Puzzelen met SQL

Het volkomen getal

'Pap, wist jij dat zes een volkomen getal is?'
Waar die kinderen tegenwoordig al niet mee thuis komen. Toen ik klein was, en trouwens nog steeds, zei ik 'u' tegen mijn ouders. Maar daar ging het even niet om. Een volkomen getal? Ik had er nog nooit van gehoord. De kinderen van tegenwoordig, inmiddels op de basisschool, hebben allang niet meer de illussie dat je als ouder alles weet.
'Een volkomen getal?', nee dat wist ik niet.

De lerares van de basisschool heeft een voorliefde voor rekenen en dat weet ze goed over te dragen op de kinderen, dat is inmiddels wel duidelijk. Eerst maar eens de definitie van een volkomen getal. Een volkomen, ook wel 'perfect getal' genoemd, is een getal waarvan de som der delers gelijk is aan het getal zelf. Alleen de delers, niet het getal zelf.
Het laagste volkomen getal is zes, zoals dochterlief al zei. De delers van zes zijn: een, twee en drie. De som van een, twee en drie is zes. Zes is dus een volkomen getal. En als er een volkomen getal is, dan weet je dat er meer zijn. Probeer zelf de volgende maar eens uit te rekenen.
Omdat ik daar geen zin in heb en omdat ik een Oracle database voor mijn neus heb staan, gaan we de kracht SQL inzetten om een paar volkomen getallen uit te laten rekenen. Maar ja, hoe pak je zoiets aan?

Het idee

We willen beginnen met een lijst van getallen, samen met bijbehorende delers. Als we dit eenmaal hebben, dan kunnen we de delers gaan sommeren en vergelijken met het bijbehorende getal. Hiervoor zijn we op zoek naar een lijst met drie kolommen. De eerste kolom het 'te testen getal', de tweede kolom met de mogelijke delers en de laatste kolom met het restant van de daadwerkelijke deling erin. Voor de zes zijn we dus op zoek naar zo'n soort resultaat:

```
6    1    0
6    2    0
6    3    0
6    4    2
6    5    1
```

Voor de eerste rij geldt: zes is het 'te testen' getal, de tweede kolom is een deler, de derde kolom is het restant van de deling: zes delen door 1 levert als restant nul. De rijen waar in de restant kolom iets anders dan een nul laat zien kunnen we niet gebruiken. Wat overblijft zijn dan de eerste drie rijen. Van deze drie rijen gaan we de delers sommeren en vergelijken met het 'te testen' getal. Dit is het idee achter de te schrijven query.

De uitwerking

Laten we eerst maar eens beginnen met een lijst met getallen.

```
SQL> select rownum rn
  2    from dual
  3  connect by level <= 10
  4  ;
        RN
----------
         1    2    3    4    5    6    7    8    9   10
```

Inmiddels een veelvuldig gebruikte truc om een lijst met getallen te genereren. Door misbruik te maken van CONNECT BY LEVEL <= 10 kun je tien rijen genereren. Volgens de documentatie zou je echter bij een hierarchische query, wat CONNECT BY tenslotte is, ook een START WITH moeten hebben. Deze eis wordt echter niet afgedwongen, het levert geen syntax fout op.
Laten we het heel netjes doen en gebruik maken van Subquery Factoring om de parameter te definieren. Subquery Factoring is niets anders dan gebruik te maken van de WITH clause. Het doet een beetje procedureel aan binnen een SQL statement.

```
SQL> with params as
  2    (select 10 max_aantal
  3       from dual
  4    ), lijst as
  5    (select rownum getal
  6       from params
  7    connect by level <= params.max_aantal
  8    )
  9  select *
 10    from lijst
 11  ;
GETAL    1    2    3    4    5    6    7    8    9   10
```

De eerste kolom – een lijst met mogelijk volmaakte getallen – hebben we vast te pakken, nu de volgende kolom – de mogelijke delers.
Voor een gegeven getal (het 'te testen' getal) willen we een kolom met de mogelijke delers hebben. Deze delers hebben als voorwaarde dat ze kleiner moeten zijn dan het 'te testen' getal. De lijst met getallen die we reeds hebben kunnen we joinen met zichzelf.

```
SQL> with params as
  2    (select 10 max_aantal
  3       from dual
  4    ), lijst as
  5    (select rownum getal
  6       from params
  7     connect by level <= params.max_aantal
  8    )
  9   ,getallen as
 10   (select l1.getal g1
 11         , l2.getal g2
 12      from lijst l1
 13     cross join
 14           lijst l2
 15     where l1.getal > l2.getal
 16   )
 17   select *
 18     from getallen
 19  ;
         G1         G2
---------- ----------
          2          1
          3          1
          3          2
          4          1
          4          2
          4          3
          5          1
          5          2
          5          3
          5          4
          6          1
          6          2
          6          3
          6          4
          6          5
...
         10          1
         10          2
         10          3
         10          4
         10          5
         10          6
         10          7
         10          8
         10          9
```

Eigenlijk is het raar wat in bovenstaande query staat. Op regel 13 staat een CROSS JOIN. Met een CROSS JOIN word een carthetisch product aangeduid. Normaliter is er voor de 'nieuwe' join syntax, inmiddels ook al weer terug te voeren naar Oracle 9i - zo nieuw is deze dus ook weer niet, een USING of ON clause nodig. Bij een CROSS JOIN is dit niet nodig. Omdat we niet echt een carthetisch product willen leggen we een beperkende voorwaarde op met een WHERE clause. Het fraaie van de ANSI join syntax is dat de join criteria en de filter criteria mooi gescheiden worden. Nu kun je de discussie aangaan of hetgeen we hier doen Join- of Filter criteria zijn. Wellicht was het beter om deze join op deze manier te schrijven:

```
    from lijst l1
         join
         lijst l2
      on (l1.getal > l2.getal)
```

Hierbij laten we deze discussie rusten.
De derde kolom die we nodig hebben is een lijst met restant na deling. Nu zijn er twee mogelijkheden om deze te bepalen De eerste is door gebruik te maken van de MOD functie – waarschijnlijk had je daaraan al gedacht. De tweede manier is door gebruik te maken van de REMAINDER functie. Er zit echter een subtiel verschil tussen beide functies. Uit de documentatie van REMAINDER:
The MOD function is similar to REMAINDER except that it uses FLOOR in its formula, whereas REMAINDER uses ROUND.
Voor onze toepassing maakt het echter niet uit of we MOD of REMAINDER gebruiken. We blijven bij de oude, vertrouwde MOD functie.

```
SQL> with params as
  2    (select 10 max_aantal
  3       from dual
  4    ), lijst as
  5    (select rownum getal
  6       from params
  7    connect by level <= params.max_aantal
  8    )
  9   , getallen as
 10   (select l1.getal g1
 11         , l2.getal g2
 12      from lijst l1 cross join lijst l2
 13     where l1.getal > l2.getal
 14   )
 15   , getallen_met_delers as
 16   (select g1
 17         , g2
 18         , mod (g1, g2) m
 19      from getallen
 20   )
 21  select *
 22    from getallen_met_delers
 23  ;
         G1          G2          M
---------- ---------- ----------
          2          1          0
          3          1          0
          3          2          1
          4          1          0
          4          2          0
          4          3          1
          5          1          0
          5          2          1
          5          3          2
          5          4          1
          6          1          0
          6          2          0
          6          3          0
          6          4          2
          6          5          1
...
```

```
        10          1          0
        10          2          0
        10          3          1
        10          4          2
        10          5          0
        10          6          4
        10          7          3
        10          8          2
        10          9          1
```

De lijst is compleet. We hebben nu de 'te testen' getallen, de mogelijke delers en het restant na deling. De volgende stap is verder filteren van de gegenereerde data. Eigenlijk willen we alleen die getallen sommeren uit de 'G2' kolom waarbij de 'M' (rest na deling) gelijk is aan nul.

```
SQL> with params as
  2    (select 10 max_aantal
  3       from dual
  4    ), lijst as
  5    (select rownum getal
  6       from params
  7    connect by level <= params.max_aantal
  8    )
  9   , getallen as
 10   (select l1.getal g1
 11         , l2.getal g2
 12      from lijst l1
 13     cross join
 14           lijst l2
 15     where l1.getal > l2.getal
 16   )
 17   , getallen_met_delers as
 18   (select g1
 19         , g2
 20         , mod (g1, g2) m
 21      from getallen
 22   )
 23   , getallen_met_som_delers as
 24   ( select g1
 25          , sum(g2) som_der_delers
 26       from getallen_met_delers
 27      where m = 0
 28      group by g1
 29   )
 30  select g1 volkomen_getal
 31    from getallen_met_som_delers
 32   where g1 = som_der_delers
 33  ;
VOLKOMEN_GETAL 6
```

Het sommeren van de delers vind plaats op regel 25 in bovenstaande code. Het filteren van de 'M' kolom vind plaats op regel 27. En zie daar het resultaat: zes, het enige getal onder de tien wat een volkomen getal is. Als we de 'Max_aantal' parameter in regel twee naar 50 dan hebben we twee volkomen getallen: 6 en 28

Met een beetje creativiteit is het mogelijk om ook de som van de delers in kaart te brengen. In onderstaande query is het gebruik van LISTAGG, een nieuwe functie in Oracle 11g Release 2 te zien om de som te kunnen tonen. Een volledige beschrijving van de LISTAGG functie zou teveel van het goede zijn voor nu.

```
SQL> with params as
  2    (select 500 max_aantal
  3       from dual
  4    ), lijst as
  5   (select rownum getal
  6     from params
  7    connect by level <= params.max_aantal
  8   ), getallen as
  9   (
 10   select l1.getal g1
 11         , l2.getal g2
 12     from lijst l1
 13    cross join
 14          lijst l2
 15    where l1.getal > l2.getal
 16   )
 17  select listagg (g2, '+') within group (order by g2)
 18         ||' = '
 19         || to_char (sum (g2)) volkomen_getal
 20    from getallen
 21   where mod (g1, g2) = 0
 22   group by g1
 23  having (g1 = sum (g2))
 24  ;
VOLKOMEN_GETAL
----------------------------------------------------------
1+2+3 = 6
1+2+4+7+14 = 28
1+2+4+8+16+31+62+124+248 = 496
```

Het eerst volgende volkomen getal is 496, daar kun je achter komen door als parameter 500 op te geven. Maak de parameter echter niet te groot, want dan kan het heel lang duren (meer dan 24 uur bij 500000) voordat je resultaat krijgt. Ook loopt het geheugen gebruik aardig op. Misschien kan je met een dedicated server op zoek gaan naar een heel groot volkomen getal, maar dan moet je verder niets met die server willen doen, voldoende geheugen hebben etc.

Wellicht is dit het juiste moment om een oplossiing van dit 'probleem' te laten berekenen door een PL/SQL procedure waarbij je efficiënter met het geheugen kunt omgaan.

Links

http://nl.wikipedia.org/wiki/Perfect_getal

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).