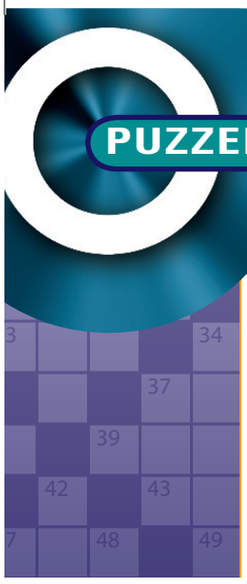

Puzzelen met SQL

Lezerspost

Feedback is altijd goed om te krijgen. Soms is het leuk en soms minder leuk. Deze aflevering staat in het teken van reacties die we kregen naar aanleiding van het voorgaande artikel over het bepalen van een volmaakt getal.

De oplettendheid van Rob van den Berg bracht een onjuistheid aan het licht die in de tekst staat. In het vorige artikel staat namelijk: "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." Dit is niet juist. Een CONNECT BY mag wel zonder START WITH gebruikt worden. Het statement waar het om ging is de volgende:

```
SQL> select rownum rn
  2    from dual
  3  connect by level <= 10
  4  ;
```

In dit statement wordt echter wel gebruik gemaakt van een functionaliteit die volgens de documentatie niet mogelijk zou moeten zijn. Niet zozeer de CONNECT BY zonder START WITH, maar het gebruik van de CONNECT BY zonder PRIOR. Volgens de documentatie is het verplicht om een PRIOR te gebruiken in een hierarchische query. En deze ontbreekt in het statement. Zo zie je maar, altijd gevaarlijk om een claim in een artikel te schrijven zonder uitdrukkelijk te controleren of het wel klopt. We zijn uiteraard blij met zulke oplettende, kritische lezers. We horen graag van jullie.

Harry Dragstra mailde ons dat het uiteraard ook mogelijk is om volmaakte getallen te bepalen met PL/SQL. Waarschijnlijk is het berekenen van volmaakte getallen beter met PL/SQL op te lossen, dan met SQL. Voor de berekening heb je namelijk geen data uit de database nodig. Het is een puur rekenkundige methode. Ook merkte hij op hoe bijzonder het eigenlijk is dat ze in de vijftiende eeuw al dit soort getallen konden berekenen, en dat is natuurlijk ook zo. Dan hebben wij het maar makkelijk met zo'n krachtige computer.
Zijn mailtje leidde er dan ook toe om met behulp van PL/SQL volmaakte getallen te berekenen en dan het liefst zo optimaal mogelijk. De procedure die Harry ons stuurde is de volgende:

```
CREATE OR REPLACE procedure get_perfect_numbers
  ( P_min_number simple_integer := 2
  , P_max_number simple_integer := 10000 )
  as
 l_max_divider    simple_integer := 0;
 l_sum_dividers   simple_integer := 0;
 l_all_dividers   varchar2(32000);
 l_min_number     simple_integer := P_min_number;
 l_max_number     simple_integer := P_max_number;
 l_sep             varchar2(1);
begin
 for n in l_min_number .. (l_max_number+1)
 loop
    if (n-1) = l_sum_dividers then
      dbms_output.put_line( 'perfect_number: '||(n-1)||'> all_dividers:
'||l_all_dividers);
    end if;
   l_max_divider  := n-1;
   l_sum_dividers := 0;
   l_all_dividers := null;
   l_sep            := null;
   for d in 1 .. l_max_divider
   loop
     if mod(n,d) = 0 then
        l_sum_dividers := l_sum_dividers + d;
        l_all_dividers := l_all_dividers||l_sep||d;
        l_sep       := '+';
     end if;
   end loop;
 end loop;
end;
```

In de code en de parameters wordt gebruik gemaakt van het datatype SIMPLE_INTEGER. De SIMPLE_INTEGER is een nieuw datatype die geintroduceerd is in Oracle 11g. De SIMPLE_INTEGER is een variant van de BINARY_INTEGER die efficienter werkt dan een normale INTEGER of zelfs een PLS_INTEGER. Deze efficientie wordt gehaald uit het achterwege laten van extra checks die voor interne verwerking van andere INTEGER types wel gedaan worden. De checks die normaliter worden uitgevoerd zijn controles op NULL en Overflow. Dit houd dus ook in dat SIMPLE_INTEGER niet NULL mag zijn, vandaar dat de variable meteen een default waarde moet hebben. Indien je een variabele hebt gedeclareerd zonder een default waarde te geven dan krijg je een compilatie foutmelding:

```
SQL> declare
  2     var simple_integer;
  3  begin
  4     null;
  5  end;
  6  /
   var simple_integer;
       *
ERROR at line 2:
ORA-06550: line 2, column 8:
PLS-00218: a variable declared NOT NULL must have an initialization
assignment
```

Maar, er is altijd een maar, de efficientie die een SIMPLE_INTEGER je biedt is alleen van toepassing als je gebruik maakt van Native Compilatie. Gebruik met de reguliere compilatie, de zogenaamde Interpreted Compilatie, heeft minder invloed op de performance. Een andere beperking van het INTEGER datatype waar je van op de hoogte moet zijn is, zijn de minimale en maximale waarden: -2147483648..2147483647.
Om erachter te komen op welke manier de PL/SQL code op de database wordt gecompileerd, kun je de V$PARAMETER view uitvragen:

```
SQL> select value
  2    from v$parameter
  3   where name = 'plsql_code_type'
  4  /
VALUE
---------------------------------
INTERPRETED
```

Zoals je in het bovenstaande resultaat kunt zien, wordt de code op mijn database niet Native gecompileerd, maar Interpreted. Het performance voordeel van de SIMPLE_INTEGER heeft op mijn database niet het gewenste effect.
Als je de code wat nader bestudeert, zit er een inefficientie in, namelijk het bepalen van de som van de delers. Het getal delen door twee levert de hoogst mogelijke deler op die een geheel getal oplevert. De deler kan dus, per definitie, nooit groter worden dan de helft van het getal. Als je dit opneemt in de code scheelt dat dus ongeveer de helft van het aantal controles dat wordt uitgevoerd. Dit gegeven in de nieuwe versie van de procedure verwerken levert een doorlooptijdswinst op. Direct lossen we ook de beperking op die het gebruik van de SIMPLE_INTEGER met zich meebrengt.

```
CREATE OR REPLACE procedure get_perfect_numbers_2
  ( P_min_number number := 2
  , P_max_number number := 10000
  , P_debug         simple_integer := 0 )
  as
 l_sum_dividers   number:= 0;
 l_all_dividers   varchar2(32000);
 l_sep            varchar2(1);
 n                number;
 n_min            number:= P_min_number;
 n_max            number:= P_max_number;
 d                number:= 0;
 d_max            number;
begin
-- begin and end with een even number... odd numbers cannot ever be
'perfect_numbers'
 if mod(n_min,2)>0 then n_min := n_min-1; end if;
 if mod(n_max,2)>0 then n_max := n_max+1; end if;
 n := n_min - 2;
 <<number_loop>>
 loop
-- iterate with incrementals of 2
   n := n+2;
   if (((n-2) = l_sum_dividers) and (n!=2))then
         dbms_output.put_line( 'perfect_number: '||(n-2)||'> all_divi
ders: '||l_all_dividers);
   end if;
-- exit before this number ( n_max+2 ) gets checked; it isn't in check
range.
   exit number_loop when (n=n_max+2);
   l_sum_dividers := 0;
   l_all_dividers := null;
   l_sep          := null;
   d              := 0;
   <<divider_loop>>
   loop
     d := d+1;
         if mod(n,d) = 0 then
       l_sum_dividers := l_sum_dividers + d;
       l_all_dividers := l_all_dividers||l_sep||d;
       l_sep      := '+';
         end if;
-- debug
   if P_debug = 1 then
         dbms_output.put_line( 'number: '||n||'> divider: '||d||'>
sum_dividers: '||l_sum_dividers||'> all_dividers: '||l_all_dividers);
   end if;
-- exit when the divider is half the number, or the number is less than
sum_dividers
         exit divider_loop when ((d=n/2) or (n<l_sum_dividers));
   end loop;
 end loop;
end;
/
SQL>
SQL> set timing on
SQL> begin
  2     get_perfect_numbers_2;
  3  end;
  4  /
perfect_number: 6> all_dividers: 1+2+3
perfect_number: 28> all_dividers: 1+2+4+7+14
perfect_number: 496> all_dividers: 1+2+4+8+16+31+62+124+248
perfect_number: 8128> all_dividers: 1+2+4+8+16+32+64+127+254+508+1016+
2032+4064
PL/SQL procedure successfully completed.
Elapsed: 00:00:14.65
```

Zoals te zien is worden er twee loops gestart. Door het gebruik van de labels (<<number_loop>> en <<divider_loop>>) kan bij het exit commando aangegeven worden welke loop moet worden verlaten. Sinds Oracle 11g is het ook mogelijk om met het continue commando aan te geven dat er verder moet worden gegaan met de volgende iteratie van een loop. De andere loop moet dan wel aangegeven worden.
Onderstaande code levert hetzelfde resultaat op, maar op een andere manier:

```
begin
  <<outer_loop>>
  for outer_indx in 1..10 loop
    <<inner_loop>>
    for inner_indx in 1..10 loop
dbms_output.put_line(inner_indx ||'*'||outer_indx||'='||outer_indx *
inner_indx);
```

```
      exit inner_loop when inner_indx >= 5;
    end loop inner_loop;
  end loop outer_loop;
end;
begin
  <<outer_loop>>
  for outer_indx in 1..10 loop
    <<inner_loop>>
    for inner_indx in 1..10 loop
      dbms_output.put_line(inner_indx ||'*'||outer_indx||'='||outer_
indx *
inner_indx);
      continue outer_loop when inner_indx >= 5;
    end loop inner_loop;
  end loop outer_loop;
end;
```

Met continue is het ook mogelijk om bepaalde iteraties uit een loop over te slaan, in plaats van de hele loop af te breken (in onderstaande code wordt de vijfde iteratie uit de inner loop overgeslagen):

```
begin
  <<outer_loop>>
  for outer_indx in 1..10 loop
    <<inner_loop>>
    for inner_indx in 1..10 loop
      continue inner_loop when inner_indx = 5;
dms_output.put_line(inner_indx ||'*'||outer_indx||'='||outer_indx *
inner_indx);
    end loop inner_loop;
  end loop outer_loop;
end;
```

Een ander voordeel van het gebruik van loop labels is dat het de leesbaarheid van de code vergroot. Het wordt veel inzichtelijker hoe de loop structuur in elkaar steekt. Het begin en het einde van de loop worden met dezelfde naam aangeduid. Er vindt echter geen validatie plaats of de namen overeenkomen. Kijkend naar het resultaat van de functie komen we erachter dat (van hoog naar laag bekeken) de delers van een getal altijd de helft zijn van de voorgaande. Als er een getal tussen de huidige deler en de voorgaande (grotere) deler zit, wat ook een deler van het getal is, dan is het dus geen volkomen getal en kunnen we uit de loop stappen.

```
CREATE OR REPLACE PROCEDURE get_perfect_numbers_3 (p_min_number NUMBER
:= 2
,p_max_number NUMBER := 10000
,p_debug      simple_integer := 0) AS
  l_sum_dividers NUMBER := 0;
  l_all_dividers VARCHAR2(32000);
  l_sep          VARCHAR2(1);
  n              NUMBER;
  n_min          NUMBER := p_min_number;
  n_max          NUMBER := p_max_number;
  d              NUMBER := 0;
  l_last_divider NUMBER := 0;
  d_max          NUMBER;
BEGIN
  -- some known perfect numbers: 6,28,496,8128,33550336,8589869056,
137438691328
  -- begin and end with an even number... odd numbers cannot ever be
'perfect_numbers'
  IF MOD(n_min, 2) > 0 THEN
    n_min := n_min - 1;
  END IF;
  IF MOD(n_max, 2) > 0 THEN
    n_max := n_max + 1;
  END IF;
  n := n_min - 2;
  <<number_loop>>
  LOOP
    -- divide the number by 2 until we are at 1
    -- iterate with incrementals of 2
    n := n + 2;
    IF (((n - 2) = l_sum_dividers) AND (n != 2)) THEN
      dbms_output.put_line('perfect_number: ' || (n - 2) || '> all_
dividers: ' || l_all_dividers);
    END IF;
    -- exit before this number ( n_max+2 ) gets checked; it isn't in
check range.
    EXIT number_loop WHEN(n = n_max + 2);
    l_sum_dividers := 0;
    l_all_dividers := NULL;
    l_sep          := NULL;
    d              := n;
    <<divider_loop>>
    LOOP
      d := floor(d / 2);
      IF MOD(n, d) = 0 THEN
        l_sum_dividers := l_sum_dividers + d;
        l_all_dividers := l_all_dividers || l_sep || d;
        l_sep          := '+';
        FOR indx IN (d + 1) .. ((l_last_divider - 1)) LOOP
          EXIT divider_loop WHEN MOD(n, indx) = 0;
        END LOOP;
        l_last_divider := d;
      END IF;
      EXIT divider_loop WHEN d <= 1;
      IF MOD(d, 2) <> 0 THEN
        d := d + 1;
      END IF;
    END LOOP;
  END LOOP;
END;

SQL> set timing on
SQL>
SQL> begin
  2     get_perfect_numbers_3;
  3  end;
  4  /
perfect_number: 6> all_dividers: 3+2+1
perfect_number: 28> all_dividers: 14+7+4+2+1
perfect_number: 496> all_dividers: 248+124+62+31+16+8+4+2+1
perfect_number: 8128> all_dividers: 4064+2032+1016+508+254+127+64+32+
16+8+4+2+1
PL/SQL procedure successfully completed.
Elapsed: 00:00:04.64
```

Zo zie je maar…. het loont wel om goed op de hoogte te zijn van de data die uitgevraagd moet worden.
De reactie van Anton Scheffer hierop was de volgende: "zoals ieder welopgevoed kind weet zijn alle even perfecte getallen gebaseerd op een Mersenne priemgetal."
Juist. Een Mersenne priemgetal? Het bestaan van volmaakte getallen was onbekend, totdat dochterlief hiermee thuis kwam. In de beschrijving van volmaakte getallen op Wikipedia wordt inderdaad gerefereerd aan Mersenne priemgetallen. Een uitleg van de Mersenne priemgetallen gaat voorbij aan dit artikel. De code die Anton heeft geschreven om volmaakte getallen te bepalen is dan ook onwaarschijnlijk snel.

```
declare
  t_s timestamp;
  PROCEDURE is_perfect(p_num IN NUMBER) IS
    t_max NUMBER := trunc(p_num / 2) + 1;
    t_cur NUMBER := 2;
    t_sum NUMBER := 1;
    t_div VARCHAR2(1000) := p_num || ' = 1';
  BEGIN
    LOOP
      IF MOD(p_num, t_cur) = 0 THEN
        t_sum := t_sum + t_cur + p_num / t_cur;
        t_div := t_div || '+' || t_cur || '+' || (p_num / t_cur);
      END IF;
      t_cur := t_cur + 1;
      EXIT WHEN t_cur > t_max;
      EXIT WHEN t_cur > p_num / t_cur;
      EXIT WHEN t_sum > p_num;
    END LOOP;
    IF t_sum = p_num THEN
      dbms_output.put_line(t_div);
    END IF;
  END;
--
  FUNCTION is_prime(p_num IN NUMBER) RETURN BOOLEAN IS
    t_max NUMBER := trunc(p_num / 2) + 1;
    t_cur NUMBER := 2;
  BEGIN
    IF p_num IN (1, 2) THEN
      RETURN TRUE;
    END IF;
    LOOP
      IF MOD(p_num, t_cur) = 0 THEN
        RETURN FALSE;
      END IF;
      t_cur := t_cur + 1;
      EXIT WHEN t_cur > t_max;
    END LOOP;
    RETURN TRUE;
  END;
BEGIN
  t_s := systimestamp;
  FOR i IN 1 .. 17 LOOP
    IF is_prime(i) THEN
      is_perfect(2 ** (i - 1) * (2 ** i - 1));
    END IF;
  END LOOP;
  dbms_output.put_line(systimestamp - t_s);
END;
/
```

De resultaten na het uitvoeren van dit script zijn dan als volgt:

```
1 = 1
6 = 1+2+3
28 = 1+2+14+4+7
496 = 1+2+248+4+124+8+62+16+31
8128 = 1+2+4064+4+2032+8+1016+16+508+32+254+64+127
33550336 = 1+2+16775168+4+8387584+8+4193792+16+2096896+32+1048448+64+
524224+128+262112+256+131056+51
048+16382+4096+8191
8589869056 = 1+2+4294934528+4+2147467264+8+1073733632+16+536866816+32+
268433408+64+134216704+128+671
12+16777088+1024+8388544+2048+4194272+4096+2097136+8192+1048568+16384+
524284+32768+262142+65536+1310
+000000000 00:00:00.282000000
PL/SQL procedure successfully completed.
```

De volmaakte getallen worden aan het begin van de regel getoond, gevolgd door een opsomming van de delers die tot het volmaakte getal leiden. De totale duur van het berekenen van de volmaakte getallen volgens de laatste methode staat op de laatste regel van de output. En om even te benadrukken over wat voor doorlooptijd we het hebben:

```
+000000000 00:00:00.282000000
```

Het resultaat wordt in ongeveer een halve seconde getoond. Ongelofelijk.

Maar valt je iets op?

Het eerste resultaat hoort er niet bij. Het getal één is geen volmaakt getal. Het te testen volmaakte getal mag namelijk niet worden aangemerkt als deler. Door de iteratie te laten beginnen bij twee in plaats van één is dit probleem verholpen. Een leuk item in deze code is het feit dat als een deler een geheel getal oplevert, dat dan het resultaat van deze deling zelf ook een deler is. Hierdoor kan het bepalen van de volkomenheid van het getal nog verder verkort (en wellicht versneld) worden. Door de limieten van de iteratie aan te passen kan gezocht worden naar andere en grotere volkomen getallen. Tot dertig gaat dit nog steeds heel snel, maar daarboven loopt de tijd echter enorm op. Bij een waarde van eenendertig heeft mijn laptop ongeveer twee en een half uur nodig om resultaat te tonen, terwijl bij een waarde van dertig het iets meer dan twee seconden duurde.
Het is wel interessant om te zien dat er in beide oplossingen gebruik wordt gemaakt van PL/SQL waar de titel van het artikel toch duidelijk is: Puzzelen met SQL.
Het voordeel van PL/SQL is in deze dat gestopt kan worden met het bepalen van of een getal volkomen is zodra een van de regels overtreden wordt, waar we met SQL alle getallen moeten bepalen, met al hun delers en dan pas in een later stadium kunnen controleren of een getal voldoet. Gezien het resultaat (6,28,496...) wordt er dus veel te veel berekend waardoor de tijd (en het geheugen gebruik) enorm oplopen.

Met dank aan Rob, Harry en Anton voor de feedback.

Referenties

Meer informatie over Volmaakte getallen: http://nl.wikipedia.org/wiki/Perfect_getal
Uitleg van het Mersenne priemgetal: http://nl.wikipedia.org/wiki/Mersennepriemgetal

Patrick Barel is consultant bij AMIS Services. Hij is te bereiken via email. patrick.barel@amis.nl.

Alex Nuijten is Oracle-consultant bij AMIS Services. Hij is te bereiken via email. alex.nuijten@amis.nl.