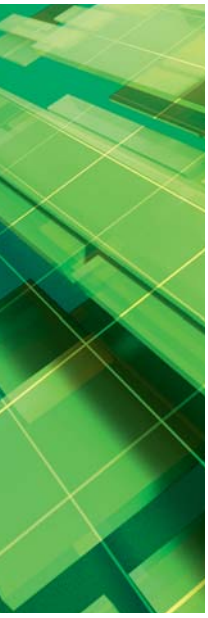


Performanceverbetering door combinatie verschillende niveaus

Optimaliseren met Multi Dimensional Expressions

Bas Stemerdink en Oscar Zonneveld

Multi Dimensional Expressions (MDX) is een nieuwe querytaal waarmee multidimensionale datastructuren, zoals een SQL Server Analysis Services (SSAS) database bevraagd kunnen worden. Dit multidimensionaal karakter maakt het bevragen van een SQL Server Analysis Services database, en de daarbij behorende cubes, door middel van MDX lastig.

Een belangrijk onderdeel van het bevragen van een cube is de tijd die SSAS nodig heeft voor het beantwoorden van de vraag (de responstijd). Om tot een acceptabele responstijd te komen is het van belang om de omgeving te tunen. Dit artikel gaat in detail in op het optimaliseren van MDX statements.
Centraal staat een case waarbij gegevens op verschillende niveaus moeten worden geaggregeerd. Bij de aggregatie moet rekening worden gehouden met het resultaat van het onderliggende niveau.

Zoals eerder gesteld is MDX geen eenvoudige taal. Vooral het multidimensionale karakter maakt het lastig om deze taal toe te passen en te doorgronden. Het optimaliseren van MDX vraagt nog meer diepgaande kennis van het gedrag van de Analysis Services engine en het gebruik van de taal MDX. Om je mee te nemen in het proces van performance-optimalisatie van MDX, is een case gedefinieerd. Deze case beschrijft het meten van doorlooptijden van Support Calls. Het probleem bij deze case is het aggregeren van gegevens op verschillende niveaus waarbij het onderliggende niveau een rol speelt. Als eerste wordt de case beschreven, waarna wordt ingegaan op de probleemstelling, om uiteindelijk te komen tot de oplossing van het performance-probleem.

Case

Bij een organisatie wordt een supportsysteem ingezet. Dit supportsysteem heeft als doel om onder andere calls van gebruikers te registreren voor diverse klanten. Wanneer een call wordt geregistreerd, wordt deze gekoppeld aan een bepaalde oplostijd. Deze oplostijd is gebaseerd op een eerder overeengekomen SLA. De beheerorganisatie heeft de verplichting om de calls zo snel mogelijk op te lossen. Voor het oplosproces is een workflow gedefinieerd. Deze workflow zorgt er voor dat de calls op een standaard manier worden afgehandeld en dat bij elk stadium een aantal zaken wordt geregistreerd, waaronder de doorlooptijd. Per klant en contract kunnen er verschillende afspraken worden gemaakt die moeten worden meegenomen in de berekeningen. Bovenstaande omgeving is een goede bron voor informatievoorziening aangaande de kwaliteit van support en het meten van het oplossend vermogen. Of, hoe lang duurt het voordat een call daadwerkelijk wordt opgelost.
Voor de implementatie van bovenstaande case is gekozen voor de inzet van Microsoft SQL Server Analysis Services. Dit geeft de mogelijkheid om analyses uit te voeren over de calls en deze af te kunnen zetten tegen de afgesloten SLA's (hoe performen we ten opzichte van wat we beloofd hebben).

Probleemstelling

Zoals aangegeven gaat dit artikel over performance-optimalisatie van MDX. Tijdens de implementatie van bovenstaande case is gebleken dat er een 'performance killer' aanwezig is. Kort samengevat is de probleemstelling: *Aggregeer gegevens op verschillende niveaus, waarbij het resultaat van het onderliggende niveau een rol speelt.* Om dit concreet te maken, wordt de case verder gedetailleerd.
Voor het laten zien van doorlooptijden behorende bij SLA's hebben we een dimensie nodig. De Dimensie Service Level Agreement heeft een hiërarchie genaamd '*by Service Agreement*'. Het laagste niveau is de status van de Service Agreement. Voor het bepalen van de measure (bijvoorbeeld de gemiddelde doorlooptijd) moet deze expliciet worden berekend. De berekening op dit niveau is een deling van de meetwaarde gedeeld door het totaal van het aantal unieke calls.

```
[Measures].[Cumulative Duration in seconds] /
                                    [Measures].[Total]
```

Het niveau boven de status van een Service Agreement betreft het SLA niveau. Voor het bepalen van bijvoorbeeld de doorlooptijd van de afhandeling van een call (binnen een SLA) hebben we het resultaat per status nodig (de afhandeling op SLA niveau is een verzameling van de doorlooptijden van de verschillende stadia). Wat het nog lastiger maakt is dat deze 'status afgehandeld' (StatusDue), voor klanten verschillend kan zijn. Zo kan het zijn dat nadat een call functioneel is afgerond (de vlag StatusDue is gezet), de call zelf nog door een aantal stadia gaat. De duur van deze acties mag dan niet worden meegeteld in het gemiddelde.

```
AVG({([Service Level Agreement].[by Service
                    Agreement].CurrentMember.Children,
       [StatusDue].&[1], [Measures].Currentmember)})
```

Het hoogste niveau van de dimensie hiërarchie is het Service Agreement zelf. Voor de Service Agreement geldt eigenlijk dat hier ook het gedrag mag worden uitgevoerd als het op het Status niveau is gedefinieerd. Echter, hierbij worden de resultaten gebruikt die op het SLA niveau zijn berekend.

```
[Measures].[Cumulative Duration in seconds] /
                                    [Measures].[Total]
```

De hiërarchie by Service Agreement is opgebouwd volgens het gedrag zoals afgebeeld in de tabel in afbeelding 1.
De volgende paragraaf gaat gedetailleerd in op het oplossen van deze betreffende probleemstelling. Bij de oplossing speelt het SCOPE statement een belangrijke rol. Dit wordt dan ook voorafgaande aan de oplossing uitgelegd. Daarna worden enkele tools beschreven die een rol hebben gespeeld bij de optimalisatie. Als laatste wordt de daadwerkelijke oplossing beschreven.

MDX SCOPE Statement

Net als bij SQL, geldt ook voor MDX dat je de beste performance behaalt door de dataset zo klein mogelijk te maken. Het verkleinen van de dataset wordt gedaan door het opbouwen van een subset van de totale dataset. Deze subset (subcube) verkrijg je door het toepassen van het SCOPE statement.
Het SCOPE statement zorgt er voor dat MDX statements die worden uitgevoerd binnen de context van de SCOPE ook daadwerkelijk worden uitgevoerd binnen deze context (de impliciete scope van een MDX statement is de totale cube).

```
SCOPE(Subcube_Expression)
SELECT  [<axis_specification>]
        [, <axis_specification>...]
FROM    [<cube_specification>]
[WHERE  [<slicer_specification>]]
END SCOPE
```

Wanneer de subcube door de SCOPE functie is gedefinieerd, kan aan deze subcube door middel van het *this* statement worden gerefereerd. In het onderstaande voorbeeld wordt eerst een subcube gedefinieerd door middel van het SCOPE statement. Via het *this* statement wordt vervolgens de waarde voor de opgegeven SCOPE op 1000 gezet (waarbij de SCOPE is bepaald op alle members van [Dimension].[by HierarchyName].[Attribute] en de daaraan gerelateerde meetwaarde [Measure].[Measure1]. De waarde 1000 vervangt hierbij de oorspronkelijke waarde die in de SSAS cube is vastgelegd voor de genoemde combinatie.

```
SCOPE([Dimension].[by HierarchyName].[Attribute].
                        members, [Measures].[Measure1]);
this=1000;
END SCOPE;
```

Om de performance van MDX statements goed te kunnen meten, moet je de query's altijd in een identieke situatie uitvoeren. Alleen op deze manier is de performance te vergelijken. Analysis Services werkt intern net zoals de Database Engine met een cache. Zodra er weer een query binnenkomt waarin data worden opgevraagd die zich nog in de cache bevinden, zullen deze worden gebruikt en wordt de database niet opnieuw geraadpleegd. Het voordeel hiervan is een aanzienlijk betere performance. Deze performance is echter niet constant, wat lastig is tijdens het tunen van de query. Je kunt het legen van de cache forceren voordat een query wordt uitgevoerd. We noemen een op die manier geleegde cache ook wel een 'cold cache'. Gebruik maken van een 'cold cache' is heel belangrijk bij het optimaliseren van MDX query's, omdat je anders de resultaten niet met elkaar kunt vergelijken.

In onze case wordt de query de eerste keer in 3 seconden uitgevoerd. Voeren we de query nog een keer uit, dan is de respons-

Level naam	Attribuut	Gewenst gedrag
Service Agreement	Service Agreement Hierarchy Name	`[Measures].[Cumulative Duration in seconds]/ [Measures].[Total]`
SLA	Service Level Agreement Priority Hierarchy Name	`AVG({([Service Level Agreement].[by Service Agreement].CurrentMember Children, [StatusDue].&[1], [Measures].CurrentMember)})`
Status	Service Level Agreement Status Hierarchy Name	`[Measures].[Cumulative Duration in seconds]/ [Measures].[Total]`

Afbeelding 1: Gewenste formules hiërarchie.

tijd 180 milliseconden. Analysis services heeft op basis van het eerste verzoek een cache opgebouwd. Deze wordt bij het tweede verzoek gebruikt. Het is moeilijk aan gebruikers uit te leggen dat iedere keer wanneer ze een nieuw verzoek sturen de performance zo slecht is maar daarna zeer acceptabel. Vanaf dit moment van lezen mag er vanuit worden gegaan dat alle resultaten tegen een 'cold cache' situatie worden uitgevoerd om goed te kunnen kijken of de wijziging een positief effect heeft op de performance.
De vraag is nu waarom de performance de eerste keer zo veel slechter is terwijl de taal zich juist leent voor analytische vragen op een geaggregeerd niveau. Juist omdat Analysis Services tijdens het processen van de OLAP cubes de mogelijkheid hebben om aggregaten te berekenen (en op te slaan) is OLAP vaak vele malen sneller dan een relationele database engine bij dit soort vragen.

Performance Tools

De vraag is hoe je meer inzicht kunt verkrijgen in hoe de engine dit verzoek oplost. Vaak wordt vergeten dat de SQL Profiler ook in combinatie met een Analysis Server kan worden gebruikt. Met de Profiler kan worden gekeken welke MDX statements binnenkomen op de Analysis Server maar tevens krijg je meer inzicht in hoe de engine het verzoek oplost, zie afbeelding 2.
Een andere tool die echt een 'must have' is bij het schrijven en optimaliseren van MDX is MDX Studio. Deze tool is geschreven door Mosha Pasumansky; één van de grondleggers van MDX en Analysis Services. Deze applicatie geeft naast de functionaliteit van een MDX editor ook informatie terug over hoe de engine de query heeft opgelost. Naast de basisinformatie als de doorlooptijd en het geheugengebruik wordt hier nog veel meer nuttige informatie teruggeven. Het gaat echter te ver om deze verder in detail te behandelen want hier kan een volledig artikel aan worden besteed. Microsoft heeft een document gepubliceerd dat veel inzicht geeft in de performanceproblemen bij MDX query's (*Identifying and Resolving MDX Query Performance Bottlenecks in SQL Server 2005 Analysis Services*) en de SQL Profiler en MDX studio geven veel informatie terug over hoe de engine de query oplost. Door deze informatie te combineren kan de oorzaak in veel gevallen snel worden gevonden.
Een goede eerste stap om te komen tot een betere MDX performance is de berekening van de meetwaarde eventueel op te delen in meerdere kleine stappen. Naast dat dit meer overzicht geeft heeft de engine hier meestal ook voordeel van omdat de executieplannen simpeler worden. Dit is vergelijkbaar met de query optimizer van SQL Server.

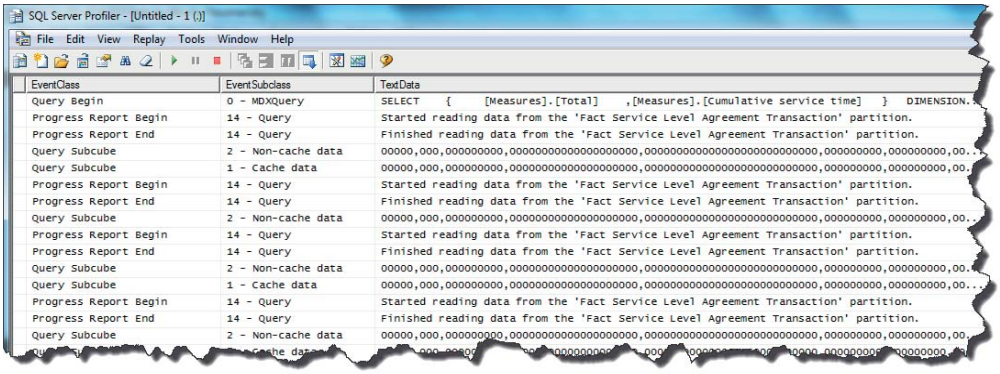
Oplossing

Het inzetten van calculated measures is een veel gebruikte oplossing om conditioneel berekeningen te laten uitvoeren. Op basis van een test (bijvoorbeeld op welk niveau bevindt de member zich) kunnen verschillende berekeningen worden uitgevoerd. In dit specifieke geval is dit niet toereikend, aangezien we op het hoogste niveau van de hiërarchie het resultaat van het onderliggende niveau nodig hebben om op een juiste manier de berekening uit te voeren.
In dit geval is de berekening niet echt op te delen, maar er is wel een extra conditie waarop iedere keer moet worden gefilterd; namelijk die van StatusDue = 1. Als deze conditie wordt weggelaten is het resultaat natuurlijk foutief maar geeft het wel een beeld of deze filter mogelijk de oorzaak is. Na verwijdering van de filter wordt de query opnieuw afgevuurd op een 'cold cache' en levert binnen 180 milliseconden het verwachte foutieve resultaat. Hiermee wordt het idee bevestigd dat dit de richting is waar we de oplossing moeten zoeken. Het filter bepaalt echter dynamisch welke rij er aan de gestelde conditie voldoet. Er moet worden gezocht naar een andere manier waarop deze kan worden toegepast, zonder dat de engine terugvalt in de *cell-by-cell* berekening.
Na op diverse manieren te hebben geprobeerd het filter onder alle omstandigheden correct en met een stabiele performance te laten berekenen is besloten om een deel op SQL niveau op te lossen. De SQL view waarin de meetwaardes worden teruggeven richting Analysis Services is uitgebreid met een extra kolom. De kolom *Cumulative_DurationInSeconds_StatusDue* geeft alleen de cumulatieve doorlooptijd terug als het filter status geldt, in alle andere gevallen wordt de waarde 0 teruggegeven.

Doordat deze kolom alleen een cumulatieve doorlooptijd heeft wanneer aan deze conditie wordt voldaan, kunnen de waarden binnen de OLAP cube gewoon worden geaggregeerd met een standaard SUM aggregatiefunctie en hoeft er niet meer op MDX niveau te worden gefilterd. We hebben nu verschillende meetwaarden tot onze beschikking, namelijk de cumulatieve doorlooptijd tot de betreffende status en onze nieuwe meetwaarde die de cumulatieve doorlooptijd bevat tot de status die mag worden gemeten volgens de afspraak. Echter, het doel is om de gebruikers één meetwaarde aan te bieden die afhankelijk van het niveau binnen de dimensie de juiste waarde teruggeeft. Om dit te realiseren wordt het eerder beschreven SCOPE statement gebruikt.

```
SCOPE([Service Level Agreement].
                [by Service Agreement].[SLA].members,
          [Measures].[Cumulative_DurationInSeconds]);
this=[Measures].[Cumulative_DurationInSeconds_
                        StatusDue]/[Measures].[Total];
END SCOPE;
```

We openen het SCOPE statement en tussen de haken kunnen de condities worden opgegeven waarvoor de scope geldt, in dit geval voor de combinatie van alle members op het niveau van het SLA attribuut binnen de hiërarchie '*by Service Agreement*' van de dimensie Service Level Agreement en de basis meetwaarde *Cumulative_DurationInSeconds*. Met het *this* statement wordt de waarde van de meetwaarde vervangen door de formule die



Afbeelding2: SQL Profiler.

hierachter is vermeld. Hier gebruiken we de nieuwe meetwaarde. Het resultaat van deze SCOPE is dat als het desbetreffende niveau wordt opgevraagd, niet de standaard SUM van alle statussen wordt getoond maar het resultaat van de formule die in de SCOPE is opgenomen.
Om de gebruikersvriendelijkheid verder te verhogen kunnen we een calculated measure introduceren. Calculated measures zijn gebaseerd op andere meetwaarden binnen de kubus. Het verschil met een standaard meetwaarde is dat alleen de definitie wordt opgeslagen en deze ook geen ruimte kost. De consequentie daarvan is dat ze runtime worden berekend. De nieuwe calculated measure krijgt een gebruikersvriendelijke naam en opmaak. Samengevat komt dit neer op de volgende code:

```
CREATE MEMBER CURRENTCUBE.[MEASURES].
                [Cumulative Duration in seconds] AS
CSTR(INT((measures.[Cumulative_DurationInSeconds]/
                        [Measures].[Total])/86400))
+ ":"
+ FORMAT
(
        CDATE(
        (([measures].[Cumulative_DurationInSeconds]/
                        [Measures].[Total])/86400)
        - INT(([measures].
        [Cumulative_DurationInSeconds]/
                        [Measures].[Total])/86400)
        )
        , "HH:mm:ss"
)
, NON_EMPTY_BEHAVIOR =
                { [Cumulative_DurationInSeconds] }
, VISIBLE = 1;
```

Nu de berekening goed is kan de OLAP cube worden 'opgeschoond', aangezien er op dit moment verschillende meetwaarden zijn gedefinieerd die alleen intern nodig zijn om tot het eindresultaat te komen en overbodig zijn voor de gebruikers. Alleen de meetwaarde uit tabel 3 hoeft zichtbaar te zijn. Bij de eerder gebruikte meetwaardes *Cumulative_DurationInSeconds* en *Cumulative_DurationInSeconds_StatusDue* kan daarom de *Visible* eigenschap op 0 worden gezet zodat ze niet worden getoond in de client tools.
De resultaten van deze oplossing worden nu in 250 milliseconden teruggegeven in vergelijking met de oorspronkelijke 3 seconden.

Conclusie

MDX is een lastige taal. Het optimaliseren van MDX is mogelijk nog lastiger. In dit artikel is specifiek ingegaan op de optimalisatie van MDX. Hierbij is inzicht gegeven in het proces om tot een oplossing te komen en tools ter ondersteuning. De beschreven case en de daarbij behorende oplossing laten zien hoe een specifieke probleemstelling opgelost kan worden.
Zoals uit de case blijkt is het niet altijd mogelijk om de optimale performance te bereiken binnen de context van de taal of oplossing waarmee men op dat moment werkt. Microsoft SQL Server Analysis Services maakt het ook mogelijk om optimalisaties op verschillende niveaus door te voeren. In bovenstaande case is dan ook gebruik gemaakt van optimalisatie op zowel SQL als MDX niveau. Het SCOPE statement heeft het mogelijk gemaakt om de oorspronkelijke waarden in de originele SSAS Cube te vervangen. Dit geeft de mogelijkheid om voor gedefinieerde combinaties (lees de context van de SCOPE) uitzonderingen te implementeren. Deze techniek is gebruikt om bij een hiërarchie het resultaat van een onderliggend niveau te gebruiken. Het gebruik van het SCOPE statement in combinatie met de aanpassingen op het SQL niveau hebben een performancewinst opgeleverd van ruim 90 procent. Om te komen tot deze aanzienlijke performanceverbetering is veel gebruik gemaakt van ondersteunende tooling zoals MDX Studio en SQL profiler. Deze tools hebben inzicht gegeven in het gedrag van de engine en de impact van aanpassingen op de performance.

Literatuur

- *Identifying and Resolving MDX Query Performance Bottlenecks in SQL Server 2005 Analysis Services (http://goo.gl/hWO6X).*
- *MDX Studio (http://goo.gl/brp39).*

Bas Stemerdink (basst@infosupport.com) en **Oscar Zonneveld** (oscarz@infosupport.com) zijn beide als BI Architect verbonden aan het Competence Center Business Intelligence en datawarehousing van Info Support.