SQL SERVER 2008

Introductie van HIERARCH

Hoe vaak kom je niet een hiërarchische gegevensstructuur tegen. Denk aan menustructuren, organisatiestructuren, bestand/mappenstructuren en stuklijsten. Windows Verkenner gebruikt bijvoorbeeld een boom om de mappen van de computer te tonen. En hoe vaak moet je zo'n structuur niet opslaan in een database. Dat kan nu met het nieuwe HIERARCHYID datatype binnen SQLServer 2008.

Het HIERARCHYID datatype is binnen SQLServer 2008 geïmplementeerd via een CLR User Defined Datatype. Je kunt er eenvoudig hiërarchische gegevens mee opslaan in de database. Als voorbeeld voor dit artikel maak ik gebruik van de organisatiestructuur zoals weergegeven in schema 1. In deze voorbeeldorganisatie is Elmy de president en heeft dus zelf geen manager. Elmy is zelf de manager van Frits en Jan. Frits is de manager van Janine, et cetera.

Deze medewerkers kunnen we opslaan in de weergegeven tabel. Hierbij wordt het veld Manager gebruikt als relatieveld naar de bovenliggende medewerker (manager).

```
CREATE TABLE Employee
    (Name  NVARCHAR(255) PRIMARY KEY NOT NULL,
     Title NVARCHAR(255) NOT NULL,
     Manager  NVARCHAR(255) FOREIGN KEY REFERENCES(Name) NULL);
```

De eerste vier medewerkers worden dan als volgt opgeslagen:

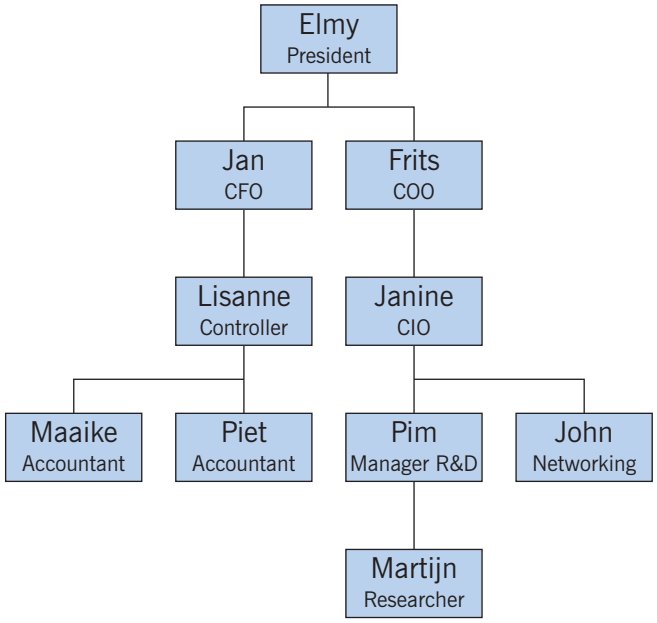
Name	Title	Manager
Elmy	President	NULL
Frits	COO	Elmy
Jan	CFO	Elmy
Janine	CIO	Frits

Om vervolgens de medewerkers op te halen die direct of indirect gemanaged worden door medewerker 'Frits' gebruik je een query die een Common Table Expression (CTE) bevat. CTE is geïntroduceerd in SQL Server 2005. Een CTE is redelijk vergelijkbaar met een subquery in de FROM clause maar heeft als voordeel dat deze recursief gegevens kan ophalen.

```
WITH MyCTE(Name, Title) AS (
  SELECT Name, Title
  FROM Employee
  WHERE Name = 'Frits'
  UNION ALL
  SELECT E.Name, E.Title
  FROM Employee AS E
  JOIN MyCTE AS M ON E.Name = M.Manager)
SELECT * FROM MyCTE
```

HIERARCHYID datatype

Een organisatiestructuur wordt ook wel een boomstructuur genoemd waarbij ieder medewerker een knoop is. Binnen een boomstructuur kan



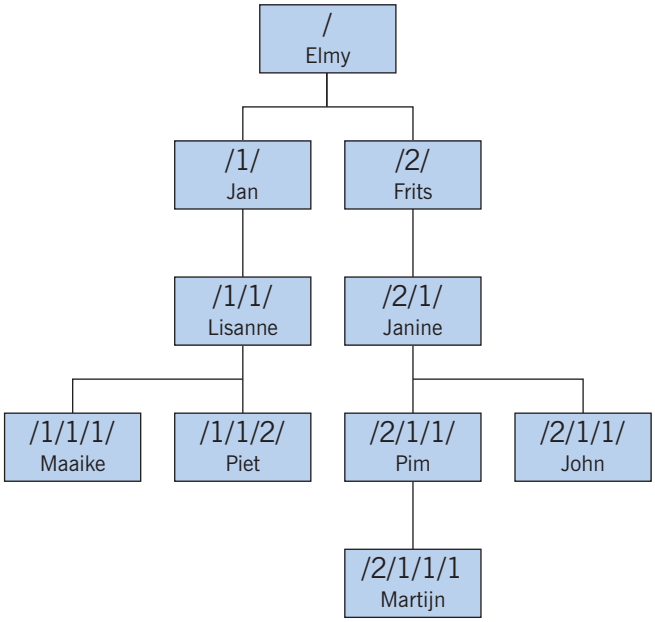
SCHEMA 1

men echter niet alleen onderliggende knopen opslaan maar ook de volgorde van de knopen onder een andere knoop. Dit gaat niet met de hier besproken oplossing, maar wel met het in SQL Server 2008 geïntroduceerde HIERARCHYID datatype. Dit datatype is zeer compact en zeer krachtig in gebruik bij het toevoegen en verwijderen van knopen in een hiërarchie. De GetDecendant()-functie zorgt voor het opslaan van een knoop vóór of juist achter een andere knoop.

Voordat we de T-SQL-code gaan behandelen, kijken we eerst hoe de opslag plaatsvindt. De positie in de hiërarchie van een knoop wordt in een binair formaat opgeslagen, zeer compact dus. De bovenste knoop, 'root' genoemd, krijgt de waarde 0x. De methode ToString() kun je gebruiken om deze binaire waarde naar een leesbare tekst om te zetten. De 0x-waarde krijgt de tekst '/'. De onderliggende knoop (voorste kind) krijgt de tekstwaarde '/1/', de volgende '/2/'. De kinderen van '/1/' krijgen de tekstwaarde '/1/1/', '1/2/2/', et cetera (zie schema 2)

Onze eerder gebruikte organisatiestructuur kunnen we opslaan met behulp van de volgende tabel. Deze tabel maakt gebruik van een HIERARCHYID-kolom genaamd OrgNode.

YID datatype



SCHEMA 2

```
CREATE TABLE Employee
    (Name NVARCHAR(255) PRIMARY KEY NOT NULL,
     Title NVARCHAR(255) NOT NULL,
     OrgNode HIERARCHYID);
```

Knopen toevoegen

Bij het toevoegen van de medewerker moeten we de OrgNode voorzien van de juiste HIERARCHTID-waarde. Als eerste voegen we medewerker Elmy toe als bovenste (root) knoop. Hiervoor maken we gebruik van de statische GetRoot()-methode voor het bepalen van de juiste waarde voor de OrgNode-kolom.

```
INSERT INTO Employee (Name, Title, OrgNode)
VALUES ('Elmy', 'President', HIERARCHYID::GetRoot());
```

Voor het toevoegen van de eerste medewerker (kind) direct onder Elmy, maken we in plaats van GetRoot() gebruik van de GetDecendant()-methode.

```
DECLARE @ParentNode HIERARCHYID;

SELECT @ParentNode = OrgNode FROM Employee WHERE Name = 'Elmy';

INSERT INTO Employee (Name, Title, OrgNode)
VALUES ('Jan', 'CFO', @ParentNode.GetDecendant(NULL, NULL));
```

Als eerste wordt de OrgNode-waarde van Elmy bepaald. Daarna is de GetDecendant()-methode met twee NULL-parameters gebruikt om de OrgNode-waarde voor Jan te bepalen. Deze parameters zijn NULL omdat de volgorde van de knopen er in dit geval niet toe doet, er zijn namelijk nog geen andere knopen onder de root.
Voor het toevoegen van de tweede medewerker direct onder Elmy en achter Jan gebruiken we ook de GetDecendant()-methode. De eerste parameter bevat nu echter de waarde van de OrgNode waarachter de nieuwe knoop moet komen. Om dit te bepalen halen we de grootste (MAX) OrgNode op van de medewerker waar de OrgNode van de ouder (bepaald via de GetAncestor(1)-methode) gelijk is aan die van Elmy. Hierdoor wordt Frits achter Jan onder Elmy opgeslagen.

```
DECLARE @ParentNode HIERARCHYID;
SELECT @ParentNode = OrgNode FROM Employee WHERE Name = 'Elmy';

DECLARE @LastChildPosition HIERARCHYID
SELECT @LastChildPosition = MAX(OrgNode) FROM Employee
WHERE OrgNode.GetAncestor(1) = @ParentNode;

INSERT INTO Employee (Name, Title, OrgNode)
VALUES ('Frits', 'COO', @ParentNode.GetDecendant(@LastChildPosi-
tion, NULL));
```

Het toevoegen van medewerkers is te vereenvoudigen door het opzetten van de AddEmployee stored-procedure zoals beschreven in listing 1. Deze stored- procedure voegt een medewerker toe door aan de hand van de naam van de manager de HIERARCHYID te bepalen.

Gegevens ophalen

De medewerkers haal je met een eenvoudige SELECT-query op. De ToString()-methode toont de tekstwaarde van de HIERARCHYID. Door te sorteren worden de medewerkers top-down opgehaald.

```
SELECT *, OrgNode.ToString() FROM Employee order by OrgNode;
```

De IsDecendant()-methode gebruiken we om te bepalen of een knoop onder een andere knoop zit, direct of via een van de ouders. De onderstaande query haalt medewerker Frits op en alle medewerkers daaronder. Deze is zeker eenvoudiger dan de eerder besproken recursieve CTE-query.

```
CREATE PROCEDURE AddEmployee
     @Name NVARCHAR(255),
     @Title NVARCHAR(255),
     @ParentName NVARCHAR(255)
AS
BEGIN
    SET NOCOUNT ON

    IF @ParentName IS NULL
      BEGIN
        -- Insert the root node
        INSERT INTO Employee (Name, Title, OrgNode)
        VALUES (@Name, @Title, HIERARCHYID::GetRoot());
      END
    ELSE
      BEGIN
        -- Get the parent node we wish to insert a descendant of
        DECLARE @ParentNode HIERARCHYID
        SELECT @ParentNode = OrgNode FROM Employee
        WHERE Name = @ParentName;
```

Adv

```
        -- Determine the last child position
        DECLARE @LastChildPosition HIERARCHYID
        SELECT @LastChildPosition = MAX(OrgNode) FROM Employee
        WHERE OrgNode.GetAncestor(1) = @ParentNode;

        -- Insert the Employee
        INSERT INTO Employee (Name, Title, OrgNode)
          VALUES (@Name, @Title,
             @ParentNode.GetDescendant(@LastChildPosition,
             NULL));
      END
END;
GO

-- Er van Uitgaande dat de tabel leeg is.
EXEC AddEmployee 'Elmy', 'President', NULL;
EXEC AddEmployee 'Jan', 'CFO', 'Elmy';
EXEC AddEmployee 'Frits', 'COO', 'Elmy';
EXEC AddEmployee 'Lisanne', 'Controller', 'Jan';
EXEC AddEmployee 'Jan', 'CIO', 'Frits';
EXEC AddEmployee 'Maaike', 'Accountant', 'Lisanne';
EXEC AddEmployee 'Piet', 'Accountant', 'Lisanne';
EXEC AddEmployee 'Pim', 'Manager R&D', 'Janine';
EXEC AddEmployee 'John', 'Networking', 'Janine';
EXEC AddEmployee 'Martijn', 'Researcher', 'Pim';
```

LISTING 1. ADDEMPLOYEE STORED-PROCEDURE

```
DECLARE @frits HIERARCHYID;
SELECT @frits = OrgNode FROM Employee WHERE Name = 'Frits';


SELECT *, OrgNode.ToString() FROM Employee
WHERE @frits.IsDescendant(OrgNode) = 1;
```

De medewerkers boven medewerker Pim haal je als volgt op.

```
DECLARE @pim HIERARCHYID;
SELECT @pim = OrgNode FROM Employee WHERE Name = 'Pim';

SELECT *, OrgNode.ToString() FROM Employee
WHERE OrgNode.IsDescendant(@pim) = 1;
```

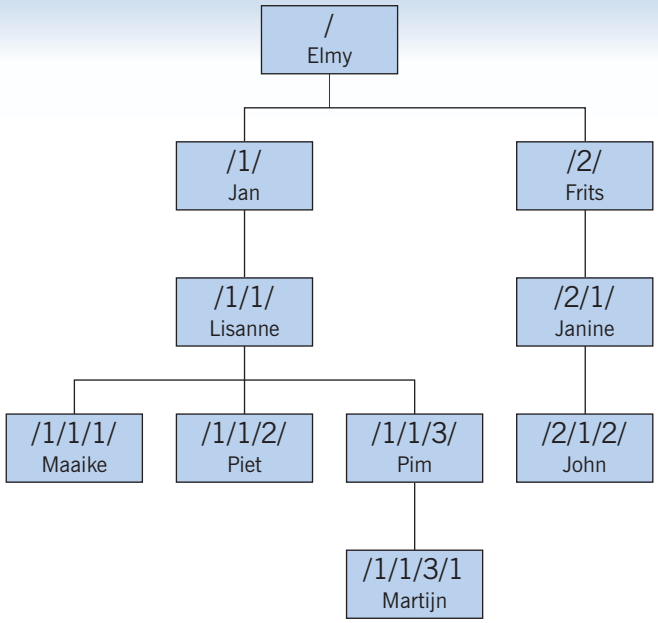
Structuur reorganiseren

Het verplaatsen van een knoop naar een andere positie in de boomstructuur is redelijk complex. Dit omdat alle knopen daaronder mee verhuisd moeten worden. De consistentie tussen de knopen moet je zelf verzorgen. Om dit te realiseren gebruik je de ReParent()-methode in de SET-clause van een UPDATE-statement. Deze functie berekent de nieuwe waarde voor de HIERARCHYID-kolom. Als voorbeeld verplaatsen we medewerker Pim onder Lisanne. Dit gebeurt met behulp van de stored-procedure MoveEmployee. Hierin wordt de grootste OrgNode van alle kinderen onder Lisanne bepaald. Daarna wordt de OrgNode van Pim bepaald, waarna aanpassing volgt van de OrgNode van Pim en alle kinderen onder Pim. Zie listing 2.

Na het verplaatsen ziet de organisatiestructuur er uit als in schema 3. De tekstwaarde voor medewerker John is overigens nog steeds '/2/1/2' terwijl het gaat om de enige medewerker onder Janine. Deze is dus niet aangepast naar '/2/1/1'. Ook dit moet je indien gewenst zelf corrigeren met behulp van een extra UPDATE-statement.

Conclusie

Het HIERARCHYID datatype werkt zeer krachtig en flexibel voor het opslaan van boomstructuren waarbij de volgorde van knopen belangrijk is, zoals bij menustructuren. Toch komen we ook een aantal beperkingen tegen. De toekomst zal uitwijzen hoe populair dit nieuwe datatype gaat worden.



SCHEMA 3

Beperkingen:

- zelf voor boomstructuur zorgen;
- er kan/mag maar één ROOT-node zijn;
- zelf consistentie van de hiërarchie verzorgen, bijvoorbeeld met GetDescendant()- en Reparent()-methodes. .net

```
CREATE PROCEDURE MoveEmployee
    @Name NVARCHAR(255),
    @ParentName NVARCHAR(255)
AS
BEGIN
    -- Determine the last child position for the Parent
    DECLARE @new HIERARCHYID;
    SELECT @new = OrgNode FROM Employee WHERE Name = @ParentName;

    SELECT @new = @new.GetDescendant(max(OrgNode), NULL)
    FROM Employee WHERE OrgNode.GetAncestor(1) = @new;

    -- Find the OrgNode of the Employee
    DECLARE @old HIERARCHYID
    SELECT @old = OrgNode FROM Employee WHERE Name = @Name;

    -- Update the Employee and all descendants
    UPDATE Employee
    SET OrgNode = OrgNode.Reparent(@old, @new)
    WHERE @old.IsDescendant(OrgNode) = 1;
END;
GO

-- Move Pim to Lisanne
EXEC MoveEmployee 'Pim', 'Lisanne';
```

LISTING 2. MOVEEMPLOYEE STORED PROCEDURE

Fons Sonnemans (fons.sonnemans@reflectionit.nl) is zelfstandig trainer en consultant in .NET, Silverlight en SQLServer.