SELECT FROM AUSTIN, TEXAS

Joe Celko over SQL en andere database-zaken

Een belangrijk onderdeel van datamining bestaat uit het zoeken naar profielen of verbanden in databases. Naar sommige profielen wordt incidenteel gezocht, naar andere op regelmatige basis. We bekijken een eenvoudige tabel met gegevens over de verkoop van onze gereedschappen.

```
CREATE TABLE Sales
(sale_nbr INTEGER NOT NULL PRIMARY KEY,
buyer VARCHAR(30) NOT NULL,
color CHAR(6) NOT NULL DEFAULT'red'
CHECK (color IN ('red', 'blue', 'yellow')),
size CHAR(1) NOT NULL DEFAULT'M'
CHECK (size IN ('S', 'M', 'L')), weight INTEGER NOT NULL);
```

Wat we nodig hebben is een nieuwe tabel voor de profielen waarin we geïnteresseerd zijn. We definiëren de nieuwe tabel.

Profiel Match

```
CREATE TABLE Profiles
(profile_nbr INTEGER NOT NULL,
test_nbr INTEGER NOT NULL,
buyer VARCHAR(30),
color CHAR(6)
CHECK (color IN ('red', 'blue', 'yellow')),
size CHAR(1) CHECK (size IN ('S', 'M', 'L')),
low_weight INTEGER,
high_weight INTEGER,
CHECK (low_weight <= high_weight)
PRIMARY KEY (profile_nbr, test_nbr));
```

Alle non-key kolommen van de Sales tabel komen terecht in de Profiles tabel, maar kunnen een NULL zijn. De NULL en '%' fungeren als "wildcards" bij onze zoektocht. We laten de query los op de Profiles tabel rijen en schrijven daarvoor één algemene query in plaats van dynamic SQL te gebruiken voor elk profiel. De basis query ziet er zo uit:

```
SELECT DISTINCT S1.*, P1.profile_nbr
FROM Sales AS S1, Profiles AS P1
WHERE S1.buyer LIKE COALESCE(P1.buyer, S1.buyer)
AND S1.color = COALESCE (P1.color, S1.color)
AND S1.size = COALESCE (P1.size, S1.size)
AND S1.weight BETWEEN COALESCE(P1.low_weight, -1)
AND COALESCE(P1.high_weight, 9999999);
```

We gaan er vanuit dat de werking van het LIKE predikaat bekend is, zodat we dat niet verder hoeven uitleggen. Laten we beginnen met een eenvoudig profiel, dat zoekt naar alle small size ('S') blauwe gereedschappen.

```
INSERT INTO Profiles
VALUES (1, 1, '%', 'blue', 'S', NULL, NULL);
```

De koper wordt gedefinieerd door een string die gevonden wordt door de '%' wildcard. De grote truc nu is om de COALESCE () functie te gebruiken zodat we zeker weten dat een NULL wordt omgezet naar dezelfde waarde die in de Sales kolom links van het vergelijkingsteken (=) staat. Dit betekent dat de predikaten die een NULL waarde toegekend kregen altijd TRUE zijn. Elke rij in de Profiles tabel werkt als een AND operator.
Nu maken we de search wat moeilijker: we willen alle blauwe of gele kleine gereedschappen vinden:

```
INSERT INTO Profiles
VALUES (2, 1, '%', 'blue', 'S', NULL, NULL);
INSERT INTO Profiles
VALUES (2, 2, '%', 'yellow', 'S', NULL, NULL);
```

De truc hier is het (profile_nbr, test_nbr) koppel. Het profile_nbr geeft een identifier voor het gehele profiel en het test_nbr werkt als een OR statement. Daarna passen we een SELECT DISTINCT toe om het test_nbr kwijt te raken en we houden alleen het benodigde profiel over. We kijken naar het volgende profiel:

```
INSERT INTO Profiles
VALUES (3, 1, '%', 'blue', 'S', NULL, NULL);
INSERT INTO Profiles
VALUES (3, 2, '%', 'yellow', 'M', NULL, NULL);
```

Het vindt kleine blauwe of middelgrote gele gereedschappen.

Waardenreeksen zijn wat ingewikkelder, ze hebben immers twee invoergegevens nodig; een beneden- en een bovengrens. De boven- en ondergrens moeten nauwkeurig bepaald worden. We kijken naar de selectie op gewicht in deze basis query:

```
AND S1.weight BETWEEN COALESCE(P1.low_weight, -1)
AND COALESCE(P1.high_weight, 9999999)
```

In de twee aangeroepen COALESCE() functies worden uiterste waarden ingevoerd, die in ons praktijkvoorbeeld nooit zullen voorkomen. Dit predikaat kan ook op een andere, iets veiliger manier omschreven worden:

```
AND S1.weight BETWEEN COALESCE(P1.low_weight,
(SELECT MIN(weight) FROM Sales))
AND COALESCE(P1.high_weight,
(SELECT MAX(weight) FROM Sales))
```

Dit ziet er al beter uit, want de subqueries naar de uitersten van de reeks hebben hier geen verband met elkaar en hoeven maar één keer berekend te worden.

Als we echt geluk hebben, staan de door MIN() en MAX() bepaalde waarden in de laatste optimizer statistics, en hoeven ze helemaal niet berekend te worden. We bekijken in een paar voorbeelden het gebruik van dit predikaat:

Zoek gereedschap met een gewicht tussen tien en twintig kilo:

```
INSERT INTO Profiles
VALUES (3, 1, '%', NULL, NULL, 10, 20);
```

Zoek gereedschap zwaarder dan 40 kilo:

```
INSERT INTO Profiles
VALUES (4, 1, '%', NULL, NULL, 40, NULL);
```

Zoek gereedschap lichter dan 25 kilo:

```
INSERT INTO Profiles
VALUES (5, 1, '%', NULL, NULL, NULL, 25);
```

Een van de problemen met een ad hoc query is, dat we alle gegevens te zien krijgen, en dat willen we natuurlijk niet. In ons voorbeeld zou een rij met alle NULLs alle rijen in de sales tabel aanroepen. Dit moeten we voorkomen door een CHECK() constraint te gebruiken voor de tabel. Die ziet er als volgt uit:

```
CHECK (COALESCE (buyer, color, size,
CAST(weight AS VARCHAR(30)) IS NOT NULL
```

Als er alleen maar NULL waarden bestaan, is het resultaat van de COALESCE functie ook NULL. De COALESCE functie kan maar één datatype verwerken en daarom moeten we het gewicht, dat een INTEGER waarde heeft, omzetten naar een character datatype. De langste kolom in de tabel is de buyer, de andere character datatypes voegen zich daar automatisch naar.

Op dezelfde wijze kunnen we een procedure vastleggen voor eenvoudige ad hoc queries die niet werken met een predikaat met een OR.

```
CREATE PROCEDURE FindWidgets
(find_buyer IN VARCHAR(30), find_color IN CHAR(6),
find_size IN CHAR(1), find_weight IN INTEGER)
AS SELECT DISTINCT S1.*
FROM Sales AS S1
WHERE S1.buyer LIKE find_buyer
AND S1.color = COALESCE (find_color, S1.color)
AND S1.size = COALESCE (find_size, S1.size)
AND S1.weight BETWEEN COALESCE(P1.low_weight,
(SELECT MIN(weight) FROM Sales))
AND COALESCE(P1.high_weight,
(SELECT MAX(weight) FROM Sales))
AND (COALESCE (buyer, color, size,
CAST(weight AS VARCHAR(30)) IS NOT NULL;
```

Omdat alle non-key kolommen in een expressie vast liggen, kunnen de meeste SQL applicaties ze helaas niet indexeren en moeten we de volledige tabel doorlopen. De laatste truc is om de Sales tabel aan te passen en een kolom toe te voegen met het profielnummer waarop gezocht wordt. De tabel hoeft nu maar één keer gegenereerd te worden en dat scheelt een hoop tijd:

```
CREATE TABLE Sales
(sale_nbr INTEGER NOT NULL PRIMARY KEY,
buyer VARCHAR(30) NOT NULL,
color CHAR(6) NOT NULL DEFAULT 'red'
CHECK (color IN ('red', 'blue', 'yellow')),
size CHAR(1) NOT NULL DEFAULT 'M'
CHECK (size IN ('S', 'M', 'L')), weight INTEGER NOT NULL,
profile_nbr INTEGER NOT NULL DEFAULT 0);
```

en daarna doen we een UPDATE, dan wordt het profile_nbr maar één keer berekend:

```
UPDATE Sales
SET profile_nbr
= (SELECT MAX(P1.profile_nbr_nbr)
FROM Sales AS S1, Profiles AS P1
WHERE S1.buyer LIKE COALESCE(P1.buyer, S1.buyer)
AND S1.color = COALESCE (P1.color, S1.color)
AND S1.size = COALESCE (P1.size, S1.size)
AND S1.weight BETWEEN COALESCE(P1.low_weight,
(SELECT MIN(weight) FROM Sales))
AND COALESCE(P1.high_weight, (SELECT MAX(weight) FROM Sales)))
WHERE profile_nbr = 0;
```

Omdat een eenmalige verkoop aan een veelheid van profielen kan beantwoorden, moeten we er een kiezen. In ons voorbeeld hebben we de MAX() gekozen, maar de MIN() had ook gekund. Dit impliceert dat de getallen van profile_nbr een zekere betekenis hebben - zouden hogere getallen betere profielen zijn? ●

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).