‘Denali’, nieuwste SQL Server-aanwinst

De belangrijkste nieuwste features op een rij

Paul van Wingerden

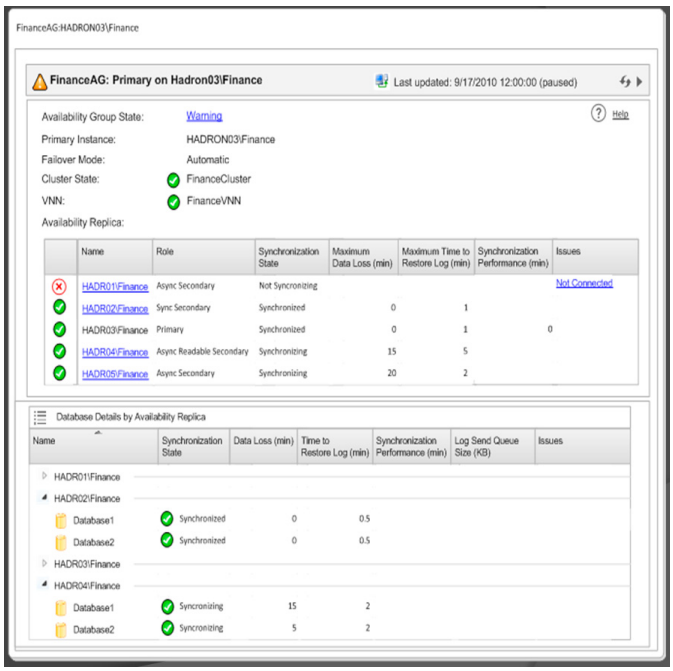
Tijdens PASS in Seattle heeft Ted Kummert, general manager van het SQL Server-team, de eerste CTP van SQL Server ‘*codename Denali*’ gereleased. Samen met de release zijn er nog een aantal additionele producten gepresenteerd zoals Parallel DataWarehouse, een Beta met database developer tools genaamd ‘Juneau’, en ‘Atlanta’ een cloud service die helpt met het configureren van SQL Server.

Er zijn drie grote thema’s waaronder de nieuwe functionaliteiten kunnen worden gegroepeerd: Mission critical, Business Intelligence en Developer features.

Mission critical

In het mission critical segment zitten belangrijke onderdelen om in een moderne omgeving een high available en scalable applicatie te bouwen. Een belangrijk onderdeel hiervan is dat alle zaken die met High avaliability en disaster recovery te maken hebben vanuit een tool te configureren zijn. Alle onderdelen zijn in de *SQL Server Management Studio* gegroepeerd (Clustering, Mirroring, Logshipping etc.).

Nieuw hierbij is dat er applicatiegroepen kunnen worden gedefinieerd die samen een failover doen als dit nodig is. Het doel is om de eventuele downtime te minimaliseren. Ook zijn er verbeteringen die de gebruikte resources beter benutten, zoals het geval is met database mirroring waarbij er ‘*readable secondaries*’ zijn. Hierdoor kunnen die worden ingezet voor bijvoorbeeld reportingdoeleinden en staat de hardware niet te wachten tot het moment dat er een failure is. Het onderwerp Mission Critical is zeer uitgebreid en komt wellicht in een volgend artikel uitgebreider aan de orde.



Business Intelligence

Er zijn een groot aantal nieuwe zaken aan het BI-front, maar de twee belangrijkste die in het oog springen zijn:

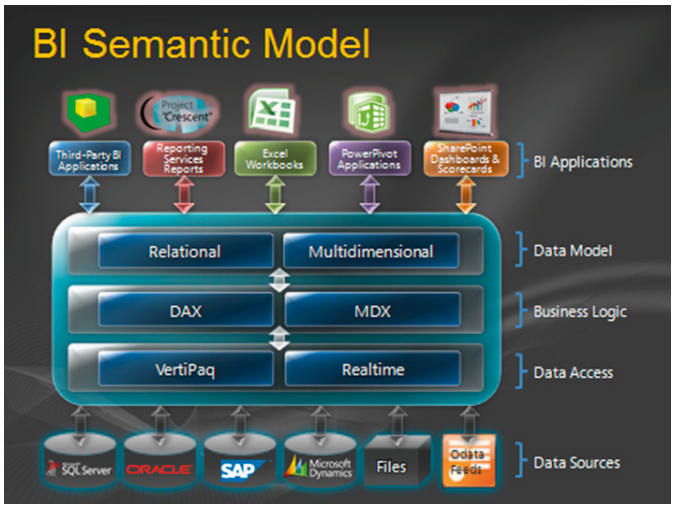
- Een ad-hoc reporting- en datavisualisatie experience. Dit staat voorlopig bekend onder de codenaam ‘*Crescent*’. Een van de dingen die in het oog springen zijn de ‘*animaties*’ die met data kunnen worden gemaakt. Hiermee ontstaan een soort 4D-grafieken met tijd als de vierde dimensie
- Een nieuw Business Intelligence Semantic Model (BISM) in Analysis Services dat de motor is van ‘Crescent’ en andere BI front ends zoals Excel, Reporting Services en Sharepoint. Simpel gesteld is het BI Semantic Model een relationeel model met tabellen en relaties. Het heeft BI artifacten zoals hiërarchieën en KPIs. Het brengt de mogelijkheden van SMDL modellen samen met een veel van de eigenschappen van het UDM. Het is goed om te realiseren dat het gene vervanger is van het UDM.

Het BISM ziet eruit als een 3-lagen-model met hierin de volgende drie lagen:

- Een Datamodel dat zichtbaar is voor de client. Dit heeft zowel relationele als mulitidimensionale interfaces. Zowel Excel als Crescent kan hiervan gebruik maken met MDX of DAX (Data Analysis Expression) queries.
- Een Business Logic laaag waarin de intelligentie van het model verwerkt is. Dit model wordt gedefinieerd door middel van DAX of MDX.

Een Data Access laag die de data van verschillende bronnen (rela-

tioneel, applicaties, flat files, etc.) combineert. De data kan zowel cached of realtime verwerkt worden. In het geval dat er voor cached gekozen wordt komt de data uit de Vertipaq modele (die zorgt voor zeer grote compressie van de data, zowel in memory als op disk). Hierbij is er geen noodzaak voor indexen het opslaan van aggregates of query tuning. Als er voor realtime verwerking gekozen wordt gaan de queries direct naar de onderliggende data sources. Hierdoor verwerkt het source systeem de data en hoeft er geen kopie (extract) van de data geladen te worden in de Vertipaq engine.



Developer features

Er zijn een flink aantal nieuwe features in SQL Server codename 'Denali'. Er is in dit artikel geen ruimte om deze allemaal stuk voor stuk uitputtend te beschrijven. Hieronder volgt een selectie van een aantal in het oog springende nieuwe features.

Ondersteuning van sequences

Met enige regelmaat keert de vraag terug of SQL Server nu ook sequences ondersteunt zoals sommige andere databases ook doen. Met de release van SQL Server codename *'Denali'* is het nu zover! Maar wat is nu een zogenaamde sequence?

Een sequence object is een object dat een *'sequence'* van numerieke waarden genereert. De waarden die worden gegenereerd kunnen beïnvloed worden tijdens het creëren van de sequence. Een sequence werkt hetzelfde als een Identity column. Het grote verschil is dat een sequence niet beperkt is tot een enkele tabel, maar databasebreed gebruikt kan worden. De volgorde van de sequence kan oplopend of aflopend zijn. Net zoals bij de identity is het interval te beïnvloeden. Een sequence kan waarden opnieuw gebruiken (cycling) als dat ingesteld wordt. Een applicatie kan naar een sequence object refereren en de waarden die erdoor gegenereerd worden gebruiken om als sleutels te gebruiken. Om een Sequence aan te maken gebruik je het CREATE SEQUENCE statement.

Dit statement ziet er als volgt uit:

```
CREATE SEQUENCE [schema_name . ] sequence_name
        [ <sequence_property_assignment> [ ,…n ] ]
    [ ; ]

<sequence_property_assignment>::=
{
    [ AS { built_in_integer_type | user-defined_integer_type } ]
        | START WITH <constant>
            | INCREMENT BY <constant>
            | { MINVALUE <constant> | NO MINVALUE }
            | { MAXVALUE <constant> | NO MAXVALUE }
            | { CYCLE | NO CYCLE }
            | { CACHE [<constant> ] | NO CACHE }
}
```

Wanneer is het nou handig om een sequence te gebruiken?

- In het geval dat een applicatie één serie van nummers moet delen tussen verschillende tabellen of meerdere kolommen in een tabel;
- Het handig is als een sleutel reeds bekend is, voordat een record in de database toegevoegd wordt;
- Een applicatie vereist dat een serie van nummers hergebruikt kan worden, nadat een maximum (of minimum) is bereikt. Dit kan het geval zijn als je een nummer van 1 tot 10 laat lopen en daarna weer bij 1 wilt beginnen;
- In het geval een applicatie een range van sequence nummers in een keer wil alloceren. Er kan dan bijvoorbeeld een blok van twintig nummers in een keer worden gereserveerd. Die kunnen dan gebruikt worden zonder dat andere gebruikers van de database gaten kunnen maken in de gereserveerde range. Dit kan bijvooorbeeld wel met een identity column gebeuren als er op hetzelfde moment een andere gebruiker records toevoegt. Gebruik om dit te bewerkstelligen de stored procedure *get_sequence_range*;
- En in de laatste plaats als je een applicatie van een ander databaseplatform migreert naar SQL Server. Er zijn dan zo min mogelijk aanpassingen in de code noodzakelijk.

Een sequence is veel flexibeler dan het gebruik van een indentity column. Stel dat je de stapgrootte wilt veranderen of het maximum van de sequence. Dit kan zonder dat de onderliggende tabellen die gebruik maken van de sequence moeten worden veranderd. Dit is in het geval van zeer grote tabellen, of tabellen met veel relaties een erg vervelend en tijdrovende klus om te programmeren.

Verbeterde Paging met OFFSET en FETCH

Veel ontwikkelaars die wel eens een (web)applicatie hebben geschreven die heel veel data ophaalt zijn tegen het probleem aangelopen hoe je op een efficiënte manier het aantal records kunt limiteren en tegelijkertijd een volgende of vorige resultset kunt ophalen. Met andere woorden: hoe kan je paging implementeren? Hiervoor zijn verschillende oplossingen mogelijk waarop in dit artikel niet allemaal ingegaan zal worden. Een van de meest gebruikte is die waarbij de ROW_NUM() functie gebruikt wordt. Dit ziet er als volgt uit:

```
WITH a AS (
   SELECT ROW_NUMBER() OVER(ORDER BY Name) AS RowNum
   ...
   FROM
   )
   SELECT ... FROM a WHERE AS BETWEEN 21 AND 40
   ORDER BY RowNum;
```

Er wordt hier een Common Table Expression gebruikt. Met SQL Server codename *'Denali'* kan dit op een veel elegantere manier woden gedaan. Met de nieuwe OFFSET en FETCH NEXT clausules van het SELECT statement ziet dit er als volgt uit:

```
SELECT ... ORDER BY ..
OFFSET 10 ROWS
FETCH NEXT 20 ROWS ONLY
```

Er zijn een aantal voorwaarden waaraan voldaan moet worden om gebruik te kunnen maken van OFFSET and FETCH.

De eerste is dat de volgorde van de resultset gespecificeerd wordt met een ORDER BY clausule. Zonder dit is er geen gedefinieerde volgorde en werkt dit mechanisme niet. Verder moet de ORDER BY clausule die door de query wordt gebruikt unieke waarden teruggeven (of een combinatie van kolommen in de ORDER BY clausule).

Het gebruik van OFFSET en FETCH om een paging mechanisme te implementeren vereist dat de query voor iedere *'pagina'* uitgevoerd wordt met andere waarden voor de OFFSET. Het mooie aan deze implementatie is dat deze waarden als een variabele of een sub-select doorgegeven kunnen worden.

Dit ziet er in het geval van het gebruik van een expressie als volgt uit:

```
ORDER BY Achternaam ASC
  OFFSET @StartingRowNumber - 1 ROWS
  FETCH NEXT @EndingRowNumber - @StartingRowNumber + 1 ROWS ONLY
```

De variabelen *@StartingRowNumber* en *@EndingRowNumber* dienen gedeclareed te worden .

Het kan nog wat flexibeler worden gebruikt door het specificeren van een subquery:

```
ORDER BY Achternaam ASC
    OFFSET @StartingRowNumber ROWS
    FETCH NEXT (SELECT Pagesize
FROM dbo.ApplicatieInstellingen
WHERE AppSettingID = 1) ROWS ONLY;
```

Het bovenstaande voorbeeld gebruikt voor de pagesize een tabel met *'ApplicatieInstellingen'*. Die zou ook bijvoorbeeld per gebruiker aan te passen zijn.

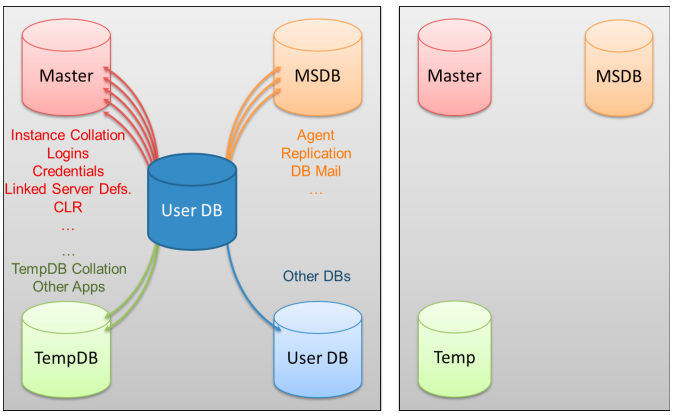
Doordat de query iedere keer wordt gedraaid, kunnen de resultaten verschillen tussen de runs als er intussen data wordt gemuteerd. Dit is een belangrijk verschil ten opzichte van het gebruik van een cursor. Als het belangrijk is dat dit niet gebeurt zijn er twee oplossingen:

- Voer de query uit met behulp van een transactie en gebruik het snapshot of serializable isolation level. Dit is alleen aan te raden voor batch-processen en niet voor user input. De transactie blijft dan immers onvoorspelbaar lang openstaan;
- Gebruik een cursor. Ook dit is niet aan te raden om dezelfde reden als hierboven beschreven.

In de praktijk zal het voor het implementeren van paging geen probleem zijn dat de resultaten kunnen verschillen als er onderliggende data wordt gemuteerd. Dit kan zelfs een voordeel zijn, doordat er verderop geen fouten ontstaan bijvoorbeeld dat een product uit voorraad is wat gekozen wordt uit een lijst, etc.

Verhoogde portabiliteit van databases.
Ongetwijfeld zijn er veel DBA's of developers die wel eens een database hebben verplaatst naar bijvoorbeeld een andere server. Een van de meest voorkomende problemen hierbij is dat er dan instellingen zijn die *'buiten'* de database om zijn geregeld. De meest bekende is ongetwijfeld de login. De login wordt in de master database gedefinieerd en moet opnieuw worden aangemaakt als de database naar een andere databaseserver (instance) wordt verplaatst. Zo zijn er nog veel meer dingen die in de master database worden geregeld zoals de collation van de instance, mogelijke definities van linked-servers, CLR procedures, credentials en nog veel meer kleinere dingen die het verplaatsen van een database *'breken'*.

In MSDB staan settings en jobs van de SQL-Agent, replicatie instellingen en oa de database mail configuratie gegevens. Omdat de tempDB de instellingen van de master overneemt bij het creëren (na iedere reboot) kunnen er bij het gebruik van temp tables en sorts ook nog van allerlei dingen anders gaan dan op de server waarvandaan de database gekopieerd is, Dit zijn dus allemaal zaken die het verplaatsen van de database net even lastiger maken dan een simpele sp_detach en sp_attach_db en het veranderen van de servernaam in de connection string.



In SQL Server codename *'Denali'* is hiervoor een oplossing die Contained Database heet. In een zogenaamde Contained database zijn alle database-instellingen en de bijbehorende metadata in de database opgeslagen en zijn er geen configuratie-afhankelijkheden op de databaseserver. Andere afhankelijkheden zijn bijvoorbeeld tabelnamen in gerelateerde databases (3 part names) of system views. Gebruikers (users) kunnen een connectie opzetten met de database zonder eerst op de database engine door middel van een login te authenticeren. Door deze ontkoppeling van user en login kan een database gemakkelijk verplaatst worden.

Om vast te stellen of er sprake is van uncontained objecten is er een dynamic management view (DMV) beschikbaar genaamd sys.dm_db_uncontained_entities. Door deze op te vragen kan de ontwikkelaar een overzicht krijgen welke objecten en of feaures moeten worden aangepast om er een volledige contained database van te maken. Op de volgende MSDN link is veel uitgebreidere informatie beschikbaar over contained databases: *http://bit.ly/cB5LH1*

Filetable
Een groot deel van data die in bedrijven omgaat is ongestructureerde of semi-gestructureerde data. Deze data staat vaak in losse bestanden in een foldersysteem (NTFS) en wordt benaderd door middel van Win32 APIs en niet door een relationele database. Een begin van een hybride oplossing is in SQL2008

geïntroduceerd in de vorm van FILESTREAMS. Dit is een mengvorm van SQL Server tables en bestanden die door SQL Server worden beheerd. Doordat SQL Server het beheer van de bestanden doet, is het mogelijk om met behoud van ACID properties zowel de tabellen als de bestanden te benaderen. De security wordt door SQL Server geregeld en kan uitgebreider zijn dan wat mogelijk is als het gaat om permissies/security met NTFS. Immers NTFS weet niets van de business rules die in de database geprogrammeerd zijn.
Een van de redenen dat de bestanden in een filesysteem worden geplaatst en niet in een blob is dat bij het benaderen van de bestanden er geen gebruik wordt gemaakt van de data cache van SQL Server, maar dat er een pointer naar de file wordt gebruikt waardoor er gewone WIN32 APIs kunnen worden gebruikt (Fileread, etc). Hierdoor kan er ook data gestreamd worden, iets wat anders veel moeilijker of helemaal niet mogelijk is.

In SQL Server codename '*Denali*' zijn er verbeteringen doorgevoerd aan het FILESTREAM type. Deze hebben als doel de barrière die er bestaat tussen data in fileservers en data in tabellen te verminderen. Door het Filetype type te activeren wordt er een Windows Share aangemaakt die te gebruiken is door Windows applicaties zonder de (kleine) aanpassingen die er in SQL2008R2 en eerder nodig waren. Deze aanpassingen bestonden uit het ophalen van een safehandle via SQL Server waarmee de bestanden konden worden benaderd. In de Denali-release kan dit zonder aanpassingen van de applicatie waardoor iedere Win32 applicatie hiervan gebruik kan maken zonder de SQL specifieke zaken van de safehandle te implementeren. De in SQL Server aanwezige Full tekst search kan ook deze filetables indexeren . Hierdoor kan er een query worden geschreven die gecombineerd wordt met andere kolommen in de database.

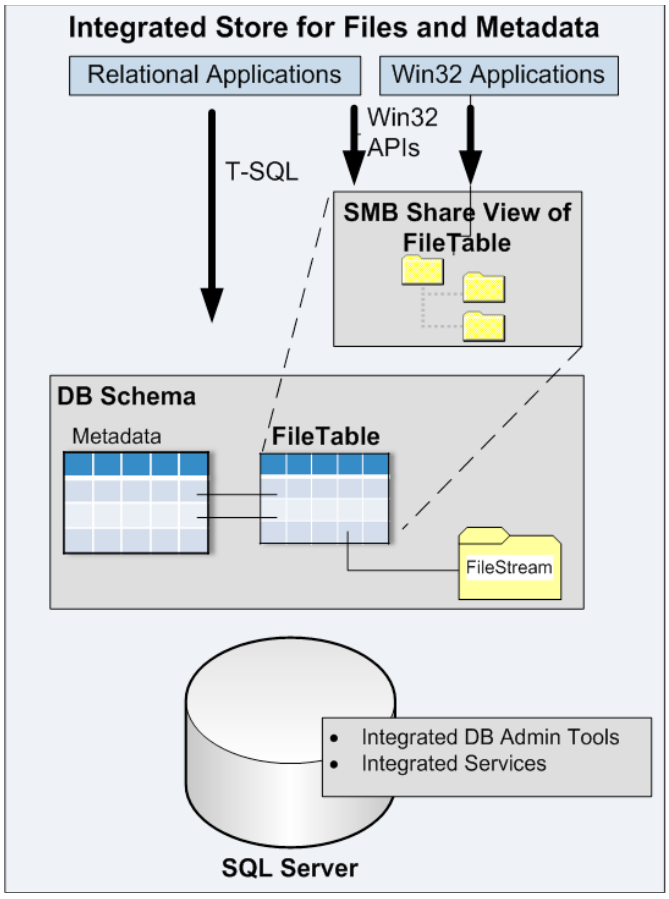
Een filetable is geïmplementeerd als een vaste tabelstructuur waarin (o.a) de filestream zelf, de hierarchy waar de file zich op disk bevindt en allerlei attributen van de file (last edited, filesize etc.) Iedere Filetable is een hierarchy van folders en files waarbinnen iedere rij naar een file verwijst.

Het mooie van deze implementatie ten opzichte van Filestreams in SQL Server 2008 is dat het volledig transparant is. Als er via de commandline of via explorer bestanden naar de share worden gekopieerd die door de Filetable worden beheerd, dan zijn die bestanden direct zonder speciale handelingen zichtbaar in de Filetable. Dit geldt uiteraard ook als er bijvoorbeeld door een Office-applicatie een bestand wordt gewijzigd: de file attributes zijn direct in T-SQL zichtbaar zonder tussenkomst van code. Bewerkingen op het Filetable type zijn te combineren met andere T-SQL statements. Ook worden alle SQL Server adminstration tools en backup/restore, replication logshipping en mirroring ondersteunt.

Dit zijn slechts een aantal van de meest in het oog springende nieuwe developer features die in deze beperkte ruimte passen.

Links

SQL Server teamblog: http://blogs.technet.com/b/dataplatforminsider/
Download Denali: http://www.microsoft.com/sqlserver/en/us/product-info/future-editions.aspx
MSDN: http://msdn.microsoft.com/en-us/library/bb418432(SQL.10).aspx



Paul van Wingerden, is Developer evangelist bij Microsoft. Zijn specialisatie: SQL Server, BI, webdevelopment, Windows Live. Paul is te breiken via email. paulvanw@microsoft.com en blogt op www.paulvanw.com.