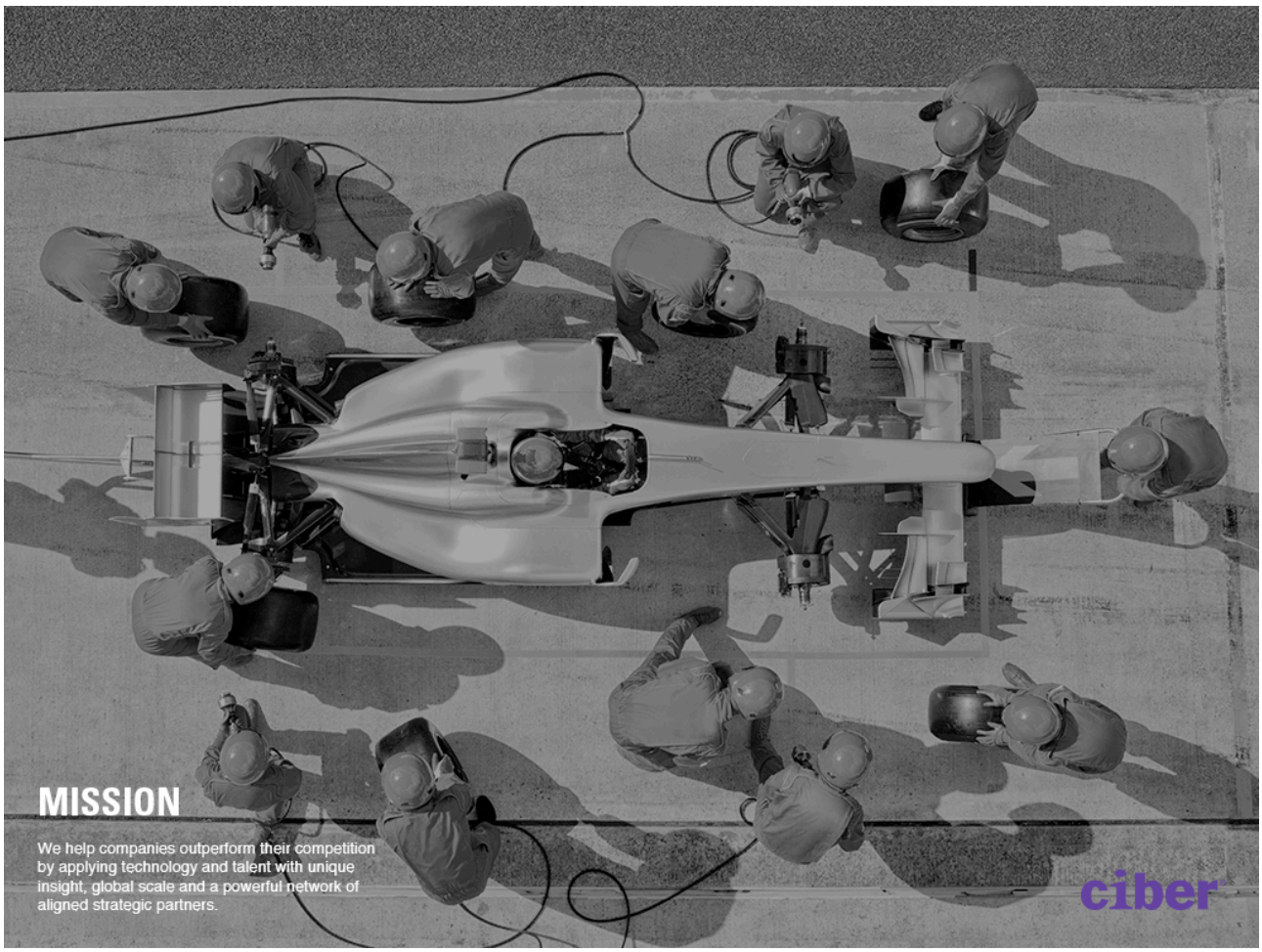
Doe-het-zelf-sessie

Oracle performance-onderzoek en -optimalisatie



Rob van Wijk

Eindhoven, 8 oktober 2015
Nieuwegein, 14 oktober 2015

Inleiding

Iedereen die met database werkt, komt met enige regelmaat in aanraking met performance-problemen: de ontwikkelaar die ziet dat zijn zoekvraag, die in een fractie van een seconde resultaat had moeten opleveren, er in productie vele seconden over doet, de DBA die opeens van gebruikers telefoontjes krijgt dat de database die hij beheert niet meer vooruit te branden is en de BI-er wiens management-rapportages niet meer voor 7 uur ‘s ochtends tot een einde gekomen zijn. In het begin lijkt het een mysterie. De druk is groot om snel maar iets te doen: de ontwikkelaar herschrijft zijn zoekvraag naar een functioneel equivalente zoekvraag, de DBA past wat database parameters aan, of de BI-er voorziet zijn zoekvraag van optimizer hints. Want tijd om uitgebreid onderzoek te doen is er niet, lijkt het. De hoop is dat met wat geluk de druk verlicht kan worden. Later komt dan het echte onderzoek en de oplossing.

Het probleem met deze methode - hoe begrijpelijk ook - is dat ze zelden in eerste instantie het performance-probleem oplossen of verlichten. Zonder onderzoek is het gewoon een gok. En als je “geluk” hebt, en het probleem lijkt verholpen, maar je weet nog steeds niet de oorzaak, dan blijf je zitten met het terechte gevoel dat je niet weet wat je overkomen is, en dat het morgen weer kan gebeuren. Of je doet net alsof je het wel weet en je verkondigt mythes als “IN is altijd sneller dan EXISTS” of “CURSOR_SHARING moet altijd op FORCE staan”.

Feit is dat de Oracle-database zeer volwassen is en uitstekend geïnstrumenteerd. De database geeft ons veel mogelijkheden om te onderzoeken waar de tijd wordt gespendeerd. Met kennis van de meetmethodes kan zo’n beetje alle informatie worden opgevraagd die je je maar wenst. Het enige dat dan nog kan gebeuren, is dat je niet weet hoe je deze informatie moet interpreteren. Vaak drijft echter de reden en/of oorzaak meteen naar boven. En zo niet, dan heb je de getallen paraat, waarmee hulp zoeken op een Oracle-forum of met Oracle-support gemakkelijk is geworden.

Het zwaartepunt van deze sessie ligt daarom bij de diverse meetmethodes, en daarnaast zal aandacht besteed worden aan belangrijke onderwerpen die bijdragen aan het begrijpen van de oorzaak van veel performance-problemen, waaronder bindingsvariabelen en peeking, histogrammen en partities. Uiteraard zullen we diverse performance-problemen niet alleen onderzoeken, maar ook daadwerkelijk oplossen.

De onderwerpen zijn:
1) De twee categorieën performance-problemen
2) SQL*Trace en tkprof
3) Explain plan
4) AWR
5) Statistieken
6) Optimaliseren
7) Histogrammen
8) Bindingsvariabelen en peeking
9) PL/SQL-bulkverwerking
10) Gepartitioneerde tabellen

Het belangrijkste doel dat vandaag geprobeerd wordt te bereiden, is dat je ervan overtuigd raakt dat er altijd eerst onderzocht moet worden waar het echte probleem zit, om vervolgens het probleem met één gerichte actie op te lossen natuurlijk.

Voorbereiding

Zorg ervoor dat je een Oracle 12.1.0.2 database geinstalleerd hebt op je laptop. Dit kan je op twee manieren doen tegenwoordig.

De makkelijkste manier is om in Virtualbox een prebuilt VM te downloaden. Kies hiervoor de “Database APP Development VM” op:
http://www.oracle.com/technetwork/community/developer-vm/index.html#dbapp

Andere manier is om zelf de database apart te downloaden en installeren. Ga hiervoor naar:
http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index-092322.html
en kies hier zelf de juiste variant, afhankelijk van het besturingssysteem waarin je het wilt installeren.

Deze sessie is ontwikkeld om te werken met SQL*Plus. Dat kan binnen je VM, maar fraaier is om dit in je host-omgeving te doen. Hiertoe moet je de Instant Client downloaden voor jouw besturingssysteem op:
http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html
Kies hier de “Instant Client Package - Basic” en de “Instant Client Package - SQL*Plus” en volg de installatie-instructies om de omgevingsvariabelen goed te zetten.

Om te kunnen verbinden vanuit je host naar de guest VM, moeten ze in hetzelfde netwerk zitten. Een manier is om in Virtualbox -> Instellingen -> Netwerk een Host-only adapter te selecteren, naam vboxnet0, promiscuous modus “Alle toestaan”. Je VM krijgt nu een IP-adres, bij mij 192.168.56.101, maar zoek die zelf op op je VM. Dit IP-adres gebruik je om een verbinding te maken op je host, bijvoorbeeld met EZConnect: sqlplus sys/oracle@192.168.56.101/CDB1 as sysdba. Of je gebruikt het IP-adres in je tnsnames.ora.

Als je de prebuilt VM gebruikt, dan heb je een container database CDB1 en een pluggable database ORCL. In deze sessie werken we in de pluggable database. Maak in de ORCL pluggable database een gebruiker aan met je eigen code (bijvoorbeeld rwijk) en ken de DBA rol aan deze gebruiker toe. Installeer de bekende EMP en DEPT tabellen in je eigen schema -en dus niet (alleen) in scott/tiger- door het script demobld.sql te draaien. Dit script kan je vinden op internet door te googlen op “demobld.sql”.

Voer tenslotte onder de gebruiker SYS in de container database de volgende commando’s uit:

```
alter session set container = orcl;
grant execute on dbms_alert to <jouw gebruikersnaam>;
```

Kom alsjeblieft voorbereid naar de sessie, want het zou zonde zijn als je de tijd moet spenderen aan het werkend krijgen van je Oracle-database, in plaats van aan de stof zelf.

1. De twee categorieën performance-problemen

Een actieve sessie is altijd hetzij werk aan het doen (verbruiken van cpu), hetzij aan het wachten. De doorlooptijd van een activiteit is de optelsom van deze twee. Een performance-probleem kan je daarom indelen in één van de volgende twee categorieën:
1. de sessie doet voornamelijk niets omdat het ergens op aan het wachten is
2. de sessie doet voornamelijk teveel vanwege één of andere inefficiëntie

In geval 1 kan de sessie bijvoorbeeld wachten omdat het een rij probeert te bewerken die al door een andere sessie is vergrendeld. Of omdat meerdere sessies op een latch zitten te wachten. In geval 2 zie je dat een zoekvraag meer systeembronnen gebruikt dan nodig.

Een goede eerste stap voor het onderzoeken en uiteindelijk oplossen van een performance-probleem is daarom om de categorie te bepalen. Als systeembeheerder kan je bijvoorbeeld het UNIX-commando “top” gebruiken. Maar ook de v$-views bevatten genoeg informatie om dit te bepalen.

a) Lees inde Oracle Database Reference over de views V$SESSION, V$SESSION_WAIT, V$SESSION_LONGOPS en V$SESS_IO en schrijf een scriptje dat alle actieve sessies toont en voldoende informatie om de categorie te kunnen bepalen. Toon in ieder geval de module kolom van V$SESSION. Geïnstrumenteerde systemen zetten daar namelijk informatie in over de module die momenteel draait.

b) Start een losse SQL*Plus-sessie op (sessie A) en start script ***opoo1b_installeer***. Bekijk in een andere sessie (sessie B) met behulp van het in 1a gemaakte script met welke categorie je te maken hebt. Als je klaar bent, start je script ***opoo1b_einde*** in sessie B om één en ander netjes te beëindigen.

c) Start een losse SQL*Plus-sessie (A) op en start script ***opoo1c_installeer***. Bekijk in een andere sessie (B) met behulp van het in 1a gemaakte script met welke categorie je te maken hebt. Als je klaar bent, start je script ***opoo1c_einde*** in sessie B om één en ander netjes te beëindigen.

d) Bekijk eens vluchtig waarop je allemaal kunt wachten, door de volgende zoekvraag uit te voeren: ***select name from v$event_name order by name***

Extra) Pas het script uit opgave 1a dusdanig aan dat je nog meer nuttige zaken te weten kunt komen over drukken en/of wachtende sessies. Bijvoorbeeld: het SQL-commando dat momenteel uitgevoerd wordt en het object waarop wordt gewacht, als het wacht.

2. AWR

AWR staat voor Automatic Workload Repository. Het is het gereedschap voor de DBA die wil kunnen zien wat er allemaal gebeurt op zijn gehele database qua performance, over alle sessies heen. Het is de opvolger van statspack en bestaat al sinds versie 10. Vaak geeft dit teveel ruis weer als er een relatief klein probleem is in de database. Andere keren biedt AWR een uitstekende eerste aanknopingspunt om te bepalen in welke richting je moet zoeken.

a) Maak een eerste AWR-snapshot door uit te voeren ***exec dbms_workload_repository.create_snapshot()***. Voer vervolgens vijf willekeurige zoekvragen uit, maakt niet uit welke. Dat mag in de huidige sessie, maar mag ook in een nieuwe sessie. Daarna voer je nogmaals een ***exec dbms_workload_repository.create_snapshot()*** uit.

b) Ga naar je virtuele machine en log daar via je terminal applicatie en sqlplus in en voer uit @$ORACLE_HOME/rdbms/admin/awrrpt en beantwoordt de vragen. Kies voor 'text' als soort uitvoer en selecteer de laatste twee snapshot ID's. Bekijk het resultaatbestand vluchtig om een idee te krijgen welke informatie je allemaal hierin kunt vinden.

Extra) Herhaal opgave 2b, maar kies nu 'html' en open het resultaatbestand in een browser, eventueel door het bestand eerst naar je host te ftp-en.

3. SQL*Trace en tkprof

Als eenmaal is bepaald dat het probleem in categorie 2 valt, oftewel er wordt (te) veel werk uitgevoerd, dan is het zaak om te bepalen waar de tijd precies gespendeerd wordt. Dit kan het allerbeste gedaan worden door een langzaam proces of zoekvraag te "tracen". Je zet dan SQL*Trace, zijnde Oracle's instrumentatie, aan. Zolang SQL*Trace is geactiveerd, schrijft Oracle veelvuldig diverse performance-gerelateerde informatie weg in een .trc bestand, in de folder die gespecifieerd staat in V$DIAG_INFO name = 'Diag Trace'. Het trace-bestand zelf is voor een ongeoefend oog nauwelijks leesbaar, en daarom is er tkprof. Tkprof is een programma buiten de database, dat een trace-bestand omzet in een leesbaarder tekstbestand.

a) bekijk waarde trace-bestanden terecht zullen komen op jouw laptop. Type hiertoe in: ***select * from v$diag_info where name = 'Diag Trace'***. Bekijk daarna hoe je trace-bestand gaat het door een: ***select * from v$diag_info where name = 'Default Trace File'***.

b) Start een tweede SQL*Plus-sessie op en start het script ***opoo3_installeer***, zonder te kijken wat dit doet.

c) Zet tracing in je eerste sessie aan door in te typen: ***alter session set sql_trace true***. Start de procedure opoo3 door in te typen ***exec opoo3*** en sluit alle cursoren door je sessie te beëindigen. Type daarom in: ***disconnect*** en log daarna weer aan. Als je wilt, kun je zien dat er door de laatste stap rijen in het trace-bestand zijn geschreven die beginnen met #STAT. Deze rijen bevatten de informatie die leidt tot de sectie "Row Source Operation" in het uiteindelijke tkprof-bestand.

d) Navigeer naar de Diag Trace folder die je hebt bekeken in vraag 3a. Bekijk eerst eens alle opties voor tkprof, door allen ***tkprof*** in te typen. Bekijk vervolgens het trace-bestand. Type daarna in:

tkprof <trace-bestand> a.txt sys=no sort=prsela exeela fchela

En bekijk vervolgens het bestand a.txt

e) Alle informatie in het tkprof-bestand duidt precies aan wat er daadwerkelijk is gebeurd. Waar in dit proces werd de meeste tijd gespendeerd?

f) Lees in de Database SQL Tuning Guide hoofdstuk 18 “Performing Application Tracing”: https://docs.oracle.com/database/121/TGSQL/tgsql_trace.htm#CHDHIIHC t/m “Library Cache Misses in TKPROF” () en probeer alle getallen in het tkprof-bestand te begrijpen. Schrijf de secties/getallen op die niet helemaal duidelijk zijn.

Extra) Maak een kopie van procedure opoo3 door de broncode op te halen uit de user_source view. Noem de nieuwe procedure opoo3_kopie en breng hierin een versnelling aan. Toon dit aan door een nieuw tkprof-bestand te fabriceren van de aangepaste procedure.

Extra2) Experimenteer met het commando ***alter session set events ‘10046 trace name context forever, level N’***. Als N=0, dan is dit equivalent met ***alter session set sql_trace false***. Als N=1, dan is dit equivalent met ***alter session set sql_trace true***. Vul zelf de waarden 4, 8 en 12 in en bekijk wat de gevolgen zijn in je trace-bestand en het tkprof-bestand.

4. Explain plan

Een methode om alleen het plan op te vragen van een SQL-commando, is via het explain plan commando. Nadeel hiervan is dat je niet zeker bent dat het plan dat je ziet, ook het plan is dat gebruikt zal worden in de applicatie. Dit komt omdat omgevingsvariabelen, optimizer-instellingen en/of bindingsvariabelen anders kunnen zijn in de applicatie. Toch zijn er twee belangrijke voordelen:
- de zoekvraag zelf hoeft niet uitgevoerd te worden: alleen de parse-fase wordt doorlopen
- je krijgt te zien welk predicaat in welke stap wordt toegepast

a) Lees in hoofdstuk 7 van de Oracle Database SQL Tuning Guide “Reading Execution Plans” 7.1 t/m 7.2.1, dus stoppen bij “Viewing Parallel Execution with EXPLAIN PLAN”: https://docs.oracle.com/database/121/TGSQL/tgsql_interp.htm#TGSQL94618.

b) Vraag het plan op van het SQL-commando uit script opoo4b:

```
select /*+ use_nl(s e) use_nl(e d) */
       e.ename
     , d.dname
     , s.grade
  from emp e
     , dept d
     , salgrade s
 where e.deptno = d.deptno
   and e.sal between s.losal and s.hisal
   and d.dname like '%S%'
 order by d.dname
     , e.ename
```

c) Een plan wordt altijd van binnenuit gelezen en bij stappen van gelijke diepte wordt de eerstgenoemde eerder uitgevoerd. Het gemakkelijkst is dit te zien door de stappen in

een boomstructuur te plaatsen. Teken een boom voor het plan uit de vorige vraag en negeer ID 0 (SELECT STATEMENT) en zet ID 1 bovenaan de boom.

d) Een blad uit een boom kan altijd uitgevoerd worden. Een knoop die uit meerdere takken bestaat, kan alleen uitgevoerd worden als alle onderliggende takken zijn uitgevoerd. Er wordt van links naar rechts gewerkt in de boom. Zet met deze regels in het achterhoofd de stappen in volgorde van uitvoering.

e) Schrijf code (pseudocode of PL/SQL) voor hoe deze zoekvraag uitgevoerd zal worden.

Extra) Zoals gezegd, kan het plan dat je met een explain plan verkrijgt, afwijken van het plan dat je in de “Row Source Operation” van het tkprof-bestand ziet. Vraag het plan maar eens op van het UPDATE-commando uit hoofdstuk 3, die je uit het tkprof-bestand kunt halen. Waar komt het verschil vandaan?

5. Statistieken

Om de cost-based optimizer goed plannen te kunnen laten genereren, heeft hij actuele statistieken nodig van alle betrokken objecten. Tot en met versie 8 werd het ANALYZE-commando gebruikt, maar vanaf versie 9 gebruiken we de package DBMS_STATS.
Lees hoofdstuk 12 Managing Optimizer Statistics: Basic Topics, onderdeel 12.4 t/m 12.4.6 door. Van Gathering Optimizer Statistics Manually t/m Gathering Statistics for Volatile Tables Using Dynamic Statistics:
https://docs.oracle.com/database/121/TGSQL/tgsql_stats.htm#TGSQL406

Maak de tabel opoo5 aan door script ***opoo5_installeer*** te draaien.

a) Draai een explain plan van “select nr from opoo5”. Kloppen de geschatte cardinaliteiten? Hoe komt dat?

b) Draai script ***opoo5b***, zonder naar de inhoud van dit script te kijken. Draai weer een explain plan. Hoe verklaar je de getallen?

c) Bekijk nu de parameters van de procedure dbms_stats.gather_table_stats in de Oracle Database PL/SQL Packages and Types Reference:
http://docs.oracle.com/database/121/ARPLS/d_stats.htm#ARPLS059
Vergaar zelf statistieken op de tabel opoo5 en toon dat de getallen nu wel kloppen.

Extra) Bekijk de overige procedures van de dbms_stats package en krijg een idee wat er allemaal mogelijk is.

Verwijder tabel opoo5 door script ***opoo5_einde*** te draaien.

6. Optimaliseren

a) Bij de tabel KLANTEN worden attributen waarvan het verloop van hun waarde gedurende de tijd bijgehouden moeten worden, opgeslagen als KLANT_PROFIELEN. De tabel KLANT_PROFIELEN kent een twintigtal van dit soort flexibele attributen. Dit is overigens slecht gemodelleerd, want attributen van een verschillend datatype worden

nu in één uniform datatype geperst en zoekvragen worden onnodig complexer door vagere naamgeving. Het is echter een situatie gebaseerd op een praktijkvoorbeeld. Een zoekvraag die alle klanten ophaalt inclusief de huidige waarde van 5 van de 20 flexibele attributen, moet geoptimaliseerd worden.

Draai het script ***opoo6a_installeer*** om de situatie aan te maken. In het script ***opoo6a*** staat de zoekvraag die geoptimaliseerd moet worden. Als je klaar bent, draai je ***opoo6a_einde*** om de installatie ongedaan te maken.

Tip: kijk of je het aantal tabelbenaderingen kunt verminderen.

b) Er is een probleem met een zoekvraag in een applicatie. Nu duurt deze boven de seconde, wat onacceptabel is in een scherm. Aan jou het verzoek om de zoekvraag dusdanig te optimaliseren om deze vele factoren te versnellen.

Draai het script ***opoo6b_installeer*** om de situatie aan te maken. In het script ***opoo6b*** staat de zoekvraag die geoptimaliseerd moet worden. Als je klaar bent, draai je ***opoo6b_einde*** om de installatie ongedaan te maken.

Tip: gebruik je gezonde verstand om te zien of alles in de juiste volgorde wordt gedaan.

7. Histogrammen

Histogrammen dienen om fijnere statistieken te vergaren over de mogelijke waarden van een kolom, zodat je een beter idee krijgt hoe vaak een waarde voorkomt in een kolom. Zonder histogrammen gebruikt de cost based optimizer bij een zoekvraag als “select * from tabel where kolom = waarde” informatie als:

- NUM_ROWS - het aantal rijen in de tabel
- LOW_VALUE - de laagst voorkomende waarde van de kolom
- HIGH_VALUE - de hoogst voorkomende waarde van de kolom
- NUM_NULLS - het aantal rijen met een NULL-waarde voor de kolom
- NUM_DISTINCT - het aantal verschillende waarden (exclusief NULL)

Lees hoofdstuk 11 van de Oracle Database SQL Tuning Guide tot en met 11.5 Height-Balanced Histograms (Legacy):
http://docs.oracle.com/database/121/TGSQL/tgsql_histo.htm#TGSQL366
Draai script ***opoo7_installeer***.

a) Draai de zoekvraag uit het script ***opoo7*** en vraag het plan op met een explain plan commando. Kan je de formule achterhalen hoe deze cardinaliteit is berekend?

b) Het zou fijn zijn als deze zoekvraag gebruik maakt van de index op de Status-kolom. Maak daarom een frequentie-histogram aan om dit voor elkaar te krijgen. Vraag de kolom Histogram van de view USER_TAB_COL_STATISTICS uit om dit te controleren.

c) Maak een zoekvraag waarin je van een frequentie-histogram het volgende toont:
   - een volgnummer van het bakje (1 t/m aantal bakjes (=bucket))
   - de waarde van het bakje
   - het aantal rijen met deze waarde, oftewel de grootte van het bakje

Tip: Gebruik de analytische functie LAG

d) Maak van de eerder gemaakte histogram een topfrequentiehistogram. Bekijk het resultaat via de USER_HISTOGRAMS view. Bekijk de cardinaliteiten van de zoekvraag uit script opoo7, terwijl je de waarde verandert van 0 naar 4.

Extra) Lees paragraaf 11.6 over hybride histogrammen.

Als je klaar bent, draai je ***opoo7_einde*** om de objecten op te ruimen.

8. Bindingsvariabelen en peeking

Om niet ieder SQL-commando iedere keer te hoeven parsen, worden alle cursoren die eenmaal geparsed zijn, bewaard in de library cache, onderdeel van de shared pool. Als je opnieuw een SQL-commando uitvoert en die blijkt al te bestaan, dan wordt het opgeslagen plan gebruikt. Bindingsvariabelen spelen een belangrijke rol in het optimaal hergebruiken van alle cursoren in de library cache. Zonder bindingsvariabelen zijn zoekvragen als “select * from emp where empno = 7839” en “select * from emp where empno = 7566” niet identiek en staan er twee cursoren in de library cache. Als bindingsvariabelen waren gebruikt, dan was de zoekvraag “select * from emp where empno = :B1” geweest en stond deze slechts eenmaal in de library cache. Met als gevolg een beter hergebruik van cursoren en slechts eenmaal een parse-actie.

a) Bekijk de vier stukken tekst uit het script ***opoo8*** en onderzoek deze door ze door SQL*Trace/tkprof te halen. Welke variant gebruikt geen bindingsvariabelen?

b) Draai het script ***opoo8b_installeer***. Schrijf een script dat tabel OPOO8B tweemaal vult met 10.000 rijen: (1,‘Omschrijving 1’) t/m (10000,‘Omschrijving 10000’). Eenmaal rij-voor-rij-verwerking met een impliciete cursor (een INSERT ... VALUES commando) en eenmaal EXECUTE IMMEDIATE zonder gebruik te maken van bindingsvariabelen. Onderzoek beide methodes en bepaal welke factor verschil er zit tussen de twee alternatieven. Als je klaar bent, verwijder je de tabel door ***opoo8b_einde*** te draaien.

Tevens is het goed om te weten dat in versie 9 en 10 er nog geen sprake was van adaptieve cursor sharing. In deze versies werd altijd het plan gekozen van de allereerste keer dat de cursor werd geparsed.

c) Draai het script ***opoo8c*** voor een demo van peeking en adaptieve cursor sharing. Druk steeds op een toets als je verder wilt.

Extra) Lees hoofdstuk 15 Controlling Cursor Sharing tot en met 15.1.3 Adaptive Cursor Sharing: http://docs.oracle.com/database/121/TGSQL/tgsql_cursor.htm#TGSQL848

9. PL/SQL-bulkverwerking

PL/SQL-bulkverwerking is het alternatief voor grotere verwerkingsklussen die niet met SQL alleen opgelost kunnen worden. Met het verstrijken van de Oracle-versies en de

steeds uitgebreidere mogelijkheden van Oracle SQL, wordt de noodzaak tot inzet van PL/SQL-bulkverwerking weliswaar steeds kleiner, toch is het goed om deze optie achter de hand te houden. Zeker als je minder handig met de nieuwere functionaliteiten van Oracle SQL bent.

Met PL/SQL-bulkverwerking kunt je het aantal executies van een DML-commando drastisch omlaag halen, wat weer leidt tot snellere verwerking. Begin met het draaien van script ***opoo9_installeer***.

a) Bekijk de code van het script ***opoo9a*** en schrijf op hoe vaak je verwacht dat het SELECT- respectievelijk het INSERT-commando een parse-, execute- en fetch-opdracht zullen doen.

b) Zet tracing aan, draai het script ***opoo9a*** en maak en bekijk het tkprof-bestand. Waren dit de getallen die je verwachtte?

c) Lees in de Oracle Database Reference over de parameter PLSQL_OPTIMIZE_LEVEL. Zet deze met een ALTER SESSION op 0 en herhaal de stappen van vraag b.

d) Schrijf code zodanig dat:
- het hetzelfde doet als script opoo9a: kopieren van t1 naar tabel t2
- het selecteren met een zelf in te stellen bulkgrootte werkt
- de INSERT's in dezelfde porties worden uitgevoerd

Gebruik hiervoor de LIMIT-clausule en het FORALL-statement. Voorbeelden hiervan zijn te vinden in hoofdstuk 12 van de Oracle Database PL/SQL Language Reference: http://docs.oracle.com/database/121/LNPLS/tuning.htm#LNPLS012
Maak hiervan weer een tkprof-bestand om het verschil te zien.

Extra) Hoe had de code nog optimaler geschreven kunnen worden? Staaf dit met een tkprof-bestand.

Draai het script ***opoo9_einde*** om de gebruikte tabellen op te ruimen.

10. Gepartitioneerde tabellen

Partitionering wordt vaak als een stuk gereedschap gezien om performance mee op te krikken. In een OLTP-omgeving is dit echter vrij lastig en zul je juist moeite moeten doen om dezelfde snelheid te behouden. Gegevenspakhuizen kunnen wel mooi profiteren van partitionering als de partities ze in staat stellen om slechts een klein deel van een grote tabel in een extra partitie te selecteren. De volgende vragen zullen een aantal voor- en nadelen laten zien. Draai allereerst script ***opoo10_installeer***, om twee tabellen met dezelfde inhoud te maken: één gepartitioneerde tabel en een "normale" tabel.

a) Bekijk de scripts ***opoo10a1*** en ***opoo10a2*** en trace deze. Kan je het verschil verklaren? Wat kan je doen om de performance gelijk te houden? En wat is daar weer het nadeel van?

b) Verwijder alle rijen met een datum voor 1-1-2000, door de scripts ***opoo10b1*** en ***opoo10b2*** te draaien.

c) Schrijf een zoekvraag die de som van alle notabedragen in 2003 ophaalt van beide tabellen. Vraag de plannen op van beide zoekvragen en draai ze terwijl je met “set timing on” de doorlooptijd opvraagt.

Draai script ***opoo10_einde*** om de tabellen op te ruimen.