SELECT FROM AUSTIN, TEXAS

Joe Celko over SQL en andere database-zaken

In de wereld om ons heen doet zich een tamelijk algemeen probleem voor, dat betrekking heeft op tijdsduur. Vermeden moet worden dat twee gebeurtenissen elkaar overlappen. Het is uiterst moeilijk in hetzelfde stadion op hetzelfde tijdstip twee verschillende wedstrijden te spelen, twee auto's op hetzelfde tijdstip op dezelfde plek te parkeren enzovoort.

We bekijken een tabel die informatie bevat over de registratie (*enrollment*) van verzekeringspolissen. De namen van de kolommen spreken voor zich.

```
CREATE TABLE Enrollments
 (client_id CHAR(9) NOT NULL,
 policy_nbr INTEGER NOT NULL,
 start_date DATE NOT NULL,
 end_date DATE NOT NULL,
 PRIMARY KEY (client_id, start_date),
 CHECK (end_date >= start_date));

INSERT INTO Enrollments VALUES ('111111111', 1, '2000-01-
   01', '2000-04-30');
INSERT INTO Enrollments VALUES ('111111111', 1, '2000-06-
   01', '2001-12-31');
INSERT INTO Enrollments VALUES ('222222222', 1, '2000-01-
   01', '2000-06-30');
INSERT INTO Enrollments VALUES ('222222222', 2, '2000-07-
   01', '2001-12-31');
INSERT INTO Enrollments VALUES ('333333333', 1, '2000-01-
   01', '2000-06-30');
INSERT INTO Enrollments VALUES ('333333333', 1, '2000-07-
   01', '2001-12-31');
INSERT INTO Enrollments VALUES ('444444444', 1, '2000-01-
   01', '2000-06-30');
INSERT INTO Enrollments VALUES ('444444444', 1, '2000-07-
   01', '2000-11-30');
INSERT INTO Enrollments VALUES ('444444444', 1, '2001-03-
   01', '2001-06-30');
INSERT INTO Enrollments VALUES ('444444444', 1, '2001-07-
   01', '2001-12-31');
INSERT INTO Enrollments VALUES ('555555555', 1, '2000-01-
   01', '2000-01-30');
INSERT INTO Enrollments VALUES ('555555555', 1, '2000-02-
   01', '2000-02-28');
INSERT INTO Enrollments VALUES ('555555555', 1, '2000-03-
   01', '2000-03-31');
INSERT INTO Enrollments VALUES ('555555555', 1, '2000-04-
   01', '2000-04-30');
INSERT INTO Enrollments VALUES ('555555555', 1, '2000-05-
   01', '2000-05-30');
```

Gaten dichten

We nemen aan dat de frontend ervoor zorgt dat geen enkele cliënt een polis heeft met overlappende datums; de tijdperioden zijn wel of niet aaneengesloten. Wordt een polis vernieuwd en de datums zijn opeenvolgend, dan worden de twee rijen samengevoegd tot één. Bijvoorbeeld:

```
('333333333', 1, '2000-01-01', '2000-06-30')
('333333333', 1, '2000-07-01', '2001-12-31')
```

wordt:

```
('333333333', 1, '2000-01-01', '2001-12-31')
```

Op de traditionele manier gebeurde dit door het bestand te sorteren op client_id, policy_nbr en start_date, het record te lezen en te vergelijken met het vorige record, ze samen te voegen als ze aaneensluitend zijn enzovoort, tot het gehele bestand is doorlopen. Hetzelfde kan in SQL gedaan worden met cursors, maar dat gaat erg langzaam.

Maar hoe pak je dit aan op een relationele manier? De cursor of traditionele wijze was gebaseerd op het één voor één samenvoegen van perioden. Maar wij denken graag in termen van hele sets en niet in *sequential processing*.

Er zijn een paar algemene patronen om tot een oplossing te komen. We kunnen een bestaande rij updaten en consolideren, maar dan moeten we de originele rijen zien kwijt te raken. We kunnen de geconsolideerde data invoegen en daarna de originele rijen die in de consolidatie begrepen zijn, wissen.

Maar wat is het kenmerk van aaneensluitende tijdperioden? Dat er geen gat tussen zit. Wat betekent dat? Als we de tijdperioden in de geconsolideerde periode bij elkaar optellen, is de lengte van de geconsolideerde periode precies gelijk aan de som van de originele perioden. Dat idee leidt tot een eerste poging om tot een oplossing te komen.

```
SELECT E1.client_id, E1.policy_nbr, E1.start_date,
       E2.end_date
  FROM Enrollments AS E1,
       Enrollments AS E2
 WHERE E1.client_id = E2.client_id
   AND E1.policy_nbr = E2.policy_nbr
   AND E1.start_date < E2.end_date;
```

Deze query levert voor elke polis en cliënt alle mogelijke paren van start- en einddatums op. Maar dit is niet de definitieve oplossing. Het heeft niets te maken met gaten of consolidaties van elke bestaande periode. De vuistregel is: we zoeken naar groepskenmerken, dus gebruiken we een GROUP BY .... HAVING-clausule.

```
SELECT E1.client_id, E1.policy_nbr, E1.start_date,
       MAX(E2.end_date)
  FROM Enrollments AS E1,
       Enrollments AS E2
 WHERE E1.client_id = E2.client_id
   AND E1.policy_nbr = E2.policy_nbr
   AND E1.start_date < E2.end_date
 GROUP BY E1.client_id, E1.policy_nbr, E1.start_date
HAVING COUNT(E2.start_date) > 1
   AND (SELECT COUNT(E3.start_date)
      + SUM (CAST (E3.end_date - E3.start_date) DAY
            AS INTEGER)
        FROM Enrollments AS E3
       WHERE E3.client_id = E1.client_id
         AND E3.policy_nbr = E1.policy_nbr
         AND E3.start_date BETWEEN E1.start_date
                               AND MAX(E2.end_date))
           = SUM (CAST ((MAX(E3.end_date)
                       - E3.start_date) DAY
                  AS INTEGER) + 1;
```

De GROUP BY presenteert ons de laatst mogelijke einddatum. Het eerste predikaat in de HAVING-clausule is gemakkelijk te begrijpen; die vertelt ons dat de consolidatie meer dan één periode bevat. Het tweede predikaat is lastiger. Als er geen gaten voorkomen, is het aantal dagen in de consolidatie gelijk aan de som van het aantal dagen in de geconsolideerde tijdperioden. Maar het is niet mogelijk de juiste getallen te verkrijgen met een simpele telling. Dat is de wet van de ordinale versus de kardinale getallen; er moet één bij opgeteld worden om de kardinaliteit van een set in een rubriek te verkrijgen. Een voorbeeld: de periode 01-01-2001 tot en met 05-01-2001 telt vijf dagen, terwijl

```
CAST ((DATE '2001-01-05' - DATE '2001-01-01') DAYS
AS INTEGER)
```

als uitkomst 4 geeft. Jammer genoeg werkt deze query ook niet. Met datums als:

```
('666666666', 1, '2000-01-01', '2000-01-02')
('666666666', 1, '2000-01-03', '2000-01-04')
('666666666', 1, '2000-01-05', '2000-01-06')
```

is de uitkomst:

```
('666666666', 1, '2000-01-01', '2000-01-04')
('666666666', 1, '2000-01-03', '2000-01-04')
```

Daar hadden we echter al eerder een truc op gevonden, die we nog eens herhalen. Hier komt een oplossing, die gebruik maakt van een afgeleide tabel en een GROUP BY om de vroegste datum te verkrijgen.

```
SELECT client_id, policy_nbr,
       MIN (start_date),
       end_date
  FROM ( {{ above query }} )
       AS Latest_Dates (client_id, policy_nbr,
                        start_date, end_date)
  GROUP BY client_id, policy_nbr, end_date;
```

Kan deze query verbeterd worden? Jazeker, door een drieweg self-join te gebruiken in plaats van de afgeleide tabel.

Om het probleem af te ronden: voeg deze rijen in de *enrollment*-tabel en voer deze opdracht uit.

```
DELETE FROM Enrollment
 WHERE EXISTS
       (SELECT *
          FROM Enrollment AS E1
         WHERE E1.client_id = Enrollment.client_id
           AND E1.policy_nbr = Enrollment.policy_nbr
           AND Enrollment.start_date
               BETWEEN E1.start_date AND E1.end_date
           AND Enrollment.end_date
               BETWEEN E1.start_date AND E1.end_date
           AND (Enrollment.start_date > E1.start_date
                OR Enrollment.end_date < E1.end_date);
```

Dit geeft als uitkomst dat er één rij van een cliënt en een polis in zit, maar niet identiek aan een andere rij. ●

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).