SELECT FROM AUSTIN, TEXAS

Joe Celko over SQL en andere database-zaken

De Set-theorie was een grote vooruitgang in de wiskunde. Voordat Gregor Cantor deze theorie ontwikkelde, werden verzamelingen van getallen behandeld als *sequences*. Een set is een compleet geheel met alle daarbij behorende eigenschappen, die echter niet altijd hoeven te gelden voor de individuele onderdelen.

SQL is een set-georiënteerde taal, en het is vaak mogelijk informatie boven water te krijgen zonder dat iets bekend is over de individuele elementen in de tabellen, weergegeven door die sets.

Onlangs kreeg ik een klus waarbij we te maken hadden met een bestaande SQL-database. We gingen op zoek naar opgeslagen procedures die dezelfde tabellen-subset gebruikten. Met die informatie konden we bepalen of we views gingen gebruiken of dat we de procedures zouden consolideren. De tabel die we bouwden zag er zo uit:

```
CREATE TABLE ProcTabs (procname CHAR(25) NOT NULL,
tablename CHAR(25) NOT NULL,
PRIMARY KEY (procname, tablename));
```

U herinnert zich misschien een vorige column over relationele deling. In dat geval zou u een tabel bouwen met één combinatie van tabellen; vervolgens deelt u de ProcTabs-tabel door deze delingstabel en zo verkrijgt u de namen van de procedures.

Helaas bestaat het schema uit zestig tabellen. De formule voor alle mogelijke combinaties omvat ook faculteiten. Kortom, er is niet genoeg opslagruimte voor die tabel.

Hoe zou een wiskundige bepalen of twee sets gelijk zijn? Als set A en set B even groot zijn en er één-op-één mapping bestaat die gelijk is aan elk van de originele sets, zijn ze hetzelfde. We kunnen een mapping doen door een INNER JOIN tussen de twee tabellen uit te voeren.

```
SELECT P1.procname AS proc1,@l: < P2.procname AS proc2,
COUNT(*) AS tally_of_procs
 FROM ProcTabs AS P1, ProcTabs AS P2
 WHERE P1.procname  P2.procname[1]
  GROUP BY P1.procname, P2.procname HAVING (SELECT
COUNT(*)[2]
          FROM ProcTabs AS P11
         WHERE P11.procname = P1.procname)
        = (SELECT COUNT(*)[3]
             FROM ProcTabs AS P111, ProcTabs AS P222
            WHERE P111.procname = P1.procname
              AND P222.procname = P2.procname
              AND P111.tablename = P222.tablename)
    AND (SELECT COUNT(*)[4]
           FROM ProcTabs AS P22
          WHERE P22.procname = P2.procname)
        = (SELECT COUNT(*)[2]
             FROM ProcTabs AS P111, ProcTabs AS P222
            WHERE P111.procname = P1.procname
              AND P222.procname = P2.procname
              AND P111.tablename = P222.tablename);
```

Het zwakke punt van deze aanpak is dat we niet precies weten welke de tabellen-verzamelingen zijn; we weten alleen hoeveel tabellen zich in de procedures bevinden. We werken op een hoger aggregatieniveau.

U kunt de aggregaatfuncties en de HAVING-clausule toepassen om van met de GROUP BY-clausule geformeerde groepen bepaalde karakteristieken vast te stellen. Een eenvoudige gegroepeerde tabel met drie kolommen ziet er dan bijvoorbeeld zo uit:

```
SELECT col1, col2
 FROM Foobar
 GROUP BY col1, col2 HAVING ..
```

U kunt met deze HAVING-clausules de volgende eigenschappen van de groepen bepalen:

Denken in aggregaten

```
HAVING COUNT (DISTINCT col_x) = COUNT (col_x)[5]
HAVING COUNT(*) = COUNT(col_x);[6]
HAVING MIN(col_x - <const>) = -MAX(col_x - <const>)[7]
HAVING MIN(col_x) * MAX(col_x) < 0[8]
HAVING MIN(col_x) * MAX(col_x) = 0[9]
HAVING MIN(col_x) * MAX(col_x) > 0[10]
HAVING MIN(col_x) = -MAX(col_x)[11]
HAVING MIN(SIGN(col_x)) = MAX(SIGN(col_x))[12]
HAVING MIN(ABS(col_x)) = 0;[13]
HAVING MIN(ABS(col_x)) = MIN(col_x) [14]
HAVING MIN(col_x) = -MAX(col_x)[11]
HAVING MIN(col_x) * MAX(col_x) = 0[15]
HAVING MIN(col_x) < MAX(col_x)[16]
HAVING MIN(col_x) = MAX(col_x)[17]
```

Ik herinner u er nogmaals aan dat als er geen GROUP BY-clausule is, de HAVING-clausule de hele tabel als als één groep zal behandelen, geheel volgens de standaards SQL-89 en -92. Dat betekent dat als u op de gehele tabel een van bovenstaande tests wil toepassen, u een constante in de SELECT-lijst moet gebruiken.

Een voorbeeld maakt dat misschien inzichtelijker. Gegeven is een tabel met een kolom van unieke opeenvolgende getallen die begint bij 1 en oploopt. Als u een nieuwe rij invoegt, moet u een volgnummer gebruiken dat nog niet in de kolom voorkomt - oftewel: vul de gaten. Alleen als er helemaal geen gaten zijn, kunt u het eerstvolgende hoge getal in de volgorde gebruiken.

```
CREATE TABLE Foobar (seq_nbr INTEGER NOT NULL PRIMARY KEY
CHECK (seq > 0),@l:   name CHAR(5) NOT NULL);
INSERT INTO Foobar VALUES (1, 'Tom'), (2, 'Dick'), (4,
  'Harry'), (5, 'Moe');
```

Hoe komt u erachter of er gaten zijn?

```
 EXISTS (SELECT 'gap'[18]
           FROM Foobar
         HAVING COUNT(*) = MAX(seq_nbr))
```

U kunt "SELECT seq_nbr" niet gebruiken, omdat de kolomwaarden niet gelijk zullen zijn binnen de enkele groep gemaakt van de tabel; de subquery zal dus mislukken met een *cardinality violation*. "SELECT *" faalt op dezelfde manier, omdat de asterisk geconverteerd wordt naar een door de SQL-engine uitgezochte kolomnaam. Hier volgt het statement voor een tussenvoeging:

```
INSERT INTO Foobar (seq_nbr, name) VALUES (CASE WHEN
  EXISTS[19]
              (SELECT 'no gaps'
                 FROM Foobar
                HAVING COUNT(*) = MAX(seq_nbr))
         THEN (SELECT MAX(seq_nbr) FROM Foobar) + 1
         ELSE (SELECT MIN(seq_nbr)[20]
                 FROM Foobar
                WHERE (seq_nbr - 1)
                      NOT IN (SELECT seq_nbr
                                FROM Foobar)
               AND seq_nbr > 0) - 1, 'Celko');
```

Als 1 voorkomt in de seq_nbr-kolom, moet de ELSE-clausule deze bijzondere omstandigheid aankunnen, zodat er geen illegale nul verschijnt. Dan volgt het echt lastige gedeelte: wachten tot de volledige geladderde uitkomsten van de subquery zijn berekend voordat een substract-actie mogelijk is; als u "MIN(seq_nbr -1)" of "MIN(seq_nbr) -1" in de SELECT-lijst vermeldt, kan dit het gebruik van indexen in vele SQL-producten onmogelijk maken. ●

Noten

1. Verschillende procedures
2. Grootte van set A
3. Grootte van de mapping
4. Grootte van set B
5. col_x heeft alle onderscheidende waarden@vn:6. De kolom heeft geen NULL's
7. col_x wijkt evenveel af boven als onder de constante waarde
8. MAX is positief en MIN is negatief
9. MIN of MAX afzonderlijk of beide zijn nul
10. col_x is of helemaal positief of helemaal negatief
11. col_x wijkt evenveel af boven als onder nul
12. col_x is helemaal positief, helemaal negatief of overal nul
13. col_x heeft minstens één nul
14. col_x >= 0 (dit kunt u overigens ook met een WHERE-clausule doen)
15. Of MIN of MAX of beide zijn nul
16. col_x heeft meer dan één waarde (kan sneller zijn dan een telling (*) > 1)
17. col_x heeft één waarde of NULL's
18. Hierbij werkt elke constante
19. Er zijn geen gaten
20. Er zijn gaten

Joe Celko (www.celko.com) is onafhankelijk consultant en lid van het ANSI X3H2 Database Standards Committee. Hij is auteur van diverse boeken over SQL. Als SQL-specialist schijft hij behalve voor Database Magazine voor het blad *Intelligent Enterprise* (voorheen DBMS).