777] = 2500
13     )
14 /
```

EMPNO	ENAME	SAL
7777	VAN WIJK	2500

1 ligne sélectionnée

Tous les calculs sont exécutés sur chaque partition. Pour le constater, enlevons le filtre « deptno = 10 » et affichons le *deptno*. Comme partition ou comme indicateur ? Pour commencer, testons ce qui se passe si nous définissons *deptno* comme indicateur :

```
SQL> select empno
2      , ename
3      , sal
4      , deptno
5  from emp
6  model
7    return updated rows
8    dimension by (empno)
9    mesures (ename,sal,deptno)
10   rules
11   ( ename[7777] = 'VAN WIJK'
12     )
13 /
```

EMPNO	ENAME	SAL	DEPTNO
7777	VAN WIJK		

1 ligne sélectionnée

Une ligne est créée, comme attendu, avec un *deptno* NULL. Testons maintenant avec *deptno* comme partition :

```
SQL> select empno
2      , ename
3      , sal
4      , deptno
5  from emp
6  model
7    return updated rows
8    partition by (deptno)
9    dimension by (empno)
```


```

10  measures (ename, sal)
11  rules
12  ( ename[7777] = 'VAN WIJK'
13  )
14 /
  
```

EMPNO	ENAME	SAL	DEPTNO
7777	VAN WIJK		30
7777	VAN WIJK		20
7777	VAN WIJK		10

3 lignes sélectionnées

Une nette différence ! Dans la table EMP, il n'y a que des *deptno* 10, 20 et 30, ce qui fait donc trois partitions dans le modèle. La règle est appliquée aux trois, ce qui donne donc trois nouvelles lignes.

 *Pourrait-on inclure deptno comme dimension ? Oui, mais cela forcerait à indiquer dans la règle le numéro de département comme caractéristique de l'indicateur : par exemple, ename[7777, 10] = 'VAN WIJK', ename[7777, 20] = 'DINIMANT', ename[7777, 30] = 'SCHNEIDER'. Plus largement, les dimensions permettent de définir les axes qui interviendront dans le calcul des indicateurs. Au contraire, les calculs sont exécutés à l'identique d'une partition à l'autre ; on peut donc dire que les partitions sont précisément les éléments qui ne jouent pas dans le calcul.*

Pour l'instant, ces exemples ne vous ont sans doute pas convaincu d'utiliser la clause MODEL. Nous avons ajouté des tas de lignes de code SQL pour un résultat très simple : une utilisation créative de l'opérateur ensembliste UNION ALL et de la table DUAL nous aurait permis de faire la même chose. L'objectif de cette première partie était simplement de vous montrer les bases de l'utilisation de la clause MODEL. Dans la seconde partie, je vous montrerai les références multi-cellules, les modèles de références et les itérations. Et c'est là que les choses deviendront autrement plus intéressantes.

Source [Script SQL contenant toutes les requêtes utilisées](#)

II - Statistiques et simulation

II-A - Références multi-cellules

Il est possible de traiter plusieurs cellules avec une seule règle, en utilisant une *référence multi-cellules*. Pour illustrer cette notion, je vais introduire une nouvelle table de démonstration avec une clé primaire composée, afin de pouvoir travailler sur des dimensions multiples. Cette table contient les ventes mensuelles de deux livres en 2005 et 2006. Voici la requête de création :

```

SQL> create table sales
2  as
3  select 'The Da Vinci Code' book, date '2005-03-01' month, 'Netherlands' country, 5 amount from
dual union all
4  select 'The Da Vinci Code', date '2005-04-01', 'Netherlands', 8 from dual union all
5  select 'The Da Vinci Code', date '2005-05-01', 'Netherlands', 3 from dual union all
6  select 'The Da Vinci Code', date '2005-07-01', 'Netherlands', 2 from dual union all
7  select 'The Da Vinci Code', date '2005-10-01', 'Netherlands', 1 from dual union all
8  select 'The Da Vinci Code', date '2005-02-01', 'United Kingdom', 15 from dual union all
9  select 'The Da Vinci Code', date '2005-03-01', 'United Kingdom', 33 from dual union all
10 select 'The Da Vinci Code', date '2005-04-01', 'United Kingdom', 47 from dual union all
11 select 'The Da Vinci Code', date '2005-05-01', 'United Kingdom', 44 from dual union all
12 select 'The Da Vinci Code', date '2005-06-01', 'United Kingdom', 11 from dual union all
13 select 'The Da Vinci Code', date '2005-08-01', 'United Kingdom', 2 from dual union all
  
```

```

14 select 'The Da Vinci Code', date '2005-05-01', 'France', 2 from dual union all
15 select 'The Da Vinci Code', date '2005-08-01', 'France', 3 from dual union all
16 select 'The Da Vinci Code', date '2006-01-01', 'France', 4 from dual union all
17 select 'Bosatlas', date '2005-01-01', 'Netherlands', 102 from dual union all
18 select 'Bosatlas', date '2005-02-01', 'Netherlands', 55 from dual union all
19 select 'Bosatlas', date '2005-03-01', 'Netherlands', 68 from dual union all
20 select 'Bosatlas', date '2005-04-01', 'Netherlands', 42 from dual union all
21 select 'Bosatlas', date '2005-05-01', 'Netherlands', 87 from dual union all
22 select 'Bosatlas', date '2005-06-01', 'Netherlands', 40 from dual union all
23 select 'Bosatlas', date '2005-07-01', 'Netherlands', 31 from dual union all
24 select 'Bosatlas', date '2005-08-01', 'Netherlands', 26 from dual union all
25 select 'Bosatlas', date '2005-09-01', 'Netherlands', 22 from dual union all
26 select 'Bosatlas', date '2005-10-01', 'Netherlands', 23 from dual union all
27 select 'Bosatlas', date '2005-11-01', 'Netherlands', 88 from dual union all
28 select 'Bosatlas', date '2005-12-01', 'Netherlands', 143 from dual union all
29 select 'Bosatlas', date '2006-01-01', 'Netherlands', 31 from dual union all
30 select 'Bosatlas', date '2006-02-01', 'Netherlands', 18 from dual union all
31 select 'Bosatlas', date '2006-03-01', 'Netherlands', 15 from dual union all
32 select 'Bosatlas', date '2006-04-01', 'Netherlands', 11 from dual union all
33 select 'Bosatlas', date '2006-05-01', 'Netherlands', 17 from dual union all
34 select 'Bosatlas', date '2006-06-01', 'Netherlands', 9 from dual union all
35 select 'Bosatlas', date '2006-07-01', 'Netherlands', 12 from dual union all
36 select 'Bosatlas', date '2006-08-01', 'Netherlands', 20 from dual union all
37 select 'Bosatlas', date '2006-09-01', 'Netherlands', 4 from dual union all
38 select 'Bosatlas', date '2006-10-01', 'Netherlands', 5 from dual union all
39 select 'Bosatlas', date '2006-11-01', 'Netherlands', 1 from dual union all
40 select 'Bosatlas', date '2006-12-01', 'Netherlands', 1 from dual
41 /
  
```

Table créée

Le livre titré *Bosatlas* a une ligne chaque mois, mais seulement aux Pays-Bas (*Netherlands*). Le *Da Vinci Code* a été vendu dans les trois pays, mais pas tous les mois. Les colonnes *book* (livre), *month* (mois) et *country* (pays) forment la clé primaire de cette table. Pour que les mois apparaissent plus clairement, je modifie ainsi le format de date :

```

SQL> alter session set nls_date_format = 'fmmonth yyyy'
2 /
  
```

L'exemple suivant présente comment doubler les ventes de *Bosatlas* à partir de juin 2006 :

```

SQL> select book
2      , month
3      , country
4      , amount
5 from sales
6 model
7      return updated rows
8      partition by (country)
9      dimension by (book,month)
10     measures (amount)
11     rules
12     ( amount['Bosatlas',month > date '2006-06-01'] =
13       amount['Bosatlas',cv(month)] * 2
14     )
15 /
  
```

BOOK	MONTH	COUNTRY	AMOUNT
Bosatlas	juillet 2006	Netherlands	24
Bosatlas	août 2006	Netherlands	40
Bosatlas	août 2006	Netherlands	8
Bosatlas	octobre 2006	Netherlands	10
Bosatlas	novembre 2006	Netherlands	2
Bosatlas	décembre 2006	Netherlands	2

6 lignes sélectionnées

L'indicateur *amount* a maintenant deux dimensions. Pour nous référer à une cellule du modèle, nous devons donc spécifier à la fois un livre et un mois. Avec une seule règle, je modifie six cellules, grâce à l'expression *month > date '2006-06-01'* à gauche du = de la règle. A droite, la fonction *cv* est utilisée pour se référer à la valeur correspondante côté gauche, *cv* indiquant la valeur courante. A la place de *cv(month)*, j'aurais pu aussi utiliser *cv()*, car il n'y a qu'une référence multi-cellules. Même avec plus de références multi-cellules, *cv()* reste utilisable, tant que la dimension concernée est évidente. Si ce n'est pas le cas, vous aurez une ORA-32611 :

ERREUR à la ligne N :
 ORA-32611: utilisation incorrecte de l'opérateur MODEL CV

Pour indiquer l'ensemble des valeurs d'une dimension, plutôt qu'un sous-ensemble, utilisez le mot-clé ANY :

```
SQL> select book
2      , month
3      , country
4      , amount
5  from sales
6  model
7  return updated rows
8  partition by (country)
9  dimension by (book, month)
10 measures (amount)
11 rules
12   ( amount[any, date '2005-08-01'] = 200
13     )
14 order by book, month
15 /
```

BOOK	MONTH	COUNTRY	AMOUNT
Bosatlas	août 2005	Netherlands	200
The Da Vinci Code	août 2005	United Kingdom	200
The Da Vinci Code	août 2005	France	200

3 lignes sélectionnées

Dans cette règle, le mot-clé ANY est utilisé pour indiquer tous les livres qui ont eu des ventes en août 2005. Les références multi-cellules sont également possibles avec le mot-clé FOR (par exemple : *amount['Bosatlas', for month from date '2005-03-01' to date '2005-08-01' increment 1]*), avec BETWEEN (par exemple: *amount['Bosatlas'], month between date '2005-03-01' and date '2005-08-01'*)), et avec tous les autres opérateurs de comparaison.

II-B - Modèles de référence

Les modèles de référence sont des sous-modèles dans un modèle. Le modèle principal peut utiliser toutes les valeurs du modèle de référence, mais ces valeurs ne peuvent pas directement être placées dans le SELECT. Nous parlons ici de données auxiliaires, qui ne sont lisibles que par le modèle principal. Pour illustrer cette idée, voici une table supplémentaire contenant les prix des deux livres :

```
SQL> create table prices
2  as
3  select 'Bosatlas' book, 42.95 price from dual union all
4  select 'The Da Vinci Code', 19.95 from dual
5  /
```

Table créée

Les deux lignes de la table des prix sont fournies au modèle principal par le modèle de référence :

```
SQL> select book
```

```

2      , month
3      , country
4      , amount
5      , to_char(turnover, '99G990D00') turnover
6  from sales
7  where month between date '2005-07-01' and date '2005-12-31'
8  model
9      reference prices on (select book, price from prices)
10     dimension by (book)
11     measures (price)
12  main result
13     partition by (country)
14     dimension by (book, month)
15     measures (0 as turnover, amount)
16     rules
17     ( turnover[any,any] = amount[cv(),cv()] * price[cv(book)]
18       )
19  order by book
20      , month
21  /

```

BOOK	MONTH	COUNTRY	AMOUNT	TURNOVER
Bosatlas	juillet 2005	Netherlands	31	1.331,45
Bosatlas	août 2005	Netherlands	26	1.116,70
Bosatlas	septembre 2005	Netherlands	22	944,90
Bosatlas	octobre 2005	Netherlands	23	987,85
Bosatlas	novembre 2005	Netherlands	88	3.779,60
Bosatlas	décembre 2005	Netherlands	143	6.141,85
The Da Vinci Code	juillet 2005	Netherlands	2	39,90
The Da Vinci Code	août 2005	United Kingdom	2	39,90
The Da Vinci Code	août 2005	France	3	59,85
The Da Vinci Code	octobre 2005	Netherlands	1	19,95

10 lignes sélectionnées

Le modèle de référence contient donc lui aussi des dimensions et des indicateurs, mais il ne peut avoir ni partition, ni règle propre. Notez également que les deux modèles sont maintenant nommés : le modèle de référence est nommé *prices* (introduit par le mot-clé REFERENCE) et le modèle principal *result* (préfixé par MAIN). L'indicateur *price* du modèle de référence est utilisé dans le modèle principal, au moyen de l'expression *price[cv(book)]*. Enfin, nous avons créé un nouvel indicateur nommé *turnover* (chiffre d'affaires), initialisé à zéro pour toutes les cellules existantes.

II-C - Itération

L'itération permet de répéter la même règle plusieurs fois sur le même modèle. Dans l'exemple ci-dessous, les ventes du *Da Vinci Code* sont élevées à la puissance quatre, sans utiliser la fonction POWER, juste pour vous montrer le fonctionnement de l'itération. Pour cela, il nous fait multiplier le nombre de ventes trois fois par le nombre original, comme ceci :

```

SQL> select book
2      , month
3      , country
4      , a1 amount
5      , a2 amount_to_the_fourth
6  from sales
7  where book = 'The Da Vinci Code'
8      and country = 'Netherlands'
9  model
10     partition by (country)
11     dimension by (book,month)
12     measures (amount a1, amount a2)
13     rules iterate (3)
14     ( a2[any,any] = a2[cv(),cv()] * a1[cv(),cv()]
15       )
16  order by month
17  /

```

BOOK	MONTH	COUNTRY	AMOUNT	AMOUNT_TO_THE_FOURTH
The Da Vinci Code	mars 2005	Netherlands	5	625
The Da Vinci Code	avril 2005	Netherlands	8	4096
The Da Vinci Code	mai 2005	Netherlands	3	81
The Da Vinci Code	juillet 2005	Netherlands	2	16
The Da Vinci Code	octobre 2005	Netherlands	1	1

5 lignes renvoyées.

Nous plaçons le mot-clé ITERATE, suivi du nombre d'itérations, directement après RULES. L'indicateur a1 contient le nombre de ventes original, inchangé. L'indicateur a2 contient initialement les nombres 5, 8, 3, 2 et 1, comme a1. A chaque itération, ces nombres sont multipliés par a1. Après la première itération, a2 contient 25, 64, 9, 4 et 1, et après la deuxième 125, 512, 8 et 1. Les résultats finaux sont présentés dans le jeu de données ci-dessus.

Pour travailler avec les itérations, il est très pratique d'utiliser le mot-clé UNTIL, suivi d'une expression booléenne permettant d'éviter les itération inutiles. Le terme ITERATION_NUMBER, qui donne le numéro de l'itération en cours, peut être utilisé dans la règle. Attention, il commence à 0. Dans l'exemple ci-dessus, il irait donc de 0 à 2.

Pour vous montrer comment UNTIL et ITERATION_NUMBER fonctionnent, voici un exemple où l'on génère une prévision des ventes futures de chaque livre en supposant qu'elles baisseront de 75% par an. Chaque année, nous vendrons donc seulement 25% des ventes du même mois de l'année précédente (en arrondissant vers le bas). L'itération doit se répéter jusqu'à ce qu'on ne vende plus un seul livre de toute l'année. Voici la requête pour ce faire :

```
SQL> select book
2      , country
3      , to_date(to_char(y) || to_char(m), 'yyyymm') month
4      , amount
5 from sales
6 where book = 'Bosatlas'
7      and extract (year from month) = 2006
8 model
9      partition by ( book, country)
10     dimension by ( extract(year from month) y, extract(month from month) m)
11     measures (amount, 0 max_monthly_amount)
12     rules upsert all
13     iterate (100) until (max_monthly_amount[2007+iteration_number,1] < 4)
14     ( amount[2007+iteration_number,any]
15       = trunc(amount[2006+iteration_number,cv()]/4)
16       , max_monthly_amount[2007+iteration_number,1]
17       = max(amount)[2007+iteration_number,m between 1 and 12]
18     )
19 order by y, m
20 /
```

Cet exemple est beaucoup plus complexe que ceux que nous avons vus jusqu'à maintenant. Comme vous pouvez le constater dans la clause WHERE, nous commençons par les 12 lignes du livre *Bosatlas* en 2006. La colonne *month* est séparée en deux dimensions, l'année (*year*) et le numéro du mois (*month*), ce qui permet de traiter en une seule itération les douze mois d'une année complète.



Pour empêcher l'apparition de la dernière année avec des ventes à zéro tous les mois, j'ai créé un indicateur auxiliaire appelé *max_monthly_amount*. Cet indicateur est alimenté par une fonction analytique, *max(amount) [2007+iteration_number, month_number between 1 and 12]* et donne le plus haut nombre de ventes mensuelles pour l'année courante. Cet indicateur est utilisé dans la clause UNTIL. Les itérations doivent s'arrêter quand ce maximum est inférieur à 4. En effet, tant qu'il y a au moins 4 ventes, cela donnera des ventes supérieures à 0 l'année suivante. Avec les données courantes, seules deux des 100 itérations prévues sont exécutées, renvoyant ce résultat :

BOOK	COUNTRY	MONTH	AMOUNT
Bosatlas	Netherlands	janvier 2006	31
Bosatlas	Netherlands	février 2006	18

Bosatlas	Netherlands	mars 2006	15
Bosatlas	Netherlands	avril 2006	11
Bosatlas	Netherlands	mai 2006	17
Bosatlas	Netherlands	juin 2006	9
Bosatlas	Netherlands	juillet 2006	12
Bosatlas	Netherlands	août 2006	20
Bosatlas	Netherlands	septembre 2006	4
Bosatlas	Netherlands	octobre 2006	5
Bosatlas	Netherlands	novembre 2006	1
Bosatlas	Netherlands	décembre 2006	1
Bosatlas	Netherlands	janvier 2007	7
Bosatlas	Netherlands	février 2007	4
Bosatlas	Netherlands	mars 2007	3
Bosatlas	Netherlands	avril 2007	2
Bosatlas	Netherlands	mai 2007	4
Bosatlas	Netherlands	juin 2007	2
Bosatlas	Netherlands	juillet 2007	3
Bosatlas	Netherlands	août 2007	5
Bosatlas	Netherlands	septembre 2007	1
Bosatlas	Netherlands	octobre 2007	1
Bosatlas	Netherlands	novembre 2007	0
Bosatlas	Netherlands	décembre 2007	0
Bosatlas	Netherlands	janvier 2008	1
Bosatlas	Netherlands	février 2008	1
Bosatlas	Netherlands	mars 2008	0
Bosatlas	Netherlands	avril 2008	0
Bosatlas	Netherlands	mai 2008	1
Bosatlas	Netherlands	juin 2008	0
Bosatlas	Netherlands	juillet 2008	0
Bosatlas	Netherlands	août 2008	1
Bosatlas	Netherlands	septembre 2008	0
Bosatlas	Netherlands	octobre 2008	0
Bosatlas	Netherlands	novembre 2008	0
Bosatlas	Netherlands	décembre 2008	0

36 lignes sélectionnées

Dans cette requête, le mode utilisé est UPSERT ALL. C'est l'un des trois modes possibles, les deux autres étant UPDATE et UPSERT. UPSERT est le mode par défaut (nous l'avons donc implicitement utilisé dans tous les exemples précédents). Le mode UPDATE interdit la création de nouvelles lignes, contrairement à l'UPSERT. Enfin, la différence entre UPSERT et UPSERT ALL est que ce dernier crée de nouvelles cellules même si la partie gauche de la règle contient des références sur des variables (dites "références non-positionnelles"). Par exemple, si la partie gauche de la règle contient le mot-clé ANY, le mode UPSERT ne rechercherait les correspondances que dans les cellules existantes. UPSERT ALL crée de nouvelles cellules, une pour chaque correspondance avec une valeur de dimension non-positionnelle.

 La notion de référence non-positionnelle et le fonctionnement exact du UPSERT ALL par rapport à l'UPSERT font l'objet d'un autre billet de Rob Van Wijk, non encore traduit, mais disponible ici en anglais :  [About Oracle: UPSERT ALL](#)

D'autres fonctions et possibilités sont disponibles avec la clause MODEL, qui valent vraiment la peine d'être étudiées, mais je vais me contenter de les citer rapidement. Tout d'abord, les fonctions PRESENTV et PRESENTNNV, le prédicat IS PRESENT et les mots-clés KEEP NAV et IGNORE NAV. En bref, ils vous permettent de distinguer entre cellules manquantes et cellules existantes contenant un NULL. Ensuite, il y a la fonction PREVIOUS, pour faire référence à la valeur de l'itération précédente dans une clause UNTIL. Enfin, les clauses AUTOMATIC ORDER et SEQUENTIAL ORDER, qui permettent à Oracle de décider lui-même dans quel ordre les règles doivent être évaluées. Toutes ces clauses et fonctions sont assez clairement expliquées dans la documentation.

Dans la dernière partie, je tenterai d'expliquer les utilisations pratiques de la clause MODEL. Je présenterai quelques solutions à des problèmes auparavant insolubles, ainsi que des solutions plus efficaces à des problèmes courants (les habitués des *OTN Forums* verront sans doute à quoi je fais allusion). Cette troisième partie est en cours de publication (en néerlandais) dans la revue *OGh Visie*. Elle sera prochainement disponible (en anglais) sur mon blog.

Source Script SQL contenant toutes les requêtes utilisées