2006:041 HIP

EXAMENSARBETE

Databasanalys

Gustaf Persson

Luleå tekniska universitet

Högskoleingenjörsprogrammet
Datateknik
Institutionen för Systemteknik
Avdelningen för Datalogi

2006:041 HIP - ISSN: 1404-5494 - ISRN: LTU-HIP-EX--06/041--SE

Förord

Examensarbetet databasanalys har genomförts på Ericsson, Lindholmen Göteborg under sommaren och hösten 2005. Jag vill tacka alla som varit med och medverkat till att arbetet har fungerat så bra framförallt personal och då speciellt min handledare Anna Ekebro på Ericsson.

Sammanfattning

Examensarbetets huvudsakliga syfte var att ta fram en metodik och en manual för hur tester kan genomföras på ett program som arbetar mot en databas. Manualen innehåller exempel på hur verktyg används, hur data skall tolkas, olika problem som kan uppstå med mera.

Med metodikens hjälp skall användaren kunna hitta de delar av programmet som kan förbättras med hänseende på prestanda. Manualen skall vara så pass lättanvänd att den skall kunna användas av någon som inte har alltför djupa kunskaper inom databaser eller mjukvaruutveckling.

Resultatet av arbetet är denna rapport och en manual som personal på Ericsson kan använda vid testning av program som använder sig av databaser och då främst Oracle-databaser.

Abstract

The main goal of the thesis work was to create a methodology and a manual for tests of program working against a database. The manual contains exampels on how to use tools, interpret data and how some common problems can be solved or avoided.

With the help of the methodology the user should be able to conduct tests to find the parts of a program that can be enhanced. The manual should be easy enough so that only basic knowledge of the subject is needed to be able to use it..

The result of the work is this report and a manual that staff at Ericsson can use to test programs accessing a database, foremost Oracle databases.

Innehållsförteckning

1 Inledning

Anledningen till att jag sökte examensarbetet på Ericsson var på grund av att arbetet verkade vara något som jag själv kunde ha nytta av samt att Ericsson verkade ha verklig användning för resultatet. Jag har också läst en del kurser som involverat databaser och dessa har jag tyckt varit mycket intressanta.

Bakgrunden till själva arbetet är Ericssons system för nätverksinventering som utvecklas på Lindholmen. Detta system, som bygger på en tredjepartsprodukt, samlar in information om ett nätverk och sparar detta i en relationsdatabas. Denna information används sedan som beslutsstöd och informationssystem för reparationer, uppgraderingar och planeringar av framtida förändringar eller investeringar gällande nät och utrustning.

Analysen syftade till att ta fram en metodik som kan användas på Ericsson för att främst prestandatesta databasen och mjukvaran som är kopplad till databasen. Väldigt lite arbete hade gjorts på detta område innan examensarbetet startade.

Prestanda är viktigt i denna typ av mjukvara då det ofta finns ett visst fönster av tid att genomföra vissa uppgifter på. I det här fallet är det meningen att mjukvaran skall ladda in data under natten och den huvudsakliga användningen av systemet sker på dagen. Det är dock inte bara denna laddning av data som skall kunna genomföras under natten utan då skall även säkerhetskopiering och annat underhållsarbete genomföras för att störa den normala driften på dagen så lite som möjligt. Det är därför viktigt att mjukvaran arbetar så effektivt som möjligt så andra rutiner såsom säkerhetskopiering av databasen kan utföras.

Det finns även konkurrenter på marknaden som har liknande produkter och när en kund väger för- och nackdelar mot varandra är prestanda en del av många som vägs in.

2 Syfte

Syftet med arbetet är:

- Ta fram en metodik som kan användas som en manual eller guide för att i framtiden prestandatesta system.
- Utvärdera verktyg som kan användas i denna process.
- Ge förslag på inställningar av databasen.
- Ge förslag på administrativa uppgifter som syftar till att förbättra prestanda.

Problemet som arbetet skall lösa är att det idag finns liten kompetens på området på den enhet där arbetet kommer att utföras. Meningen med arbetet är att hitta sätt att förbättra en redan existerande produkt samt att i viss mån höja kompetensen på området.

En utvärdering av de vanligaste verktygen som finns samt rekommendationer på hur data från dessa skall tolkas är det som arbetet främst syftar till.

3 Avgränsningar

De avgränsningar som gjorts gällande detta arbete är:

- Det är endast inställningar i databasen och den programvara som utgör databasen som har tagits i beaktande. Saker som minnen, hårddiskar och annan hårdvara har inte testats.

- Operativsystemet som databasen exekveras i har inte heller tagits i beaktande och inga inställningar har förändrats under processen.

Att hårdvara och operativsystem har utelämnats beror dels på att arbetet skulle ha blivit för brett samt att det redan finns kravdokument gällande hårdvara och operativsystemet som inte går att förändra.

4 Teori

4.1 Bakgrund

Den databas som analysen har utförts på är en databas från tillverkaren Oracle. Databasen är en såkallad relationsdatabas. Data sparas i tabeller, som enkelt kan beskrivas som kalkylblad med rader och kolumner. Dessa tabeller kan sedan kopplas samman för att få relationer. Med hjälp av frågespråket SQL, Structured Query Language, kan information från dessa tabeller hämtas och samtidigt filtreras och sorteras.

Det tredjepartssystem som Ericsson använder till inventeringsystem och som använder databasen kommer från tillverkaren Cramer. Cramer tillhandahåller databasstruktur och ett Application Programming Interface eller API för att göra ändringar i databasen.

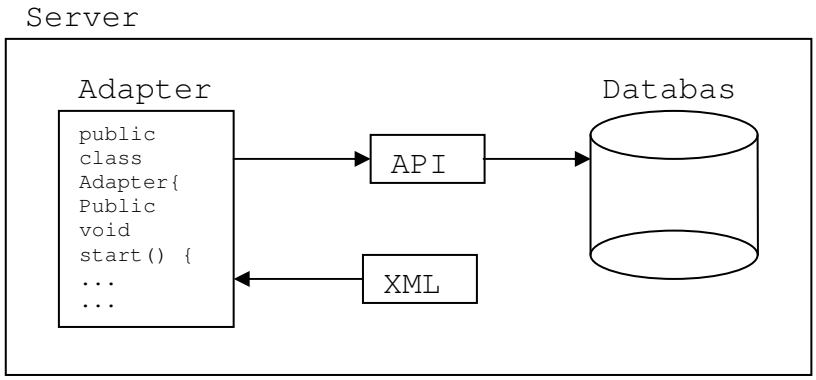
I denna databas lagras information om ett nät. Ett nät består av noder, kort, länkar och mycket mer information. Det är viktigt för tillexempel en operatör att ha information över alla sina resurser för att kunna utnyttja dem effektivt. Genom att ha en god överblick över vilka resurser som finns tillgängliga kan operatören minimera sin överkapacitet och maximalt utnyttja de resurser som redan finns.

För att ladda databasen med information används en såkallad adapter. Denna adapter är utvecklad på Ericsson med hjälp av Cramers API samt egenutvecklad kod.

Data som adaptern skall läsas in är formaterat i XML-format, eXtensible Markup Language. Det format som dessa XML-filer är skapade efter är specificerat i en såkallad DTD, Data Type Definition. Adapterns jobb är att avkoda dessa filer med hjälp av en parser och sedan placera dessa data i databasen. Det är kring denna process som tyngdpunkten av arbetet har lagts.

Databasen måste innehålla testdata för att tester ska kunna genomföras. Testdata fanns redan i systemet då arbetet påbörjades genom de tester som görs för att testa XML-filer och adapterns funktioner. Dessa data ser i mångt och mycket ut som data i ett verkligt system. Testerna speglar alltså ett produktionssystem rätt så bra även om mängden data kan komma att bli större i ett sådant.

4.2 Systemet



Figur 1 Beskrivning av system.

Systemet som gör uppladdningen av nätverksdata är enkelt i sin uppbyggnad. En server innehåller en databas och en adapter. För att ladda upp data i systemet exekveras adaptern och med hjälp av det API som redan nämnts läggs data in i systemet.

4.3 Testning

Redan från arbetets start var det bestämt att tre verktyg skulle testas. Dessa tre verktyg kommer alla från tillverkaren av databasen. De tre verktygen heter:

- TKProf
- Explain Plan
- Oracle Expert

Utöver dessa tre genomfördes också tester med ett fjärde verktyg som också kommer från databastillverkaren nämligen:

- Statspack

Information om hur dessa verktyg kan och skall användas finns att tillgå från tillverkaren i form av manualer. Se [12.2].

Testarbetet kan delas in i tre huvudsakliga kategorier.

- Applikation
  Hur applikationen använder databasen. Tillexempel hur SQL exekveras.

- Instans
  Inställningar av databasen specifikt för att optimera prestanda för applikationen.

- Lagringsstruktur
  Hur data skall lagras på ett optimalt sätt. Tillexempel var databasens filer skall läggas för att optimera användningen av hårdvara.

Verktygen är gjorda för att assistera i antingen ett eller flera av dessa områden. Tyngdpunkten i arbetet har legat på de två första punkterna, alltså applikationen och instansen. Detta främst på grund av att installationen och placeringen av databasens filer är mer eller mindre upp till slutanvändaren av systemet.

4.4 Förutsättningar

För att kunna genomföra tester så är det viktigt att få data från det system som testas. I fallet Oracle fås den viktigaste informationen från dess såkallade SQL-tracesystem. Då användaren anger att systemet skall spåra den SQL som exekveras av databasen sparas all den informationen i såkallade spårfiler. Det är dessa spårfiler som senare används i en del av verktygen.

En annan viktig förutsättning är möjligheten att få statistik med tid. I databasen finns en inställning som kan göras vilken medger att den statistik som tracesystemet skapar innehåller tidsinformation. Utan denna tidsinformation får användaren ingen information om hur lång tid tillexempel en SQL-sats har tagit att exekvera.

Databasen körs på en dedikerad hårdvara. Med detta menas att det bara är databasen samt adaptern samt en instans av webbservern Apache som körs samtidigt på hårdvaran. För mer information om hur testmiljön och systemet i allmänhet fungerar se [5].

För mer information om dessa två grundläggande saker se [12.2] samt bilaga A.

4.5 Verktygen

De fyra verktyg som kommer att testas och användas är som tidigare nämnts Tkprof, Explain plan, Oracle expert samt Statspack. Tre av dessa medföljer databasen som standard medan verktyget Oracle expert köps i ett paket från tillverkaren.

4.5.1 Tkprof

Tkprof är ett verktyg som används för att skapa rapporter från spårfiler som databasen genererar. I de genererade rapporterna kan utläsas hur lång tid det tar att exekvera SQL-satsen, hur mycket data databasen måste läsa för att utföra en SQL-sats med mera.

Informationen från detta verktyg kan främst användas till applikations- samt instansförbättringar. Det är även möjligt att utläsa information om förändringar i lagringsstrukturen men denna information är svårare att utläsa.

Tkprof är ett verktyg som skapar rapporter. Rapporterna visar hur SQL-satser har exekverats i databasen med information som exekveringstider, hur mycket data som lästs, skrivits med mera. Informationen i rapporten säger ingenting om hur denna exekvering kan förbättras utan det är upp till användaren att genom kunskaper förbättra exekveringen. Det krävs alltså att användaren har åtminstone en grundläggande kunskap i hur databasen fungerar och hur den skapar en plan för att genomföra exekveringen.

Verktyget Tkprof finns i alla standardinstallationer av en databas från Oracle. Det används direkt från kommandoraden i operativsystemet där databasen körs.

Se [12.2] samt bilaga A [6.1] för mer information.

4.5.2 Explain plan

Explain plan är ett verktyg som används för att visa vilken exekveringsplan som databasen kommer att välja vid exekveringen av SQL-satsen.

Detta verktyg kan vara bra att använda för att se om exekveringsplanen liknar det som kan förväntas för en specifik SQL-sats. Verktyget visar också det ungefärliga antalet rader som kommer att returneras av satsen. Det allra viktigaste användningsområdet för verktyget är att det genom att testa olika SQL-satser som ger samma resultat går att se vilken exekveringsplan som är den bästa och på så sätt välja den sats som ger bäst prestanda.

Rapporterna som Tkprof skapar innehåller också en exekveringsplan. Skillnaden mellan exekveringsplanen i Tkprof och Explain plan är att exekveringsplanen i Tkprof är den exekveringsplan som verkligen användes då uttrycket exekverades och hämtade data från databasen medan en exekveringsplan i Explain plan är den plan databasen tror att den ska använda med utgång från den statistik och data som finns om tabeller och liknande. Att det inte blir samma resultat i Explain plan och Tkprof kan bero på att statistiken i systemet inte är aktuell eller att data har förändrats under körningens gång.

Genom att se skillnaderna mellan Tkprof och Explain plans exekveringsplaner går det att avgöra om någonting behöver åtgärdas. Tillexempel går det att se att ny statistik behöver samlas in över de tabeller som satsen använder.
Se [12.2] samt bilaga A [6.3] för mer information om verktyget.

4.5.3 Oracle expert

Oracle expert är ett grafiskt verktyg som ingår i ett prestandapaket från Oracle som heter Tuning pack.

Oracle expert är ett lättanvänt verktyg som använder sig av automatisk inhämtning av data från databasen. Med hjälp av dessa data skapas rekommendationer som användaren kan använda för att förbättra prestanda på sin databas. Rekommendationerna kan enkelt användas direkt från verktyget i de script som automatiskt skapas av verktyget.

Se kapitel [12.5] samt bilaga A [6.5] för mer information om Oracle Expert.

4.5.4 Statspack

Statspack är ett verktyg som används under körning för att samla in data om databasen. Statspack använder sig av databasens statistiktabeller för att skapa sin rapport.

För att skapa en rapport görs två såkallade snapshots. Den data som kommer att synas i rapporten kommer att komma från tiden mellan två snapshots. Rapporten innehåller en mängd data om vad som skett under tiden mellan två snapshots. Den visar hur de olika minnespoolerna användes, SQL som exekverades, antalet kompileringar, antalet commit och mycket mer.

Verktyget är bra att använda som en historik. Genom att kontinuerligt göra snapshots skapas en sorts historik av hur systemet ser ut i normalläge. Om prestandaproblem uppstår kan denna historik användas för att se vilka skillnader som finns mellan normalläget och problemet. Detta kan göra det enklare att snabbt finna problem och åtgärda dessa.

Se kapitel [12.2] samt bilaga A [6.6] för mer om hur Statspack används.

4.6 Testmiljö

Databasen är som redan tidigare nämnts från tillverkaren Oracle. Versionsnummer på den databas som har använts för testerna i detta arbete är 9.2.0.4.0 och den är konfigurerad som en OLTP-databas. Maskinen eller servern som databasen körs på är en Sun Fire 480 med 4 GB internminne. Operativsystem är Solaris 5.9.

Påverkan på testerna utifrån som tillexempel fördröjningar i nätverk är minimala med tanke på att all exekvering sker på samma system. Dessa faktorer har dock inte testats då de inte ligger inom ramen för arbetet.

Flera användare kan samtidigt vara inloggade och använda databasen. Dessa användare tillför väldigt lite arbete på maskinen då de inte utför några längre eller avancerade uppgifter. De flesta tester som gjorts har gjorts med endast en användare inloggad.

4.7 Testdata

För att kunna testa verktygen samt adaptern och kunna skapa en metodik måste någon form av testdata användas. I detta fall består testdata av XML-filer vilka symboliserar delar av ett nätverk.

Ett exempel på en XML-fil kan vara:

```
<node name="Cisco 12008" ip_address="127.0.0.1">
          <card name="STM OC-196" ip_address="123.123.123.123" />
          <card name="STM OC-196" ip_address="123.123.123.124" />
          <card name="STM OC-196" ip_address="123.123.123.125" />
</node>
```

Ur denna struktur hämtas data och skickas till databasen.

Som tidigare nämnts måste också databasen lämna någon sorts data som verktygen kan användas på, se [4.4].

5 Metod

För att kunna göra strukturerade tester på databasen måste en metod tas fram som gör att testerna kan upprepas och förvandlas till information som kan användas i en förbättringsprocess. Det är också viktigt att en struktur finns så det är enkelt att se vilka tester som gjorts, hur de gjordes och under vilka förutsättningar de gjordes. Detta för att senare använda detta material även i ett helt annat eller liknande system.

5.1 Grundläggande metodik

Genomförandet av tester kan göras på en rad olika sätt beroende på förutsättningarna. Det grundläggande är att först veta och förstå vilka förutsättningarna är. För att komma fram till detta kan en del enkla frågor ställas som tillexempel:

- Kan databasen förändras beroende på vad som hittas under tester?
- Kan kod som använder sig av databasen förändras?
- Hur ser avtal med tredjepartsleverantörer ut?

Genom att kunna sitt system och databas är det också möjligt att få fram de förutsättningar som gäller. Om databasen bygger på en arkitektur levererad av tredje part så kan i vissa fall testerna endast användas till att uppmärksamma leverantören om fel eller brister som hittas.

Den grundläggande iden till metodik är att det finns ett prestandaproblem som skall lösas. Utifrån denna ide byggs ett troligt scenario över vad som händer upp. Genom att sedan prova olika lösningar kan skillnader mellan de olika lösningarna hittas och vidare utvecklingar av lösningarna prövas. Hela arbetet är en sorts iterering tills en tillfredsställande lösning har hittats.

Att göra mätningar under test är viktigt för att kunna återupprepa sina tester och för att ha bevis för att en viss förändring verkligen ger bättre prestanda. Speciellt när det gäller inställningar på databasen så är testning synnerligen viktigt så att arbetet inte blir en ren gissning. Att kunna återupprepa det man tidigare gjort och applicera det på liknande problem är grunden till allt som kan liknas vid vetenskapligt arbete.

Tester och prestanda skall finnas med i planen för varje system som utvecklas. Prestanda är ingenting som tillkommer i slutet av ett projekt. Prestanda skall vara en del i både design och implementationsfasen av ett projekt. Genom att tidigt förstå och testa ett systems prestanda är det möjligt att undvika att sent i projekt bli tvungen att införa ändringar i systemet på grund av prestandaproblem. Att tidigt ha med detta i processen spar alltså både tid och pengar.

Se [12.1], [12.4], [12.6] samt [12.7] för mer information angående de olika tillvägagångssätt och metoder som kan användas.

5.2 Data

De data som används då tester skall genomföras bör likna data som kommer att användas i ett produktionssystem. Framförallt bör mängden data vara minst lika stor eller större än i ett tänkt produktionssystem då framförallt stora mängder data ofta är svårare att hantera än små.

Delar av databasen som kan påverkas av hur representativ de data som finns i ett system under tester är tillexempel:

Användning av index
Användningen av index styrs av hur data i tabellen ser ut. Om tester görs och datamängden tillexempel inte stämmer kan användaren tro att ett index är oanvändbart eller onödigt.

Exekveringsplaner
Exekveringsplaner kan förändras beroende på hur data ser ut.

Exekveringstid
Stora skillnader i exekveringstider gentemot ett verkligt system kan uppstå om tester med framförallt felaktiga mängder data görs.

Minnesanvändning
Minnesanvändningen i ett system med lite data kan vara mycket mindre än ett system med stora mängder data.

Att ha rätt mängd data i systemet kan också hjälpa till att klargöra vilka krav som kommer att ställas på övriga komponenter i systemet samt hårdvaran som systemet kommer att användas på.

Behandlingen och bearbetningen av data bör ske på ett sådant sätt att det liknar ett verkligt scenario.

Ett testsystem som inte innehåller representativa data till form eller mängd kan ge helt felaktig information till den som testar systemet.

Det är alltså viktigt för testerna att den som testar har tillgång till representativ data. Det är också viktigt att samma data senare kan användas för att göra liknande tester för att se hur förbättringar eller ändringar påverkar processen.

5.3 Genomförande tester

För att genomföra tester är det viktigt att precisera vilket område som skall testas. Att göra tester av alla funktioner i databasen i ett enda stort test när det handlar om förbättringar är ingen bra ide då mängden data och arbetet med att sortera denna kan bli enormt.

Det enklaste sättet är som sagt att rikta in sig på ett område. Testerna kan delas upp i huvudområden enligt [4.3]. Dessa huvudområden kan i sin tur delas upp i underkategorier för att strukturera och förenkla upprepning vid senare tester.

Om det tillexempel är ett Java-program som innehåller SQL kod som kommer att exekveras kan testerna inriktas mot en speciell del av programmet, en klass eller en funktion.

5.3.1 Val av verktyg

Valet av verktyg görs ofta genom den typ av test som skall genomföras. I vissa fall är dock verktygen gjorda för att kunna användas på mer än ett område och då är det bra om användaren vet om ett specifikt verktyget kan ge värdefull information vid testet..

För tester av SQL så är Tkprof och Explain plan egentligen de enda två verktyg som behövs. Utifrån informationen från dessa verktyg går det att förändra koden och databasen så den presterar maximalt under de förutsättningar som finns.

För tester av databasen generellt såsom inställningar, lagringsstruktur och annat är oftast informationen lite mer fragmentarisk. Oracle expert är ett enkelt verktyg som ger snabba svar även för användare med begränsade förkunskaper. Verktyget är dock väldigt beroende av att de tester som utförs speglar verkliga förhållanden för att inte ge felaktiga rekommendationer.

Statspack är ett bra verktyg för att få fram information om parameterinställningar och annat i databasen. I vissa fall går det också att utläsa från Tkprof vilka inställningar som kan göra så databasen får högre prestanda.

Att få fram information om instansen och hur den ska förändras ställer ofta lite högre krav på användarens kunskaper om databasen då det ofta är viktigt att förstå hur saker påverkar och påverkas av vissa inställningar. Med lite erfarenhet kan det vara fullt möjligt även för en vanlig användare att göra vissa inställningar som kan förbättra prestanda till viss del.

Valet av verktyg styrs av dels vilka tester som skall utföras samt vilken information som dessa tester skall generera. Följande kapitel ger några exempel på hur tester kan genomföras.

5.3.2 Utförande applikationstester

För att testa en applikations prestanda vid tillexempel problem vid en funktion kan följande enkla metod användas:

1. Förbered test, starta tillexempel skapandet av spårfiler om det verktyg som valts kräver detta.
2. Genomför en exekvering av applikationen som problemet finns i.
3. Använd valt verktyg för att få information om vad som sker i databasen under körning.
4. Analysera problemet och ta fram en eller ett antal möjliga lösningar på problemet.
5. Implementera den lösning som troligen fungerar bäst.
6. Börja om vid punkt ett.

Denna iterering kan sedan pågå tills en lösning som är tillfredställande har hittats. Det är inte alltid den första lösningen som är den bästa så det är viktigt att testa i alla fall ett par olika lösningar om inte problemet är väldigt trivialt.

För att få en mall att använda kan det vara bra att göra ett användarfall för de scenarion som skall testas. Detta innebär de operationer som ska göras skrivs ned och kartläggs för att sedan testas med avseende prestanda. Det är då enkelt att ta fram det användarfall som användes för att göra om testet med en ny metod.

5.3.3 Utförande instans och lagringstest

Att utföra tester på instansen och på lagringsstrukturen är lite mer komplicerat än vad ett applikationstest är. Förändringar på instansen innebär ofta förändringar av parametrar i databasen och det är då viktigt att förstå vad dessa parametrar påverkar. Ofta kan ett värde ha en positiv effekt för en specifik funktion som databasen utför medan den för en annan kan ha negativ effekt. En grundläggande kunskap om hur databasen fungerar och hur de olika delarna arbetar tillsammans krävs för att förstå resultat och genomföra förändringar.

Testen görs i princip på samma sätt som ett test av en applikation. Test genomförs med applikation, förändringar genomförs och slutligen test igen för att se skillnader.

5.4 Exempel applikationstest

För att illustrera hur det kan gå till att genomföra ett test av en applikation genomförs här ett enkelt exempel.

Applikationen som skall testas är en enkel applikation som läser en rad i en tabell. Problemet är att läsningen uppfattas som långsam av användarna.

I applikationen finns en SQL-sats som ser ut som följande:

```
select * from big_table, small_table where
object_name='Test';
```

Genom att skapa en spårfil på denna SQL-sats och använda Tkprof på spårfilen genereras följande rapport:

```
select big_table.*
from
 big_table, small_table where big_table.object_name='Test'


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        4      0.17       0.17          0       8029          0          40
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6      0.17       0.17          0       8029          0          40

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 61

Rows     Row Source Operation
-------  ---------------------------------------------------
     40  NESTED LOOPS
     20   TABLE ACCESS FULL SMALL_TABLE
     40   TABLE ACCESS FULL BIG_TABLE
```

Det första som bör göras är att kontrollera att SQL-satsen har rätt syntax och verkligen ger det resultat som förväntas. En enkel kontroll av denna sats visar att tabellen `small_table` inte finns med i resultatet eller används någon annanstans i SQL-satsen. Denna tabell kan alltså tas bort. Resultatet blir då:

```
select big_table.*
from
 big_table where big_table.object_name='Test'


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0        402          0           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.01          0        402          0           2

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 61
```

```
Rows     Row Source Operation
-------  ---------------------------------------------------
    2   TABLE ACCESS FULL BIG_TABLE
```

Detta exekverar nu snabbt och uppfattas som väldigt snabbt av användaren. Det går dock att se värdet 402 för kolumnen query. Detta innebär att för att få fram resultatet måste databasen läsa 402 block. Vidare går det att se att exekveringsplanen i slutet av rapporten innebär en såkallad full scan på tabellen big_table. För att undvika detta skapas ett index.

Ett index skapas på den kolumn som används i where-satsen på SQL-satsen, `object_name` i detta fall, och då genereras följande resultat:

```
select big_table.*
from
 big_table where big_table.object_name='Test'


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          5          0           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.01          0          5          0           2

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 61

Rows     Row Source Operation
-------  ---------------------------------------------------
0      SELECT STATEMENT
 1       TABLE ACCESS (BY INDEX ROWID) OF 'BIG_TABLE
   2        INDEX (RANGE SCAN) OF 'BIG_TABLE_INDEX' (NON-UNIQUE)
```

Query-kolumnen har nu ett värde av 5 så databasen behöver nu endast läsa 5 block för att läsa informationen. Det är också tydligt att sökningen nu bara tar 0.01 sekund medan det i det första exemplet tog 0.17 sekunder.

I kapitel 12 referens 3 och 4 går det att läsa vilka tekniker som använts och teorin som ligger bakom exemplet, tillexempel hur man aktiverar spårning och hur index skall användas..

5.5 Exempel instanstest

För att genomföra detta instanstest används verktyget Statspack.

Detta test kommer att vara väldigt enkelt och endast för att visa hur det kan gå till då en Statspack-rapport skapas.

Först skapas den första snapshoten, detta görs direkt i SQL*Plus tillexempel:

```
SQL> exec statspack.snap;
```

```
PL/SQL-procedur õr utf÷rd utan fel.
```

Efter detta exekveras applikationen, i detta fall hämtar applikationen all data ur tabellen big_table.

När applikationen exekverat klart skapas ännu ett snapshot enligt ovan. Med hjälp av dessa två snapshots kan nu en rapport genereras:

```
SQL> @ spreport

Current Instance
~~~~~~~~~~~~~~~~
♀   DB Id    DB Name     Inst Num Instance
----------- ------------ -------- ------------
 1873217334 TEST                1 test

Instances in this Statspack schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
----------- -------- ------------ ------------ --------
 1873217334        1 TEST         test         TEST

Using 1873217334 for database Id
Using          1 for instance number

Completed Snapshots

                            Snap                     Snap
Instance     DB Name          Id   Snap Started    Level Comment
------------ ------------ ----- ----------------- ----- -------
test         TEST             1 26 Sep 2005 15:00     5
                              2 26 Sep 2005 15:06     5


Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Ange võrdet f÷r begin_snap: 1
Begin Snapshot Id specified: 1

Ange võrdet f÷r end_snap: 2
End   Snapshot Id specified: 2


Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is sp_1_2.  To use this name, press
<return> to continue, otherwise enter an alternative.
Ange võrdet f÷r report_name: test.txt
```

Detta skapar en rapport med en hel del information om hur användningen av databasen såg ut, delar av rapporten kan ses nedan:

STATSPACK report for

DB Name	DB Id	Instance	Inst Num	Release	Cluster	Host
TEST	1873217334	test	1	9.2.0.1.0	NO	

	Snap Id	Snap Time	Sessions	Curs/Sess	Comment
Begin Snap:	1	26-Sep-05 15:00:59	9	5.2	
End Snap:	2	26-Sep-05 15:06:51	9	5.9	
Elapsed:		5.87 (mins)			

Cache Sizes (end)
~~~~~~~~~~~~~~~~~

Buffer Cache:	24M	Std Block Size:	8K
Shared Pool Size:	48M	Log Buffer:	512K

Load Profile
~~~~~~~~~~~~

	Per Second	Per Transaction
Redo size:	1,842.35	216,169.33
Logical reads:	24.48	2,872.67
Block changes:	10.09	1,183.33
Physical reads:	1.16	136.67
Physical writes:	0.03	3.67
User calls:	5.74	673.67
Parses:	0.94	110.00
Hard parses:	0.07	8.00
Sorts:	0.51	59.67
Logons:	0.01	0.67
Executes:	1.86	218.33
Transactions:	0.01	

% Blocks changed per Read:	41.19	Recursive Call %:	51.94
Rollback per transaction %:	0.00	Rows per Sort:	46.02

Instance Efficiency Percentages (Target 100%)
~~~~~~

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	95.24	In-memory Sort %:	100.00
Library Hit %:	93.62	Soft Parse %:	92.73
Execute to Parse %:	49.62	Latch Hit %:	100.00
Parse CPU to Parse Elapsd %:	45.83	% Non-Parse CPU:	82.81

Shared Pool Statistics	Begin	End
Memory Usage %:	40.48	41.81
% SQL with executions>1:	71.06	76.07
% Memory for SQL w/exec>1:	53.30	70.24

Top 5 Timed Events
~~~~~~~~~~~~~~~~~~

Event	Waits	Time (s)	% Total Ela Time
control file sequential read	86	1	34.68
CPU time		1	28.05

```
db file scattered read                    35          0    20.15
control file parallel write              114          0     9.03
db file sequential read                   13          0     7.13
...
...
```

Ur denna information kan användaren utläsa hur mycket minne de olika poolerna använt, hur mycket data som skrivits till databasens filer. Det finns även en del råd i rapporten som talar om för användaren vad som kan vara en lämplig förändring för att förbättra sitt system.

För mer djupgående information om hur rapporter och data kan tolkas se bilaga A [6] – [8].

6 Slutsats verktyg

Av de fyra testade verktygen var det tre som fungerade riktigt bra och kunde användas effektivt för att göra tester.

Dessa tre var Tkprof, Explain plan och Statspack.

De två verktygen Tkprof och Explain plan hör mer eller mindre ihop då man använder de på ungefär samma sätt och för samma sorts tester. I många fall kompletterar dessa verktyg varandra genom att visa på skillnader mellan verklighet och statistik. Tkprof är ett enkelt verktyg som kan användas för att kontrollera all SQL som utvecklarna skriver. Det är enkelt att se vilken SQL som kan behöva konstrueras om för att uppnå den prestanda som man hade tänkt sig.

Statspack är bra att använda för att skapa sig en historik över vad som händer med databasen i det långa loppet. Man kan enkelt schemalägga verktyget så det med jämna mellanrum tar sina ögonblicksbilder. Dessa kan vid senare tillfällen plockas fram för att användas i arbete med prestandaproblem.

Oracle Expert var det verktyg som tillförde minst av verktygen. Verktyget är dock väldigt enkelt att använda då kunskapsnivån för att just använda verktyget inte behöver vara så hög vilket kan vara positivt. Dock så bör den som inför förändringarna som verktyget föreslår förstå hur detta kan påverka databasen. Verktyget ingår i ett paket som kostar en hel del pengar och i nästa version av databasen finns inte verktyget kvar längre.

Slutsatsen blev att den som skall lära sig att genomföra tester bör lära sig och använda verktygen Tkprof, Explain plan och Statspack. Dessa finns nästan alltid tillgängliga i alla installationer av databasen och är ovärderliga i prestandaarbetet.

7 Resultat

Resultatet av arbetet har blivit en manual som personal på Ericsson skall kunna använda för att utföra enkla tester på databasen. Den har också en del generella samt specifika tips på hur produkten som utvecklas på inventeringsavdelningen kan förbättras. Se bilaga A.

Delar av resultaten har också använts i systemet för att förbättra prestanda på laddning av data i databasen. Framförallt är det då förändringar i den SQL som systemet använder som ändrats för att ge högre prestanda.

8 Problemutvärdering

Svårigheter och problem under arbetets gång har lösts dels med hjälp av vidare litteraturstudier men även med viss hjälp av personal på Ericsson. Det svåra med ett liknande arbete är att det är väldigt svårt att innan själva arbetet startar veta vilka teorikunskaper som krävs. Detta är någonting som till viss del växer fram under arbetets gång då ny problem hittas eller uppstår. Att lära sig allt inom detta område är i princip omöjligt då det tillexempel finns över 150 manualer från tillverkaren av databasen.

Arbetet och problemen som hörde till arbetet kändes väl avgränsade och det fanns hela tiden ett mål mot vilket arbetet var riktat mot. Detta var bra eftersom planeringen av arbetet blev väldigt enkel då slutmålet var väldefinierat.

Uppföljningar av arbetet utfördes kontinuerligt av personal på Ericsson som en del av förbättringsarbetet av produkten och för att få mer kompetens på området.

De största mer praktiska problemen under arbetets gång har varit:

- Testdata
- Delad miljö
- Databasmodell
- SQL

Testdata
De testdata som fanns tillgänglig var alltför liten för att göra kunna göra riktigt stora tester. I början av arbetet fanns tillräckligt med saker att göra och förbättra utan att använda stora mängder data men senare kom behovet att göra försök med större mängder. Detta löstes genom att skriva ett litet program som skapade testdata i tillräckliga mängder. Vid avslutningen av det egentliga arbetet kunde dessa tester genomföras och fler problem blev då också uppenbara.

Delad miljö
Arbetet genomfördes i en delad miljö vilket innebar att andra jobbade samtidigt med systemet. Deras arbete påverkade inte testerna men vissa saker som skulle testas kunde fördröjas för att inte påverka det dagliga arbetet. Detta gjorde att vissa saker ibland fick planeras ganska noga när de skulle genomföras. Senare delen av arbetet var större delen av personalen på semester så då var problemet mer eller mindre obefintligt.

Databasmodell
Att förstå och lära sig databasmodellen som hela arbetet byggde på var ett av de allra första problemen. Att se alla kopplingar och tänka igenom all lösningar som gjorts var ett ganska så stort arbete. Modellen var egentligen inte svår att förstå utan det var antalet relationer och tabeller som gjorde att det till en början kändes som en ganska övermäktig uppgift. Detta klarnade eftersom men en fördel hade varit om en del av detta arbete kunnat genomföras i förväg för att påskynda starten av själva arbetet.

SQL
En del av den SQL som användes var svår att förstå speciellt då vissa SQL-satser kan fylla flera vanliga A4-sidor. Att lära sig de tekniker och funktioner för SQL som finns i databasen har varit den absolut största svårigheten med arbetet. De grundläggande kurserna som behandlat SQL på universitet har lagt en god grund men det finns en hel del mer att lära sig.

9 Diskussion

Arbetet har fungerat väldigt bra och mer eller mindre följt den tidplan som ritades upp innan början på arbetet. Vissa motgångar är det alltid då man arbetar med saker som till stora delar är nytt men i det stora hela har det fungerat bra.

Det har också varit väldigt enkelt att få hjälp på Ericsson då alla som arbetat med projektet finns väldigt nära till hands. Frågeställningar om kod och tankegångar bakom hur saker har gjorts har oftast varit väldigt enkla att få förklarade för sig.

Tidigt i arbetet märktes att en teoretisk bakgrund att stå på vad det gäller databasen är väldigt viktigt för att inte komma in på sidospår som inte leder någonstans. Manualer, forum samt diskussiongrupper är ett enkelt sätt att inte falla i samma grop som andra redan fallit i. Det finns otroligt mycket information på Internet och i manualerna.

En grundläggande förståelse av SQL och vad som är viktigt att tänka på vid skrivning av SQL är väldigt viktigt. Ofta är det väldigt enkelt att dra en logisk slutsats om varför en sats inte presterar bra och rätta till felet. I vissa fall är det en aning svårare men efter att i ord skrivit ned vad SQL-satsen skall göra är det ofta enkelt att konstruera en ny, snabbare sats som förbättrar prestanda många gånger.

Av ett arbete som detta lär man sig också mycket om systemet man jobbar med och man får också en inblick i hur det går till i lite större, verkliga, projekt. Att arbetet också var någonting som man praktiskt kunde använda sig av på Ericsson var nog också en fördel eftersom folk intresserade sig för arbetet som utfördes på ett annat sätt än om resultatet endast hade varit en lång rapport efter tre månader.

10 Förkortningar

XML – eXtensible Markup Langugar
DTD – Data Type Definition
SQL – Structured Query Language
API – Application Programming Interface
OLTP – OnLine Transaction Processing

11 Referenser

[1] *Database Performance Planning* (2001). Andrew Holsworth: Oracle Corporation. Part No. A96532-01.

[2] *Database Performance Guide and Reference* (2001). Michele Cyran: Oracle Corporation. Part No. A87503-02.

[3] *Database Administrator's Guide* (2002). Ruth Baylis: Oracle Corporation. Part No. A96521-01.

[4] *Database Concepts* (2001). Lenore McGee Luscher: Oracle Corporation. Part No. A88856-02.

[5] *Data Tuning with the Oracle Tuning Pack* (2001). Lisa M. Jamen: Oracle Corporation. Part No. A86647-01.

[6] *Designing and Tuning for Performance* (1999). Michele Cyran: Oracle Corporation. Part No. A76992-01.

[7] *Tuning Third-party Vendor Oracle systems* (2003). Mike Ault: Rampant Techpress. ISBN 0-9740716-3-3.

Bilaga A – Tuning Guide

Prepared (also subject responsible if other)
EAB/Z/SNN Gustaf Persson

No.

Approved | Checked | Date 2005-08-12 | Rev PA1 | Reference

Oracle Tuning guide for NIM

1 Introduction

In this paper tuning of the NIM database is explained. The database is running Oracle9i (9.2.0.4, OLTP) on a Sun Solaris platform. The parts considered mostly are the work that can be done with the tools Tkprof, Explain plan, Oracle Expert and Statspack to improve the performance of the database. The tuning is a little crippled by the fact that the NIM system uses third party software. Therefore some of the improvements to the system might not be possible to perform.

1.1 Scope

This document covers most of the testing and tuning that should be done during development. It is also a guideline for administrators on the field to use if there are performance problems in a NIM system in production. The tuning is concentrated on the Oracle database and not the operative system or physical details that can affect performance on the database, e.g. disk, memory, processors.

There is also another restriction, the system relies on a third party application; that is code that cannot be changed. This third party product builds the database; database changes are also under restrictions.

1.2 Audience

Developers, testers and system administrators are the main audience for this document. Basic knowledge in the Unix and Oracle environment is needed. Basic SQL knowledge is also required, especially for the application tuning.

1.3 Typing conventions

System elements

Command and parameter names, program names, parameters, paths, URLs and directory names are written in Courier.

For example:

The trace files are stored in:
`/opt/oracle/product/9.2.0.4/admin/alphadb/udump`

User input

A command that must follow an exact syntax is written in bold courier.

Example:

```
ls -l | grep ora
```

Command variable

Bold italic courier indicates user input.

Example:

```
SQL> grant alter session to cramer_user;
```

Tool parameters

Non-mandatory parameters are inside brackets.

Example:

```
explain plan [into plan_table] for select * from dual;
```

Can be written as:

```
explain plan for select * from dual;
```

or

```
explain plan into my_plan_table for select * from dual;
```

GUI elements

Buttons in the GUI are inside vertical lines.

Example:

Press the |Cancel| button.

Information box

Grey boxes are especially valid for the type of reader that is written at the top of the box. Other readers are encouraged to read the boxes to as the boxes improve the understanding of the material.

Example:

Developer

Oracle is a database.

1.4 Variables

Table 1 is the main variables that are used in this guide. The last column can be used to enter the values for the current installation.

Variable	Description	Value
DBServer	Hostname of the Resource Manager database server.	
oracle_user	Unix user on the DBServer.	
ResourceDatabase	SID for the Resource Manager database.	
sys_password	System administrator password for the Resource Manager database.	
cramer_user	Oracle username used by the uploadadapter.	
perfstat_password	Password for the Statspack perfstat user.	
db_user	Normal user on the database used for testing.	
db_user_password	Password for the db_user.	
user_dump_dest	Path to trace files	

Table 1 Main variables.

1.5 Abbreviations

Abbreviations used in this document.

Abbreviation	Explanation
OLTP	OnLine Transaction Processing
SQL	Structured Query Language
I/O	Input/Output to and from disk drives
CPU	Central Processing Unit, Processor
XML	eXtensible Markup Language
JDBC	Java DataBase Connector
API	Application Programming Interface
GUI	Graphical User Interface
DBA	DataBase Administrator
DDL	Data Definition Language

Table 2 Abbreviations.

2 Tuning overview

Testing a database can be done in as many ways as there are databases. The common goal for all testing and tuning is that the database is going to be faster, more responsive and more effective.

Testing an Oracle database has three mayor parts. The first is Application tuning. This is testing the code that is executed on the database. The second part is the Oracle internals; hit rates, shared pool and such. The third and last is the storage. How the data files are written, I/O in general.

Often there is some third party software involved in the database. This is a problem when tuning the database because changes cannot be done without help from the third party vendor. When tuning a database with some third party software involved one might have to consider jumping a few steps (1-5 foremost) in the steps to not void any support or agreements made with third part. Tests might be done on the other parts also but the results might only be used to enlighten the third party and try to convince them that a change in the code is required.

There is also a question of when and where to stop the tuning. The tuning should have a goal set up. Worst performing code will always exists so there has to be something telling the person tuning the database when the goals are met and then to stop.

A database is under continuous change and therefore performance should always be monitored so that problems can be avoided at an early stage.

The golden rule is: **don't try to fix what is not broken**. If a program is following the requirements and the end users finds the database responsive and well functioning then there is generally nothing to fix.

3 NIM adapter and third party

The NIM adapter loads the database with data. The NIM adapter is written in Java and uses a JDBC connection to execute SQL on the database.

The NIM adapter uses XML files, which holds the information about the network. These files are parsed into temporary tables from which they then are inserted to the resource database using both the Cramer API and SQL statements. The working of the adapter and the system is explained by further down and by figure 1.

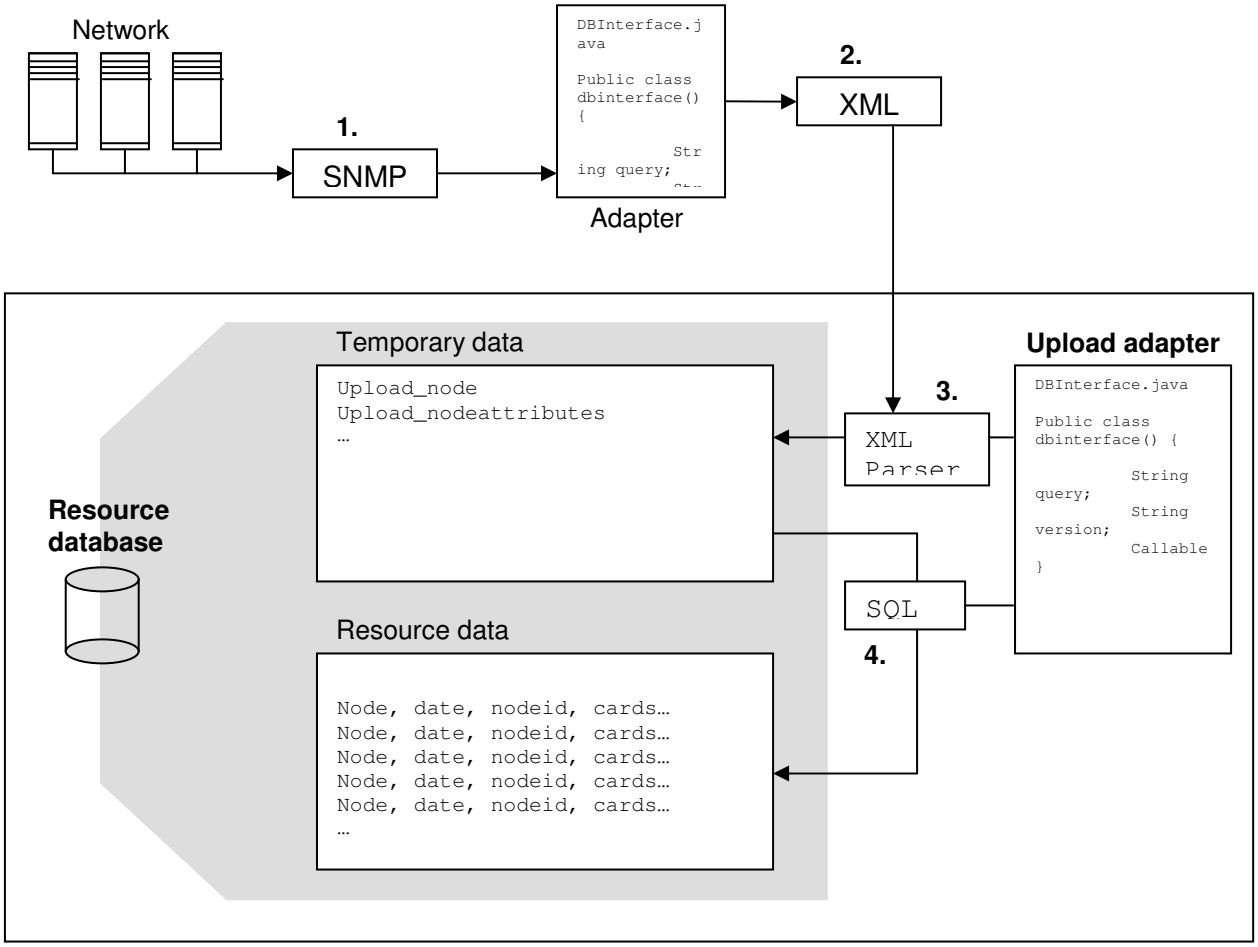


Figure 1 NIM System.

This picture shows a conceptual picture of how the NIM system works. The NIM system can be described with figure 1 and the following explanation.

1 Information from the network is retrieved in some way, say SNMP.

2 An adapter changes this information into an XML-file and the file is saved on the server in the customer network.

3 The information from the XML-file is parsed to temporary storage in the resource database by a parser in the upload adapter. In this stage mapping files are used to create unified names of the nodes.

4 The upload adapter inserts the information from the temporary storage into the resource database.

Cramer is the third party application that is partly used for inserting data into the database. This means the design and the code used by many of the functions utilized cannot be altered. The only thing possible is to give advice to the third party when a problem is found and let them take a decision to whether there is a problem or issue and so create a solution.

Changes in the database like indexes and such is also a part of the third party solution because the design of the resource database is part of the Cramer application and all changes should be cleared with them before implementation.

All this creates a situation where more tuning might have to be done on the instance instead of on application because the control of the application much lies on third part.

4 Methodology

Performance problems are often addressed to the administrator by the end users. There is often information like that the application is slow, it hangs or other not so very specific information.

First step is to figure out where the problem lies. Setting up a user case and collect data during the case can help in this work. The data collected can be used in tools or interpreted in any other way to tell the administrator what the problem is.

When the problem is found a solution must be constructed. The solution can involve changing the database internals, change the application or changing the actual hardware for example.

If there are more than one solution they has to be graded somehow so that the most likely best solution is tested or implemented first. All solutions should be tested on a non-production system before they are implemented so that they do not have any unknown side effects.

When a solution is tested and found to be working it can be implemented in a production environment. If the solution is a solution also in the production environment the problem is solved. To know that a solution is really solving something it is important to measure before and after to have some evidence of a change. If the solution did not work go back and take another possible solution and test again or collect more data and see if something was overlooked the first time.

The evidence part of the solution is always important because it removes the guessing if the same problem arises again later or somewhere else. Evidence can be a performance report from before and after or a clocked user case with a before and after result.

Finding out what is wrong is always easier if the administrator or DBA knows his system. If for example the end users complain about long response times and the administrator checks and they are the same as last week it can be concluded that nothing dramatic has happened to the system. Even tough it should be seen as a bad sign that end users start to complain about the application.

History of what is normal is often a good tool to have when trying to find what is causing bad performance. It is easier to see what the problem could be when there is some possibility to compare the normal situation with the bad situation.

5 Create trace files

In order to be able to do a tuning session on the database Oracles SQL trace facility must be enabled. Also the option to use timed statistics should be enabled.

The data obtained from the trace facility contains:

- Parse, execute, and fetch counts.
- CPU and elapsed times.
- Physical reads and logical reads.
- Number of rows processed.
- Misses on the library cache.
- Username under which each parse occurred.
- Each commit and rollback.

This data is used in the tuning process to see how resources are used and if there are room for improvements.

Errors on the command line means something did not work. Check the reference text again and see that the command was entered correctly.

5.1 Enable SQL trace for session

Because of the behavior of the program it is not possible to trace only one session because there are a number of different sessions working during one upload. To avoid this problem a trigger can be created that starts tracing for a specific username.

1 Log on to the server using the command:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database using the commands:

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect sys/sys_password@ResourceDatabase as sysdba;
```

3 To create the trigger write the following in the console:

Note: `cramer_user` must be in uppercase. If there are compilation errors redo the input. If you make syntax error while entering data end the input with / and replace the trigger.

```
SQL> create [or replace] trigger CRAMER_USER.trace_trigger
  2> after logon on database
  3> begin
  4> if(user='CRAMER_USER') then
  5> execute immediate 'alter session set
timed_statistics=true';
  6> execute immediate 'alter session set sql_trace=true';
  7> end if;
  8> end;
  9> /
```

4 The cramer user has to be able to run the trigger at logon:

```
SQL> grant alter session to cramer_user;
```

The system is now ready for testing. Perform a system test by using all different parts of the system.

5.2 Finding the SQL trace files

The trace files are saved in a directory that is specified in the Oracle environment. To find out what directory that is use the following procedure.

1 Log on to the server using the command:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database using the commands:

```
oracle_user@DBServer> sqlplus /nolog
```

```
SQL> connect sys/sys_password@ResourceDatabase as sysdba;
```

3 To find out what the path is, issue the SQL statement:

```
SQL> select value from v$parameter where
name='user_dump_dest';
```

4 Exit SQL*Plus using the command:

```
SQL> exit
```

5 Change directory to the directory found in step 3:

```
oracle_user@DBServer> cd /value/from/step/three
```

6 View the files in the directory and sort them with `grep` or similar:

```
oracle_user@DBServer> ls -l | grep 'Jun 17'
```

The trace files can be renamed to be easier to manage. Trace files no longer used can be discarded.

For reading trace files see [6.1].

5.3 Disable SQL trace for session

After all the testing is done the SQL trace for the cramer user should be disabled to not affect performance.

1 Log on to the server using the command:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database using the commands:

```
oracle_user@DBServer> sqlplus /nolog
```

```
SQL> connect sys/sys_password@ResourceDatabase as sysdba;
```

3 To disable the trigger use the command:

```
SQL> drop trigger CRAMER_USER.trace_trigger;
```

4 If no more tracing is going to be done the privilege granted to the user can be revoked using command:

```
SQL> revoke alter session from CRAMER_USER;
```

Now all tracing is stopped and the database is in its initial state.

5.4 Database performance impact

By using the trace facility the machine running the database is of course affected considering performance. The trace consumes CPU time, I/O and other resources and thus affects the performance on the things being tested. But without this data there is no possibility at all to find the underlying problems and also, if the database is heavily loaded already, the overhead introduced by using the trace facility can be neglected.

Also the timed statistics is creating more stress on the machine but this is very valuable information in the tuning process. The same arguments stand for this as for the trace facility itself. It is possible to do without the timed statistics but they contain valuable information. The overhead introduced by the timed statistics is less than five percents. A tuning without enough information will become a less effective tuning so using as much information as possible is always desirable.

The timed statistics is as standard enabled on an Oracle database; controlling the `statistics_level` parameter can show this, see [11.4].

6 Tuning tools

The Tkprof, Explain plan, Statspack and Oracle Expert tools are fairly easy to use and the results from them are rather easy to interpret. The tools Tkprof, Explain plan and Statspack are installed on a standard installation of Oracle. Oracle Expert is part of the tuning pack that is available from Oracle. The tools covers the whole scope of tuning one can do on an Oracle database.

The trace files created by Oracles trace facility must be transformed into a readable format. Tkprof can do this transformation.

Testing should always be done with an account other than `sys`. `sys` should be used for starting, shutting, restoring and other administrative tasks like doing grants, not for testing. This is because the `sys` user does not work like a normal user and therefore should not be used as one. Also, do not change anything in the `data dictionary` if not told to do so by Oracle support. Changes in the dictionary can be a good way to crash the database. Create accounts with DBA privileges instead of using the `sys` account.

6.1 Using Tkprof

To use Tkprof we need to know where the files from the trace facility are stored. To get that path see [5.2].

6.1.1 Tkprof command

1 Log on to the server using command:

```
rlogin DBServer -l oracle_user
```

2 Use the instructions in section [5.2] to find the trace files.

3 To use the Tkprof tool on the data use the command:

```
oracle_user@DBServer> tkprof trace_file output_file
[print=number][sort=parameter][sys=yes/no]
```

See [11.1] for more parameters and information.

4 To view the file created use the command:

```
oracle_user@DBServer> more output_file
```

6.2 Reading the Tkprof result

Information on how to read the results from the tool see [7.1].

6.3 Using Explain plan

Explain plan only uses the SQL statements so no trace file is needed. The trace files transformed by Tkprof can be used as a starting point to find the interesting statements.

Explain plan only tests a statement, in other words it never alters the database in any way. In this way the plan from Explain plan can be seen as the expected execution and the explain plan in the Tkprof report is how it ended up in reality.

6.3.1 Explain plan command

1 Log on to the server using the command:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database using the command:

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect db_user/db_user_password@ResourceDatabase
```

3 Use the command on the selected SQL statement:

```
SQL> explain plan for SQLstatement;
```

See [11.2] for more information and parameters. Also see [11.4] if no plan table exists.

4 To view the Explain plan for the statement use:

```
SQL> select * from table(dbms_xplan.display);
```

6.3.2 Explain plan in SQL*Plus

There is an easy way to view the plan of a statement directly when it is entered in SQL*Plus:

1 Log on to the server using the command:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database using the command:

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect db_user/db_user_password@ResourceDatabase
```

3 Set autotrace to on:

```
SQL> set autotrace on explain;
```

If a statement now is written in SQL*Plus the explain plan will be shown. See [11.4] if no plan table exists.

4 To turn off autotrace use:

```
SQL> set autotrace off;
```

There are more uses for the autotrace command. See Chapter 11 in Oracle9i Database performance tuning guide, page 391.

6.4 Reading the Explain plan result

Information on how to read the results from the tool see [7.1].

6.5 Using Oracle Expert

Oracle Expert is the simple to use GUI tool that an administrator or developer can use to find problems and optimizations in the database. The tool gives the user notifications on problems and also solutions to them in the form of scripts or instructions. These can later be implemented to make the database perform better.

Oracle Expert can be used both for tuning purposes and during development.

6.5.1 Preparation

To use Oracle Expert a user with the `select_any_table` privilege is needed. The easiest way to do this is to create a user and give that user DBA privileges.

1 Log on to the resource database server:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database:

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect sys/sys_password@ResourceDatabase as sysdba
```

3 Create a new user to be used by the Oracle Expert tool:

```
SQL> create user USERNAME identified by USERPASS
  2  default tablespace tools;
```

Note: If the `TOOLS` tablespace does not exist, choose another tablespace but do not use `SYSTEM`.

4 Give the new user privileges that is sufficient to run the tool:

```
SQL> grant dba to USERNAME;
```

Now the user has sufficient privileges to use the Oracle Expert tool.

6.5.2 Starting the tool

The Oracle Expert tool is installed with the Oracle tuning pack. If the tuning pack is installed on the server the tool can be found directly on the server. If the tuning pack is not installed on the server the tool can be executed on a PC with the tuning pack installed and executed against the database over the network. It can conduct application, instance and space management tuning.

6.5.2.1 Unix

To bring up the enterprise manager console on a Unix server use commands:

1 Log on to the resource database:

```
rlogin DBServer -l oracle_user
```

2 Start the Enterprise Manager Console:

```
oracle_user@DBServer> oemapp console
```

3

Figure 2 Enterprise manager login.

The enterprise manager console starts, choose launch standalone (see figure 1) and press |OK|.

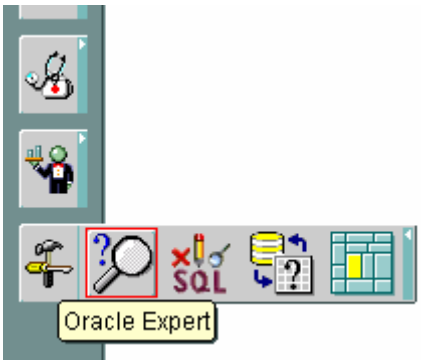


Figure 3 Oracle Expert pop-out.

Check in the toolbox in the left down corner and see if Oracle Expert can be found in the pop-out menu (see figure 2). If not, it is not installed and the tool has to be executed from somewhere else or installed.

6.5.2.2 Windows

If the tool has been installed on a PC the program is started by:

Start-> Programs-> Oracle - databasename**-> Enterprise Management Packs-> Tuning-> Oracle Expert.**

If not found the tool is not installed. The tool has to be executed from somewhere else or installed.

6.5.3 Logging on

At startup of the tool a window appear and the user is asked whether to log on to a management server or to a standalone repository. If a management server exists it can be used and the user credentials are then the existing management user. If there is no management server the user chooses standalone repository.

The information asked from the application is:

- Username
- Password
- Service

Username and password is from the user created in [6.5.1]. Service is the connection string to the database being examined. A connection string is constructed like:

host*:*port*:*sid

For example:

cramer.ericsson.se:1521:ResourceDatabase

or

133.24.196.201:1521:ResourceDatabase

When the connection is made for the first time a question if a repository should be created on the user is asked. Answer yes on this question. A repository is built for the user. This repository is used for storing data retrieved while using the tool.

6.5.4 Create tuning session

1 There is no need to use the tuning session wizard, press |Cancel|.

2 If the database you are going to tune is in the tree to the left jump to step 5. If it is not in the tree mouse right-click in the left area and choose new from the menu to create a new connection.

3 In the dialog add new database service that appears enter all the fields with the values:

Display name: ***Anyname***
Username: ***Username*** (as from [6.5.1])
Password: ***Password*** (as from [6.5.1])
Service: ***ResourceDatabase***
Host: ***DBServer***
Connect as: ***Normal***

4 Click |OK|.

5 Click the newly created connection in the tree to the left.

6 Mouse right-click 'Tuning Session' and choose new.

A new tuning session is now available.

6.5.4.1 Scope tab

In the window to the right there are now some parameters to consider. The tuning scope and the tuning session characteristics is the choices and values that help the tool in the tuning work. These parameters need to be set by the user to values corresponding to how the database is configured and what tuning that should be conducted.

Try to give as much information as possible to the tool before starting the tuning. This will help the tool in the process to create results that are useful.

A typical setting of the *'Tuning scope'* would be:

- Check for Instance Optimizations: ***Checked***
- Check for SQL Reuse: ***Checked***
- Check for Appropriate Space Management: ***Checked***
- Check for Optimal Data Access: ***Checked***
- Perform comprehensive index evaluation on tables referenced by worst performing SQL statements: ***Checked***

A typical setting of the *'Tuning session characteristics'* would be:

- Application Type: ***OLTP***
- Downtime Tolerance: ***None***
- Peak Logical Write Rate: ***Medium*** (this means ~50 insert, delete, update / second)
- Forms Applications Used: ***No***
- Comprehensive Analysis: ***Yes***

This will give a complete tuning session regarding most parts of the database.

6.5.4.2 Collect tab

The collect tab shows when collection of data and statistics were last done. It is also here the collection of data starts by pressing the |Collect| button.

When the collect button is pressed a dialog is shown asking for information about the system being tuned. Type in as much information as possible, at least the memory size of the machine being examined should be entered. Click the |Advanced| button to enter even more information.

After entering the information and the |OK| button is pressed the data collection process is started. It saves data from a time span of 15 minutes in 4 runs as default. During this time the database should be utilized as in normal use.

When the collection is done the review data and the results are created.

6.5.4.3 Review tab

The review tab shows all the collected data from which the recommendations are built upon.

All the different aspects of the database are recorded in the review tree. Instance and parameter settings are all available in the tree structure.

It is also possible to edit the collected data and the rules used by Oracle Expert by clicking on the end nodes. This can be done to prevent the tool to make faulty assumptions.

6.5.4.4 Recommendations tab

Here all the recommendations that Oracle Expert gives are listed.

By pressing the |Generate| button recommendations are built using the data from the review tab.

Recommendations on indexes, instance and so on are divided into their own directories and by simply clicking them all the results are there.

The hard part with this is to figure out what recommendations can be a good idea to implement and what is not. It is possible to decline the recommendations by mouse right-click and press decline. If now the |Generate| button is pressed the tool removes the declined recommendations and removes them from the scripts.

6.5.4.5 Scripts tab

Here the scripts that implements the recommendations and their location in the file system is listed.

6.5.5 Reading Oracle expert result

Information on how to read the results from the tool see [7.2].

6.6 Using Statspack

Statspack is a tool or more a script that needs a form of installation to start working with. The data from the Statspack tool is in the form of a report and shows performance and other information over a certain time period. This data is then analyzed to see things that can need improvement.

Statspack is also a good tool to use weekly to have some background information to address problems. By having a normal picture of the database it is possible to do a snapshot when things are bad and see what differs from the normal.

Running of the tool should be done after the database has been warmed up. That means that a report done just after starting the database will show excessive physical I/O because the memory buffers are empty and now are filled with data. When the database has been up for some time the buffers are containing the most used blocks and is in its “normal” state.

Reading and understanding of the Statspack report requires a little more understanding of the Oracle environment than the other tools.

Note that in order to use Statspack effectively the database need to have timed statistics enabled. See [11.4] for statistics level and timed statistics.

6.6.1 Installation

Installation of the tool is by execution of the `spcreate.sql` script that is found in the `ORACLE_HOME/rdbms/admin` directory.

1 Log on to the resource database server:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database:

```
oracle_user@DBServer> sqlplus /nolog
SQL> connect sys/sys_password@ResourceDatabase as sysdba
```

3 Execute the following command to install the tool:

```
SQL> @ ?/rdbms/admin/spcreate
```

4 The first question is what password the `perfstat` user should have. Enter any password.

5 Second is what tablespace is to be used by the tool. The `SYSTEM` tablespace cannot be used and should not be used. Select the `TOOLS` tablespace that should exist or create a new tablespace for the tool.

6 A temporary tablespace is needed for the tool. Choose a temporary tablespace other than `SYSTEM`.

Information on errors is printed on the screen. If errors are produced, view the files according to the instructions on the screen.

6.6.2 Create snapshots

The data collection with Statspack is done with snapshots. By doing a snapshot values are collected and the report is formed from this values.

A snapshot should be no longer than fifteen minutes and taken during either normal operation or abnormal operation. This is because if the snapshot is from a longer time span the bad thing affecting performance short times is evened out over the longer time period and so does not look so bad.

It can be very useful to have a snapshot from normal operation to compare to a snapshot from abnormal operation as already mentioned.

To create a snapshot:

1 Log on to the resource database:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database:

```
oracle_user@DBServer> sqlplus /nolog
SQL> connect perfstat/perfstat_password@ResourceDatabase
```

3 Issue the command:

```
SQL> exec statspack.snap;
```

this will create the first part needed to create the report.

4 Now let the database do the work that should be tested. This might involve starting a batch job like the upload adapter or simply letting the users do what they always do.

5 Issue the command:

```
SQL> exec statspack.snap;
```

this will create the endpoint of the report.

6.6.3 Create a report

1 Log on to the resource database:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database:

```
oracle_user@DBServer> sqlplus /nolog
SQL> connect perfstat/perfstat_password@ResourceDatabase
```

3 Issue the command:

```
SQL> @ ?/rdbms/admin/spreport
```

4 If there are any snapshots done a list of snapshots will show up and a question is asked about what id the `begin_snap` should have. Look in the list of snapshots and choose the appropriate id.

5 Another question about what should be the `end_snap` id is asked. Choose appropriate id from the list.

6 Now a question on filename is asked, if none is submitted a default filename will be used. Submit a filename if appropriate. The file with the report will be placed in the current directory, the directory from where the `sqlplus /nolog` command was done in step 2.

6.6.4 Delete snapshots

1 Log on to the resource database:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database:

```
oracle_user@DBServer> sqlplus /nolog
SQL> connect perfstat/perfstat_password@ResourceDatabase
```

3 Issue the command:

```
SQL> @ ?/rdbms/admin/sppurge
```

4 Choose what snapshots to delete according to the instructions on the screen.

5 Exit SQL*plus:

```
SQL> exit
```

6.7 Reading Statspack results

Information on how to read the results from the tool see [7.2.1.2].

7 Tuning

7.1 Tkprof and Explain plan interpretation

After the Tkprof tool has been used to transform the trace files (see [6.1]) the resulting reports need to be interpreted. The information that can be retrieved from a report can help a user to determine what is causing problems in a system and correct these problems.

The Tkprof report will contain execution plans or explain plans. As stated earlier, this is the plan that was actually used executing the statement opposed to the result from Explain plan, which is the expected execution plan.

The explain plan in the Tkprof report is explained in [7.1.2].

7.1.1 Reading Tkprof report

How to interpret a Tkprof report is an art in itself. First of all the layout of the report, the columns and what they mean:

```
********************************************************************************
SELECT USER
FROM
  DUAL

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          3          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0          3          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 59
```

Table 3 Tkprof report.

This is a typical part of a Tkprof report. At the top of the report the SQL that is analysed is shown. In this case a selection of all rows in the column user from the table dual. The different statements are divided by a row of star signs.

The columns of the report displays the following:

7.1.1.1 Call column

Call is the type of call executed; there are three different call types:

Parse – The construction of the SQL statement.
Execute – The execution of the SQL statement.
Fetch – The numbers of fetch's done by the SQL statement.

7.1.1.2 Count column

The count column is the number of times the particular call has been made. In this example parse, execute and fetch is done once.

7.1.1.3 CPU column

The CPU column shows how much CPU time the code took to run.

If the `timed_statistics` parameter is false this column will show all zero.

7.1.1.4 Elapsed column

Elapsed shows how much total time that was spent on the code. In this example all of these figures shows zero as the time result. Of course nothing takes no time to execute but the granularity of the clock is 1/100 second and if the value is beneath that it shows up as zero. The timing granularity is also dependent on the operating system.

If the `timed_statistics` parameter is false this column will show all zero.

7.1.1.5 Disk column

Disk is how many physical reads that have been done, that is how many times the database have been forced to retrieve data from the physical disks.

This is also referred to as physical reads.

Administrator

High numbers in the disk column might indicate that some sorting is being done on disk instead of in memory. An increase of the `sort_area_size` parameter might be a solution to this.

7.1.1.6 Query column

Query is the number of buffers gotten in consistent mode. Consistent mode means that the blocks or buffers are gotten as they were in the beginning of the execution of the statement. This might involve using rollback data to be able to use the values that were in place at beginning of execution.

This might also be referred to as consistent gets.

7.1.1.7 Current column

Current reads is exactly what it states. The buffers gotten, blocks or values as they are right now. This is often part of an update statement where new data is entered into the database.

Also referred to as db block gets.

7.1.1.8 Parse row

Parse is showing the work done by the database to construct an executable statement of the statement that was sent to the database. This might be normal SQL, PL/SQL code or even Java code.

Developer

Think of this as a normal programming language, when your function or procedure is parsed you can use it a million times without parsing it again. If the number of parses is the same as the number of executions then there can be room for improvement, if not parse to execute is 1:1 of course. Check whether there was a miss in the library cache (see [7.1.1.12]), if there is only one miss then the statement or code has been hard parsed once and soft parsed the rest of the times.

A hard parse is when nothing is in the shared pool and the whole statement/code need to be constructed. A soft parse is when there is a hit in the cache but some parts of the code still needs to be parsed. Soft parse is cheaper than hard parse but even a soft parse uses resources so they should be avoided. A soft soft parse is when the database is able to reuse a certain statement or procedure so that the parse number in the Tkprof report does not change. This is accomplished by using cursors and is done per session.

Even if parses does not take a lot of CPU time they might create wait time so that other processes have to wait and thus creates a longer response time. Problems creating this might be not using bind variables in the SQL statements.

Developer and Administrator

The lack of using bind variables might create a situation where the library cache is emptied from useful SQL. This happens because there are a lot of statements parsed that is not unique and instead could have used the same SQL statements that is already in the cache instead of creating new ones. These statements will push other statements out of the library cache and so create more parsing because the statement pushed out must be parsed again.

Programming with bind variables can correct this. See the sources - All about binds [10].

One other thing can be wrong coding (if java especially) where the programmer tells Oracle to parse the statement every time it is executed. Oracle will do this; it does everything we ask for even if it is consuming resources that could have been used in other places.

Developer

In java the “rule of thumb” can be said to be:

Prepare in constructor
Bind and execute in your functions as many times as wanted

PARSE
loop
    bind
    execute
end loop
CLOSE

By following this simple rules the code is parsed in the beginning of the execution and can then be reused as many times as wanted without being re-parsed. Look at `PrepareStatement` in the Java API to avoid repeated parsing.

7.1.1.9 Execute row

An execution is just what it sounds. This is when the database executes the statement, whether it is java or simple SQL. An execution can take place lots of times depending on how the program accessing the database is programmed.

If the statement is executed a few times and consumes a lot of CPU-time this might be an indication that the statement can be improved. If the statement is executed a lot of times and has a long execution time even a small improvement in the statement can impact the total quite a lot.

Even though it might seem best to reduce the physical I/O (disk in Tkprof report) one should always concentrate on reducing the logical I/O. The logical I/O stands for a read of a block that might be found in the buffer cache (query + current = logical I/O:s in Tkprof report); this seems good because the buffer cache is in memory but that data has sometime not long ago been read to memory from disk. So by removing much of the work done on the logical I/O then there is no need to read it in using physical I/O at an earlier stage either. It is a win-win situation.

As always think of what the statement is doing, is it retrieving a million rows and executing thousands of times then there is maybe not something odd with a big number of logical I/O:s, but if the statement is short and executed once, then improvement should be possible.

7.1.1.10 Fetch row

Fetch is the operation where the database retrieves the data requested from the user or the SQL statement. A fetch consists of rows from the database. The number of fetches doesn't have to be the same as the number of rows retrieved because the database can take an array of rows in each fetch. So the number of rows doesn't have to be the same as the number of fetches.

The same rules apply to fetch as execution. On inserts, updates and such no rows are fetched and the fetch count is zero.

7.1.1.11 Total row

The total column is simply a summary of the figures. Here it is easy to see if any of the totals is very high and so needs to be rewritten.

7.1.1.12 Misses in library cache

A miss in the library cache is when a statement needs to be reparsed or reconstructed totally from the beginning. Every statement needs to be parsed at least one time but if the same statement is used many times it can be reused. This is because all parsed statements are put in the cache and from there Oracle can retrieve it again if the same statement is being executed again. Lots of misses is bad but there has to be at least one miss for each statement for the first time it is parsed.

Developer

Misses in the library cache might be a result of bad coding, see [7.1.1.8].

Administrator

A miss in the library cache might also be an indication that the shared pool is too small. This can be seen if there are a lot of reparsing that is not due to bad program design (see [7.1.1.8]). If the pool is to small the database will have to reparse a lot of statements and parsing is expensive.

Increasing the size of the shared pool might fix this problem.

7.1.1.13 Optimizer mode

What approach the optimizer chooses to be the best for the statement.

Developer and Administrator

To “bypass” the optimizer and get a different explain plan use hints. See [7.1.2.4].

7.1.1.14 Parsing user id

Id of the user that executed the statement. Might be the number representing the user or the username.

7.1.1.15 Report summary

At the bottom of all Tkprof reports a summary of all the SQL executed is shown. It is one total for the non-recursive statements and one for the recursive statements. Non-recursive statements sum the code that is executed directly to the database from the user. Recursive totals sum code that is executed either from functions or internal Oracle code that is executed for storage management and such. If combining the two totals in the elapsed columns for the two totals tables an estimate of the total time for SQL statements can be made.

7.1.2 Reading Explain plans

Administrator

This chapter is for developers foremost but also good to know for administrators.

The explain plan is concentrated on only one statement. There are a lot of different things to look into when reading an execution plan.

Explain plans that are found in the Tkprof report has less information than those retrieved from the Explain plan tool. They contain the numbers of rows, row source and the operation. They are interpreted the same though.

The columns of this report are somewhat self-explained. Id is what number a certain operation has. Operation is in what way Oracle interprets and executes a certain part of the statement. Name is the name of the table or the index used in the statement. Rows is the number of rows accessed for the operation. Bytes are the number of bytes read. Cost is a figure showing how expensive the execution was and time is the time the operation took to finish.

The lower table is how the table would look in a Tkprof report. It looks virtually the same; the biggest difference is the numbers in the end of the operation. cr is how many consistent reads the operation did. r is the number of physical reads done. w is the number of writes and us is the time used to execute the statement.

```
Plan hash value: 749696591
-------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |    26 |   338 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| T    |    26 |   338 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------
```

Table 4 Explain plan from the Explain plan tool.

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      0  SELECT STATEMENT  (cr=1634 r=0 w=0 time=497926 us)…
      …  …
---------------------------------------------------------------
```

Table 5 Explain plan from a Tkprof report.

The explain plan in table 3 above can be explained in two simple steps:

```
TABLE ACCESS FULL| T
```

The row above goes through all the rows in the table T and returns them as the result.

```
SELECT STATEMENT
```

The select statement retrieves the rows from the row below it and presents them to the user.

This is the way that the data from the explain plan is read. The data retrieval starts in the leafs (right most side) and ends in the root (left most side, top). Data is formed by different aggregations and joins to give the requested result in the node. It can be easier to understand if drawn like a tree:

Figure 4 Explain plan tree.

Here it is easy to see that id one gives the result back to id zero.

Executing a statement gives:

```
explain plan for select object_id from t where object_id=161;
```

```
Plan hash value: 749696591
-------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     2 |    26 |    56   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |     2 |    26 |    56   (2)| 00:00:01 |
-------------------------------------------------------------------------
```

Table 6 Explain plan.

If creating an index on the table T and then executing the command:

```
explain plan for select object_id from t where object_id=161;
```

The result is:

```
Plan hash value: 1933149062
-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |    13 |     1   (0)| 00:00:01|
|*  1 |  INDEX RANGE SCAN| T_INDEX |     1 |    13 |     1   (0)| 00:00:01|
-------------------------------------------------------------------------------
```

Table 7 Explain plan.

The result is the same as that in table 5 but here the operation of id 1 has changed from `table access full` to `index range scan`. This indicates that we now are using an index instead of scanning the whole table. Name column now indicates the name of the index instead of the table name.

Using two tables and executing the statement:

```
explain plan for select t.object_id, t2.object_name from t,t2
where t.object_id=t2.object_id;
```

```
Plan hash value: 1666661573
-------------------------------------------------------------------------------
| Id  | Operation             | Name    | Rows  | Bytes |TempSpc| Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |         | 41732 |  1752K|       |   295 |
|*  1 |  HASH JOIN            |         | 41732 |  1752K|  1024K|   295 |
|   2 |   INDEX FAST FULL SCAN| T_INDEX | 41732 |   529K|       |    25 |
|   3 |   TABLE ACCESS FULL   | T2      | 48532 |  1421K|       |    52 |
-------------------------------------------------------------------------------...
```

Table 8 Explain plan.

This explain plan shows that there first is a fast full scan of t1 that is using the index t_index and fetching 41732 rows, then there is a full access of t2 fetching 48532 rows. On row two the hash join statement will first take the first row from t that it got from the index fast full scan and compare that to the first row in t2, return this data down to row one. This procedure will be iterated until there are no more rows to compare (some data removed from report). If drawing a tree it would look like:

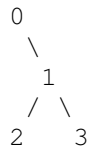


Figure 5 Explain plan tree.

7.1.2.1 Indexes and Throw-away

The first and maybe easiest thing to look for in an explain plan is for operations that involves `table access full`. These operations go through every row in a table and if the table is large this will take a long time. If there is a table consisting of 20000 rows and the operation is to fetch 10 out of these, then the SQL statement should use some sort of where clause and use an index to search the table.

Developer

If a big percent of the rows is going to be fetched then there might be no point in using an index, in fact the index might slow down operations in that case. So by examining how many rows the table consists of and how many of these rows that are fetched it is possible to see if an index should be implemented.

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      0  HASH JOIN  (cr=1634 r=0 w=0 time=497926 us)
 111870   TABLE ACCESS FULL CARDINSLOT (cr=1440 r=0 w=0 time=106760 us)
     52   TABLE ACCESS BY INDEX ROWID SLOT (cr=194 r=0 w=0 time=3065 us)
     52    INDEX RANGE SCAN SLOT_CARD_FK_I (cr=180 r=0 w=0 time=2502 us)(object id…
```

Table 9 Explain plan.

The example above shows typical throw-away behavior. Accessing thousands of rows to throw them away.

Developer and Administrator

Test if the index is usable. If the where clause contains functions on the values the index might be rendered unusable. Consider creating a function-based index. The statement might also be improved by rewriting it using other where clauses.

Also using indexes creates wait time. If for example an index is in the buffer and a SQL statement heavily relies on that index it will read it often. We will have lots of consistent gets and consistent gets creates wait time that other operations might be forced to wait for.

Always consider if a particular index is needed. Do not create an index because it might be needed; create it because it is needed.

Indexes do not sort data in any particular order. Sorting has to be done on the data to obtain it in the order wanted.

The conclusion is that indexes are not good for all things and if not implemented and used in the right way might be totally useless and increase the resource usage.

7.1.2.2 Joins

If the statement accesses a number of tables then there are some different ways Oracle can choose when joining the results. Different joins give different speed. As standard Oracle tries to deliver the last row as fast as possible, this means that there might be different ways to for example retrieve the first row almost instantly but the last row in the result set a few minutes away. This is not something that is a standard way of doing things but the possibility exists.

The effectiveness of the different join methods is hard to explain, they depend on the cost based optimizer that processes the statement and comes up with a result. The most important thing with a statement is to create a code that is as simple as possible but yet able to give the demanded result.

Developer

Keep a lookout for any Cartesian and nested loop outer joins, as they often will induce poor performance. Often it can be faulty SQL statements that cause them to degrade performance.

7.1.2.3 Differences between Tkprof and Explain plan

One good way of finding out problems or why statements performs poorly is to view the differences between the explain plan in Tkprof and the explain plan from the tool Explain plan.

The plan found in the Tkprof report is what really happened during execution and the plan from Explain plan is what the optimizer thinks will happen. If there are big differences in the plan the optimizer might have to little information to go on for making its decisions. For example if the data from Explain plan says there will be 300 rows returned and the real execution in the Tkprof report says it returned 2 rows indicates that something might be wrong.

The table might have to be analyzed or histograms might have to be constructed to give the optimizer enough data to work with. The main idea is that the optimizer can't make decisions on execution if it does not have the right information.

Administrator

For more info on gathering statistics, see [7.1.2.5].

7.1.2.4 Hints

Hints can be used to force the optimizer to do a certain operation. Instead of calculating what is the best method the programmer can write a hint in the code and the optimizer will use it.

A hint can be implemented like this.

```
select /*+row_first*/ object_name from all_objects;
```

Where `/*+row_first*/` is the hint.

Developer

Hints can be useful to create a more responsive application or other non-standard behavior. A hint can also remove the standard behavior of the optimizer and so create a less efficient question.

Hints can be used to test if an index is making a question perform better with or without the index. To force Oracle to not use an index on a certain table insert the `/*+full(table_name)*/` hint.

Example:

```
select name, number from cust where name='NAME';
```

If an index is implemented on the name in the cust table simply use:

```
select /*+full(cust)*/ name, number from cust where
name='NAME';
```

This will force Oracle to do a full scan on the table instead of using the index.

See chapter 5 in Oracle9i – Database performance tuning guide (page 205).

Note that hints instructing the optimizer generally not should be used but are a valuable tool when testing different statements.

7.1.2.5 Analyze

In order to give Oracle the best possible way to calculate plans for the execution of SQL the optimizer needs information. To give the optimizer this information statistics needs to be computed for the tables used.

By having old statistics or none at all the optimizer is forced to guess or use old statistics that in turn can result in a less optimal execution plan for a statement.

Administrator

Statistics should be computed as needed. How often depends on how often the data in a table is changed. If there is a big batch job running that changes a lot of the table data statistics can be computed directly after the job is done for example.

To see when a table was last analyzed use:

```
select table_name, last_analyzed from user_tables where
table_name ='TABLE_NAME';
```

To see when an index was last analyzed use:

```
select index_name, last_analyzed from user_indexes where
index_name ='INDEX_NAME';
```

To gather statistics use the DBMS_STATS package as described in Chapter 3, Oracle9i – Database performance tuning guide (page 157).

The lack of current statistics can be a cause to why explain plans does not show the expected result in regards to for example usage of indexes.

At the same time statistics is not some sort of magic wand that always increases performance. Test with Tkprof before and after and see the differences in explain plans and figures to see if the statistics made the statement execute faster.

7.1.2.6 General

Almost all possible statements can be done as one statement instead of a number of statements. Advanced statements with a lot of inline SQL can be really costly. Also using functions in the where clauses can be costly, a function can render indexes unusable if the indexes are not built to use functions. Trying to limit the numbers of rows in the explain plan is a simple and easy way to get a better performing statement.

Also have a lookout for wrongly written statements; this can be an easy way to improve performance on a statement.

7.2 Oracle expert results

The real problem with Oracle Expert is to decide what parts of the suggestions that should be implemented and if a solution can be implemented in a running environment or if it is required to restart the database and so on. All of these questions are up to the administrator to decide. Also in a production environment one has to think of the possibility for database stall.

The recommendations that are created by the tool are always right in some sense but not always right for the specific instance. The results might depend on a session that did not use all indexes that exists in the database and therefore the tool thinks that these index are unnecessary even if they are not. The tool creates recommendations on what it knows, the recommendations has to be reviewed by someone who knows the database and what the data in it is all about in order to see what recommendations that can be dismissed.

The results and the recommendations are easily viewed in the tool and by choosing what statements that should be in the final scripts are easy. There is also often some sort of evidence available to why the tool made the recommendation. By reading the explanation or proof it should be possible to figure out if the recommendation is something that is isolated to the test just done or something that is always appearing.

7.2.1 Implement recommendations

7.2.1.1 SQL Changes

1 Copy the script files to the resource database.

2 Log on to the resource database server by issuing the command:

```
rlogin DBServer -l oracle_user
```

3 Move to the path where the script is situated:

```
oracle_user@DBServer> cd /path/to/scripts
```

4 Run the following command in order to log on to the database:

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect sys/sys_password@ResourceDatabase as sysdba
```

5 Execute the script.

```
SQL> @scriptfilename;
```

7.2.1.2 Instance parameters

Some instance parameters can be changed dynamically while the database is running. They should always be set in the init.ora file so that they exist also when the instance has been shut down. Most new systems make use of server parameter file.

Be sure to not change or add any parameters on a production setting before knowing the implications or knowing how to reverse the setting.

Setting a parameter in the spfile is done by:

1 Log on to the resource database server:

```
rlogin DBServer -l oracle_user
```

2 Log on to the database

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect sys/sys_password@ResourceDatabase as sysdba
```

3 To change a parameter issue the following command:

```
SQL> alter system set parameter_name=value
scope=[spfile,memory,both];
```

for example:

```
SQL> alter system set timed_statistics=true scope=both;
```

7.3 Statspack

A Statspack report is very large and contains a lot of information. This chapter will therefore concentrate on some key pieces of the report.

The report from Statspack is the most advanced of the tools in this document to read, as it is not written in stone what to do with certain figures. The report is a valuable tool and even the basics can be really useful.

7.3.1 Reading the Statspack report

The top part of a Statspack report looks like this:

```
STATSPACK report for

DB Name         DB Id    Instance     Inst Num Release     Cluster Host
------------ ----------- ------------ -------- ----------- ------- ------------
ORADB         2222222222     database        1 9.2.0.4.0   NO      comp

            Snap Id     Snap Time      Sessions Curs/Sess Comment
            ------- ------------------ -------- --------- -------------------
Begin Snap:       1 08-Aug-08 08:08:08        8         8
  End Snap:       2 08-Aug-08 08:23:08        8         8
   Elapsed:               15.00 (mins)

Cache Sizes (end)
~~~~~~~~~~~~~~~~~
               Buffer Cache:       400M      Std Block Size:         8K
           Shared Pool Size:       240M          Log Buffer:      512K
```

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:           35,000.00             2,000.00
              Logical reads:            3,500.00               135.00
              Block changes:              250.00                12.50
             Physical reads:                5.00                 0.20
            Physical writes:                3.50                 0.15
                 User calls:              100.0                  4.00
                     Parses:               85.00                 3.50
                Hard parses:                0.30                 0.01
                      Sorts:                5.50                 0.25
                     Logons:                0.05                 0.00
                   Executes:              200.00                 9.00
               Transactions:               25.00

  % Blocks changed per Read:    10.0    Recursive Call %:               85.00
 Rollback per transaction %:    0.00       Rows per Sort:              150.00

Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:  100.00       Redo NoWait %:              100.00
            Buffer  Hit   %:   99.95    In-memory Sort %:              100.00
            Library Hit   %:   99.95        Soft Parse %:               99.95
         Execute to Parse %:   99.95         Latch Hit %:              100.00
Parse CPU to Parse Elapsd %:   99.95     % Non-Parse CPU:               99.95

 Shared Pool Statistics        Begin   End
                               ------  ------
             Memory Usage %:   40.00   40.00
    % SQL with executions>1:   75.00   75.00
  % Memory for SQL w/exec>1:   70.00   70.00

Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                                     % Total
Event                                               Waits    Time (s) Ela Time
--------------------------------------------- ------------ ----------- --------
CPU time                                                        25,000    80.00
latch free                                         850,000       3,500    15.00
log file sync                                      350,000       1,250     5.00
log file parallel write                            360,000         450     1.50
db file parallel write                               2,500          50      .20
          -------------------------------------------------------------
```

Table 10 Statspack report example.

This is only the top part of the report. Lots of more data is in the report. Some of the other data is also covered in this document.

7.3.1.1 Basic information

In the top of the report all information about the database that the report is done on is contained. The database id, instance name, version and other information are listed here.

The interesting information here is how long the time span the report is calculated from. This should be no more than fifteen minutes and is obvious in the Elapsed column. In this particular report the time is 15.0 minutes.

7.3.1.2 Cache sizes (end)

This part of the report shows how the cache sizes looked like at the end snapshot. That is how large the different caches in the database were.

This information can be seen as an information overview about the database and information that can be used to read the rest of the report.

7.3.1.3 Load profile

The load profile shows what work that was done during the time between the two snapshots. The numbers are somewhat not interesting here. The important thing is that the transaction row is more than zero because else no work has been done.

If the Statspack does not contain a typical load or any load at all the report is not representative and therefore the figures are not valid and should not be used. If they are used wrong conclusions can be made and changes made to the database with this information may degrade performance.

Numbers in this section is also used to read the rest of the report.

Basic observations from this section can be:

- Hard parses should be as low as possible. This part is often an application issue. This might also show up as a high memory usage and high percentage of CPU parsing later in the report.
- Rollback per transaction should be as low as possible. If this is high the transactions executed must roll back to get consistent data which can be costly. This is also often an application issue.
- Logons should be a low number. If logon is very high and the database is not an OLTP with thousands of users there might be some problem in the application.
- If the number of transactions per second is very high in the application might commit to often. This might cause problem in writing to the redo log files.

7.3.1.4 Instance Efficiency Percentages (Target 100%)

This part of the report shows how efficient the database used the different caches and how parsing and execution was done. All the figures in these columns have the figure 100 as a goal. This is the goal but there are exceptions. The important thing is to know why it is this way and if everything works fine like this then no need to worry. An investigation should always be done to answer question to why the figures are one or the other way.

The different values are explained in table 9 below.

Buffer Nowait	Shows if buffers asked for are ready to read or if there is some contest for getting the resource.
Buffer Hit	Shows how much of the data needed for operations that already reside in memory so that it does not need to be read from disk. A high number is preferred as it is a lot faster to read from memory than disk. This is part of the SGA and the init.ora variable is db_cache_size (or might be db_block_buffers).
Library Hit	This shows how often SQL that is going to be executed is found in the library cache. A small percentage of library hit might indicate that the library cache is to small, that is that the shared pool is to small. It might also show that the program makes poor or no use of bind variables.
Execute to Parse	Show a value of how many times a statement is executed on each parse. If it is a web application with short sessions that executes one statement this figure can be rather low. If it is an application with long sessions the figure should be high. session_cached_cursors and open_cursors are init.ora variables that affect this.
Parse CPU to Parse Elapsd	The ratio between the time that the CPU did really parse a statement compared to the time it all took including the time waiting to start parse. Minimizing the wait time makes elapsed=CPU.
Redo NoWait	Shows how much wait has been done by a transaction to write its redo data. init.ora variable affecting this is log_buffer.
In-memory Sort	How much of the sorting operations that was done in memory without swapping out to the users temporary tablespace. Sorting in memory is a lot faster than on disk. sort_area_size is the init.ora variable that decides the size of this.
Soft Parse	The amount of SQL that can be reused. This increases the performance instead of a total reparse of a statement. Not using bind variables so that SQL cannot be reused might give a low figure. Also a small shared pool might push out statements before they are reused, increasing the size of the shared pool can fix the problem. session_cached_cursors can also save statements for reuse.
Latch Hit	The percent of successfully acquired latches. If low, check top 5 timed events for latch free. If it exists check the latch activity of the report for further investigation.
Non-Parse CPU	The higher the more work was done not parsing. If parsing takes up a considerable amount of the total time it might have to be investigated.
Memory Usage	Total amount of memory used by instance. High memory use might decrease performance.

SQL with executions>1	SQL with more than one execution.
Memory for SQL w/exec>1	Amount of memory used by SQL with more than one execution.

Table 11 Statspack values

Why the values are low can be caused by a lot of different things. The important thing is that by having the report it is possible to see what the problem is and by this information it is possible to search for a solution.

Developer and Administrator

The figures in this section are just ratios or just number or figures. It is not a goal to get all of these to a hundred and then to think that everything is ok. Different applications give different results. A problem may not be directly linked to a bad ratio; instead there might be some underlying cause that has to be investigated. Or as Oracle writes:

"*Database tuning never must be driven by hit ratios. They only provide additional information to understand how the instance is operating.*" (MetaLink note 228913.1) Oracle corp.

7.3.1.5 Top 5 Timed Events

These events are the ones that had the longest times during the snapshot. The events are described in the Oracle9i - Reference Manual from page 717 and in the Oracle9i – Database performance tuning guide from page 660.

By removing some of the events that create wait the database will increase in performance because more work will be done and no waiting on resources will occur.

Administrator

Concentrate on the top five timed events and check up in the reference manual what they mean. Some of the wait events may be connected and some like CPU time is not a wait event, it just shows how much CPU that was used.

7.3.1.6 Buffer pool advisory

In the report Statspack is also doing forecasts or advisories. This is meant to help the administrator to size buffers to the right size.

The buffer pool advisory shows how the buffer that holds data would contribute to different sizes.

The advisory looks like:

```
Buffer Pool Advisory for DB: ALPHADB  Instance: alphadb  End Snap: 2
-> Only rows with estimated physical reads >0 are displayed
-> ordered by Block Size, Buffers For Estimate

        Size for  Size      Buffers for  Est Physical          Estimated
P    Estimate (M) Factr         Estimate   Read Factor     Physical Reads
--- ------------ ----- ---------------- ------------- ------------------
…
D             288    .7           35,730          1.00             10,355
D             336    .8           41,685          1.00             10,355
D             384    .9           47,640          1.00             10,355
D             432   1.0           53,595          1.00             10,355
D             480   1.1           59,550          1.00             10,355
D             528   1.2           65,505          1.00             10,355
…[text removed]
```

Table 12 Example of a Buffer pool advisory table.

In this report it is easy to see that after the estimate size 144 MB (column two) the estimated physical reads (last column) will stay the same. So a size of 144 MB would be fine in this case. If memory is a rare resource it might be a good idea to decrease the buffer cache size and move that memory to more resource consuming areas.

Administrator

The advisory is just what it is named, advisory. Depending on when, where and how the snapshot was done it might or might not be a good representation of the database. Change the variables with care on a production instance, as usual.

7.3.1.7 PGA memory advisory

The Program Global Area is a memory area that only a certain process or server can use. It is not shared as the System Global Area. The PGA is allocated at connection from a client for example when a server process is started. The PGA holds space for sorts and other parts of a program.

```
PGA Memory Advisory for DB: ALPHADB  Instance: alphadb  End Snap: 2
-> When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value
   where Estd PGA Overalloc Count is 0

                                    Estd Extra    Estd PGA   Estd PGA
PGA Target    Size           W/A MB   W/A MB Read/      Cache  Overalloc
  Est (MB)   Factr         Processed Written to Disk     Hit %      Count
---------- ------- ---------------- ---------------- -------- ----------
        17     0.5         12,110.7            956.7     93.0        149
        25     0.8         12,110.7            764.3     94.0        106
        33     1.0         12,110.7            203.2     98.0         27
        40     1.2         12,110.7            203.2     98.0         19
…[text removed]
```

Table 13 Example of a PGA memory advisory table.

The goal in this report is to have an Overalloc Count that is zero, which means that there was enough memory during the session for sorting and other user data.

7.3.1.8 Shared pool advisory

As for the PGA memory there is also an advisory for the shared pool.

```
Shared Pool Advisory for DB: ALPHADB  Instance: alphadb  End Snap: 2
-> Note there is often a 1:Many correlation between a single logical object
   in the Library Cache, and the physical number of memory objects associated
   with it.  Therefore comparing the number of Lib Cache objects (e.g. in
   v$librarycache), with the number of Lib Cache Memory Objects is invalid

                                                          Estd
Shared Pool    SP       Estd         Estd     Estd Lib LC Time
   Size for  Size  Lib Cache    Lib Cache   Cache Time   Saved  Estd Lib Cache
  Estim (M) Factr   Size (M)      Mem Obj   Saved (s)   Factr    Mem Obj Hits
----------- ----- ---------- ------------ ------------ ------- ---------------
        144    .6        127       12,774        6,257     1.0       1,517,060
        176    .7        127       12,774        6,257     1.0       1,517,060
        208    .9        127       12,774        6,257     1.0       1,517,060
        240   1.0        127       12,774        6,257     1.0       1,517,060
        272   1.1        127       12,774        6,257     1.0       1,517,060
…[text removed]
```

Table 14 Example of a Shared pool advisory table.

This is the same as the two other advisories, the estimate sizes in column one shows how the values would differ with different settings.

7.3.1.9 Parameters

At the end of the report all the init.ora initialization parameters are listed. These are the values that the instance first started with and will be started with during a restart of the database.

7.4 Scaling issues

The adapter is not subjected to any real scaling issues due to the fact that it only can be one adapter running at the same time. There is only one process that is doing the data load and that is not subject to change in the near future. All tuning of the database can be concentrated on giving this only process as much resources as possible.

The only other things that can affect performance are if there are any users logged on to the Cramer GUI and accessing data that way. But loading of data will generally be done on non-office hours so that event is not very common or likely.

If in the future there will be more than one adapter loading data into the database this will change. Some processes might have to wait on others and this might increase the time an upload takes to carry out. To obtain the same performance as with one adapter faster hardware might have to be considered.

Also if more then one end users data is to be stored in the same database performance might become an issue. This should tough be more of a machine sizing issue if all the programming is done right.

The programming of the adapter can also play a big part of the applications possibility to scale well. If all the right techniques are used while developing the program then problems can be avoided or minimal. If the program uses non-efficient coding and excessive parsing then scaling can create huge load times and big problems.

Especially logical I/O is costly in an environment with many transactions/second. When the data is read from the buffer it is locked and this lock might create wait time for other processes. The less logical I/O the less locks created and therefore a more scalable program. Locks and other stuff that prohibits simultaneous access to data are serialization devices that will decrease performance in a scaling environment.

As always rigorous testing should always be done before the program enters production environment.

For more information on scaling see Oracle9i – Performance planning in [10].

8 Conclusion

The NIM system had not been subject to any performance testing before the work in this document was done. This has made it possible for some unnecessary and easy to find errors or mistakes to exist in the product. Simple testing could have eliminated these things at an early stage if it had been conducted.

To work with a third party can be both easy in the way that all or parts of the software can be used directly out of the box. But it is also restraining in the way that things cannot be changed, code is locked and bugs must be reported to third party. There are a lot of things to think about when dealing with software from third party to not violate any contracts or void agreements.

Sometime testing takes time. It is not easy to look at such a complicated system and state what is going wrong by simply looking at a number. Often there can be a couple of things that makes a statement execute slow for example. Experience of the system and its data is a valuable tool when trying to find problems.

Knowing the system is a vital part of the tuning process. Without knowing anything about the system and how it is used it is a lot harder to know where to start, if not impossible.

Testing should be done in some sort of scientific manor. That is do not use guidelines or rule of thumbs without actually testing what they would do with the system. If a new idea of how to do something is constructed, test before, implement, test after and make a conclusion of what the results means. Is the result what could be expected? Is there any side effect? Was the testing done during a normal period or abnormal; is the results a good picture of how it will look in reality? All of these questions might have to be answered if the change or idea has big implications, only a few might need an answer if the change is small.

9 The future

The future in this context will be the new release of Oracle, 10g. In this new release of Oracle some things have been removed and some new things added.

The Oracle Expert tool has been removed from the tuning tools. Instead a new set of tools has been introduced. Among them:

- SQL Tuning Advisor
- SQLAccess Advisor
- Automatic Statistics Gathering
- Automatic Workload Repository - AWR
- Automatic Database Diagnostic Monitor - ADDM

The new sets of tools are supposed to give the user an even easier way of tuning the database. For more reading see:

http://www.oracle.com/technology/products/manageability/database/pdf/twp03/TWP_manage_automatic_performance_diagnosis.pdf

Tkprof, Explain plan and Statspack is still there and are used as in this paper.

10 Sources

[1] Oracle corp. [2002] Oracle9i – Database concepts, release 2.
[www] http://www.stanford.edu/dept/itss/docs/oracle/9i/server.920/a96524.pdf
Oracle part number A96524-01.

[2] Oracle corp. [2002] Oracle9i – Database performance tuning guide.
[www] http://www.stanford.edu/dept/itss/docs/oracle/9i/server.920/a96533.pdf
Oracle part number A96533-02.

[3] Oracle corp. [2002] Oracle9i – Performance planning.
[www] http://www.stanford.edu/dept/itss/docs/oracle/9i/server.920/a96532.pdf
Oracle part number A96532-01.

[4] Oracle corp. [2002] Oracle9i – Reference.
[www] http://www.stanford.edu/dept/itss/docs/oracle/9i/server.920/a96536.pdf
Oracle part number 96536-02.

[5] Gorman Tim. [2001] The Search For Intelligent Life in the Cost-Based Optimizer.
[www] http://www.evdbt.com/SearchIntelligenceCBO.doc

[6] Kyte Tom. [2005] All about binds.
[www] http://asktom.oracle.com/pls/ask/z?p_url=download_file%3Fp_file%3D3256111530431915054&p_cat=AllAboutBinds.zip&p_company=10

[7] Howard J. Rogers.[2004] DBA Advice.
[www] http://www.dizwell.com/html/dba_tips.html

10.1 Good reading

[1] AskTom – Everything about Oracle:
http://asktom.oracle.com

[2] Grid adoption for large Cramer implementation
http://www.chinagrid.net/grid/Conference/GridWorld2005/OracleKey.ppt

11 Appendix

11.1 Tkprof

11.1.1 Command reference

```
tkprof   trace_file   output_file
[explain = username/password]
[aggregate = yes/no]
[insert = stat_file_name]
[print = number]
[record = sql_file_name]
[sort = parameter]
[sys = yes/no]
[table = schema.table_name]
```

Parameter	Explanation	
explain	Used to retrieve explain plan in the report. Not needed, the explain plan will be in the report anyway.	
aggregate	Tells tkprof whether similar statements should be grouped together.	
insert	Creates a script that can be used to insert the data into a table.	
print	Tells Tkprof how many statements to print in the report.	
record	Creates a script with a copy of each SQL statement that can be used for the purpose of copy-and-paste and such	
sort	Used to sort the data in the trace file based on the criteria beneath.	
	prscnt	Number of times parsed.
	prscpu	CPU time spent parsing.
	prsela	Elapsed time spent parsing.
	prsdsk	Number of physical reads from disk during parse.
	prsqry	Number of consistent mode block reads during parse.
	prscu	Number of current mode block reads during parse.
	prsmis	Number of library cache misses during parse.
	execnt	Number of executes.
	execpu	CPU time spent executing.
	exeela	Elapsed time spent executing.
	exedsk	Number of physical reads from disk during execute.
	exedsk	Number of physical reads from disk during execute.
	exeqry	Number of consistent mode block reads during execute.
	execu	Number of current mode block reads during execute.
	exerow	Number of rows processed during execute.
	exemis	Number of library cache misses during execute
	fchcnt	Number of fetches.
	fchcpu	CPU time spent fetching.
	fchela	Elapsed time spent fetching.

	fchdsk	Number of physical reads from disk during fetch.
	fchqry	Number of consistent mode block reads during fetch.
	fchcu	Number of current mode block reads during fetch.
	fchrow	Number of rows fetched.
sys	Tells Tkprof whether to remove all system SQL calls. These calls are done for space management and similar	
table	The table that Tkprof uses to insert values for the explain plan when explain parameter is used.	

Table 15 Tkprof parameters.

11.1.2 Example

Question for top-ten most CPU and time consuming statements:

```
tkprof oracl_ora_324.trc report.txt sort=exeela,fchela print=10
```

11.1.3 References

[1] Chapter 10, page 347 in Oracle9i – Database performance tuning guide.

11.2 Explain plan

11.2.1 Command reference

```
explain plan [set statement_id = string in single quotes]
[into plan table name]
for SQLstatement;
```

Parameter	Explanation
set statement_id	Creates a unique id for the statement so that the plan table later can be queried with that id.
Into	Inserts the data in a different table than the default plan_table.
For	Takes the statement that the plan is to be built upon.

Table 16 Explain plan parameters.

11.2.2 Example

```
explain plan into new_plan_table for select * from dual;
```

11.2.3 References

[1] Chapter 9, page 315 in Oracle9i – Database performance tuning guide.

11.3 Statspack

11.3.1 References

[1] Chapter 21, page 607 in Oracle9i – Database performance tuning guide.

[2] spdoc.txt in `ORACLE_HOME/rdbms/admin directory.`

11.4 FAQ

Q: Cant revoke alter session for user?

A: Check if the user has the alter session privilege by issuing:

```
select * from dba_role_privs where grantee='USER';
```

Q: Don't know if a trigger exists?

A: To check all triggers that exists in the database use:

```
select trigger_name from all_triggers;
```

To check triggers that is owned by a specific user use:

```
select trigger_name from user_triggers;
```

Q: Explain plan has no table to insert its results?

A: The `plan_table` that Explain plan uses for its results may not have been constructed at some earlier point. There is a script that creates a plan table, it is situated at the path: `ORACLE_HOME/rdbms/admin/utlxplan.sql`

1 Log on to the Oracle resource database server by issuing the command:

```
rlogin DBServer -l oracle_user
```

2 Run the following command in order to log in to the database:

```
oracle_user@DBServer> sqlplus /nolog

SQL> connect sys/sys_password@ResourceDatabase as sysdba
```

3 Execute the script and create synonyms with the grant and the plan table is ready to use.

```
SQL> @ ?/rdbms/admin/utlxplan.sql;
SQL> create public synonym plan_table for plan_table;
SQL> grant all on plan_table to public;
```

This will make it possible for all users to run explain plans. If that is not wanted exchange public with the user that should be able to run the explain plan.

Q: What table and column does *index_name* belong to?

A: To retrieve what table and column an index belongs to use:

```
select table_name, column_name from dba_ind_columns where
index_name='INDEX_NAME';
```

Q: Are there any indexes on table *table_name*?

A: To retrieve the indexes on a particular table use:

```
select index_name, uniqueness from user_indexes where
table_name = 'TABLE_NAME';
```

To see which columns that is indexed:

```
select index_name, column_position, column_name from
user_tables order by index_name, column_position;
```

Q: The max_dump_file_size is reported to be too small?

A: The maximum size of the dump file is set by a parameter in the Oracle environment. To change this for a session the command is:

```
SQL> alter session set max_dump_file_size=value;
```

Where value are either an integer in bytes or the string unlimited for unlimited file size.

Q: How can I see all of these Oracle session/instance parameters?

A: The parameters that affect Oracle can be viewed with a simple command:

```
SQL> show parameter parameter_name;
```

Where parameter name can be the complete name or a substring of the name.

Q: What is the statistics_level parameter?

A: The statistics_level parameter tells Oracle how much statistics it should collect as standard. The default value is typical; in this mode the database will collect timed_statistics.

To control the parameter use:

```
SQL> show parameter statistics_level;
```

If this is set to none no time statistics are collected and the `timed_statistics` parameter must be set, timed statistics can easily be set on in the instance by:

```
SQL> alter session set timed_statistics=true;
```

Q: Why is the Tkprof report saying parse=execute when testing in SQL*Plus?

A: This is the way that SQL*Plus works, it sends every statement off to the parser so the parse count will equal the execute count.

Q: I want to trace all SQL statements executed on the server, how?

A: To trace all SQL that is executed on the server set the sql_trace parameter in the init.ora file to true and restart the database.

Note: This should only be done during testing because the I/O subsystem may become filled with writing to the trace files.