Modelleren in SQL (2)

De wondere wereld tussen GROUP BY en ORDER BY

De afgelopen jaren heeft Oracle aan iedere nieuwe release van de database nieuwe functionaliteit toegevoegd, onder andere aan de SQL-taal. Alhoewel veel van 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 10g van de Oracle database is geïntroduceerd. In dit artikel, het laatste uit een serie van 2, wordt de MODEL-clause verder uitgediept.

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 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. Dit artikel behandelt een aantal aspecten die in het vorige artikel (zie Optimize nr. 1, 2007) onbesproken zijn gebleven: for-loops, regelspecifieke functies, Fibonacci-reeks, Reference Model en enkele praktijkvoorbeelden.

For-loops

For loops bieden de mogelijkheid om regels in UPSERT modus te definiëren die in staat zijn om met multi-cel verwijzingen aan de linkerkant van de regel nieuwe cellen toe te voegen aan het array. Omdat de waarden die door een for-loop worden doorlopen al op voorhand bekend zijn is het mogelijk de regel uit te schrijven in meerdere regels, namelijk een nieuwe regel voor iedere iteratiewaarde. Door deze eigenschap geldt dat multi-cel verwijzingen die met een for-constructie worden opgebouwd worden gezien als positionele celverwijzingen en dus in UPSERT modes nieuwe cellen zullen creëren.

Er zijn drie manieren om met behulp van een for-lus te itereren over dimensiewaarden, met behulp van waardelijsten, aftelbare reeksen of subquery's.

- **Waardenlijst**
  Deze vorm wordt toegepast als een dimensiewaarde een opsombare reeks betreft

```
sales[for product_name in ('LAPTOP',
'DESKTOP'),2007] = 0
```

  doet hetzelfde als de volgende twee regels

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

  Het is ook mogelijk om te itereren over combinaties van waarden uit verschillende dimensies. In dit geval wordt de dimensie vervangen door een combinatie van verschillende dimensies:

```
Sales[for (product,year) in (('LAPTOP',2007),
('DESKTOP',2007))] = 0
```

- **Aftelbare reeksen**
  Deze vorm kan gebruikt worden voor numerieke reeksen met een vaste stapgrootte. De dimensieverwijzing ziet er als volgt uit:

```
sales[ANY,for year from 1998 to 2007
increment 1] = 0
```

  Voor oplopende reeksen wordt het keyword decrement gebruikt. Omdat Oracle numerieke bewerkingen toestaat op datumvelden is het ook mogelijk om te itereren over een dimensie met datumvelden.

- **Subquery**
  Deze vorm is vergelijkbaar met de waardenlijst, maar nu worden de waarden niet opgesomd maar geselecteerd uit een tabel in de database.

```
sales[for product in (select product_name from
product_information),for year from 1990 to 2007
increment 1] = 0
```

  Deze regel bouwt een carthesisch product tussen de producten en de jaren 1990 tot en met 2007 en vult iedere cel in dit product met de waarde 0. Ook in deze vorm is het mogelijk de waarden uit verschillende dimensies via een subquery te vullen.

Regelspecifieke functies

Met de cv() functie wordt in een expressie de actuele dimensiewaarde uitgevraagd. De naam van de dimensie zoals gedefinieerd in de DIMENSION BY sectie wordt als parameter aan de functie meegegeven. Deze functie wordt hoofdzakelijk gebruikt in combinatie met een multicel-verwijzing in de dimensie aan de linkerkant van de regel.

Binnen een celverwijzing is het niet noodzakelijk bij de cv() functie de naam van de dimensie op te geven. Impliciet wordt verwezen naar de dimensie in de DIMENSION BY definitie op de positie van de dimensieloze cv() verwijzing in de celverwijzing. De cv() functie mag onderdeel uitmaken van een expressie. De volgende regel:

```
target[ANY, for year from 1997 to 2007] = 1.1 *
sales[cv(), cv()-1]
```

stelt de target voor alle producten voor de jaren 1997 t/m 2007 op tien procent maal de sales uit het voorgaande jaar.

```
Presentv () en presentnnv()
```

De presentv-functie heeft een celverwijzing als parameter en twee expressies als parameters. Deze functie controleert of er voor de meetwaarde op de het snijpunt van de genoemde dimensiewaarden een cel bestaat. Als de cel bestaat wordt de waarde van de eerste expressie teruggegeven, bestaat de waarde niet dan wordt de waarde van de tweede expressie teruggegeven.

De volgende regel:

```
Sales[for product_name in (select product_name from product_
information),year from 1997 to 2007] = presentv(sales[cv(),cv()],
sales[cv(),cv()],0]
```

...bouwt een carthesisch product op tussen de producten en de jaren 1997 tot en met 2007. Voor iedere combinatie in dit product wordt gevraagd of er een meetwaarde bestaat. Is de cel niet aanwezig dan zal er een cel worden toegevoegd aan het array en worden voorzien met de waarde 0.

Merk op dat de functie presentv() iets anders doet dan de functie nvl(). De presentv functie controleert alleen het bestaan van de cel en vraagt niet de waarde uit. Een cel die de waarde NULL heeft is voor de functie presentv() een bestaande cel. De functie presentnnv() combineert het gedrag van presentv() met het gedrag van nvl(). Behalve het bestaan van de cel wordt dus een extra controle op de waarde NULL verricht. Het bestaan van een cel kan ook worden uitgevraagd met de conditie IS PRESENT in combinatie met een CASE-WHEN voorwaarde:

```
Sales[for product_name in (select product_name from product_
information),year from 1997 to 2007] = case when sales[cv(),cv()] IS
PRESENT THEN sales [cv(),cv()] else 0 end
```

Fibonacci-reeks

Het volgende voorbeeld laat een query zien die een recordset met de eerste 200 Fibonacci-getallen opbouwt. Als extra wordt een benadering voor de Gulden Snede bepaald[1].

```
select i, to_char(fib) fib, phi gulden_snede
from (select 1 i from dual)
model
dimension by (i)
measures (1 fib, 1 phi)
rules upsert
( fib[1] = 1
, fib[2] = 1
, fib[for i from 3 to 200 increment 1] =
      fib[cv(i)-1] + fib[cv(i)-2]
, phi[ANY]=fib[cv(i)]/fib[cv(i)-1]
)
```

In de hoofdquery wordt de waarde 1 geselecteerd uit dual voor de initiële dataset. Dit record wordt in de MODEL-clause omgezet in een ééndimensionale array *i* waarbij voor ieder element in de array twee meetwaarden worden gedefinieerd, *fib* voor de fibonacci-getallen en *phi* voor de benadering van de Gulden Snede. In de regels wordt voor iedere waarde van i het bijbehorende Fibonacci-getal berekend. In de eerste twee regels worden de eerste twee Fibonacci-getallen gedefinieerd door

1. *Fibonacci getallen is een getallenreeks met als kenmerk dat een getal in de reeks gelijk is aan zijn twee voorgangers. De eerste twee getallen in de reeks zijn gelijk aan 1. Deze reeks heeft als eigenschap de naarmate de reeks vordert de verhouding tussen twee opeenvolgende getallen in de Fibonacci reeks een constante waarde benadert. Deze constante wordt ook wel de Gulden Snede genoemd. De zoektocht naar de Da Vinci Code in het gelijknamige boek begint met deze reeks.*

deze hard op 1 te zetten. De derde regel berekent voor de elementen 3 tot en met 200 het overeenkomstige Fibonacci-getal uit de reeks door de voorgaande twee getallen op te tellen. Hier wordt gebruik gemaakt van de cv() functie in combinatie met een expressie (cv(i)-1 en cv(i)-2) waarmee de posities van deze twee elementen in het array worden bepaald.
Tenslotte wordt bij ieder Fibonacci-getal de bijbehorende waarde van de Gulden Snede bepaald door het huidige getal te delen door zijn voorganger met behulp van de ANY operator.
Het is dus interessant om te zien dat recursieve verwijzingen zijn toegestaan binnen de regels. Oracle is in staat recursieve afhankelijkheden tussen regels te herkennen en om de regels in de juiste volgorde te evalueren.

Reference Model

Tot zover is in dit artikel gekeken naar het deel van de model-clause dat in staat is om de dataset te manipuleren. Dit deel van de MODEL-clause wordt ook wel het MAIN-model genoemd. Voordat het MAIN-model wordt gedefinieerd kunnen er één of meerdere reference-models worden gedefinieerd.
Het verschil met een main-model is dat er geen regels op worden gedefinieerd zodat er sprake is van een statisch model. De arrays die als reference-model worden gedefinieerd zijn in het MAIN model te benaderen. Een reference-model fungeert dus als een lookup tabel.
Een reference model bestaat net als het main-model uit dimensies en meetwaarden. De dimensie- en meetwaarden worden altijd opgehaald via een subquery. Analoog aan het main-model wordt het resultaat van de query met behulp van DIMENSION BY- en MEASURES-statements omzet in een array. Ieder reference model krijgt een unieke naam binnen het model die als prefix gebruikt wordt in het main-model om celverwijzingen mogelijk te maken. De onderstaande query bevat een voorbeeld met een reference model:

```
select *
from (
select distinct product_name, 2007 year
from   product_information
)
model
reference sales_hist
on (select product_name, to_number(to_char(order_date,'YYYY')) year,
sum(quantity*unit_price) 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, to_number(to_char(order_date,'YYYY'))
   )
dimension by (product_name,year)
measures(sales sales)
ignore nav
main main_model
dimension by (product_name, year)
measures(0 sales, 0 target)
rules upsert all
( target[ANY, for year from 1998 to 2007 increment 1] = 1.1 * sales_
hist.sales[cv(product_name), cv(year)-1]
, sales[ANY, ANY] = sales_hist.sales[cv(product_name), cv(year)]
)
```

In het reference model wordt een lijst sales_hist met saleswaarden opgebouwd voor product- en jaar-combinaties met behulp van een subquery. Met IGNORE NAV wordt aangegeven dat verwijzingen naar niet bestaande elementen de standaardwaarde 0 opleveren. In het main-model wordt de lijst sales_hist tweemaal geraadpleegd. De eerste keer om het target te berekenen door tien procent bij de sales van het voorgaande jaar, de tweede keer worden de sales_hist waarden gekopieerd naar de sales meetwaarden in het main-model.

Praktijkvoorbeelden

1. Crosstabs in Discoverer

De waarden op de assen in een kruistabel worden opgebouwd aan de hand van de unieke dimensiewaarden die de snijpunten vormen in de dataset. In combinatie met het gebruik van pagina-items kan dit leiden tot ongewenst gedrag. Immers, een andere waarde in een pagina-item kan leiden tot een andere verzameling van snijpunten en dus een andere combinatie van unieke as-waarden. Een wijziging van een filterwaarde in een pagina-item resulteert in een heropbouw van de assen waardoor het gedrag van het rapport voor de gebruiker een onrustig verdrag vertoont. Met name in de gevallen waarin er sprake is van een laag aantal dimensiewaarden is dit storend.
Deze situatie kan worden ondervangen met behulp van een MODEL-clause. In het model worden alle pagina-items en assen als dimensie gedefinieerd. In een regel wordt voor iedere dimensie een for-lus opgebouwd die òf via een opsomming òf via een subquery alle mogelijke dimensiewaarden opbouwt. In de expressie wordt met een present-functie afgevraagd of er op het snijpunt van de dimensiewaarden een meetwaarde aanwezig is. Indien deze niet aanwezig is, zal de regel een dummywaarde toevoegen. Hiermee wordt geborgd dat voor iedere filterwaarde de verzameling van as-waarden constant is.

2. Debiteurensaldo op een variabele peildatum

Vanuit de debiteurenadministratie komt de vraag voor een rapport dat op een willekeurige peildatum het saldo van alle debiteuren bepaalt. Veelal worden dergelijke rapporten gebaseerd op een tijdelijke tabel die middels een procedure wordt opgebouwd. Met behulp van een model-clause is het mogelijk een zeer elegante oplossing te maken in één SQL-statement. De volgende query is dusdanig geconstrueerd dat voor iedere factuur voor een periode het saldo op dagniveau kan worden gequeried.

```
select customer_id, invoice_id, datum, saldo
from
(
select distinct customer_id, invoice_id, datum
from  v_debiteuren_mutaties
)
model
reference deb_mut
on (select invoice_id,datum,sum(bedrag_mutatie) bedrag_mutatie
    from   v_debiteuren_mutaties
    group by invoice_id,datum
   )
dimension by (invoice_id, datum)
measures     (bedrag_mutatie bedrag_mutatie) ignore nav
main debiteuren_standen
partition by (customer_id, invoice_id)
dimension by (datum datum)
measures (0 saldo)
rules upsert
(  saldo[to_date('30-12-2005','DD-MM-YYYY')]=0
,  saldo[for datum from to_date('31-12-2005','DD-MM-YYYY') to to_
date('31-12-2006','DD-MM-YYYY') increment 1]
   = saldo[cv()-1] + deb_mut.bedrag_mutatie[cv(invoice_id),cv(datum)]
)
```

De view v_debiteuren_mutaties bevat een opsomming van mutaties op facturen. Debetposten resulteren in positieve mutatiebedragen, creditposten tot negatieve mutatiebedragen. De datum van de transactie is de mutatiedatum en mutaties worden gesommeerd per mutatiedatum. De debiteurenmutaties worden in het reference model deb_mut opgeslagen. Met de IGNORE NAV optie wordt afgedwongen dat celverwijzingen in de lijst op een datum waarvoor er geen mutatie aanwezig alsnog een mutatiebedrag van 0 laten zien voor de factuur. Voor iedere factuur wordt een model opgebouwd, dus er wordt gepartioneerd over customer_id en invoice_id. In dit model speelt dus alleen de dimensie datum ten bate van de peildatum een rol. In de meetwaarde saldo wordt het saldo van de factuur elke dag in de datumdimensie bepaald.
In de regels vindt een iteratie plaats over een zekere periode. Het saldo voor iedere dag wordt berekend door de mutaties op de factuur op deze dag op te tellen bij het saldo van de factuur op de voorgaande dag. Indien er geen mutaties op de betreffende dag zijn geweest, levert de verwijzing naar deb_mut de waarde 0 op. Dit gedrag is afgedwongen door de optie IGNORE NAV.
Omdat het SQL-statement voor iedere datum een saldo terug geeft, kan de peildatum in de vorm van een where-clause worden opgegeven:

```
where datum = trunc(sysdate) – 7
```

Het werd nog interessanter toen bleek dat het debiteurensaldo niet aansloot bij het debiteurensaldo in het grootboek. Dit is voor de boekhouder niet acceptabel. Ook hier bood de model clause uitkomst. Er wordt een nieuw reference-model gedefinieerd, opgebouwd met grootboekmutaties per factuur (journaalposten). Er wordt een meetwaarde op de dimensiedatum toegevoegd voor het grootboeksaldo per datum. De berekening van het grootboeksaldo is analoog aan de berekening van het debiteurensaldo.
Met de query worden snel de facturen gevonden waarvan de journaalposten niet aansluiten met de debiteuren. De boekhouder was erg blij met de query en gebruikt nu een rapport dat op deze query gebaseerd is om de aansluiting tussen Debiteuren en Grootboek te maken.

Conclusie

De MODEL-clause is een krachtige uitbreiding op SQL die de mogelijkheid biedt om complexe vragen op een elegante wijze op te lossen. De MODEL-clause biedt een alternatief voor gevallen waar snel wordt uitgeweken naar procedurele oplossingen zoals het opbouwen van tijdelijke tabellen in een stored procedures of pipelined functies. De mogelijkheden die in de regels worden geboden zijn krachtig en generiek genoeg om vele regels procedurele code samen te vatten in enkele regels. Daarnaast wordt de mogelijkheid geboden om extra rekenkundige bewerkingen uit te voeren die normaliter in Excel zouden worden uitgevoerd. Aan een recordset kunnen nieuwe kolommen worden toegevoegd die gebaseerd zijn op één of meerdere bestaande kolommen. Doordat de dataset in een array-achtige structuur wordt omgezet waarbij ieder element apart adresseerbaar is, ontstaat de mogelijkheid afhankelijkheden tussen de verschillende records te definiëren. Behalve het manipuleren van bestaande records bestaat de mogelijkheid om nieuwe records toe te voegen aan de dataset. Ook deze nieuw aangemaakte records kunnen relaties bevatten met bestaande records waardoor de mogelijkheid ontstaat op recursieve wijze een dataset uit te breiden.
De introductie van modellen leidt tot een andere manier van denken over data en de manier waarop deze worden bevraagd in de database. Vooralsnog wordt de model-clause gezien als een curiositeit in de database. Het zal nog enige tijd duren voordat database-ontwikkelaars de mogelijkheden zullen ontdekken en in de praktijk toepassen. Ik ben benieuwd naar de nieuwe features van Oracle 11.

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.