SELECT FROM AUSTIN, TEXAS

Joe Celko over SQL en andere database-zaken

Hier is leuk puzzeltje dat door een zekere Ian in een nieuwsgroep werd geplaatst. Hij schreef:

Ik werk in een ziekenhuis en heb een lastige query waar ik hulp bij nodig heb. Als een patiënt een fysiotherapeutische behandeling nodig heeft, bezoekt hij een PT (Physical Therapist) of een PTA (-assistent). Meestal komen patiënten 10 tot 12 keer achter elkaar. Ik ben op zoek naar een query die voor elke patiënt het respectieve aantal behandelingen (Visits) telt. Als een patiënt vier of meer opeenvolgende keren een assistent (PTA) bezoekt, zonder bij de therapeut zelf (PT) geweest te zijn, moet dat gesignaleerd moet worden. Ik denk dat ik een counter moet hebben die reset naar 1 bij elke patiënten-ID, en tevens bij elke verandering van therapeutentype. Als de counter 3 bereikt, moet de rij teruggeplaatst worden in de result set. Hoe kan ik dat het beste aanpakken? Met een cursor?

Query therapie

Cursors zijn bijna nooit een goede oplossing, maar aan de andere kant stelt SQL niet veel voor als het om sequences gaat. We beginnen eerst maar eens een goede DDL te maken. Ik neem hierbij aan dat een patiënt hooguit 1x per dag op therapie komt.

```
CREATE TABLE Visits
(patient_id INTEGER NOT NULL,
 visit_date DATE NOT NULL DEFAULT CURRENT_TIMESTAMP,
 therapist_type CHAR(3) NOT NULL
             CHECK (therapist_type IN ('PT', 'PTA')),
  PRIMARY KEY(visit_date, patient_id));
```

We laden wat testgegevens in deze tabel. Eerlijk gezegd was het makkelijker geweest om een grote hoeveelheid willekeurige dummydata te genereren. Maar een column is maar kort, dus we bedenken een paar representatieve gevallen.

```
INSERT INTO Visits VALUES (1, '2001-01-01', 'PTA');
INSERT INTO Visits VALUES (1, '2001-02-01', 'PTA');
INSERT INTO Visits VALUES (1, '2001-03-01', 'PTA');
INSERT INTO Visits VALUES (1, '2001-04-01', 'PTA');
INSERT INTO Visits VALUES (1, '2001-05-01', 'PT');
INSERT INTO Visits VALUES (2, '2001-01-02', 'PTA');
INSERT INTO Visits VALUES (2, '2001-02-02', 'PTA');
INSERT INTO Visits VALUES (2, '2001-03-02', 'PTA');
INSERT INTO Visits VALUES (2, '2001-04-02', 'PT');
INSERT INTO Visits VALUES (3, '2001-01-03', 'PTA');
INSERT INTO Visits VALUES (3, '2001-02-03', 'PTA');
INSERT INTO Visits VALUES (3, '2001-03-03', 'PTA');
INSERT INTO Visits VALUES (3, '2001-04-03', 'PTA');
INSERT INTO Visits VALUES
  (3, '2001-05-03', 'PTA');
INSERT INTO Visits VALUES
  (4, '2001-01-04', 'PT');
INSERT INTO Visits VALUES
  (4, '2001-02-04', 'PT');
INSERT INTO Visits VALUES (4, '2001-03-04', 'PT');
INSERT INTO Visits VALUES (4, '2001-04-04', 'PT');
INSERT INTO Visits VALUES (4, '2001-05-04', 'PT');
INSERT INTO Visits VALUES (5, '2001-01-01', 'PTA');
INSERT INTO Visits VALUES (5, '2001-02-01', 'PTA');
INSERT INTO Visits VALUES (5, '2001-03-01', 'PTA');
INSERT INTO Visits VALUES (5, '2001-05-01', 'PTA');
INSERT INTO Visits VALUES (5, '2001-04-01', 'PT');
INSERT INTO Visits VALUES (6, '2001-01-02', 'PTA');
INSERT INTO Visits VALUES (6, '2001-02-02', 'PTA');
INSERT INTO Visits VALUES (6, '2001-03-02', 'PT');
INSERT INTO Visits VALUES (6, '2001-04-02', 'PTA');
INSERT INTO Visits VALUES (6, '2001-05-02', 'PTA');
INSERT INTO Visits VALUES (6, '2001-06-02', 'PTA');
```

Steve Kass van de Drew University bedacht een oplossing die gebruik maakt van een tijdelijke tabel en wat complexe logica om de bezoeken tussen begin- en einddatum in ogenschouw te nemen.

```
INSERT INTO Temp (patient_id, first_visit, last_visit,
consecutive_visits)
SELECT V1.patient_id,
      V1.visit_date AS first_visit,
      V2.visit_date AS last_visit,
      (SELECT COUNT (visit_date)
         FROM Visits
        WHERE visit_date
             BETWEEN V1.visit_date AND V2.visit_date
          AND Patient = V1.patient_id
          AND Patient = V2.patient_id)
```

```
  FROM Visits AS A
       INNER JOIN
       Visits AS B
       ON V1.visit_date < V2.visit_date
          AND V1.patient_id = V2.patient_id
          AND 'PA'
              NOT IN
              (SELECT therapist_type
                 FROM Visits
                WHERE visit_date
                     BETWEEN V1.visit_date AND
V2.visit_date
                 AND Patient = V1.patient_id
                 AND Patient = V2.patient_id)
         AND 4
             <= (SELECT COUNT (visit_date)
                  FROM Visits
                 WHERE visit_date
                     BETWEEN V1.visit_date AND
V2.visit_date
                  AND Patient = V1.patient_id
                  AND Patient = V2.patient_id);

SELECT Y.patient_id, Y.first_visit, Y.last_visit
       MAX (consecutive_visits)
  FROM (SELECT X.patient_id,
               MIN(X.first_visit),
               X.last_visit,
               MAX(consecutive_visits)
          FROM Temp AS X
         GROUP BY X.patient_id, X.last_visit) AS Y
 GROUP BY Y.patient_id, Y.first_visit;
```

De tijdelijke werktabel is eigenlijk overbodig, maar zonder kreeg Mr. Kass een interne MS-SQL server error. Dit geeft mij het gevoel dat er sprake is van een simpele query in een veel te complex statement. Mijn eerste poging zag er zo uit.

```
SELECT V2.patient_id
  FROM Visits AS V1, Visits AS V2
 WHERE V1.visit_date < V2.visit_date
   AND V1.patient_id = V2.patient_id
   AND 'PTA' = ALL
       (SELECT V3.therapist_type
          FROM Visits AS V3
         WHERE V3.patient_id = V1.patient_id
           AND V3.visit_date
               BETWEEN V1.visit_date AND V2.visit_date)
 GROUP BY V2.patient_id
HAVING SUM (CASE WHEN V1.therapist_type = 'PTA'
                 THEN 1 ELSE 0 END) >= 4;
```

V1 en V2 zijn de bronnen voor de begin- en einddata van de behandelingen van elke patiënt. Alle bezoeken, de V3-kopie van de tabel, moesten liggen tussen de begin- en einddata en het type PTA-bezoeken is vanaf hier het derde predikaat. De GROUP BY en HAVING clausules werden verondersteld de vier bezoeken te tellen. Ik wilde ook graag het "=ALL" predikaat gebruiken, maar dan had ik net zo goed een NOT EXISTS() predikaat kunnen schrijven. Een andere meedenker, Steve Dassin, kwam er al snel achter dat dit verkeerde uitkomsten gaf bij opeenvolgende PTA´s. Je krijgt namelijk patiënten 1, 3 en 6; en 6 is fout. Mijn eerste gedachte was om ">=4" te veranderen in ">4" om van patiënt 6 af te komen - maar dit werkt nog steeds niet als we patiënt 7 toevoegen.

```
INSERT INTO Visits VALUES (7, '2001-01-01', 'PTA');
INSERT INTO Visits VALUES (7, '2001-02-01', 'PTA');
INSERT INTO Visits VALUES (7, '2001-02-10', 'PTA');
INSERT INTO Visits VALUES (7, '2001-02-11', 'PT');
INSERT INTO Visits VALUES (7, '2001-03-01', 'PTA');
INSERT INTO Visits VALUES (7, '2001-04-01', 'PTA');
INSERT INTO Visits VALUES (7, '2001-04-11', 'PTA');
```

En zoals Steve al aantoonde: deze query leverde mij alleen de som van drievoudige getallen op en geen telling van het werkelijke aantal bezoeken. Ik had geen rekening gehouden met het Cartesische vermenigvuldigingseffect. Ik troostte mezelf met de gedachte dat als ik een fout maak in een nieuwsgroep, het me weer materiaal voor een column zou opleveren. Dit leidde tot mijn volgende poging.

```
SELECT DISTINCT V2.patient_id
  FROM Visits AS V1, Visits AS V2, Visits AS V3
 WHERE V3.visit_date
      BETWEEN V1.visit_date AND V2.visit_date
   AND V1.visit_date < V2.visit_date
   AND V1.patient_id = V2.patient_id
   AND V1.patient_id = V3.patient_id
 GROUP BY V2.patient_id, V1.visit_date, V2.visit_date
HAVING COUNT (V3.visit_date) = 4
   AND SUM (CASE WHEN V3.therapist_type = 'PTA'
                 THEN 1 ELSE 0 END) = 4;
```

Deze query gebruikt hetzelfde basisidee, maar is een enorme re-write. Er zijn twee data die de grenzen bepalen en één datum daartussen om patiëntenbezoeken te tellen. De HAVING clausule telt de behandelingen tussen begin- en einddata en telt de PTA´s daartussen.

```
SELECT V2.patient_id, V1.visit_date, V2.visit_date
```

is een betere oplossing, omdat het tijdelijke informatie oplevert en je laat weten of een patiënt meer dan 4 bezoeken heeft afgelegd. Maar het probleem gold alleen de patiënt-ID, dus we verwijderen de duplicaten met SELECT DISTINCT. Omdat hier geen geladderde subqueries inzitten, denk ik dat het een betere performance geeft dan mijn eerste benadering. Probeer het eens uit. ●

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 schrijft hij behalve voor Database Magazine voor het blad *Intelligent Enterprise* (voorheen DBMS).