SQL SERVER 2008

Nieuwe features in T-SQL

Bij de introductie van de CLR-integratie in SQL Server bestond bij velen de angst dat T-SQL zou verdwijnen. Maar verbeteringen aan T-SQL, zoals common table expressions, ranking functions en structured error handling, toonden het tegendeel al aan. T-SQL is en blijft dé taal om data mee te manipuleren in SQL Server. En in SQL Server 2008 komen we weer een aantal gave features tegen die T-SQL nog krachtiger maken dan het al was.

In dit artikel begin ik met een aantal kleine verbeteringen te tonen om daarna in wat meer detail te kijken naar Grouping Sets, een nieuwe manier om te groeperen, en naar het Merge statement. Verbeteringen zijn niet altijd grote veranderingen. Voor developers die niet van typen houden is de code in listing 1 al een leuke verbetering. Allereerst kunnen we een variabele nu declareren en initialiseren in één statement. Daarnaast krijgen we een increment operator (+=). De derde verbetering die we in de listing zien, is het Insert Values statement. Waar we met dit statement tot nu toe slechts één record tegelijk konden toevoegen aan een tabel, kunnen we nu, door komma's gescheiden, meer records toevoegen in één statement. De increment operator werkt ook met character datatypes. Bovendien bestaan ook de varianten -=, *=, /=, != en &=.

Een andere gave, schijnbaar kleine, maar zeer nuttige nieuwe feature zijn de vernieuwde datum datatypes. Misschien vallen datatypes eigenlijk niet onder T-SQL, maar ze gaan onze T-SQL code wel gemakkelijker maken. Eindelijk is er een datatype, Date, dat alleen een datum bevat, dus zonder tijd. Net zo bevat het nieuwe Time datatype alleen een tijd. Dus we hebben niet langer conversies nodig om het tijd-gedeelte van een datum kwijt te raken. Bovendien is er een datatype, DateTimeOffset, die timezone-aware is, wat zoveel inhoudt dat het verschil met de UTC-tijd wordt bijgehouden. Tot slot is het bereik van datums opgerekt. De ondergrens is nu het jaar 0 (in plaats van 1753) en de precisie kun je instellen tot een maximum van 100 nanoseconde. Het oude DateTime datatype veranderde niet. Het nieuwe bereik zit in de al genoemde nieuwe datatypes en in het DateTime2 datatype.

Grouping Sets

In veel situaties zijn we niet geïnteresseerd in detailrecords zoals ze in de database zitten, maar in aggregaties daarvan. Daarvoor hebben we sinds jaar en dag de Group By clause in het Select statement. Ook in rapportages gebruiken we het concept van groeperen veel. Bij elkaar horende records worden in groepen getoond en vaak eindigt een groep met een subtotaal voor die groep. Bovendien eindigt het rapport dan vaak met een totaal generaal. Maar wat als je niet een rapportagetool zoals Reporting Services gebruikt om deze groepen te definiëren, maar in een resultset vergelijkbare tussentotalen wilt tonen?

De Group By clause van het Select statement kent twee extra operatoren, Rollup en Cube. Rollup beschouwt de kolommen waarover je groepeert als een soort hiërarchie in een dimensie en voegt op levelovergangen totaalrecords toe. De Rollup lijkt daarmee op een drill-down in een hiërarchie. De Cube operator is vergelijkbaar maar beperkt zich niet tot een ééndimensionale drill-down. Voor alle denkbare levels worden totaalrecords toegevoegd. Listing 2 maakt een tabel met wat gegevens erin. In het screenshot zien we het resultaat van het gebruik van de Rollup en Cube op deze voorbeeld-data.

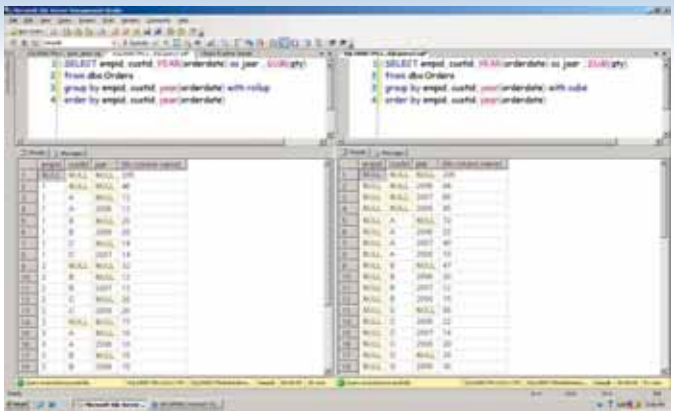
Rollup en Cube zijn leuk maar nogal beperkt omdat ze niet flexibel zijn. Je kunt niet zelf kiezen op welke levels je wel en geen tussentotalen wilt. Gelukkig kan je de functionaliteit nabootsen door voor elk gewenst tussenresultaat een query te schrijven en van alle resultaten één resultaat te maken met de Union All. Listing 3 geeft precies hetzelfde resultaat als

```
DECLARE @i int = 0

SET @i += 2

INSERT TABLE Test
VALUES (@i)
    , (@i + 2)
    , (@i + 4)
```

LISTING 1

```
USE tempdb
GO
CREATE TABLE dbo.Orders
(
  orderid   INT        NOT NULL,
  orderdate DATETIME   NOT NULL,
  empid     INT        NOT NULL,
  custid    VARCHAR(5) NOT NULL,
  qty       INT        NOT NULL,
  CONSTRAINT PK_Orders PRIMARY KEY(orderid)
);

INSERT INTO dbo.Orders
  (orderid, orderdate, empid, custid, qty)
VALUES
  (30001, '20060802', 3, 'A', 10),
  (10001, '20061224', 1, 'A', 12),
  (10005, '20061224', 1, 'B', 20),
  (40001, '20070109', 4, 'A', 40),
  (10006, '20070118', 1, 'C', 14),
  (20001, '20070212', 2, 'B', 12),
  (40005, '20080212', 4, 'A', 10),
  (20002, '20080216', 2, 'C', 20),
  (30003, '20080418', 3, 'B', 15),
  (30004, '20060418', 3, 'C', 22),
  (30007, '20060907', 3, 'D', 30);
GO
```

LISTING 2



SCREENSHOT MET HET RESULTAAT VAN HET GEBRUIK VAN DE ROLLUP EN CUBE

de Rollup uit het screenshot, maar nu door zelf de tussenresultaten te maken. Met deze oplossing winnen we flexibiliteit omdat we elk gewenst resultaat via een Union aan de resultset kunnen toevoegen. Helaas is de query uit listing 3 ongeveer tweemaal zo traag als de variant met de Rollup operator.

SQL Server 2008 introduceert Grouping Sets. Zoals de naam al suggereert kunnen we hiermee groeperen over verschillende sets van kolommen. Met als resultaat dat we de verschillende Group By's uit listing 3 samenvoegen tot een Grouping Set. Onze query komt er dan uit te zien zoals in listing 4. Ook deze query geeft exact hetzelfde resultaat als de query met de Rollup uit het screenshot en ook met dezelfde performance. Maar we zijn wel heel flexibel geworden.

Als we geen totaal generaal willen, halen we de lege Grouping Set, de haakjes met niets ertussenin, gewoon weg. Maar we kunnen ook een extra Grouping Set toevoegen, zoals bijvoorbeeld (custid). Dit levert records op met een totaal over alle jaren en over alle werknemers voor elke specifieke klant. En omdat alles met één scan wordt ingelezen, bespaar je veel 'Reads' ten opzichte van de oude manier en is de performance dus goed.
Naast de functionaliteit is ook de syntax zeer flexibel. We kunnen meer Grouping Sets per Group By gebruiken. Dat levert dan een cartesisch product van Grouping Sets op. Om het te vereenvoudigen kun je kolommen die in elke Grouping Set voorkomen er ook uit halen en direct na de Group By clause zetten. Zie voor alle mogelijkheden en voor leuke voorbeelden BOL.

```
SELECT empid, custid, YEAR(orderdate) as jaar , SUM(qty)
FROM dbo.Orders
GROUP BY empid, custid, YEAR(orderdate)

UNION ALL

SELECT empid, custid, null , SUM(qty)
FROM dbo.Orders
GROUP BY empid, custid

UNION ALL

SELECT empid, null, null , SUM(qty)
FROM dbo.Orders
GROUP BY empid

UNION ALL

SELECT null, null, null , SUM(qty)
FROM dbo.Orders
GROUP BY empid, custid, YEAR(orderdate)
```

LISTING 3

```
SELECT empid, custid, YEAR(orderdate) as jaar , SUM(qty)
FROM dbo.Orders
GROUP BY GROUOIBG SETS
(
    (empid, custid, YEAR(orderdate))
    , (empid, custid)
    , (empid)
    , ()
)
ORDER BY empid, custid, YEAR(orderdate)
```

LISTING 4

Met de Grouping Sets komt ook een nieuwe functie, namelijk Grouping_ID(). De Grouping_ID functie is vergelijkbaar met de reeds bestaande Grouping-functie, maar flexibeler in het gebruik. De oude Grouping-functie geeft een extra kolom in de resultset die aangeeft of een record is toegevoegd door het gebruik van Rollup of Cube. De Grouping_ID() functie geeft met een getal niet alleen aan dat een record is toegevoegd aan de resultset, maar ook om welke groep het gaat. Als je in het eerder gebruikte voorbeeld een kolom Grouping_ID(empid, custid, year(orderdate)) opneemt, krijg je 0, 1, 3 of 7 terug. 7 (binair 111) betekent dat voor alle drie genoemde kolommen een null is ingevuld (de lege Grouping Set) en dat we dus met het totaal generaal record te maken hebben. 3 (binair 011) betekent dat er over empid 'normaal' gegroepeerd is en we dus het totaal van een specifieke werknemer over alle klanten over alle jaren bekijken. Met een Case is deze functie (of eigenlijk de resulterende bitmask) gemakkelijk te vertalen naar iets zinvols om alle null-waarden in de resultset mee te vervangen.

Bovendien neemt de performance toe omdat we met minder IO's toekunnen

Merge

In veel situaties willen we een tabel wijzigen op basis van een andere tabel. Denk bijvoorbeeld aan een dimensietabel in een datawarehouse. Als in de onderliggende OLTP productiedatabase veranderingen optreden, moeten die ook in het datawarehouse doorgevoerd worden. Vaak moeten we dan code schrijven die controleert of records al bestaan of niet, en op basis daarvan update, insert of delete statements uitvoeren.
De code in listing 5 maakt een tabel Products en een tabel Mutaties. Via de mutaties-tabel krijgen wij de wijzigingen door. Als een product in zowel de products-tabel als de mutaties-tabel voorkomt, moeten we de voorraad bijwerken. Een product dat wel in de mutaties-tabel

```
CREATE TABLE Products (id int, name nvarchar(20), qty int)
go
INSERT into Products VALUES
    (1, 'Bicycle', 0),
    (2, 'Roller blades', 0),
    (3, 'Soccer ball', 0);
GO
CREATE TABLE Mutaties (id int, name nvarchar(20), qty int)
go
INSERT into Mutaties VALUES
    (1, 'Bicycle', 5),
    (2, 'Roller blades', 10),
    (4, 'Baseball ball', 15);
```

LISTING 5

voorkomt maar niet in de products-tabel moeten we toevoegen. En een product waarvoor het omgekeerde geldt, zullen we uit de products-tabel verwijderen. Met de juiste inner en outer joins is dit niet heel lastig te schrijven, maar het nieuwe Merge statement maakt het ons nog gemakkelijker. Bovendien neemt de performance toe omdat we met minder IO's toekunnen.

Het Merge statement koppelt twee tabellen via het Using keyword en met een join-achtige syntax aan elkaar. Vervolgens kunnen we via een soort Case expressie, dus met when … then … constructies, de verschillende mogelijkheden afvangen. Daartoe hebben we de keywords (NOT) MATCHED en SOURCE NOT MATCHED tot onze beschikking. Dit tezamen levert het statement op zoals te zien in listing 6.

```
MERGE Products P USING Mutaties M on M.id = P.id
WHEN MATCHED -- AND ...
    THEN UPDATE SET P.qty += M.qty
WHEN NOT MATCHED
    THEN INSERT (id, name, qty) VALUES (M.id, M.name, M.qty)
WHEN SOURCE NOT MATCHED
    THEN DELETE;
```

LISTING 6

```
CREATE TYPE MutatieType AS TABLE (id INT, name NVARCHAR(100), qty
INT);
GO

-- Pass table variable as a parameter to a stored procedure
CREATE PROCEDURE UpdateProducts (@tvp MutatieType READONLY)
AS
    MERGE Products P USING @tvp M on M.id = P.id
    WHEN MATCHED -- AND ...
      THEN UPDATE SET P.qty += M.qty
    WHEN NOT MATCHED
      THEN INSERT (id, name, qty) VALUES (M.id, M.name, M.qty)
    WHEN SOURCE NOT MATCHED
      THEN DELETE;
GO

-- Test: declare & populate variable of the TABLE type
DECLARE @list AS MutatieType;

INSERT INTO @list VALUES
    (1, 'Bicycle', 5),
    (2, 'Roller blades', 10),
    (4, 'Baseball ball', 15);

EXEC myProc @list;
```

LISTING 7

```
using (SqlConnection con = new SqlConnection(
       "server=.;database=testdb;integrated security=sspi"))
{
    con.Open();

    using (SqlCommand com = new SqlCommand("dbo.UpdateProducts",
con))
    {
        com.CommandType = CommandType.StoredProcedure;
        com.Parameters.AddWithValue("tvp", table);
        com.ExecuteNonQuery();
    }

    con.Close();
}
```

LISTING 8

Table Valued Parameters

Het zou gaaf zijn als we het beschreven Merge statement in een stored procedure kunnen gebruiken, waarbij we de mutaties als parameter aan de procedure meegeven. Dat eerste is niet moeilijk, maar een tabel als parameter meegeven was tot nu toe onmogelijk. In SQL Server 2008 krijgen we die mogelijkheid echter wel. Stap 1 daarin is het maken van een user defined type (alias type) van het type tabel. Daarna kunnen we een stored procedure een parameter meegeven van dit eigen datatype. Zo'n zogenaamde table valued parameter moet wel altijd het nieuwe keyword READONLY meekrijgen. Als je in de stored procedure toch probeert te schrijven naar deze parameter, zal de stored procedure niet compileren. De code van listing 7 maakt het user defined type en de stored procedure. Uiteraard kan je de stored procedure ook aanroepen vanuit ADO.NET code. De C# code in listing 8 laat dit zien. In de aanroep van Parameters.AddWithValue is tvp de naam van de parameter van onze stored procedure en Table een variabele in de C# code van het datatype DataTable. Deze variabele kan de DataSource van een DataGridView zijn die disconnected is. De getoonde code werkt de database met een enkele aanroep bij, met alle wijzigingen in de DataGridView.

Ten slotte

De beschreven nieuwe features van T-SQL in SQL Server 2008 vormen niet een complete lijst van vernieuwingen en aanpassingen. In mijn ogen zijn het wel mooie en vooral nuttige aanpassingen die ons in staat stellen efficiëntere code te schrijven. .net

Peter ter Braake (trainsql@live.nl) is zelfstandig SQL Server trainer en consultant.