SELECT FROM AUSTIN, TEXAS

Joe Celko over SQL en andere database-zaken

Als we artikelen bestellen dan is de prijs afhankelijk van de hoeveelheid. We bekijken een getrapte prijstabel.
Elke trap geeft een reeks met een onder- en een bovengrens aan, waarbinnen de aankoophoeveelheid moet vallen.

```
CREATE TABLE PriceList
(item_nbr INTEGER NOT NULL,
step INTEGER NOT NULL,
low INTEGER NOT NULL,
high INTEGER NOT NULL,
CHECK (low < high),
price DECIMAL (12,4) NOT NULL,
PRIMARY KEY (item_nbr, step));
```

```
INSERT INTO PriceList VALUES (101, 1, 1, 20, 5.00);
INSERT INTO PriceList VALUES (101, 2, 21, 99, 4.00);
INSERT INTO PriceList VALUES (101, 3, 100, 1000, 3.00);
INSERT INTO PriceList VALUES (205, 1, 1, 99, 17.00);
```

We kunnen een query doen naar een artikel en een bepaald afnamevolume om de prijs te verkrijgen. Maar zo simpel is het niet. Iemand zou kunnen informeren naar de prijs bij een bepaalde hoeveelheid die buiten de reeksen valt. Misschien bestaat er een kloof of een overlapping bij de grenswaarden. Onbedoeld of toevallig, maar helaas komt dit in de realiteit voor.

In ons voorbeeld levert een query naar 600 exemplaren van product "101" wel resultaat op, maar niet bij product "205". Natuurlijk kunnen we aan de bovengrens van de laatste trap van elke reeks een absurd hoge waarde toekennen, die in de realiteit nooit zal voorkomen. Dit zal in de praktijk best werken, maar in theorie is het niet de beste manier. De vrees bestaat toch dat iemand op zekere dag een order plaatst voor 9999 producten "101". Het is een stuk makkelijker om reeksen zonder bovengrenzen te gebruiken.

We kennen daarvoor een NULL-waarde toe aan de kolom met de bovengrenzen, die daardoor oneindig worden.

```
CREATE TABLE PriceList
(item_nbr INTEGER NOT NULL,
step INTEGER NOT NULL,
low INTEGER NOT NULL CHECK (low > 0),
high INTEGER,  -- null means infinite
CHECK (low < high),
price DECIMAL (12,4) NOT NULL,
PRIMARY KEY (item_nbr, step));
```

Staffel-korting

We bepalen nu prijsniveau 3 voor artikel "205" bij een afname groter dan 501 exemplaren.

```
INSERT INTO PriceList VALUES (205, 3,
501, NULL, 13.50);
```

De query wordt eenvoudig:

```
SELECT O1.item_nbr, O1.quantity, P1.price
FROM PriceList AS P1, Orders AS O1
WHERE O1.quantity
BETWEEN P1.low
AND COALESCE (P1.high, O1.quantity);
```

Een ander probleem vormen doublures of gaten. Als de prijs voor product "205" bepaald is voor een reeks van 1 tot en met 99 stuks en voor een reeks van 101 tot en met 500, dan ontbreekt de hoeveelheid van 100. Andersom, als de prijs bepaald is voor een reeks van 1 tot en met 100 stuks en op het volgende niveau van 100 tot en met 500, dan bestaan er twee prijzen voor een afname van 100 stuks.

Gaten en doublures lossen we op door ze met een query te zoeken. De volgende query levert alle artikelen in the PriceList tabel op.

```
SELECT item_nbr
FROM PriceList GROUP BY item_nbr
HAVING SUM((high - low + 1) * (low + high) / 2)
= ((MAX(high) - MIN(low) + 1)
* (MIN(low) + MAX(high)) / 2)
AND MAX(high) - MIN(low) + 1 = SUM(high - low + 1);
```

Dit is een moeilijk stukje wiskunde. De optelling van alle gehele getallen tussen a en b kan weergegeven worden met de formule

(n * (a + b) / 2), daarbij is n het aantal gehele getallen oftewel (b - a + 1). Effectief komt de berekening van het totaal neer op het vermenigvuldigen van het gemiddelde met het aantal keren dat het voorkomt.

Er zijn geen doublures of gaten, wanneer de som van alle gehele getallen die in de reeks vallen gelijk is aan de som van alle gehele getallen tussen de minimum lage en maximum hoge grenswaarden en de lengte van de totale reeks gelijk is aan de optelling van alle reeksen.

Eigenlijk werkt deze query niet helemaal goed bij gebruik van NULLs voor een oneindig hoge waarde.
We moeten dan in de HAVING clause "high" veranderen in "COALESCE(high, (low +1))". ●

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).