Werken met recursie vereist speciale constructies in SQL

Recursief aanroepen van coding

Toon Loonen

Recursieve relaties in het gegevensmodel komen in de praktijk veel voor en zijn bijzonder krachtig. In het vorige artikel in Database Magazine 4 is dit verder uitgewerkt. In dit artikel worden de modellen vertaald naar coding. Minder bekend in een database-omgeving is het recursief aanroepen van coding, iets dat in talen als C of LISP heel gewoon is. Toch komt deze werkwijze weinig voor. Reden voor Toon Loonen om dit eens nader uit te werken.

Met het gegevensmodel zoals beschreven in het genoemde vorige artikel is het ook gewenst om de structuur van deze gegevens op een scherm of rapport af te beelden, bijvoorbeeld het gehele organogram van een bedrijf of de *bill of material* van een product. Het organogram zou er dan als volgt uit kunnen zien:

```
Code        Naam Afdeling
--------------------------
X           Bedrijf Xxx
  A         Divisie A
    A1      Sectie A1
      A11   Afdeling A11
      A12   Afdeling A12
      A13   Afdeling A13
    A2      Sectie A2
      A21   Afdeling A21
      A22   Afdeling A22
      A23   Afdeling A23
  B         Divisie B
    B1      Sectie B1
      B11   Afdeling B11, enzovoort
Enzovoort
```

Niet alle implementaties van SQL bieden ondersteuning voor recursie. Oracle biedt deze ondersteuning wel. Hierbij kan bovenstaande lijst worden gemaakt met de volgende query:

```
column afd_code format A20;
select lpad(‘ ‘, 2*(level-1)) || afdeling_code
afd_code, naam
from afdeling
start with afdeling_code_hoger is null
connect by prior afdeling_code =
afdeling_code_hoger ;
```

Voor meer informatie over deze constructie en de SQL-standaard voor recursie, zie[1].

Stored procedures

Als het RDBMS recursie niet ondersteunt zal er voor deze lijst wat meer geprogrammeerd moeten worden. Maar ook in meer ingewikkelde situaties volstaat bovenstaande constructie niet meer en is procedurele coding nodig.
Als aan de hand van een formuletabel, zoals bovenstaand, berekeningen uitgevoerd moeten worden, inclusief het aanroepen van subformules, moet er vaak gebruik gemaakt worden van een stored procedure die zichzelf weer aanroept: een recursieve stored procedure.

> Het is niet eenvoudig controle op rondzingen uit te voeren

Er volgt nu eerst een eenvoudig voorbeeld van recursie in een stored procedures (in Oracle). De procedure wordt aangeroepen met als argument een getal, als dit getal groter is dan 0, dan wordt het getal geprint, 1 van het getal afgetrokken en dezelfde procedure wordt weer aangeroepen.

```
create or replace procedure x (a in number)is begin
   dbms_output.enable(1000000);
   if a > 0 then
    dbms_output.put_line(‘par = ‘ || to_char(a) );
      x (a-1);
   else
    dbms_output.put_line(‘IGNITION’ );
   end if;
end;
— Testen met:  exec x(128);
```

Sybase ondersteunt geen recursieve SQL-query, maar wel het recursief aanroepen van stored procedures. Hiermee is het ook mogelijk om bovenstaand rapport van het organogram te maken. Op basis van het gegevensmodel in het vorige artikel kan hiervoor de volgende stored procedure gebruikt worden:

> In principe wordt met recursie bereikt dat het aantal niveaus onbeperkt is

```
create procedure afdeling_report
  @afdeling_code char(4) = NULL,
  @level smallint = 1
as
begin
  if (@afdeling_code is null )  – haal code van
het bedrijf op
  begin
    select @afdeling_code = afdeling_code
    from   afdeling
    where  afdeling_code_hoger is null
    – eventueel verdere controles op invoer
  end

  declare @afdeling_code_c char(4)
  ,       @afdeling_naam_c varchar(40)
  ,       @level2          smallint
  ,       @spaties1        varchar(32)
  ,       @spaties2        varchar(32)

  if @level = 1
  begin
    select   @afdeling_code_c = afdeling_code
    ,        @afdeling_naam_c = afdeling_naam
    from   afdeling
    where  afdeling_code = @afdeling_code
    select @spaties1 = space(32)
    print " %1! %2! %3!", @afdeling_code_c,
                          @spaties1,
@afdeling_naam_c
  end

  select @spaties1 = space(@level * 2)  – maak
string van spaties
  select @spaties2 = space(32 - @level * 2)
  select @level2   = @level + 1

  declare afdlng_curs cursor for
    select afdeling_code, afdeling_naam
    from   afdeling
    where  afdeling_code_hoger = @afdeling_code

  open    afdlng_curs
  fetch   afdlng_curs into   @afdeling_code_c
  ,                          @afdeling_naam_c

  while   @@sqlstatus = 0
  begin
    print "%1! %2! %3! %4!", @spaties1, @afde-
ling_code_c,
                             @spaties2, @afde-
ling_naam_c
    exec afdeling_report
         @afdeling_code = @afdeling_code_c,
         @level = @level2

    fetch   afdlng_curs into   @afdeling_code_c
    ,                          @afdeling_naam_c

  end – while   @@sqlstatus = 0

  close       afdlng_curs
  deallocate  cursor afdlng_curs
end – procedure
```

Deze procedure kan worden uitgevoerd met:

```
exec afdeling_report @afdeling_code = NULL
exec afdeling_report @afdeling_code = "X"
exec afdeling_report @afdeling_code = "B"
```

In bovenstaande coding zien we het statement

```
  exec afdeling_report @afdeling_code =
@afdeling_code_c, ...
```

Hierin roept de procedure zichzelf weer aan. Door het meegeven van het niveau weet de procedure hoeveel spaties er ingesprongen moet worden voordat de code wordt afgedrukt.

Let op het mogelijk "rondzingen" in deze structuur. Als bij het invullen van de gegevens wordt aangegeven dat:
- Afdeling D valt onder C;
- Afdeling C valt onder B;
- Afdeling B valt onder D.

dan zal bovenstaand rapport de volgende structuur tonen:

```
  B               Divisie B
    C             Sectie C
      D           Afdeling D
        B         Divisie B
          C       Sectie C
            D     Afdeling D, enzovoort
```

De torens van Hanoi

De torens van Hanoi vormen waarschijnlijk het bekendste voorbeeld van een "moeilijk probleem" dat met een heel klein recursief programma kan worden opgelost.

De opgave is om de vijf schijven van de tweede naar de derde toren te brengen. Er mag echter maar een schijf per keer worden verplaatst naar een andere toren. Verder mag er nooit een grotere maat schijf op een kleinere liggen. De eerste toren kan als tijdelijke opslag worden gebruikt. Op internet is veel informatie over deze opgave te vinden, tik "torens van Hanoi" in (bijvoorbeeld op www.google.nl) en men vindt prachtige uitwerkingen waarmee ook gespeeld kan worden.

Hieronder staat de coding van een Sybase stored procedure, die de verplaatsingen aangeeft.
De oplossing voor een toren van X (= 5) schijven is in feite simpel:
1. Breng een toren van X-1 (= 4) schijven van twee naar een;
2. Breng de laatste schijf naar drie;
3. Breng de toren van X-1 (= 4) schijven van een naar drie.
Natuurlijk mogen we stap een en drie niet in een keer doen, dat moeten we ook weer in dezelfde drie stappen doen, nu met X-2 schijven enzovoort. Uiteindelijk komen we op een schijf en die kunnen we gewoon van naar de gewenst plek overbrengen.
In een programma:

```
/* ********************************************** **
** Torens van Hanoi
** Syntax: Sybase Transact SQL
** Parameters: aantal levels, van toren, naar toren
** ********************************************** */
create procedure toren_hanoi
  @niveau smallint,
  @van smallint,
  @naar smallint
as
begin
  declare @temp smallint — tijdelijke plaats voor rest
  ,       @niveau2 smallint
  select  @temp = 1
  if (@niveau > 15 or @niveau < 1
  or @van not in (1,2,3)
  or @naar not in (1,2,3))
  begin
    print "fout in parameters"
    return 1
  end  — controle invoer

  if (@niveau = 1)
  begin
    print "schijf van %1! naar %2!", @van, @naar
  end
  else
  begin
    select @niveau2 = @niveau - 1    — telkens het aantal schijven met
1 verlagen
    select @temp = 6 - @van - @naar  — de tijdelijke opslag
    exec toren_hanoi @niveau2, @van, @temp
    print "schijf van %1! naar %2!", @van, @naar
    exec toren_hanoi @niveau2, @temp, @naar
  end  — @niveau = 1
end    — procedure
```

Een test van het programma met vier schijven van twee naar drie geeft het volgende resultaat:

```
exec toren_hanoi 4, 2, 3

schijf van 2 naar 1
schijf van 2 naar 3
schijf van 1 naar 3
schijf van 2 naar 1
schijf van 3 naar 2
schijf van 3 naar 1
schijf van 2 naar 1
schijf van 2 naar 3
schijf van 1 naar 3
schijf van 1 naar 2
schijf van 3 naar 2
schijf van 1 naar 3
schijf van 2 naar 1
schijf van 2 naar 3
schijf van 1 naar 3
```

Bij vier schijven hebben we al 15 verplaatsingen te doen. Het aantal verplaatsingen bij N schijven is (2 tot de macht N) min 1. Bij 12 of 13 niveaus is er voldoende uitvoer van dit programma om dit gehele nummer van Database Magazine te vullen.

Het rapport zal in een oneindige loop raken of op een andere manier in de fout lopen, bijvoorbeeld door gebrek aan geheugen of (bij Sybase) doordat het aantal niveaus waarop stored procedures elkaar mogen aanroepen een maximum (van 16) overschrijdt.

Het is niet eenvoudig om bij het invoeren van een afdeling (of andere gegevens, bijvoorbeeld een bill of material) deze controle op rondzingen uit te voeren. Daarom is een rapport als het bovenstaande bruikbaar om deze controle uit te voeren. Als bij het maken van het rapport de betreffende fout (meer dan 16 niveaus) wordt geconstateerd, dan zal er meestal een dergelijke fout in de gegevens zitten. Meer dan 16 niveaus komt bij dit soort situaties namelijk zelden voor.

> Zoals overal in modellering moeten ook hier betekenisvolle codes worden vermeden

Bij de test van het rapport kan deze situatie een keer ingebracht worden om te zien of het rapport inderdaad op de gewenste manier "vast loopt" en tevens is dit een simpele manier om gegevens tot 16 niveaus af te drukken. Oracle heeft deze beperking van 16 niveaus niet. Hierbij is het dan ook nodig om in de code een controle op het niveau aan te brengen. In bovenstaand programma is dat mogelijk door direct na de start een foutmelding te geven en de verwerking af te breken als de waarde van de tweede parameter (level) een afgesproken maximum (bijvoorbeeld 15) overschrijdt. Voor verdere toelichting op foutafhandeling, zie[2].

Twee werktabellen

Het is mogelijk om hetzelfde rapport te maken zonder het recursief aanroepen van dezelfde stored procedure. Hiervoor definieert men twee werktabellen.

1. In de eerste stap wordt werktabel 1 gevuld met de gegevens van het bedrijf (1 record).
2. In de volgende stap wordt werktabel 2 gevuld met de gegevens van werktabel 1 plus de kinderen van de records in werktabel 1 (de divisies, niveau 2); daarna maakt men werktabel 1 leeg en kopiert werktabel 2 weer naar 1.
3. In de derde stap vult men werktabel 2 met de gegevens van werktabel 1 plus de kinderen van de divisies (niveau 3); daarna maakt men werktabel 1 weer leeg en kopieert werktabel 2 weer naar 1, enzovoort tot er geen nieuwe records meer worden toegevoegd.
4. Daarna kan vanuit de werktabel het rapport worden gemaakt. Ook hier moet worden opgelet dat er geen oneindige loop ontstaat door "rondzingen" zoals hierboven beschreven is. Men bouwt dus altijd een controle in op een maximaal aantal verwachte niveaus.

In principe wordt met recursie bereikt dat het aantal niveaus onbeperkt is. Praktisch gezien zullen er toch vaak ergens beperkingen optreden. Bijvoorbeeld bij het rapport hierboven moet beslist worden hoeveel ruimte er nodig is voor het inspringen op het laagste niveau, om de kolom aan te geven waar de naam van de afdeling wordt afgedrukt. Ook Sybase kent de beperking van maximaal 16 niveaus bij het recursief aanroepen van stored procedures. In de praktijk is dit echter nog nooit een probleem gebleken, meestal is acht of tien niveaus al een acceptabele bovengrens.

In het voorbeeld hiervoor hebben de afdelingen van bedrijf X een code gekregen die afgeleid is van de bovenliggende afdeling plus een volgcijfer. De afdelingen van sectie A1 zijn dus gecodeerd als A11, A12 en A13. Zoals overal in de gegevensmodellering moeten zeker ook hier betekenisvolle codes worden vermeden. Voor een test is deze code prima geschikt, maar men codeert de afdelingen in de praktijk anders. Zeker mag uit deze code niet de code van de hoofdafdeling worden afgeleid door de laatste positie weg te halen. Als bij een reorganisatie een afdeling van plaats in de hiërarchie verandert, zou ook de code van deze afdeling moeten wijzigen met alle gevolgen voor de vele andere plaatsen (niet alleen in computersystemen) waar deze code wordt gebruikt.

Conclusie

Recursieve relaties vormen een abstracte maar bijzondere krachtige constructie bij het opstellen van een logisch of fysiek gegevensmodel. Het valideren, rapporteren of op een andere wijze werken met recursieve structuren vereist speciale constructies in SQL (mogelijk in Oracle) of procedurele coding, waarbij gebruik gemaakt kan worden van het recursief aanroepen van stored procedures, iets wat nog zeer weinig voorkomt.

Toon Loonen (toon.loonen@cgey.nl, toon.loonen@inter.nl.net) is als consultant werkzaam bij Cap Gemini Ernst & Young.

Literatuur

1. Cannan, Recursief SQL. Database Magazine 1999/2.
2. Loonen, Foutafhandeling in SQL coding. Database Magazine 2003/2.