Modelleren in SQL

De wondere wereld tussen GROUP BY en ORDER BY

In 1995 leerde ik werken met Oracle databases. De versie van toen was versie 7.2. In deze versie was de SQL redelijk beperkt tot hetgeen in de ANSI-standaard was voorgeschreven. Een SQL statement bevat een SELECT, FROM en WHERE clause. Met GROUP BY worden groeperingen in de recordset aangebracht, een ORDER BY sorteert tenslotte de data in de aangegeven volgorde.

Afgelopen jaren is er door Oracle aan iedere nieuwe release van de database nieuwe functionaliteit toegevoegd, onder andere aan de SQL-taal. Deze nieuwe functionaliteit is niet altijd even laagdrempelig en ook in de manuals wordt de functionaliteit erg cryptisch beschreven. Bekend voorbeeld zijn de analytische functies (zie Optimize nr. 3, 2006) die in Oracle 9i zijn geïntroduceerd. Alhoewel deze functies zeer krachtig zijn en de mogelijkheid bieden complexe vraagstukken op elegante wijze op te lossen worden ze door Oracle-ontwikkelaars in de praktijk weinig toegepast.
Hetzelfde geldt voor de Model-clause die in release van de Oracle database 10g is geïntroduceerd. In de Manual "Datawarehousing Guide" wordt er één hoofdstuk (toevallig het hoofdstuk na de analytische functies) aan gewijd. Na het lezen (en herlezen) van het hoofdstuk en na het uitvoeren van enkele experimenten blijkt dat het een bijzonder krachtige toevoeging te zijn aan de SQL-taal. In dit artikel wordt de MODEL clause verder uitgediept en wordt afgesloten met een aantal voorbeelden uit de praktijk.

Dimensies, Meetwaarden en Regels

Kort samengevat biedt de Model-clause de mogelijkheid de uitkomsten van een SQL statement na te bewerken. Deze nabewerking vindt plaats nadat het SQL statement de inhoud van de recordset is vastgesteld – dit is nadat de GROUP BY de data gegroepeerd heeft en er de groeps- en analytische funties heeft uitgevoerd. De logische plaats van de MODEL-clause is dus tussen het GROUP BY en voor de ORDER BY clause. De volgende nabewerkingen zijn mogelijk dankzij de MODEL-clause:

1. Toevoegen van nieuwe records aan de recordset
2. Toevoegen van nieuwe kolommen aan de recordset als expressie gebaseerd op andere kolommen of rijen.
3. Manipuleren van kolomwaarden in de recordset.

Een SQL-model is een multidimensionale representatie van de uitkomsten van een query. Dit model wordt opgebouwd in de MODEL-clause, die de uitkomsten van een query omzet in een multi-dimensionele structuur. Deze structuur wordt opgespannen door één of meerdere (afgeleide) kolommen die fungeren als dimensies. Op de snijpunten van waarden in de verschillende dimensies worden meetwaarden gedefinieerd. Meetwaarden ontstaan door expressies op een kolom of meerdere kolommen toe te passen. Het is zelfs mogelijk om nieuwe meetwaarden te definiëren die geen relatie hebben met de oorspronkelijke query. In de multidimensionale structuur geldt dat ieder element wordt geadresseerd door een combinatie van waarden uit de dimensies, de zogenaamde snijpunten. Op ieder snijpunt worden één of meer meetwaarden gedefinieerd. De waarde NULL is geldig als dimensie en als meetwaarde. Met behulp van regels is het mogelijk om wijzigingen door te voeren in de multidimensionale structuur. De volgende datamanipulaties zijn mogelijk:

1. Toevoegen van een nieuwe waarde aan de bestaande verzameling van waarden in de dimensie,
2. Manipuleren van een meetwaarde op ieder mogelijk snijpunt van de dimensiewaarden,
3. Toevoegen van meetwaarden op ieder mogelijk snijpunt van de dimensiewaarden.

Nadat de evaluatie van de regels wordt de gemanipuleerde multidimensionele structuur terugvertaald naar een recordset waarop selecties, voorwaarden en sorteringen worden toegepast. De mogelijkheid bestaat de dataset op te delen in partities. Een partitie bevat één of meerdere dimensies met een statische verzameling van waarden en hebben verder geen afhankelijkheden met andere dimensies. Voor iedere combinatie van

deze waarden wordt een multidimensionale structuur opgebouwd. Het volgende voorbeeld laat zien hoe de uitkomst van een query wordt omgezet in een SQL-model[1]:

```
select product_name, periode, sales
from (
select product_name, sum(unit_price*quantity) sales, to_
char(trunc(order_date, 'MM'),'YYYY-MM') periode
from product_information pi
,    order_items oi
,    orders o
where oi.order_id = o.order_id
and   oi.product_id = pi.product_id
group by product_name , trunc(order_date, 'MM')
)
model
partition by (periode)
dimension by (product_name)
measures (sales sales)
rules (upsert sales[ANY] = presentv(sales[cv()],sales[cv()],0)
)
```

Dit statement deelt de uitkomsten op in drie delen

1. Een partitie met de dimensie PERIODE.
   Alle records in de recordset worden opgedeeld per periode.
2. Per partitie een dimensie PRODUCT_NAME.
   Per partitie wordt een ééndimensionale structuur opgebouwd met daarin de waarden die in de dimensie PRODUCT_NAME voorkomen in de partitie.
3. Een meetwaarde SALES die gebaseerd is op de kolom SALES in de query. Sales wordt gemeten op ieder snijpunt van PRODUCT_NAME (dimensie) en PERIODE (partitie).

In de rules-sectie van het model wordt een eenvoudige regel gedefinieerd. In de query wordt gebruik gemaakt van een inner view waarin de recordset wordt opgebouwd. De gegevens worden gegroepeerd per periode en product zodat voor alle combinaties van deze twee kolommen de sales wordt berekend. Het model wordt opgebouwd door de uitkomst van de query opnieuw op te halen en daarna aan te bieden aan de model-clause. Het is ook mogelijk het model rechtstreeks te baseren op de query.

```
select product_name,  periode, sales
from product_information pi
,    order_items oi
,    orders o
where oi.order_id = o.order_id
and   oi.product_id = pi.product_id
group by product_name , trunc(order_date, 'MM')
model
partition by (to_char(trunc(order_date, 'MM'),'YYYY-MM') periode)
dimension by (product_name)
measures (sum(unit_price*quantity) sales)
rules (upsert sales[ANY] = presentv(sales[cv()],sales[cv()],0)
)
```

Beide query's hetzelfde resultaat. In de tweede query zijn de kolomexpressies opgenomen in de modeldefinitie waar in het eerste voorbeeld alle expressies in de inner query zijn verwerkt. In het model wordt sales als measure gedefinieerd als totaal van de prijs maal aantal. Deze sommatie blijft nodig vanwege de groepering over periode en product. Hetzelfde geldt voor periode die als expressie in de partitiedefinitie wordt bepaald. Alle in de modelclause gedefinieerde attributen worden kunnen in de select clause worden opgehaald. Met de MODEL-clause wordt het dus mogelijk nieuwe kolommen toe te voegen!
Vanwege het array-achtige aspect van een model geldt dat ieder snijpunt is gedefinieerd als een unieke combinatie van waarden uit alle benoemde dimensies. Als er in de uitkomsten van de query meerdere records voorkomen zodat de combinatie van dimensiewaarden niet meer uniek is, is er een aggregatie nodig om deze weer uniek te maken. Alhoewel het niet vereist is, kan als vuistregel gesteld worden dat de dimensies ook altijd als GROUP BY moeten worden benoemd.

Regels

Nadat het model met behulp van partities, dimensies en meetwaarden is opgebouwd worden regels gedefinieerd waarmee de elementen van het model worden bijgewerkt. De regels worden in de rules-sectie van de model-clause gedefinieerd.
Een regel bestaat uit een celverwijzing en een toewijzing in de vorm van een expressie. De celverwijzing selecteert één of meerdere cellen in het array die voor bewerking in aanmerking komen. De uitkomst van de expressie wordt toegekend aan de cel. Een expressie kan op zijn beurt ook bewerkingen uitvoeren op één of meerdere celverwijzingen. Meerdere regels worden met behulp van een komma-teken gescheiden.
Een celverwijzing kan een verwijzing betreffen naar één enkele cel, maar ook een verwijzing naar meerdere cellen tegelijkertijd. In dit laatste geval wordt gesproken over multi-cel verwijzing. Multi-cel verwijzingen kunnen zowel in het selectie deel als in het expressie deel van de regel worden toegepast. Met een multi-cel verwijzing in de selectie worden in één regel meerdere cellen bewerkt. Een multi-cel verwijzing in de expressie maakt het mogelijk een uitkomst toe te kennen aan een cel die afhankelijk is van de inhoud van meerdere cellen.
Een celverwijzing is de verwijzing naar een meetwaarde op een snijpunt van dimensiewaarden. Hierbij geldt dat de volgorde waarin de dimensiewaarden genoemd worden bepaald wordt door de volgorde waarin de dimensies in de DIMENSION BY sectie zijn benoemd. Dimensies die in de PARTITION BY sectie zijn genoemd worden niet gebruikt in de snijpunten, deze zijn constant voor de hele partitie:

1. *De voorbeeld query's zijn gebaseerd op de tabellen in het OE-schema dat standaard in een Oracle 10g database wordt meegeïnstalleerd:*

<Meetwaarde>[<dimensiewaarde 1>, .., <dimensiewaarde n>]

Er wordt – net als in PL/SQL – gebruik gemaakt van associatieve indexeringen. Een snijpunt van dimensies wordt dus geïdentificeerd door waarden uit de verschillende dimensies. De celverwijzing:

```
sales['LAPTOP', 2006]
```

verwijst dus naar de waarde van de meetwaarde sales op het snijpunt van de dimensiewaarden LAPTOP en JAAR. Bij celverwijzingen wordt onderscheid gemaakt tussen positionele en symbolische verwijzingen. Beide verwijzingen vertonen verschillend gedrag en hebben dus invloed op de wijze waarop de regels evalueren.

Positionele verwijzing

Positionele verwijzing houdt in dat er in de celverwijzing constanten worden gebruikt om een snijpunt aan te duiden. Er wordt gerefereerd naar een gefixeerde positie in de array. In de onderstaande regel wordt positioneel verwezen naar het snijpunt (LAPTOP,2007):

```
sales['LAPTOP',2007] = 0
```

De regel zal de meetwaarde op het snijpunt de waarde van de expressie 0 toekennen. Indien er geen meetwaarde bekend was op dit snijpunt zal het aan het array worden toegevoegd.

Symbolische verwijzing

Een symbolische verwijzing filtert dimensiewaarden op basis van een booleaanse conditie. De verwijzing in de dimensie is dus een afvraging die de waarde TRUE of FALSE teruggeeft voor alle dimensiewaarden. Het is dus mogelijk dat er meerdere dimensiewaarden aan de conditie voldoen.

De verwijzing `sales[product_name='LAPTOP', year=2007] = 0` vindt dezelfde cel als in het voorbeeld met de positionele verwijzing. De volgende verwijzingen:

```
sales[product_name like '%LAPTOP%', year >= 2007]
= 0
```

zal meerdere cellen vinden.

Met de symbolische verwijzing:

dimensie IS NOT NULL or dimensie IS NULL

wordt gerefereerd naar alle voorkomende dimensiewaarden. Hiervoor mag ook de wildcard verwijzing ANY worden gebruikt. De regel

```
sales[ANY, year >= 2007] = 0
```

stelt de sales voor alle producten vanaf het jaar 2007 op 0. Ook in de expressies mag gebruik worden gemaakt van multicel verwijzingen. Dit moet wel in combinatie met een groepsfunctie gebeuren zodat de expressie uiteindelijk één waarde oplevert:

```
Sales['LAPTOP',2007] = AVG(sales)['LAPTOP' , year
BETWEEN 2004 and 2006]
```

Deze stelt de sales voor het product LAPTOP voor het jaar 2007 op het gemiddelde van de sales over de jaren 2004 tot en met 2006. De aggregatiefunctie kan ook in combinatie met analytische functies worden gebruikt. In de volgende query:

```
select product_name,periode,sales,sales_ytd
from product_information pi
,    order_items oi
,    orders o
where oi.order_id = o.order_id
and   oi.product_id = pi.product_id
group by product_name , trunc(order_date, 'MM') model dimension by
(product_name,to_char(trunc(order_date, 'MM'),'YYYY-MM')
periode)
measures (sum(unit_price*quantity) sales, 0 sales_ytd)
rules
( upsert sales[ANY,ANY] = presentv(sales[cv(),cv()],sales[cv(),cv()],0)
, upsert  sales_ytd[ANY, ANY]
  = sum(sales) over (partition by product_name,substr(periode,1,4)
order by periode rows unbounded preceding)
)
order by 1,2
```

wordt de behalve een meetwaarde sales voor een product en voor een periode ook de meetwaarde sales_ytd berekend met de cumulatieve sales in het jaar voor het product.

Regelevaluatie

IGNORE.NAV en KEEP.NAV

Het is toegestaan in de regels te verwijzen naar snijpunten waarop geen meetwaarde bekend is. Deze verwijzing levert de waarde NULL op. Dit standaardgedrag wordt veranderd door bij de definitie van de meetwaarde de optie IGNORE NAV op te geven:

```
measures (sales sales) IGNORE NAV
```

De tegenhanger van IGNORE NAV is de optie KEEP NAV. Afhankelijk van het type van de data levert een verwijzing naar een niet-bestaande cel een standaardwaarde op:

- 0 voor numerieke meetwaarden

- '' voor alfanumerieke meetwaarden
- '01-JAN-2001' voor datums
- NULL voor alle overige datatypen.

Verwijzingen naar sales op niet bestaande snijpunten leveren de waarde 0 op.

De regel:

```
sales[ANY,2007] = (sales[cv(),2005] +
sales[cv(),2006] ) / 2
```

levert altijd een waarde op. Zonder IGNORE NAV levert de expressie de waarde NULL op indien er voor het product geen sales bekend is in het jaar 2005 òf 2006.

UPDATE, UPSERT en UPSERT ALL

Het gedrag dat de regels op de verzameling van elementen in het model wordt beïnvloed door aan de regel een UPDATE, UPSERT of UPSERT ALL eigenschap toe te kennen. Met deze eigenschap wordt aangegeven of de waarde van een cel gewijzigd mag worden of er nieuwe cellen ontstaan op basis van de celverwijzingen in het linker deel van de regel. Helaas laat de documentatie van Oracle het precieze verschil tussen deze opties enigszins in het midden en vergt het in de praktijk nogal wat probeerwerk om het juiste gedrag uit te zoeken.
Globaal kan worden gesteld dat het gedrag van opties als volgt wordt gekenmerkt:

- UPDATE
  Deze optie werkt uitsluitend met positionele verwijzingen. Als de meetwaarde op het snijpunt van de dimensiewaarden een waarde bevat zal het worden gewijzigd. Als er op het snijpunt geen meetwaarde bestaat zal deze aan de array worden toegevoegd met als waarde de uitkomst van de expressie. Symbolische verwijzingen hebben geen effect op de array.
- UPSERT
  In deze modus worden er alleen cellen toegevoegd aan de array indien er sprake is van positionele verwijzingen. Wel is het mogelijk om via symbolische verwijzingen wijzingen door te voeren op uitsluitend de cellen die in de array voorkomen.
- UPSERT ALL
  Met UPSERT ALL ontstaat UPSERT gedrag voor regels met symbolische verwijzingen in het linkerdeel. Er wordt een carthesisch product opgebouwd van alle waarden in de dimensies die voldoen aan de symbolische verwijzingen in het linkerdeel van de regel. Cellen die bestaan worden gewijzigd met de uitkomst van de expressie, cellen die nog niet bestaan zullen worden toegevoegd aan de array en krijgen het resultaat van de expressie toegewezen.

Het gedrag kan globaal voor alle regels worden of per regel individueel verschillend worden gezet. Zonder expliciete opgave werken alle regels in UPSERT modus.

Volgorde in regel evaluatie

De volgorde waarin de waarden binnen een regel worden is niet deterministisch, vergelijkbaar met de volgorde waarin de volgorde van de records in een select statement wordt getoond. De volgordelijkheid wordt afgedwongen door een order by clause:

```
sales[ANY, year between 2004 and 2006] order by
year =
```

Indien er meerdere regels zijn gedefinieerd, zal Oracle zelf op zoek gaan naar de afhankelijkheden tussen deze regels. De volgorde waarin de regels worden uitgevoerd hangt af van deze analyse. Dit gedrag wordt overruled door de volgende toevoeging aan de RULES sectie:

```
RULES SEQUENTIAL ORDER
```

Waardoor de regels in de opgegeven volgorde worden uitgevoerd. Standaard werkt Oracle in de met de volgende optie:

```
RULES AUTOMATIC ORDER
```

Iteraties

De mogelijkheid bestaat om de regels meerdere malen uit te voeren met behulp van de ITERATE optie. Het voorbeeld hieronder berekent de waarde van π uit volgens de formule

$$\pi = 4 \sum_{i=0}^{\infty} (-1)^i \frac{1}{2i+1}$$

Uitgeschreven de volgende reeks:

$$\pi = 4(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\frac{1}{13}-\frac{1}{15}+\frac{1}{17}-\frac{1}{19}+....)$$

Deze formule wordt als volgt uitgewerkt in een SQL-statement:

```
select pi
from (
select 1 n from dual
)
model
dimension by (n) measures( 0 pi)
rules iterate (1000000)
(update pi[1] = pi[1]
 + 4/(1+2*ITERATION_NUMBER) *
   case when mod(ITERATION_NUMBER,2) = 0
      then 1 else -1
   end
)
```

Deze query bevat een aantal interessante elementen. Als eerste wordt de waarde 1 van dual geselecteerd en voorzien met alias n. Deze wordt in de model clause omgezet in een ééndimensionale array en er wordt op deze dimensie één meetwaarde pi gedefinieerd. De array bestaat dus uit exact één cel en dit blijft zo gedurende de evaluatie van de regel. Op deze wijze wordt er een tijdelijke variabele pi gedefinieerd die na afloop van de query de benadering van π bevat.
De regel wordt 1 miljoen maal doorlopen. De systeemvariabele ITERATION_NUMBER bevat de waarde van huidige iteratie en wordt in de regel gebruikt om de waarde van de noemer uit de breuk uit te rekenen en om te bepalen of de breuk moet worden opgeteld bij of moet worden afgetrokken van de tijdelijke variable PI.

Merk op dat tijdens de eerste iteratie ITERATION_NUMBER de waarde 0 heeft. Het bovenstaande voorbeeld itereert 1 miljoen maal. Het is ook mogelijk om een stopconditie in te bouwen. De stopconditie wordt gedefinieerd met een UNTIL optie. De volgende iteratieve regel:

```
rules ITERATE (1000000)
 UNTIL ABS(PREVIOUS(pi[1])-p[1]) < 0.00001
```

zal stoppen zodra er 1 miljoen iteraties zijn geweest òf het absolute verschil tussen opeenvolgende waarden kleiner wordt dan de opgegeven nauwkeurigheid. De functie PREVIOUS bevat de waarde van de cel in de voorgaande iteratie.

Vervolg

In het volgende nummer van Optimize zal het tweede deel van dit artikel gepubliceerd worden. Daarin komen aan de orde: for-loops, regelspecifieke functies, Fibonacci-reeks, Reference Model en enkele praktijkvoorbeelden.

Denny de Jonge (djonge@scamander.com) is Technisch Consultant bij Scamander Solutions B.V. te Nieuwegein waar hij zich bezig houdt met het realiseren van Business Intelligence en Datawarehouse oplossingen met behulp van Oracle-technologie.