Paul van Wingerden

Spatial enabled applicaties bouwen

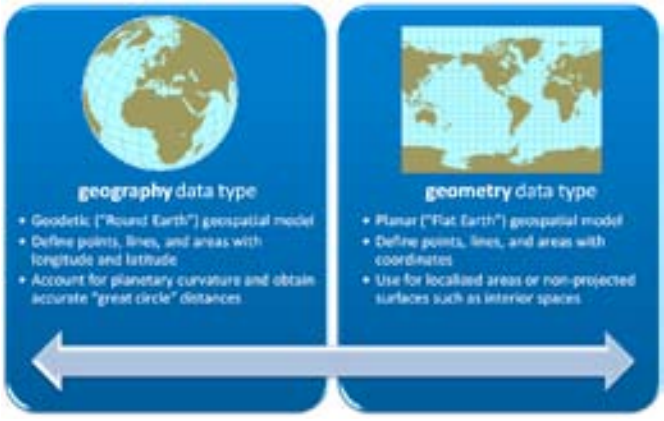
DATA OP EEN KAART TONEN

Het kan niemand ontgaan zijn dat geotechnologie, of 'kaarttechnologie' sterk in opkomst is. Doordat grote partijen als een broker van geografische data fungeren en deze door middel van partijen als Microsoft, Google en Yahoo beschikbaar worden gesteld, kunnen kleine partijen met geringe inspanning en middelen een professionele geo-applicatie in elkaar zetten. In dit artikel worden de nieuwe features van SQL Server 2008 op dit gebied behandeld.

Enkele jaren geleden was het alleen mogelijk voor zeer grote organisaties – vaak overheden – om toepassingen met uitgebreide kaarten, satelliet- en luchtfoto's te gebruiken. Vanwege de kosten van de data zelf en omdat er vele tientallen terabytes aan data worden opgeslagen in het backend. Microsoft Virtual Earth doet een aantal dingen: er worden op grote schaal satellietbeelden en luchtfoto's aangekocht of zelf gemaakt. Deze data moeten bewerkt worden op kleur, helderheid en het aansluiten van randen. Ook moeten de kaartdata zo nauwkeurig mogelijk over de foto vallen.
In dit artikel laat ik zien hoe je met behulp van SQL Server 2008 en de Virtual Earth SDK een spatial enabled applicatie kunt bouwen. De diverse onderdelen worden met behulp van LINQ, AJAX en WCF aan elkaar gekoppeld.

SQL Server spatial data

In SQL Server 2008 is er een nieuw datatype geïmplementeerd: het zogeheten spatial datatype. Dit is een CLR-datatype. Het is een native type dat automatisch beschikbaar is, er hoeft niet eerst een assembly geladen te worden. U zult wellicht denken dat het nu ook mogelijk is een lat/long in een database op te slaan, bijvoorbeeld in een string of in twee velden van het type long. Een nadeel daarvan is dat er geen bewerkingen mee gedaan kunnen worden zonder zelf de complexe berekeningen uit te voeren. De analogie met een veld van het type datetime gaat hier op. Door voor het type datetime te kiezen, kan er gerekend worden met de data. Denk aan het optellen van tien dagen bij een datum of het berekenen van het verschil tussen twee data in bijvoorbeeld minuten.



Afbeelding 1. Twee types: geography en geometry

Bewerkingen op de data

Voor ons voorbeeld gaan we een tabel creëren die er uit ziet zoals in codevoorbeeld 1 is te zien. Het locatieveld is een spatial type met welke we de locatie kunnen bewerken. Eerst moeten we data in de tabel invoeren. Het scriptje in codevoorbeeld 2 laat zien hoe dit kan. De functie STPointFromText converteert de lat/long naar het interne opslagformaat van SQL Server, waarmee gerekend en gemanipuleerd kan worden. Nu is het mogelijk de afstand tussen bijvoorbeeld Haarlem en Amsterdam uit te rekenen. Codevoorbeeld 3 toont hoe dit er uit ziet, het resultaat is te zien in afbeelding 2.
SQL Server geeft altijd een resultaat in meters terug, vandaar het delen door 1000 om de afstand in kilometers uit te drukken. Dit soort functionaliteit laat heel goed de kracht zien van het spatial type en de library met functies die hiermee kunnen werken. Het is redelijk eenvoudig om via de stelling

```
CREATE TABLE Woonplaats
(
  ID    INT IDENTITY (0, 1) NOT NULL,
  Naam    NVARCHAR(50)    NOT NULL,
  Inwoners  INT             NOT NULL,
  Locatie GEOGRAPHY         NOT NULL,
  Lat     AS Locatie.Lat,
  Long    AS Locatie.Long
)
```

Codevoorbeeld 1. Tabel creëren

```
DECLARE @g geography;

SET @g = geography::STPointFromText('POINT(52.373049 4.89304)', 4326);
INSERT Woonplaats (Naam, Inwoners, Locatie) VALUES('Amsterdam',
850000, @g);

SET @g = geography::STPointFromText('POINT(52.379493 4.637718)', 4326);
INSERT Woonplaats (Naam, Inwoners, Locatie) VALUES('Haarlem',
150000, @g);

SET @g = geography::STPointFromText('POINT(53.214882 6.567758)', 4326);
INSERT Woonplaats (Naam, Inwoners, Locatie) VALUES('Groningen',
80000, @g);

SET @g = geography::STPointFromText('POINT(52.091262 5.122748)', 4326);
INSERT Woonplaats (Naam, Inwoners, Locatie) VALUES('Utrecht',
400000, @g);
```

Codevoorbeeld 2.

```
-- Bereken afstand Haarlem Amsterdam
SELECT a.Locatie.STDistance(h.Locatie) / 1000 AS [Afstand in KM]
FROM (SELECT * FROM Woonplaats WHERE NAAM = 'Amsterdam') AS a,
     (SELECT * FROM Woonplaats WHERE NAAM = 'Haarlem') AS h
```

Codevoorbeeld 3.

```
-- Oppervlakte van een polygoon
DECLARE @g geography;
SET @g = geography::STGeomFromText('POLYGON((47.653 -122.358, 47.649
-122.348, 47.658 -122.348, 47.658 -122.358, 47.653 -122.358))', 4326);
SELECT @g.STArea() AS [Opervlakte];
```

Codevoorbeeld 4.

van Pythagoras de afstand tussen twee punten uit te rekenen. Bij een ronde aarde komt daar echter veel meer bij kijken, zie onderstaande formule.

$$d(P,Q) = 2R\arcsin\frac{1}{2}\sqrt{(\cos\beta_Q\cos\alpha_Q - \cos\beta_P\cos\alpha_P)^2 + (\cos\beta_Q\sin\alpha_Q - \cos\beta_P\sin\alpha_P)^2 + (\sin\beta_Q - \sin\beta_P)^2}$$

Bron: wikipedia

Een ander voorbeeld van een bewerking is het berekenen van de oppervlakte van een polygoon. Dit gaat zoals in codevoorbeeld 4 is te zien. De spatial operatoren kunnen gecombineerd worden met de 'normale' operatoren die in SQL Queries gebruikt kunnen worden. Om dit in normale taal uit te drukken, kun je vragen stellen zoals: 'geef mij alle objecten binnen een afstand van <n> die gemaakt zijn van rode baksteen met een verkoopprijs tussen <n> en <m>. Tot slot een voorbeeld hoe je de objecten die zich binnen een polygoon bevinden, kunt opvragen. Dit gaan we later in dit verhaal weer terug zien als we Virtual Earth en SQL Server 2008 aan elkaar gaan verbinden zie codevoorbeeld 5.

Let hier ook op de functie STPolyFromText, die hier een polygoon van een comma delimited-formaat omzet naar het native SQL Server-formaat. De query geeft alle records terug die binnen het polygoon vallen. Voor de duidelijkheid, dit hoeft niet een rechthoek te zijn, maar mag bijvoorbeeld ook een driehoek of vijfhoek zijn. Om zometeen op een eenvoudige manier plaatsen te kunnen selecteren die zich binnen een polygoon bevinden, maken we een stored procedure die we kunnen aanroepen met als parameter de polygoon. De resultset bestaat hier uit de plaatsen die binnen de polygoon vallen, zie codevoorbeeld 6.

Knoop het allemaal aan elkaar

In afbeelding 4 wordt weergegeven hoe we de data die in de database zijn opgeslagen met behulp van Microsoft Virtual

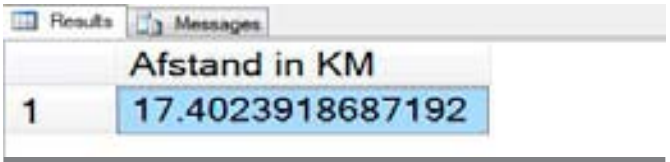
```
   SELECT Naam, Inwoners, Locatie.Lat AS Lat, Locatie.Long AS Lng
   FROM Woonplaats
   WHERE
   Locatie.STIntersects(geography::STPolyFromText(@area, 4326)) = 1
```

Codevoorbeeld 5.

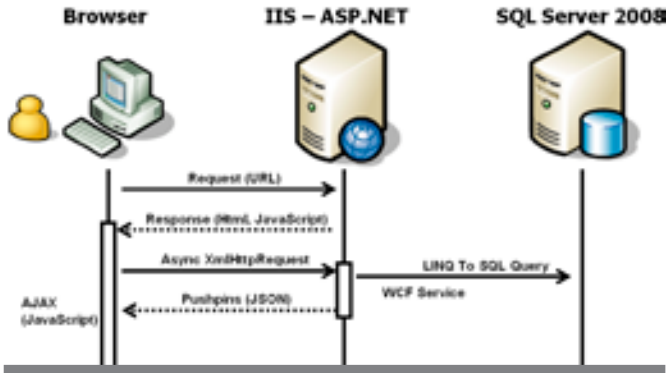
```
CREATE PROCEDURE dbo.GetWoonplaatsenInArea
   @area varchar(1000)
AS
BEGIN
   SET NOCOUNT ON

   SELECT  ID, Naam, Inwoners, Lat, Long
   FROM dbo.Woonplaats
   WHERELocatie.STIntersects(geography::STPolyFromText(@area, 4326)) = 1
END
```

Codevoorbeeld 6.



Afbeelding 2. De afstand in km tussen Amsterdam en Haarlem



Afbeelding 4. WCF-service

Earth op onze eigen ASP.NET-pagina gaan tonen.
Om de data vanuit de database te halen en op een Virtual Earth-kaartje te tekenen is een WCF-service gemaakt die in het IIS-process draait. De declarative van deze service is te zien in codevoorbeeld 7. Het namespace-attribuut zorgt ervoor dat er straks vanuit javascript met de naam MicrosoftNederland naar gerefereerd kan worden. Aan de VEService-klasse voeren we aantal methods toe. De eerste is er een om alle plaatsen uit de database op te halen. De tweede haalt de inwoners op uit een bepaald gebied. In codevoorbeeld 8 is de declaratie zien. Het attribuut OperationsContract zorgt er voor dat de functie extern aanroepbaar is. In de functie maken we gebruik van een LINQ-expressie die een query naar de database verstuurt. De resultset wordt omgezet naar een array van Woonplaats-objecten.

```
[ServiceContract(Namespace = "MicrosoftNederland")]
[AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
public class VEService
{
   ...
}
```

Codevoorbeeld 7.

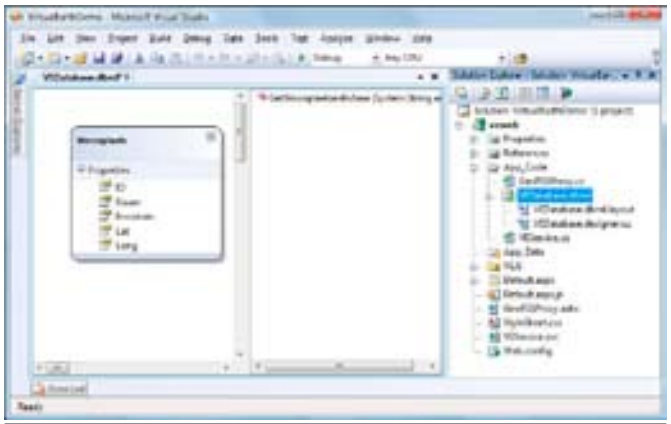
```
[OperationContract]
public Woonplaats[] GetAllWoonplaatsen()
{
   using (var db = new VEDatabaseDataContext())
   {

      var q = from wpl in db.Woonplaats
              select wpl;

      return q.ToArray();
   }
}

[OperationContract]
public Woonplaats[] GetWoonplaatsenInArea(string area)
{
   using (var db = new VEDatabaseDataContext())
   {
      return db.GetWoonplaatsenInArea(area).ToArray();
   }
}
```

Codevoorbeeld 8.



Afbeelding 5. Een 'LINQ to SQL Classes'-projectitem is toegevoegd aan de ASP.NET-applicatie/website

LINQ to SQL

LINQ to SQL wordt gebruikt om eenvoudig met de SQL Server 2008-database te communiceren. Hiervoor is aan de ASP.NET-applicatie/website een 'LINQ to SQL Classes'-projectitem toegevoegd met de naam VEDatabase.dbml. De Woonplaatstabel en de GetWoonplaatsInArea stored-procedure zijn vanuit de Server Explorer via drag and drop toegevoegd aan de VEDatabase.dbml, zie afbeelding 5. In de Designer is de Serialization Mode-property van de DataContext op Unidirectional gezet, waardoor de Woonplaats-entity class voorzien is van de DataContract- en DataMember-attributes. Hierdoor is deze class te gebruiken in de WCF-service als return value. Het Locatieveld (spatial datatypen) is niet in de Woonplaatsdefinitie opgenomen, aangezien LINQ to SQL niet met CLR-dataypes kan omgaan. Vandaar dat we gebruikgemaakt hebben van de Lat- en Long-calculated fields.

```
<body>
  <form id="form1" runat="server">
  <h1>
    Building spatial applications with VE and SQL2008</h1>
  <div>
    <input id="ButtonAll" type="button" value="Alle Woonplaatsen"
        onclick="return ButtonAll_onclick()" />
    <input id="ButtonInAea" type="button" value="In zichtveld"
        onclick="return ButtonInArea_onclick()" />
    <h3>
      Aantal = <span id="aantal"></span>
    </h3>
  </div>
  <div id='myMap' style="position: relative; width: 1000px;
      height:500px;">
  </div>
  <asp:ScriptManager ID="ScriptManager1" runat="server"
      LoadScriptsBeforeUI="false">
    <Scripts>
      <asp:ScriptReference
        Path="http://dev.virtualearth.net/mapcontrol/mapcontrol.
          ashx?v=6" />
      <asp:ScriptReference Path="~/Default.aspx.js" />
    </Scripts>
    <Services>
      <asp:ServiceReference Path="~/VEService.svc" />
    </Services>
  </asp:ScriptManager>
  </form>
</body>
```

Codevoorbeeld 9.

De user-interface

De user-interface zit in ASP.NET-pagina default.aspx. Deze pagina maakt gebruik van client-side Ajax voor de communicatie met de WCF-service. Er zijn een paar interessante stukjes html in te vinden. De <asp:ScriptReference > tag bevat een referentie naar het Virtual Earth-control. Door deze referentie komt de volledige Virtual Earth-functionaliteit tot onze beschikking. In de tag met de id 'myMap' bepalen we waar en hoe groot de kaart getoond gaat worden, zie codevoorbeeld 9.
Door de <asp:ServiceReference> tag naar de WCF-service wordt er onder water een javascriptproxy gegenereerd die de asynchrone communicatie verzorgt tussen de browser en de WCF-service. Ajax gebruikt hiervoor JSON in plaats van xml. Dit verhoogt de snelheid aangezien JSON minder overhead heeft. Het default.aspx.js-bestand bevat de Ajax-javascriptcode. Bovenin het script is een aantal references naar javascript-files en de WCF-service opgenomen. Deze referenties verzorgen Intellisense-support in Visual Studio 2008. De VEJS-referentie maakt het gebruik van de Virtual Earth SDK veel eenvoudiger, zie codevoorbeeld 10. Meer informatie hierover is te vinden op http://www.codeplex.com/vejs.

In de pageLoad-functie wordt het Virual Earth-control geïnitialiseerd. De button...onClick()-functies roepen met behulp van de gegenereerde proxies asynchroon de WCF-service aan.

```
/// <reference name="MicrosoftAjax.js" />
/// <reference path="VEJS/VeJavaScriptIntellisenseHelper.js" />
/// <reference path="VEService.svc" />

var map = new VEMap('myMap');

function pageLoad()
{
  map.LoadMap(new VELatLong(51.920807092250776, 4.481327533721924),
            7 ,'h' ,false, VEMapMode.Mode2D);
}

// haal alle woonplaatsen uit de database
function ButtonAll_onclick()
{
  resetMap();
  MicrosoftNederland.VEService.GetAllWoonplaatsen(showResult);
}

// haal alle woonplaatsen uit de database binnen een gebied.
function ButtonInArea_onclick()
{
  MicrosoftNederland.VEService.GetWoonplaatsenInArea(GetArea(),
    showResult);
}

// teken de Woonplaatsen op de kaart
function showResult(lijst) {
  map.Clear();
  $get("aantal").innerHTML = lijst.length;
  Array.forEach(lijst, function(pp) {
    var shape = new VEShape(VEShapeType.Pushpin,
    new VELatLong(pp.Lat, pp.Long));
    shape.SetTitle(pp.Title);
    shape.SetDescription(pp.Description);
    map.AddShape(shape);
    });
}
```

Codevoorbeeld 10.

Afbeelding 6. Per array-element wordt een VEShape-object geïn stantieerd en aan de kaart toegevoegd

Het resultaat (retourwaarde) van deze WCF-servicefuncties wordt door de showResult-functie verwerkt. De showResult-functie ontvangt een array met woonplaatsobjecten. Hierna wordt het mapobject leeggemaakt met de methode clear(), en wordt er over alle elementen een iteratie gedaan. Per array-element wordt een VEShape-object geïnstantieerd en aan de kaart toegevoegd. Her resultaat is te zien in afbeelding 6.

Met Virtual Earth en SQL Server 2008 zijn snel krachtige applicaties (of mashups) te bouwen die data op een kaart tonen. De bij dit artikel meegeleverde code bevat nog een aantal andere voorbeelden (onder andere het tekenen van polygonen en het consumeren van een RSS-feed).

Paul van Wingerden is Developer Platform Enthousiasmist voor ISV's bij Microsoft Nederland in de DPE-groep. Hij is via e-mail te bereiken op paulvanw@microsoft.com.

Referenties

AJAX: http://msdn2.microsoft.com/en-us/asp.net/default.aspx
ASP.NET : http://www.asp.net
Intellisense voor Virtual Earth: http://www.codeplex.com/vejs
JSON: http://en.wikipedia.org/wiki/JSON
Virtual Earth SDK: http://dev.live.com/virtualearth/sdk/
WCF: http://msdn2.microsoft.com/en-us/library/ms735119.aspx
Wikipedia: http://nl.wikipedia.org/wiki/Vorm_van_de_aarde