SQL Plan Management in Oracle11*g*

Harald van Breederode

Sinds de introductie van de Cost Based Optimizer (CBO) in Oracle7 hebben zowel database beheerders als database ontwikkelaars de wens om deze optimizer te kunnen beïnvloeden. Deze wens komt voort door het feit dat de CBO soms plotseling een suboptimaal executie plan bepaalt met een tegenvallende performance tot gevolg.

In dit artikel wordt uitgelegd welke technieken er bestaan om dit probleem aan te pakken waarbij de nadruk zal liggen op SQL Plan Management in Oracle11*g*.

Introductie

Om een SQL opdracht uit te kunnen voeren is een zogenaamd executie plan nodig. Dit executie plan wordt aangeleverd door de query optimizer, kortweg optimizer. De optimizer bepaalt mogelijke executie plannen en kiest aan de hand van bepaalde criteria het beste executie plan.

Historisch gezien zijn er binnen Oracle twee optimizers beschikbaar, de Rule Based Optimizer (RBO) en de Cost Based Optimizer (CBO). De RBO bepaalt de keuze van het executie plan op basis van de betrokken schema objecten zonder verdere kennis van de omvang van die objecten en de SQL opdracht in kwestie. De RBO is sinds Oracle10*g* officieel naar het museum verwezen en zal in dit artikel verder buiten beschouwing worden gelaten. De CBO bepaalt het beste executie plan onder andere aan de hand van de betrokken schema objecten, statistische informatie over die objecten, statistische systeem informatie, diverse database parameters en de formulering van de SQL opdracht zelf.

Over het algemeen kiest de optimizer het executie plan dat de beste performance oplevert, maar soms kiest de optimizer door omstandigheden een minder gunstig executie plan. Dit heeft doorgaans te maken met een verandering in de (omgeving van de) database zoals schema veranderingen, parameter veranderingen of nieuwe statistische informatie over het systeem en/of schema objecten. Meestal zijn deze veranderingen positief en blijft de performance hetzelfde of verbetert. Maar soms komt het echter voor dat de performance afneemt. Natuurlijk kunnen we dit laatste voorkomen door helemaal niets te veranderen aan de database maar dan is een kleitablet misschien een betere bedrijfsoplossing. In de loop der tijd zijn er verschillende mogelijkheden in de Oracle software gebouwd die ons in meer of mindere mate controle laten uitoefenen op de optimizer. Dit met als doel het voorkomen van verminderde performance ten gevolge van veranderingen in of aan de database.

Optimizer hints

De oudste techniek om de optimizer te beïnvloeden is door gebruik te maken van optimizer hints. Door middel van een hint geeft men de optimizer de opdracht om het

executie plan, of een gedeelte daarvan, op een bepaalde manier uit te voeren. De term hint is enigszins misleidend omdat het een dwingende opdracht is voor de optimizer. De hint mag alleen genegeerd worden indien daar een technische reden voor is. Met de komst van Oracle11*g* zijn de beschikbare hints gedocumenteerd in V$SQL_HINT.

Het gebruik van hints heeft voor- en nadelen ten opzichte van andere technieken om de optimizer te beïnvloeden. Een voordeel is dat hints door applicatie ontwikkelaars aangebracht kunnen worden maar dat is tevens een nadeel wat de DBA betreft. Er is immers toegang tot de applicatie code nodig om hints te kunnen aanpassen en dat is vaak een probleem voor de DBA. Een nadeel is verder dat als er op een later tijdstip een mogelijk beter executie plan mogelijk blijkt, dit door de optimizer niet bepaald kan worden omdat de hints dit blokkeren. Dus als er een nieuwe index op een tabel wordt aangebracht zal de performance van een met hints doorspekte SQL opdracht waarschijnlijk niet verbeteren ook al zou deze nieuwe index een beter executie plan mogelijk maken. Als laatste nadeel geldt dat er meestal een set van hints nodig is om de optimizer onder alle omstandigheden het gewenste executie plan te laten bepalen. Mijn advies is dan ook om hints alleen te gebruiken als laatste redmiddel.

Optimizer Plan Stability

Eén van de nieuwe mogelijkheden van Oracle8*i* is het kunnen opslaan van bestaande executie plannen in een Stored Outline. Een Stored Outline is een complete set van hints die ervoor zorgt dat de optimizer het originele executie plan kan reproduceren. Stored Outlines worden onderverdeeld in categorieën zodat verschillende applicaties voor dezelfde SQL opdracht een ander executie plan kunnen vastleggen. Stored Outlines worden in het OUTLN schema opgeslagen en kunnen door middel van Export en Import worden overgezet naar een andere database.

Een Stored Outline kan op twee manieren aangemaakt worden. De eerste manier is door de parameter CREATE_STORED_OUTLINES op TRUE te zetten of door deze een categorienaam te geven en daarna de SQL opdracht in kwestie uit te voeren. De tweede manier is door gebruik te maken van de SQL opdracht CREATE OUTLINE. Een eenmaal opgeslagen Stored Outline kan gemanipuleerd worden door de SQL opdracht ALTER OUTLINE, het DBMS_OUTLN package of het DBMS_OUTLN_EDIT package.

De optimizer kan een Stored Outline gebruiken als de sessie parameter USE_STORED_OUTLINES op TRUE staat of wanneer deze op een categorie is ingesteld. Tijdens het bepalen van executie plannen kijkt de optimizer of er een Stored Outline aanwezig is voor de huidige SQL opdracht binnen de juiste categorie. Indien dit het geval is zal de optimizer de hints uit de Stored Outline gebruiken om het gewenste executie plan te genereren. De parameters QUERY_REWRITE_ENABLED, STAR_TRANSFORMATION_ENABLED en OPTIMIZER_FEATURES_ENABLE worden overigens niet vastgelegd in een Stored Outline waardoor andere instellingen voor deze parameters kunnen leiden tot een afwijkend executie plan.

Het gebruik van Stored Outlines heeft als voordeel ten opzichte van hints dat ze volledig onder controle van de DBA staan, ze eenvoudig te implementeren zijn en

makkelijk te onderhouden zijn. Het nadeel is dat ze niet alle factoren die een rol spelen bij het bepalen van een executie plan beïnvloeden. Tot slot is het net als bij het gebruik van hints mogelijk dat betere executie plannen onopgemerkt blijven. Mijn advies is dan ook om Stored Outlines alleen te gebruiken voor de Oracle versies 8*i*, 9*i* en 10*g*.

SQL Profiles

Een gerelateerde nieuwe functionaliteit, geïntroduceerd in Oracle10*g*, zijn SQL Profiles. SQL Profiles bevatten extra informatie voor de optimizer die van belang is voor het bepalen van de kosten van een executie plan. Deze extra informatie kon in eerdere Oracle versies niet worden opgeslagen en wordt aangemaakt door de SQL Tuning Advisor welke eveneens in Oracle10*g* is geïntroduceerd.
Belangrijk is dat SQL Profiles zelf geen executie plan of hints bevatten en dat ze dus niet een bepaald executie plan afdwingen doch slechts de optimizer een handje helpen tijdens het bepalen van het beste executie plan.

SQL Plan Management

De nieuwste ontwikkeling op het gebied van executie plan beheer is SQL Plan Management in Oracle11*g*. Deze functionaliteit heeft wel de voordelen van Stored Outlines maar niet de nadelen en is dan ook de logische opvolger van Optimizer Plan Stability. SQL Plan Management slaat executie plannen op in een SQL Management Base (SMB) dat zich in de SYSAUX tablespace bevindt. De inhoud van de SMB kan uitgevraagd worden via DBA_SQL_PLAN_BASELINES en de DBA kan het ruimte gebruik van de SMB instellen en tevens bepalen hoelang executie plannen bewaard blijven als ze niet gebruikt worden door de optimizer.

Executie plannen in de SMB hebben drie belangrijke kenmerken: ENABLED, ACCEPTED en FIXED. Deze kenmerken zijn van belang tijdens het selectie proces van de optimizer. Het gebruik van executie plannen in de SMB wordt geregeld door de OPTIMIZER_USE_SQL_PLAN_BASELINES parameter welke standaard op TRUE staat. Nieuwe executie plannen die door de optimizer gevonden worden kunnen automatisch toegevoegd worden aan de SMB maar moeten eerst een EVOLVE proces ondergaan voor ze in aanmerking komen voor gebruik.

Er zijn drie methoden om executie plannen in de SMB te plaatsen. Als eerste is er de zogenaamde on-the-fly-capture die er voor zorgt dat de executie plannen van SQL opdrachten, die meerdere keren zijn uitgevoerd, worden opgeslagen. De parameter OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES activeert deze methode. Het eerste executie plan dat op deze manier opgeslagen wordt in de SMB wordt automatisch als ACCEPTED aangemerkt. De tweede methode is een bulk-load methode welke één of meer executie plannen uit de Library Cache of uit een SQL Tuning Set (STS) kan kopiëren naar de SMB. Dit gebeurt door het uitvoeren van de relevante procedure in het DBMS_SPM package. Executie plannen die op deze manier in de SMB worden geladen krijgen direct het stempel ACCEPTED. Als derde manier bestaat de mogelijkheid dat de SQL Tuning Advisor een nieuw executie plan vindt en dit direct aan de SMB toevoegt en als ACCEPTED aanmerkt.

De optimizer overweegt alleen executie plannen uit de SMB indien ze ACCEPTED en ENABLED zijn. Een executie plan dat niet ACCEPTED is kan door middel van een EVOLVE proces ACCEPTED worden indien de performance van dat executie plan beter blijkt dan de reeds aanwezige ACCEPTED executie plannen. Dit EVOLVE proces kan handmatig uitgevoerd worden door de procedure EVOLVE_SQL_PLAN_BASELINE aan te roepen in het DBMS_SPM package of door de SQL Tuning Advisor aan het werk te zetten. De SQL Tuning Advisor is in principe elke nacht actief en zal een advies geven om een executie plan op ACCEPTED te zetten en het bijbehorende SQL Profile te accepteren. Door dit SQL Profile zal het betreffende executie plan voorrang krijgen ten opzichte van andere bestaande ACCEPTED executie plannen. Door de instelling ACCEPT_SQL_PROFILES van de SQL Tuning Advisor op TRUE te zetten kan het advies geheel automatisch geaccepteerd worden zodat de DBA geen omkijken heeft naar SQL Plan Management.

Indien er twee of meer executie plannen ENABLED en ACCEPTED zijn en er geen SQL Profile aanwezig is kiest de optimizer het executie plan dat tot stand is gekomen met optimizer instellingen die het meest overeenkomen met de huidige instellingen. Indien er wel een SQL Profile aanwezig is bepaalt dat de keuze. Het is ook mogelijk om een executie plan in de SMB als FIXED te markeren. Een FIXED plan heeft altijd voorrang ten opzichte van andere executie plannen die niet FIXED zijn. Indien er twee of meer executie plannen ENABLED, ACCEPTED en FIXED zijn en er geen SQL Profile aanwezig is kiest de optimizer het executie plan dat tot stand is gekomen met optimizer instellingen die het meest overeenkomen met de huidige instellingen. Indien er wel een SQL Profile aanwezig is bepaalt dat wederom de keuze. Een eventueel aanwezige Stored Outline voor de SQL opdracht in kwestie heeft weer voorrang boven de plannen in de SMB.

Om het geheel een beetje inzichtelijker te maken een voorbeeld. Laten we aannemen dat we een tabel hebben met één index en dat de optimizer er voor kiest om voor een SQL opdracht die index te gebruiken. We nemen verder aan dat zowel het on-the-fly-capture als het gebruik van executie plannen uit de SMB geactiveerd is. Zodra de optimizer vaststelt dat voor deze SQL opdracht nog geen executie plan in de SMB aanwezig is, wordt het dan bepaalde executie plan in de SMB geplaatst en op ENABLED en ACCEPTED gezet. Hierna zal de optimizer elke keer als dezelfde SQL opdracht uitgevoerd wordt en er geen executie plan in de Library Cache aanwezig is, het plan uit de SMB in de Library Cache plaatsen en gebruiken. Als er later een nieuwe index op de tabel wordt gezet zal de optimizer mogelijk een nieuw executie plan bepalen en deze in de SMB plaatsen. Dit nieuwe executie plan is wel ENABLED maar niet ACCEPTED. De optimizer zal daarom voor het uitvoeren van de SQL opdracht het reeds aanwezige ACCEPTED executie plan gebruiken. Pas wanneer tijdens het EVOLVE proces is aangetoond dat de performance van het nieuwe executie plan daadwerkelijk beter is, wordt het nieuwe executie plan op ACCEPTED gezet. Bij een volgende uitvoering van de SQL opdracht constateert de optimizer dat er twee executie plannen in de SMB op ACCEPTED staan en zal, zoals eerder beschreven is, een keuze maken tussen die twee executie plannen. Voor een uitgebreidere demonstratie over SQL Plan Management zie http://prutser.wordpress.com/files/2009/06/spm.pdf

Door gebruik te maken van procedures in het DBMS_SPM package kunnen we onderhoud plegen op de SMB. Zo kunnen we een staging tabel aanmaken en één of meer executie plannen uit de SMB van of naar deze staging tabel kopiëren met als doel deze executie plannen naar een andere database over te zetten. Verder kunnen executie plannen uit de SMB worden verwijderd.

Er zijn verschillende strategieën denkbaar bij het inzetten van SQL Plan Management. Als eerste kan men er voor kiezen om de on-the-fly-capture simpelweg aan te zetten en voor alle SQL opdrachten het executie plan in de SMB te laten vastleggen. Nieuwe executie plannen kunnen dan alleen gebruikt worden indien is aangetoond dat het een verbetering geeft. Daarbij is het dan wel van belang om een keuze te maken voor een specifiek EVOLVE proces, automatisch of handmatig. Als tweede kan men besluiten om alleen voor bepaalde kritische SQL opdrachten het executie plan vast te leggen in de SMB. Eventueel kan men ervoor kiezen om executie plannen te ontwikkelen in een test of acceptatie omgeving en deze te copieren naar productie. Als derde mogelijkheid kan men ervoor kiezen om alleen executie plannen in de SMB te laten plaatsen door de SQL Tuning Advisor. Men laat zich dan als het ware begeleiden door de Oracle software. Dit is zeker voor de beginnende DBA een aantrekkelijke methode van werken.

SQL Plan Management is ook uitermate goed bruikbaar om tijdens een database upgrade verslechtering van SQL performance te voorkomen. Er zijn in principe twee mogelijkheden. Als eerste kunnen de bestaande executie plannen in de huidige versie van de database in een STS geplaatst worden om deze dan na de database upgrade in de SMB te laden. We gaan dus in de nieuwe versie van start met de executie plannen van de oude versie. Als de optimizer constateert dat er een beter executie plan beschikbaar lijkt te zijn kan dat alleen gebruikt worden als tijdens het EVOLVE proces wordt aangetoond dat dat plan inderdaad beter is.
Met de tweede mogelijkheid voeren we eerst de database upgrade uit en zetten de OPTIMIZER_FEATURES_ENABLE parameter op de versie van vóór de upgrade. Als voor alle SQL opdrachten de executie plannen in de SMB zijn geladen, middels de on-the-fly-capture, zetten we de OPTIMIZER_FEATURES_ENABLE parameter op de waarde van de nieuwe versie van de software. We genereren dus met de nieuwe versie software de executie plannen zoals ten tijde van de oude versie en als er in de nieuwe versie een beter executie plan wordt gevonden kan dat alleen gebruikt worden als aangetoond is dat de performance inderdaad beter is.

Natuurlijk heeft ook het gebruik van SQL Plan Management zijn voor- en nadelen. Het grootste voordeel is dat verslechtering van SQL performance op een effectieve manier wordt bestreden en dat alles onder beheer van de DBA valt. Als nadeel geldt misschien dat het begrijpen wat de software allemaal uitspookt tijdens het kiezen van het meest optimale executie plan er niet eenvoudiger op wordt. Tevens moet men beslissen op welke wijze het EVOLVE proces uitgevoerd dient te worden. Van belang hierbij is te weten dat voor het automatische EVOLVE proces de SQL Tuning Advisor noodzakelijk is waarvoor een aparte licentie nodig is.
Mijn advies is om SQL Plan Management te gebruiken wanneer verslechtering van SQL performance voorkomen dient te worden.

Conclusie

De Cost Based Optimizer kiest het beste executie plan op basis van informatie die daarvoor onder andere in de database beschikbaar is. Over het algemeen leidt dit tot een goede en stabiele database performance maar soms bepaalt de optimizer een suboptimaal executie plan. Door de jaren heen zijn er verschillende technieken ontstaan om de optimizer hiervoor te behoeden. Elke techniek heeft zijn voor- en nadelen maar met de komst van SQL Plan Management is er een definitieve oplossing voor het voorkomen van verslechtering van SQL performance. Wel is het van belang om goed na te denken over hoe en op welke manier SQL Plan Management ingevoerd dient te worden.