Melchior Brandt Corstius en Robert Tusveld

Het optimaliseren van stored procedures

SQL SERVER 2005 PERFORMANCE TUNING

Vandaag de dag krijg je als ontwikkelaar steeds vaker te maken met database-benadering en vooral met het schrijven van stored procedures. Hier komt echter meer bij kijken dan het schrijven van een aantal SQL-statements onder elkaar. Dit artikel geeft inzicht in de problemen die kunnen ontstaan bij het schrijven van stored procedures, de manier waarop SQL Server met het uitvoeren ervan omgaat en geeft vervolgens oplossingen voor deze problemen.

Als ontwikkelaar heb je een aantal manieren tot je beschikking om gegevens uit een database te lezen en wijzigingen op een database te doen. Afhankelijk van de gekozen benaderingswijze blijken er echter grote verschillen te zijn in de manier waarop SQL Server met deze soorten van databasebenadering omgaat. Deze verschillen bestaan zowel op het gebied van beveiliging als op het gebied van performance. In ons vorige artikel "Alles wat je als ontwikkelaar moet weten over indexoptimalisatie" in .NET Magazine #17 beschreven we hoe je op een optimale manier indexen gebruikt in SQL Server 2005. In dit artikel geven we een overzicht van de mogelijkheden die stored procedures je bieden en leggen we de voor- en nadelen van de diverse benaderingswijzen uit. Daarnaast wordt stilgestaan bij de performanceproblemen die kunnen ontstaan bij 'slecht' geschreven stored procedures en wat je hieraan kunt doen.

Statement-executie

Om te begrijpen hoe SQL Server statements uitvoert, kijken we naar het stappenplan dat hierbij wordt doorlopen:
Allereerst wordt gekeken of het statement zich in de cache bevindt en of het al gecompileerd is. Als dit niet het geval is, wordt eerst het **Language Processing**-deel uitgevoerd: het statement zelf wordt gecontroleerd (***Parse***), waarna de parameters gecontroleerd worden en voorzien worden van het juiste type (***Auto-Param***). Ten slotte worden de parameters van de stored procedure gekoppeld (***Bind***) aan de juiste kolommen in de gebruikte tabellen, views, et cetera. In de **Query Optimization**-stap wordt op basis van de query en de waarde van de parameters een plan opgesteld. Dit plan bevat de toegangsmethode die SQL Server gebruikt bij het uitvoeren van het betreffende statement. Denk hierbij aan zaken als table-lookups, table-scans en het samenvoegen van informatie uit bijvoorbeeld twee indexen om gegevens te retourneren. De laatste stap van de optimalisatie is het opslaan van dit plan in de cache voor hergebruik. Bij de uitvoering, tijdens de **Query Execution**-stappen, wordt een executable-plan aangemaakt (***Generate Executable Plan***), dit is het daadwerkelijk compileren van het plan. Hierna wordt de voorbereiding getroffen voor de daadwerkelijke uitvoer, zoals geheugenallocatie (***Fix Memory***), autorisatie bepalen op de tabellen, views, et cetera. (***Grant***) en vorm van uitvoeren (***DoP*** = degree of parallelism) over het aantal beschikbare processoren. Dit laatste maakt het mogelijk een enkel SQL-statement door meer processoren (op één server) gezamenlijk uit te laten voeren. Hiermee wordt het werk verdeeld en haalt iedere processor een deel van de query binnen, waarna deze uiteindelijk weer samengevoegd wordt tot één resultaat. Dit kan een aanzienlijke versnelling opleveren op multi-processor servers. Uiteindelijk vindt de daadwerkelijke uitvoering van het statement plaats (***Execute***) en wordt het plan weer in de cache geplaatst.

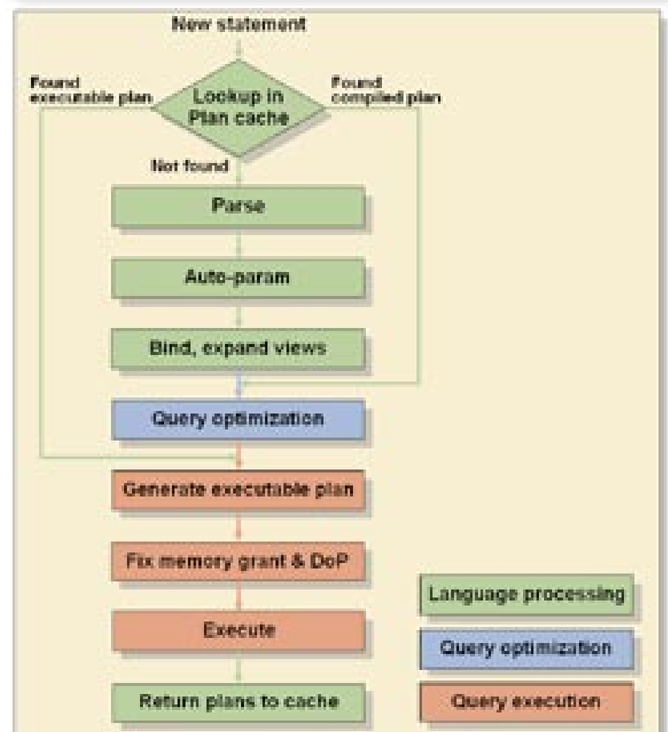
Vier vormen van statement-executie

Dynamische executie is de manier die gebruikt wordt als vanuit de programmatuur in een string een SQL-statement wordt opgebouwd. Dit statement wordt volgens eerder genoemd schema geparsed, van de juiste parametertypes voorzien (***Auto-Param***) en ten slotte in de cache geplaatst.

```
SELECT * FROM titles WHERE price = $19.99
```

Word dit executieplan vervolgens hergebruikt? Wat gebeurt er bijvoorbeeld als het volgende statement wordt uitgevoerd:

```
SELECT * FROM titles WHERE price = 19.99
```



Afbeelding 1. SQL Statement-executie

In dit geval kan SQL niet meer vaststellen dat de parameter ‘price’ van het type ‘money’ is en wordt een tweede executieplan gemaakt en in de cache geplaatst. Met andere woorden, er wordt geen hergebruik gemaakt van het in de cache aanwezige executionplan.

Typed Parameters is een betere oplossing waarbij gebruik wordt gemaakt van ‘sp_executesql’. Hiermee worden de parameters als typed parameters gespecificeerd en doorgegeven aan SQL Server. Dit biedt SQL de mogelijkheid het executionplan te blijven gebruiken uit de cache.

```
EXEC sp_executesql
   N'select * from titles where price = @price',
   N'@price money',
   19.99
```

Naast de mogelijkheid tot hergebruik biedt ‘sp_executesql’ ook betere beveiliging, aangezien op deze manier SQL-injection niet meer mogelijk is, omdat de parameters getypeerd doorgegeven worden aan SQL Server. Trucs als “’ or 1=1 --” om inloggegevens van anderen te achterhalen, zijn op deze wijze niet meer mogelijk. Door in bijvoorbeeld invoervelden hiervan gebruik te maken, is het mogelijk SQL-queries te manipuleren (indien hier niet tegen beveiligd is). Een SQL-query kan er dan als volgt uit zien:

```
SELECT * FROM users WHERE userid = 'brandt' or 1=1 --' or password...
```

Op deze wijze worden dus niet alleen de gegevens van de ene user ‘brandt’ teruggegeven, maar de gegevens van alle gebruikers in de database (wegens de “or 1=1”).

Dynamic String Execution (DSE) biedt de mogelijkheid geheel dynamisch de gewenste statements op te bouwen. Dit kan zowel in .NET-code als in stored procedures. Het voordeel hiervan is dat je zeer specifieke SQL-statements kunt bouwen die bijvoorbeeld gebaseerd zijn op de informatie die meegegeven wordt aan een C#-functie of stored procedure.

```
CREATE PROC GeefKlantInfoParamDSE
(
   @Achternaam  varchar(30) = NULL,
   @Voornaam varchar(30) = NULL,
)
AS
-- More SQL code...

SELECT @ExecStr = 'SELECT * FROM klant WHERE '

IF @Achternaam IS NOT NULL
   SELECT @Achternaam = 'Achternaam LIKE ' + QUOTENAME(@Achternaam,
  '''')
IF @Voornaam IS NOT NULL
   SELECT @Voornaam = 'Voornaam LIKE ' + QUOTENAME(@Voornaam, '''')

SELECT @ExecStr = @ExecStr + ISNULL(@Achternaam, ' ')
   +
   CASE
      WHEN @Achternaam IS NOT NULL AND @Voornaam IS NOT NULL
         THEN ' AND '
      ELSE ' '
   END
   +
   ISNULL(@Voornaam, ' ')

EXEC(@ExecStr)
Go
```

Codevoorbeeld 1.

Zo wordt hier een SELECT-statement opgebouwd afhankelijk van het meegeven van een achternaam of voornaam; zie codevoorbeeld 1. Dit is de meest dynamische en flexibele manier om een database te benaderen, maar ze heeft ook een aantal nadelen:
- De string-evaluatie vindt pas op runtime plaats en niet tijdens designtime. Hierdoor kunnen mogelijke fouten pas in een laat stadium worden gevonden.
- De queries zijn erg complex om te schrijven, wat de onderhoudbaarheid niet ten goede komt (zie het gebruikte voorbeeld).
- Er vindt geen caching en hergebruik plaats aangezien het executieplan pas op runtime wordt samengesteld.
- De uit te voeren DSE-code dient dezelfde rechten te hebben als diegene die de DSE creëert. Er zijn dus meer rechten nodig dan normaal om de query uit te voeren).
- Indien DSE niet heel zorgvuldig gebruikt wordt, biedt het mogelijkheden tot SQL-injection.

Stored Procedures. Beter dan de hiervoor genoemde methoden is het gebruik van stored procedures. Hiermee wordt een aantal problemen opgelost die de vorige methoden om SQL Server te benaderen met zich mee brengen:
- Betere controle over de toegang tot de database en de autorisatie op de database-objecten.
- Betere caching- en hergebruikmogelijkheden.
- Betere sturingsmogelijkheden voor (her)compilatie van de queries.
- Designtime-controle van de syntax.

De werking van stored procedures

De verwerking van stored procedures vindt plaats in twee delen; het aanmaken ervan (Creatie) en het gebruikt ervan (Uitvoering). Tijdens beide delen wordt er door SQL Server een aantal stappen uitgevoerd om tot een snelle en efficiënte manier van uitvoering te komen.

Stap 1. Parsing / Resolution
- Bij het aanmaken van een stored procedure worden alle referenties gecontroleerd.
- Als er verwezen wordt naar andere (nog) niet bestaande stored procedures wordt hiervoor een waarschuwing gegeven. Het is echter wel mogelijk de stored procedure aan te maken. Een voordeel hiervan is dat het hierdoor mogelijk is recursie toe te passen en dat je verwijzingen kunt maken naar nog niet bestaande procedures.
- Het is toegestaan om naar niet bestaande tabellen, views of functies te verwijzen.
- Tijdens deze stap wordt geen executieplan aangemaakt en opgeslagen.

Stap 2. Compilation / Optimization
- **Het compileren van een stored procedure vindt pas plaats bij de eerste** aanroep van de stored procedure.
- De stored procedure wordt geheel geoptimaliseerd voor de te doorlopen code op basis van de meegegeven parameters. Op basis van deze parameters wordt het executieplan opgesteld.
- Deze gecompileerde versie wordt in de cache opgeslagen.
- Alle toekomstige aanroepen maken gebruik van dit executieplan.

Stap 3. Execution / Recompilation
- Bij uitvoering wordt het executieplan uit de cache gehaald en uitgevoerd indien het plan nog geldig is.
- Er is een aantal redenen waarom een plan uit de cache ongeldig wordt en gehercompileerd moet worden:
- Het herstarten van de server of SQL Server.
- Het restoren van de database waarin de stored procedure (of één van de gebruikte objecten uit de stored procedure) zich bevindt.
- Als er tabeldefinities en/of indexen wijzigen waarvan het plan gebruik maakt.
- Als de statistieken van de gebruikte objecten wijzigen.
- Als het plan te weinig gebruikt wordt, kan SQL Server zelf beslissen het plan als ongeldig te markeren.

```
CREATE PROCEDURE GeefKlantGegevens
( @KlantNaam varchar(30) )
AS
SELECT *
FROM klant
WHERE achternaam LIKE @KlantNaam
GO
```
Codevoorbeeld 2

- Handmatig via de aanroep van DBCC FREEPROCCACHE of DBCC FLUSHPROCINDB(id).
- Veel hercompileren kan tot slechte performance leiden.

Soms is het beter om het executieplan niet in de cache plaatsen. Het hercompileren van het plan kan namelijk tot betere resultaten leiden. Hierover meer in de volgende paragrafen.
Tip: *door de interne werking van SQL Server te begrijpen en hier rekening mee te houden, kun je performanceproblemen voorkomen.*

Hercompilatie-issues

Wanneer is het aan te bevelen om te hercompileren? Er is een aantal situaties waarin SQL Server 'denkt' het meest optimale executieplan te hebben, maar waarbij dit in de praktijk niet het geval is. Laten we eens naar het volgende (erg eenvoudige) voorbeeld kijken; zie codevoorbeeld 2.
In dit voorbeeld zoeken we drie maal klanten waarbij de achternaam (deels) voldoet aan de opgegeven parameter. Er zit een groot verschil in performance bij het uitvoeren van een stored procedure met een LIKE-statement als je deze achtereenvolgens aanroept met de parameters als te zien is in codevoorbeeld 3.
De eerste aanroep bepaalt het executieplan, waarna de volgende aanroepen hetzelfde(!) plan uit de SQL Server-cache halen en uitvoeren. Stel dat in dit voorbeeld de eerste aanroep een enkelvoudig resultaat oplevert, en hierbij gebruikgemaakt is van de index en een table-lookup voor het ophalen van het ene record. Als dit nu voor de '%B%'-versie gebruikt wordt, leidt dit tot een zeer grote hoeveelheid I/O-acties aangezien er nu voor ieder record dat in de index gevonden wordt via dezelfde table-lookup opgehaald moet worden. Bij een zoekresultaat van bijvoorbeeld 100 records leidt dit tot 200 database I/O's, terwijl een tablescan waarschijnlijk slechts één of twee I/O's kost (afhankelijk van de SQL-paginagrootte).
Hoe detecteren we dit probleem en wat zijn de mogelijke oplossingen hiervoor? Start door de aanroep van de stored procedure te testen met EXEC WITH RECOMPILE. Dit forceert een hercompilatie voor de gehele procedure. Indien dit inderdaad een ander executieplan oplevert, is er een aantal opties om ervoor te zorgen dat SQL Server altijd een optimaal executieplan gebruikt tijdens het uitvoeren van een stored procedure:
1. Je kunt een stored procedure aanmaken met de optie CREATE WITH RECOMPILE, zodat iedere uitvoering van deze Stored Procedure een hercompilatie forceert voor de volledige procedure. In bovenstaand voorbeeld levert dit in elk van de drie gevallen een optimaal executieplan op.
2. Je kunt ook gebruikmaken van INLINE-hercompilatie als het een stored procedure betreft met verscheidene SQL-regels, waarvan er maar één (of enkele) zo dynamisch zijn dat deze gehercompileerd dienen te worden:

```
SELECT * FROM Klant WHERE Naam LIKE @Naam OPTION (RECOMPILE)
```

3. De beste keuze is om de stored procedure te modulariseren. Met andere woorden het opdelen van een stored procedure in kleinere delen die het maken en (her)gebruiken van executieplannen voor SQL gemakkelijker maakt.

```
EXEC GeefKlantGegevens 'Brandt'
EXEC GeefKlantGegevens 'B%'
EXEC GeefKlantGegevens '%B%'
```
Codevoorbeeld 3.

```
CREATE PROCEDURE GeefKlantGegevens
( @KlantNaam varchar(30) )
AS
IF @KlantNaam LIKE '%[%]%'
BEGIN
SELECT klant_nr, voornaam, achternaam, telnr
   FROM klant
   WHERE achternaam LIKE @KlantNaam
END
ELSE
BEGIN
SELECT klant_nr, voornaam, achternaam, telnr
   FROM klant
   WHERE achternaam = @KlantNaam
END
```
Codevoorbeeld 4.

Tip: *Test stored procedures altijd voor alle mogelijke situaties waarbij weinig of juist veel records worden geretourneerd.*

Modulariteit

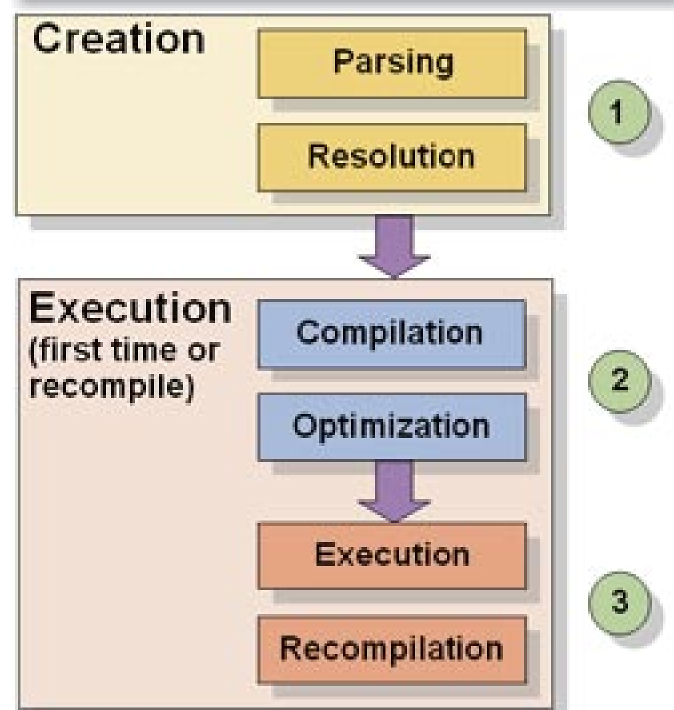
Eén van de problemen met het uitvoeren van stored procedures is het moment waarop het executieplan vastgesteld en bewaard wordt in de SQL-cache. Onderstaand voorbeeld maakt duidelijk wat er gebeurt als we de mogelijkheid bieden om zowel met als zonder wildcards op klantgegevens te zoeken in de stored procedure. We definiëren hiervoor één stored procedure, waarbij aan de hand van de invoerparameter één van beide paden uitgevoerd wordt; zie codevoorbeeld 4..

De volgende aanroep zorgt ervoor dat de tweede SELECT-query wordt uitgevoerd en dat het executieplan samengesteld wordt. Het executieplan wordt dus geoptimaliseerd voor de tweede SELECT-query en niet voor de eerste.

```
EXEC GeefKlantGegevens 'Brandt'
```

Als we vervolgens de onderstaande aanroep uitvoeren, zou je verwachten dat het executieplan voor de eerste SELECT-query wordt aangemaakt.

```
EXEC GeefKlantGegevens 'B%'
```



Afbeelding 2. Het maken en uitvoeren van stored procedures

Helaas: SQL Server heeft tijdens de eerste aanroep ook het executieplan van de IF-tak aangemaakt, echter met de meegegeven parameters van de ELSE-tak! Het is interessant om te proberen deze stored procedure weg te gooien en opnieuw aan te maken en dan eerst de wildcard SELECT uit te voeren en daarna pas de andere query. De query wordt nu toch geoptimaliseerd voor de wildcard-search. Dat de eerste aanroep ook een redelijke performance heeft, is toeval. Niet beide takken worden geoptimaliseerd. Echter, de eerste aanroep is natuurlijk nooit te voorspellen en daarmee het executieplan ook niet. Een oplossing voor dit probleem is het opdelen van stored procedures in kleinere stored procedures (modulariseren). De stored procedure komt er uit te zien als in codevoorbeeld 5.
Van 'sub' stored procedures wordt een executieplan gemaakt op het moment van de daadwerkelijke uitvoering hiervan. In dit voorbeeld wordt tevens bij de wildcard-versie de optie WITH RECOMPILE meegegeven, aangezien bij deze versie van te voren nooit te voorspellen is wat het efficiëntste executieplan is (het aantal resultaatrijen kan namelijk sterk verschillen). Het iedere keer hercompileren is hier dan ook beduidend efficiënter dan het uitvoeren met het 'verkeerde' executieplan. Test of dit in jouw situatie ook de optimale oplossing is.

Tip: Gebruik zo weinig mogelijk conditionele SQL-code in stored procedures, maar maak in plaats hiervan gebruik van losse stored procedures.

Het gebruik van SQL-tooling

Standaard wordt bij SQL Server 2005 een aantal tools meegeleverd waarmee het mogelijk wordt problemen te onderzoeken en analyses uit te voeren op databases en stored procedures. Deze tools bieden een goede start voor analysewerkzaamheden, maar dienen wel met zorg gebruikt te worden. De volgende tools kunnen helpen bij het analyseren en oplossen van problemen:

Database Engine Tuning Advisor (DTA). Met behulp van de DTA is het mogelijk advies te krijgen over mogelijke verbeteringen in de databasedefinitie. Hierbij wordt op basis van een vastgelegde workload de gebruikte executieplannen geanalyseerd en worden aanbevelingen gedaan op het gebied van:

- Statistics
- Indexes
- Indexed views
- Partitions
- Tunes Triggers

Let wel: het gaat hier slechts om adviezen. Volg deze adviezen nooit zomaar op zonder deze te analyseren en te testen. Vaak heb je aan één of twee adviezen al voldoende om de gewenste verbeteringen te bereiken. Het is echter heel goed mogelijk dat het opvolgen van alle adviezen weer tot een vermindering leidt van de reeds behaalde performancewinst. In veel gevallen worden bijvoorbeeld indexen geadviseerd die maar in een heel klein percentage van de queries gebruikt worden, terwijl alle databasemutaties hier wel beduidend trager van worden.

Tip: Volg niet alle adviezen van de DTA op zonder verder onderzoek en kijk goed welke adviezen de grootste performanceverbeteringen opleveren.

SQL Server Profiler. Met behulp van de Profiler is het mogelijk instanties van een SQL- database te monitoren. Dit kan bijvoorbeeld gebruikt worden om:

- Problemen in productieomgevingen te monitoren en te analyseren
- Deadlocks te analyseren
- Opgenomen traces opnieuw af te spelen (bijvoorbeeld om te testen of een fout daadwerkelijk opgelost is)
- Opgenomen traces opnieuw af te spelen met debugging-mogelijkheden (breakpoints, stappen, enzovoort.)

De profiler maakt het mogelijk alle mo gelijke SQL-events vast te leggen. Je kunt de vastgelegde gegevens ook gebruiken in de DTA voor tuningmogelijkheden.

Tip: Opnemen van traces uit een productieomgeving en deze analyseren in een ontwikkelomgeving kan helpen bij het oplossen van problemen of het verbeteren van de performance.

De belangrijkste tips voor ontwikkelaars

Als ontwikkelaar zal je moeten realiseren dat het benaderen van databases een grote impact kan hebben op de performance van een applicatie. Hier kun je en moet je op voorhand rekening mee houden. De onderstaande punten zijn belangrijke handvatten die je kunnen helpen bij het optimaliseren van de databaseperformance:

1. Gebruik altijd 'typed parameters' (voorkom Auto-parameterization)
2. Gebruik stored procedures of sp_executesql
3. Houdt stored procedures klein (modulair)
4. Gebruik de 'Execution-plans' om problemen te lokaliseren
5. Gebruik de DTA & SQL Profiler om je te helpen om mogelijke oplossingen te vinden
6. Test, test, test

```
-- Stored Procedure met test op gebruik van Wildcards
CREATE PROCEDURE GeefKlantGegevens
  ( @KlantNaam varchar(30) )
AS
IF @KlantNaam LIKE '%[%]%'
BEGIN
  EXEC GeefKlantGegevensMetWC @KlantNaam
END
ELSE
BEGIN
  EXEC GeefKlantGegevensZonderWC @KlantNaam
END

-- Stored Procedure met gebruik van Wildcards
CREATE PROCEDURE GeefKlantGegevensMetWC
  ( @KlantNaam varchar(30) )
  WITH RECOMPILE
AS
SELECT klant_nr, voornaam, achternaam, telnr
  FROM klant
  WHERE achternaam LIKE @KlantNaam

-- Stored Procedure zonder gebruik van Wildcards
CREATE PROCEDURE GeefKlantGegevensZonderWC
  ( @KlantNaam varchar(30) )
AS
SELECT klant_nr, voornaam, achternaam, telnr
  FROM klant
  WHERE achternaam = @KlantNaam
```

Codevoorbeeld 5.

Melchior Brandt Corstius is softwarearchitect bij Capgemini (www.nl.capgemini.com). Hij houdt zich voornamelijk bezig met Service Oriented Architecture op het .Net-platform. Melchior is te bereiken via Melchior.BrandtCorstius@Capgemini.com.

Robert Tusveld is als softwarearchitect werkzaam bij Capgemini. Hij heeft zich gespecialiseerd op het gebied van softwareontwikkeling met behulp van producten en technologieën van Microsoft. Robert is te bereiken via Robert.Tusveld@Capgemini.com.

Referenties

Kimberly Tripp: www.sqlskills.com
SQL nieuws & blogs: www.sqljunkies.com
SQL nieuws: www.sqlteam.com
MSDN SQL Server 2005: msdn.microsoft.com/sql
DTA Tutorial: msdn2.microsoft.com/en-us/library/ms166575.aspx