# Puzzelen met SQL

## *Java en de Database*

***Sommige mensen beschrijven een ontwikkelaar als 'iemand die cafeine om kan zetten in programmacode'. Als ik naar mezelf kijk, dan drink ik behoorlijk wat koffie op een dag - ongeveer 15 kopjes. Mensen met wie ik samenwerk weten dat ook dat ik mijn suikerzakjes op tafel laat liggen om aan het einde van de dag de balans op te maken. En nu weet ik wel dat het niet goed voor je is, maar ik kan het niet laten staan. Het is zelfs zo erg dat ik in het weekend hoofdpijn krijg als ik te laat mijn eerste kopje koffie drink.***

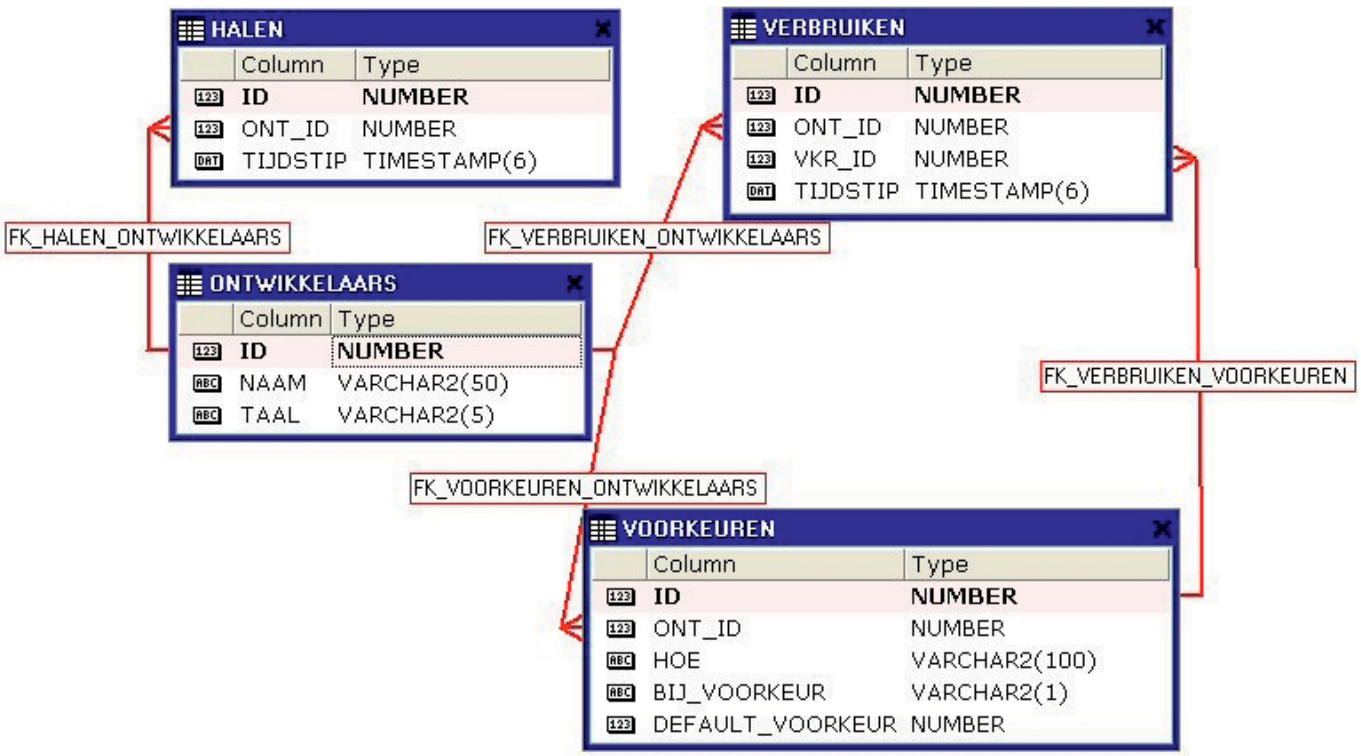
Deze 'Puzzelen met SQL' staat dan ook volledig in het teken van de koffie en dan eigenlijk het gebruik van koffie op de werkvloer.

Allereerst het datamodel.

In het datamodel zitten geen grote verassingen. Allereerst hebben we een lijst nodig met de verschillende ontwikkelaars en hun programmeertaal. Een programmeur kan meerdere koffievoorkeuren hebben, hoewel ik er zelf maar één heb, met suiker. Het verbruik wordt vastgelegd in de Verbruikstabel. Hierin wordt bijgehouden wie, wanneer, welke koffie gebruikt. In de tabel Halen wordt bijgehouden wie er koffie heeft gehaald. Om ervoor te zorgen dat een ontwikkelaar maar één default voorkeur heeft, gaan we allereerst de volgende Business Rule implementeren:

*'Een ontwikkelaar mag maximaal één default koffie voorkeur hebben'.*

Deze Business Rule is eenvoudig te implementeren met een Virtual Column. De Virtual Column is geïntroduceerd in Oracle



11g R1. Wat is een Virtual Column nu eigenlijk? Het is niet een normale column, zoals je die met een CREATE TABLE statement aanmaakt, maar deze is gebaseerd op een expressie.
In het volgende statement maken we de Virtual Column aan op basis van een Case expressie

```
alter table voorkeuren
add default_voorkeur as
   (case
     when bij_voorkeur = 'J'
     then ont_id
    end) unique
);
```

De Case expressie zorgt ervoor dat in de Virtual Column het ID van de ontwikkelaar komt te staan indien deze bij de 'bij_voorkeur' heeft aangegeven dat deze voorkeur geldt als zijn favoriet. Door deze column uniek te maken, zorg je ervoor dat er in de Virtual Column slechts één default is toegestaan. Dit is eenvoudig te testen:

Allereerst moet er wel een Ontwikkelaar aangemaakt worden:

```
SQL> insert into ontwikkelaars values (1, 'Alex', 'PLSQL');

1 row created.
```

Nu de voorkeur

```
SQL> insert into voorkeuren (id, ont_id, hoe, bij_voorkeur) values (1,
1, 'suiker', 'J');

1 row created.
```

Als we nu nog een keer proberen een voorkeur op te voeren als favoriet bij dezelfde ontwikkelaar, dan krijgen we een foutmelding - precies wat we verwacht hadden.

```
SQL> insert into voorkeuren (id, ont_id, hoe, bij_voorkeur) values (1,
1, 'suiker', 'J');
insert into voorkeuren (id, ont_id, hoe, bij_voorkeur) values (1, 1,
'suiker', 'J')
*
ERROR at line 1:
ORA-00001: unique constraint (ALEX.SYS_C0011283) violated
```

Mocht je nu niet op een Oracle 11g zitten, maar op een Oracle 10g dan is de bovenstaande Business Rule ook declaratief te valideren. In dat geval kun je gebruik maken van een Unique Function Based Index. In dat geval is de definitie:

```
SQL> create unique index Een_voorkeur
  2  on voorkeuren
  3  (case when bij_voorkeur = 'J'
  4   then ont_id
  5   end
  6  )
  7  /

Index created.
```

Alleen indien de kolom bij_voorkeur 'J' aangeeft, wordt het ID van de ontwikkelaar opgenomen in de index. Aangezien het een UNIQUE index is, kan er slechts één voorkeur per ontwikkelaar voorkomen.
Het logo van Java is een kopje waar damp vanaf komt. Naar alle waarschijnlijkheid zal het wel een kopje koffie zijn. Maar houdt dit dan ook in dat de Java-ontwikkelaars meer koffie drinken dan de PL/SQL ontwikkelaars? Aangezien we deze data tot onze beschikking hebben, kunnen we dit onderzoeken.
De gegevens die we nodig hebben komen uit de tabel Ontwikkelaars en uit de tabel Verbruiken.

```
SQL> select taal
  2       , count(*)
  3    from verbruiken v
  4    join ontwikkelaars o
  5      on v.ont_id = o.id
  6   group by o.taal
  7   order by 2 desc
  8  /


TAAL     COUNT(*)
----- ----------
PLSQL          37
JAVA           12
```

Wat in bovenstaande query opvalt, is niet zo zeer de aggregaat functie, COUNT, maar de manier van een join leggen tussen de tabellen. Oracle ondersteund sinds versie 9i de ANSI manier van joins. Eén van de grote voordelen van de ANSI join ten opzichtte van de 'ouderwetse' join is de scheiding die aangebracht wordt tussen join- en filtercriteria. Waar bij de 'ouderwetse' manier de join- en filtercriteria door elkaar staan, staan bij de ANSI join de joincriteria in de ON clause en staan de filtercriteria in de WHERE clause van een SELECT statement.
Laten we nu eens gaan bepalen wie er van iedereen het meeste koffie drinkt. Een kwestie van verbruik tellen en dan degene met het hoogste aantal tonen. Eerst maar wat iedereen verbruikt:

```
SQL> select o.naam
  2       , o.taal
  3       , count(*) cnt
  4    from ontwikkelaars o
  5    join verbruiken v
  6      on o.id = v.ont_id
  7   group by o.naam
  8           , o.taal
  9   order by cnt desc
 10  /


NAAM                 TAAL          CNT
-------------------- ----- ----------
Alex                 PLSQL          19
Patrick              PLSQL          12
Theo                 PLSQL           6
Peter                JAVA            6
Louis                JAVA            6
```

Zo te zien zijn degene die PL/SQL programeren de grootverbruikers van koffie. Maar deze query laat teveel data zien, we willen alleen maar degene zien met het meeste aantal kopjes koffie op zijn naam.
We zouden natuurlijk dit tussenresultaat kunnen filteren met ROWNUM. Maar let op, laat je je nu niet verleiden om dit te doen:

```
SQL> select o.naam
  2       , o.taal
  3       , count(*) cnt
  4    from ontwikkelaars o
  5    join verbruiken v
  6      on o.id = v.ont_id
  7   where rownum = 1
  8   group by o.naam
  9           , o.taal
 10   order by cnt desc
 11  /

NAAM                 TAAL          CNT
-------------------- ----- ----------
Alex                 PLSQL          1
```

Toeval wil dat wel de juiste naam eruit komt. De restrictie ROWNUM = 1 wordt als eerste na de JOIN uitgevoerd. Een willekeurige rij wordt opgehaald, deze wordt vervolgens geaggregeerd en gesorteerd. Dit is te zien in het Explain Plan.

```
SQL> select *
  2    from table (dbms_xplan.display_cursor())
  3  /

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
---------------
SQL_ID  469kzkqfdhb3f, child number 0
-------------------------------------
select o.naam       , o.taal       , count(*) cnt    from ontwikkelaars o
 join verbruiken v      on o.id = v.ont_id  where rownum = 1  group by
o.naam           , o.taal  order by cnt desc

Plan hash value: 729700905

--------------------------------------------------------------------------------
---------------
| Id  | Operation              | Name           | Rows  | Bytes | Cost
(%CPU)| Time     |
--------------------------------------------------------------------------------
---------------
|   0 | SELECT STATEMENT       |                |       |       |
8 (100)|          |
|   1 |  SORT ORDER BY         |                |     1 |    57 |
8  (25)| 00:00:01 |
|   2 |   HASH GROUP BY        |                |     1 |    57 |
8  (25)| 00:00:01 |
|*  3 |    COUNT STOPKEY       |                |       |       |
|          |
|*  4 |     HASH JOIN          |                |    49 |  2793 |
7  (15)| 00:00:01 |
|   5 |      TABLE ACCESS FULL| ONTWIKKELAARS |     5 |   220 |     3
(0)| 00:00:01 |
|   6 |      TABLE ACCESS FULL| VERBRUIKEN     |    49 |   637 |     3
(0)| 00:00:01 |
--------------------------------------------------------------------------------
---------------

Predicate Information (identified by operation id):
---------------------------------------------------


   3 - filter(ROWNUM=1)
   4 - access("V"."ONT_ID"="O"."ID")
```

Om dit explain plan te laten zien hebben we hier gebruik gemaakt van DBMS_XPLAN.DISPLAY_CURSOR(). Deze table function laat het explain plan zien van het laatst uitgevoerde statement. Deze functionaliteit is nieuw in Oracle 10g. Mocht je deze functionaliteit willen gebruiken en je krijgt het volgende als output:

```
SQL> select *
  2    from table (dbms_xplan.display_cursor())
  3  /

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
------------
SQL_ID  5t10uu7v11s5t, child number 0


BEGIN DBMS_OUTPUT.ENABLE(NULL); END;


NOTE: cannot fetch plan for SQL_ID: 5t10uu7v11s5t, CHILD_NUMBER: 0
      Please verify value of SQL_ID and CHILD_NUMBER;

It could also be that the plan is no longer in cursor cache (check
v$sql_plan)
```

Dan heb je je SERVEROUTPUT aan staan. Dat geeft de melding ook wel aan, maar het is eenvoudig om daar overheen te lezen. Even terug naar de originele vraag voordat ik te ver afdrijf. Hoe kunnen we dan wel op een correcte manier bepalen wie de meeste koffie verbruikt? Wel nog steeds gebruikmakend van ROWNUM = 1, maar nu via een inline view of zoals onderstaand met behulp van Subquery Factoring - de WITH clause:

```
SQL> with tussenstand as
  2  (select o.naam
  3        , o.taal
  4        , count(*) cnt
  5    from ontwikkelaars o
  6    join verbruiken v
  7      on o.id = v.ont_id
  8   group by o.naam
  9           , o.taal
 10   order by cnt desc
 11  )
 12  select *
 13    from tussenstand
 14   where rownum <= 1
 15  /

NAAM                 TAAL          CNT
-------------------- ----- ----------
Alex                 PLSQL         19
```

Het wordt weer eens tijd voor een bakkie, maar wie is er nu aan de beurt om te gaan halen? Om dit te bepalen gaan we van dezelfde 'truuk' gebruik maken als hierboven, met ROWNUM = 1. Omdat we nu eigenlijk een willekeurig iemand willen aanwijzen om koffie te halen, gaan we in de inline view de sortering laten uitvoeren door het DBMS_RANDOM package.

```
SQL> select naam
  2    from (select o.naam
  3            from ontwikkelaars o
  4           order by dbms_random.value
  5         )
  6   where rownum = 1
  7  /


NAAM
--------------------
Theo
```

In plaats van de Subquery Factoring, maken we hier gebruik van een gewone inline view. Over koffie halen gesproken wie haalt er eigenlijk het meest of juist het minst vaak koffie?
Om deze vraag te kunnen beantwoorden willen we eigenlijk een sorteren op de COUNT die voorvloeit uit de join tussen Ontwikkelaars en Halen. Eigenlijk willen we aggregeren op de ene kolom en een andere kolom laten tonen. Juist voor situaties zoals deze zijn de FIRST/LAST functies in het leven geroepen.

```
SQL> select min (naam) keep (dense_rank first order by count(*)) minst
  2       , max (naam) keep (dense_rank last order by count(*)) vaakst
  3    from ontwikkelaars o
  4    join halen h
  5      on o.id = h.ont_id
  6   group by o.naam
  7  /


MINST                VAAKST
-------------------- --------------------
Peter                Alex
```

In dit resultaat laten we diegene zien die het minst vaak koffie haalt. De First/Last functies zijn aggregaten die ervoor zorgen dat respectievelijk de eerste of de laatste rij uit een set rijen getoond worden afhankelijk van de respectievelijke positie in de set voor de aggregatie. Als de volledige set voor de aggregatie gesorteerd zou zijn op basis van de COUNT(*), dan zou je je voor kunnen stellen dat degene met het hoogste aantal bovenaan zou staan (first) en degene met de minste aantal onderaan (last). Nu zijn we niet zo zeer geinteresseerd in dat aantal, maar juist in de naam. En omdat het een aggregatie functie betreft, kunnen we een andere kolom in de aggregaat functie zetten, zoals de naam.
Zoals je kunt zien in de resultaatset, haalt Peter wel erg weinig koffie. Wordt het niet weer eens tijd voor een bakkie? Die van mij met één klontje suiker, alsjeblieft.... en neem zelf ook wat.

## Referenties

Link naar Oracle documentatie voor First/Last functies: http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96520/analysis.htm#25806

**Patrick Barel** is consultant bij AMIS Services. Hij schrijft op het blog van AMIS (http:/technology.amis.nl/blog) en op zijn eigen blog (http:/blog.bar-solutions.com).

**Alex Nuijten** is Oracle-consultant bij AMIS Services. Hij schrijft op het blog van AMIS (http:/technology.amis.nl/blog) en op zijn eigen blog (http:/nuijten.blogspot.com).

N I E U W S

## Oracle gaat Suns MySQL niet afstoten

Oracle-baas Larry Ellison heeft verklaard dat zijn bedrijf de open source databank MySQL niet zal afstoten als de fusie met Sun Microsystems definitief rond is. In de Verenigde Staten is de fusie al goedgekeurd, maar de Europese Commissie heeft meer tijd gevraagd om de deal nog eens te onderzoeken.

De mededingingsautoriteit van de Europese Commissie heeft een diepgaand onderzoek ingelast dat de vraag moet beantwoorden of de combinatie van Oracle en MySQL in één bedrijf de concurrentie op de markt verstoort. Het onderzoek van de EU moet 19 januari zijn afgerond.
Ellison vindt verder dat Europa de fusie snel goed moet keuren,.Sun verliest volgens de Oracle-topman 100 miljoen dollar per maand en hij vreest dat er meer ontslagen vallen bij Sun als goedkeuring lang uitblijft.
Juristen verklaren dat Oracle er rekening mee moet houden dat de EU het bedrijf concessies gaat opleggen, waaronder mogelijk het afstoten van MySQL. Ellison denkt echter dat het niet zo ver zal komen en hij verklaart dat Oracle dan ook niet van plan is om Suns MySQL-divisie af te stoten.