|  |  |
| --- | --- |
|  |  |

Audit de la base Oracle MARCELLO

Le Bon Marché

Workshop for Rapport d’audit

|  |  |
| --- | --- |
|  | **Version :** V 1.0  **Date de la version**: 16/10/2017  **Etat du document :** Draft |

Diffusion

|  |  |
| --- | --- |
| Nom | Société |
| … | UMANIS |
|  |  |

Historique des modifications

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Auteur | Objet de la version et liste des modifications |
| V 1.0 | 03/10/2017 | JEA | Initialisation |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

VAlidation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Emetteur | Date / Visa | Vérificateur | Date / Visa | Approbateur | Date / Visa |
| V 1.0 | JEA | 16/10/2017 |  |  |  |  |

**Date d'application :** 16/10/2017

Table des matières

[1 Objet de l’audit 4](#_Toc495935371)

[2 Environnement 5](#_Toc495935372)

[3 Les points forts 6](#_Toc495935373)

[4 Les points faibles 7](#_Toc495935374)

[5 Les opportunités d’amélioration 8](#_Toc495935375)

[6 Annexes 9](#_Toc495935376)

# Annexes

## Script de compilation des objets en mode debug

Declare

cVc\_owner constant varchar2 (30) := 'STORELAND' ;

begin

for laChose in (select name, type

from all\_plsql\_object\_settings

where type in ( 'PACKAGE'

, 'PACKAGE BODY'

, 'PROCEDURE'

, 'FUNCTION'

, 'TRIGGER'

)

and owner = cVc\_owner

and plsql\_debug = 'TRUE'

order by type asc, name asc

) loop

begin

case laChose.type

when 'PACKAGE' then

execute immediate 'alter package ' || laChose.name || ' compile' ;

when 'PACKAGE BODY' then

execute immediate 'alter package ' || laChose.name || ' compile body' ;

when 'PROCEDURE' then

execute immediate 'alter procedure ' || laChose.name || ' compile' ;

when 'FUNCTION' then

execute immediate 'alter function ' || laChose.name || ' compile' ;

when 'TRIGGER' then

execute immediate 'alter trigger ' || laChose.name || ' compile' ;

end case;

exception

when others then

dbms\_output.put\_line('Erreur sur [' || laChose.name || ', ' || laChose.type || '] : ' || sqlerrm);

end;

end loop;

end;

/

## Script de compilation des objets invalides

Declare

cVc\_owner constant varchar2 (30) := 'STORELAND' ;

begin

for laChose in (select object\_name, object\_type

from all\_objects

where object\_type in ( 'PACKAGE'

, 'PACKAGE BODY'

, 'PROCEDURE'

, 'FUNCTION'

, 'VIEW'

, 'TRIGGER'

)

and owner = cVc\_owner

and status = 'INVALID'

order by object\_type asc

) loop

begin

case laChose.object\_type

when 'PACKAGE' then

execute immediate 'alter package ' || laChose.object\_name || ' compile' ;

when 'PACKAGE BODY' then

execute immediate 'alter package ' || laChose.object\_name || ' compile body' ;

when 'PROCEDURE' then

execute immediate 'alter procedure ' || laChose.object\_name || ' compile' ;

when 'FUNCTION' then

execute immediate 'alter function ' || laChose.object\_name || ' compile' ;

when 'VIEW' then

execute immediate 'alter view ' || laChose.object\_name || ' compile' ;

when 'TRIGGER' then

execute immediate 'alter trigger ' || laChose.object\_name || ' compile' ;

end case;

exception

when others then

dbms\_output.put\_line('Erreur sur [' || laChose.object\_name || ', ' || laChose.object\_type || '] : ' || sqlerrm);

end;

end loop;

end;

/

## Reste à faire

### Vérifications de la sort\_area\_size ou (si elle est utilisée de la pga\_aggregate\_target)

NB1.. Attention vérifier si *workarea\_size\_policy =auto*=> dans ce cas, ce paramétrage n’est plus utilisé.

NB2. Dans init.ora, **sort\_area\_size=1048576**

-- Sur l'utilisation de la mémpoire pour les tris :

-- objet : produire un graphique Nombre de tris en fonction des jours de la semaine

set pages 9999

column sorts\_memory format 999,999,999

column sorts\_disk format 999,999,999

column ratio format .99999

select to\_char (snap\_time, 'day')

, avg (newmem.value - oldmem.value) sorts\_memory)

, avg (newdsk.value - olddsk.value) sorts\_disk

from perfstat.stats$sysstat oldmem

, perfstat.stats$sysstat newmem

, perfstat.stats$sysstat olddsk

, perfstat.stats$sysstat sn

where newdsk.snap\_id = sn.snap\_id

and olddsk.snap\_id = sn.snap\_id -1

and mewmem.snap\_id = sn.snap\_id

and oldmem.snap\_id = sn.snap\_id -1

and oldmem.name = 'sorts (memory)'

and newmem.name = 'sorts (memory)'

and olddsk.name = 'sorts (disk)'

and newdsk.name = 'sorts (disk)'

and newmem.value - oldmem.value > 0

group by to\_char (snap\_time, 'day')

;

----

-- objet : produire un graphique Nombre de tris en fonction des tranches horaires de la journée

set pages 9999

column sorts\_memory format 999,999,999

column sorts\_disk format 999,999,999

column ratio format .99999

select to\_char (snap\_time, 'hh24')

, avg (newmem.value - oldmem.value) sorts\_memory)

, avg (newdsk.value - olddsk.value) sorts\_disk

from perfstat.stats$sysstat oldmem

, perfstat.stats$sysstat newmem

, perfstat.stats$sysstat olddsk

, perfstat.stats$sysstat sn

where newdsk.snap\_id = sn.snap\_id

and olddsk.snap\_id = sn.snap\_id -1

and mewmem.snap\_id = sn.snap\_id

and oldmem.snap\_id = sn.snap\_id -1

and oldmem.name = 'sorts (memory)'

and newmem.name = 'sorts (memory)'

and olddsk.name = 'sorts (disk)'

and newdsk.name = 'sorts (disk)'

and newmem.value - oldmem.value > 0

group by to\_char (snap\_time, 'hh24')

;

## Mail de Christophe GONZALVEZ

### Mail de Christophe GONZALVEZ du 20171004

Il faudrait qu’en préambule tu décrives l’environnement OS et Rdbms en donnant le versioning.

Il faut également que tu donnes le délai de rétention des rapports statpack.

Il faudra aussi que

-          Le client te fournisse le plan de production afin que tu puisses faire le lien entre les activités constatées dans les statpacks et l’activité de production. Cela te permettra d’identifier des traitements gourmands.

-          Tu fasses un tour d’horizon du paramétrage des bases de données et de la configuration network

Lorsque tu auras fini tes différentes analyses, il faudra que pour chaque point tu donnes une action à réaliser.

### Mail de Christophe GONZALVEZ du 20170927

De mémoire, l’audit est motivé par une envie du client d’avoir une vision extérieure sur leur base de données en terme d’implémentation logique ( répartition tables, index, lob) et physique ( localisation tablespace, nbr d’extents, etc..).

Une analyse de l’activité de la base de données devra aussi être fait à partir de rapport STATPACK. Normalement ils doivent l’avoir installé, c’est un point à vérifier avec eux s’ils l’ont installé ou si il faut que tu le fasses. Il faudra également voir si la configuration network est correct.

:

## Sites à consulter

<http://dbspecialists.com/not-licensed-for-awr-use-statspack-instead/>

<https://perfstat.wordpress.com/2014/09/04/capturing-long-running-sql-in-statspack/>

<https://blog.dbi-services.com/awrrpt-and-spreport-in-multitenant/>

<https://www.akadia.com/services/ora_statspack_survival_guide.html>

<https://jonathanlewis.wordpress.com/category/oracle/statspack/>

<http://www.remote-dba.net/t_op_sql_high_use.htm>

Aussi :

<http://orafrance.developpez.com/dbahelp/>

## ASH (àctive session History)

Voir V$ACTIVE\_SESSION\_HISTORY 🡪 ppt\*.pdf pp 21/50

Existe-t-il ce fichier sur le serveur (Simpleash.sql) ?

### ASH: Top SQL

* Returns most active SQL in the past minute

select sql\_id

, count(\*)

, round(count(\*)

/ sum(count(\*)) over (), 2)pctload

from v$active\_session\_history

where sample\_time > sysdate -1/24/60

and session\_type <> 'BACKGROUND'

group by sql\_id

order by count(\*) desc

;

### ASH: Top IO SQL

Returns SQL spending most time doing I/Os

Similarly, can do Top Sessions, Top Files, Top Objects

select ash.sql\_id, count(1)

from v$active\_session\_history ash

, v$event\_name evt

where ash.sample\_time > sysdate –1/24/60

and ash.session\_state = 'WAITING'

and ash.event\_id = evt.event\_id

and evt.wait\_class = 'User I/O'

group by sql\_id

order by count(1) desc

;

Voir DBA\_HIST\_ACTIVE\_SESS\_HISTORY

### ASH : Bad SQL

select e.event

, e.total\_waits -nvl(b.total\_waits,0) total\_waits

, e.time\_waited -nvl(b.time\_waited,0) time\_waited

from v$active\_session\_history b

, v$active\_session\_history e

, stats$snapshot sn

Where snap\_time > sysdate - &1 --- A fournir

And e.event not like '%timer'

And e.event not like '%message%'

And e.event not like '%slave wait%'

And e.snap\_id =sn.snap\_id

And b.snap\_id = e.snap\_id-1

And b.event = e.event

And e.total\_timeouts > 100

And (e.total\_waits -b.total\_waits > 100

or e.time\_waited -b.time\_waited > 100)

;

select sum(a.time\_waited) total\_time

from v$active\_session\_history a

, v$event\_name b

where a.event# = b.event#

and sample\_time between to\_date ('201710201300', 'yyyymmddhh24mi')

and to\_date ('201710202001', 'yyyymmddhh24mi')

and b.wait\_class = 'User I/O'

;

select sess\_id

, username

, program

, wait\_event

, sess\_time

, round(100 \* (sess\_time / total\_time),2) pct\_time\_waited

from (select a.session\_id sess\_id

, decode(session\_type, 'background', session\_type, c.username) username

, a.program program

, b.name wait\_event

, sum(a.time\_waited) sess\_time

from sys.v\_$active\_session\_history a

, sys.v\_$event\_name b

, sys.dba\_users c

where a.event# = b.event#

and a.user\_id = c.user\_id

and sample\_time between to\_date ('201710201300', 'yyyymmddhh24mi')

and to\_date ('201710202001', 'yyyymmddhh24mi')

and b.wait\_class = 'User I/O'

group by a.session\_id

, decode(session\_type, 'background', session\_type, c.username)

, a.program

, b.name

)

, (select sum(a.time\_waited) total\_time

from sys.v\_$active\_session\_history a

, sys.v\_$event\_name b

where a.event# = b.event#

and sample\_time between to\_date ('201710201300', 'yyyymmddhh24mi')

and to\_date ('201710202001', 'yyyymmddhh24mi')

and b.wait\_class = 'User I/O'

)

order by 6 desc

;

Voir sys.v\_$sys\_time\_model

select case db\_stat\_name

when 'parse time elapsed' then

'soft parse time'

else db\_stat\_name

end db\_stat\_name

, case db\_stat\_name

when 'sql execute elapsed time' then

time\_secs - plsql\_time

when 'parse time elapsed' then

time\_secs - hard\_parse\_time

else time\_secs

end time\_secs

, case db\_stat\_name

when 'sql execute elapsed time' then

round(100 \* (time\_secs - plsql\_time) / db\_time,2)

when 'parse time elapsed' then

round(100 \* (time\_secs - hard\_parse\_time) / db\_time,2)

else round(100 \* time\_secs / db\_time,2)

end pct\_time

from (select stat\_name db\_stat\_name

, round((value / 1000000),3) time\_secs

from sys.v\_$sys\_time\_model

where stat\_name not in('DB time'

,'background elapsed time'

,'background cpu time'

,'DB CPU'

)

)

, (select round((value / 1000000),3) db\_time

from sys.v\_$sys\_time\_model

where stat\_name = 'DB time'

)

, (select round((value / 1000000),3) plsql\_time

from sys.v\_$sys\_time\_model

where stat\_name = 'PL/SQL execution elapsed time'

)

, (select round((value / 1000000),3) hard\_parse\_time

from sys.v\_$sys\_time\_model

where stat\_name = 'hard parse elapsed time'

)

order by 2 desc

;

Dire quel type de process pass le plus de temps à faire attendre :

select to\_char(a.end\_time,'DD-MON-YYYY HH:MI:SS') end\_time

, b.wait\_class

, round((a.time\_waited / 100),2) time\_waited

from sys.v\_$waitclassmetric\_history a

, sys.v\_$system\_wait\_class b

where a.wait\_class# = b.wait\_class#

and b.wait\_class != 'Idle'

order by 1,2

;

--

## Response-Time Analysis Made Easy

Starting at the global or system level, DBAs typically want answers to these questions:

* In general, how well is my database running? What defines efficiency?
* What average response time are my users experiencing?
* Which activities affect overall response time the most?

### Part of the answer to how well, in general, a database is running

The query above helps you determine if your database is currently experiencing a high percentage of waits/bottlenecks vs. smoothly running operations. The Database CPU Time Ratio is calculated by dividing the amount of CPU expended in the database by the amount of "database time," which is defined as the time spent by the database on user-level calls (with instance background process activity being excluded). High values (90-95+ percent) are good and indicate few wait/bottleneck actions, but take this threshold only as a general rule of thumb because every system is different

select metric\_name,

value

from SYS.V\_$SYSMETRIC

where metric\_name IN ( 'Database CPU Time Ratio'

, 'Database Wait Time Ratio'

)

and intsize\_csec = (select max(intsize\_csec)

from SYS.V\_$SYSMETRIC

)

;

You can also take a quick look over the last hour to see if the database has experienced any dips in overall performance by using this query:

select end\_time

, value

from sys.v\_$sysmetric\_history

where metric\_name = 'Database CPU Time Ratio'

order by 1

;

And, you can get a good idea of the minimum, maximum, and average values of overall database efficiency by querying the V$SYSMETRIC\_SUMMARY view with a query such as this:

select Case metric\_name

When 'SQL Service Response Time' then 'SQL Service Response Time (secs)'

When 'Response Time Per Txn' then 'Response Time Per Txn (secs)'

Else metric\_name

End metric\_name

, Case metric\_name

When 'SQL Service Response Time' then round((minval / 100),2)

When 'Response Time Per Txn' then round((minval / 100),2)

Else minval

End mininum

, Case metric\_name

When 'SQL Service Response Time' then round((maxval / 100),2)

When 'Response Time Per Txn' then round((maxval / 100),2)

Else maxval

End maximum

, Case metric\_name

When 'SQL Service Response Time' then round((average / 100),2)

When 'Response Time Per Txn' then round((average / 100),2)

Else average

End average

from sys.v\_$sysmetric\_summary

where metric\_name in ( 'CPU Usage Per Sec'

, 'CPU Usage Per Txn'

, 'Database CPU Time Ratio'

, 'Database Wait Time Ratio'

, 'Executions Per Sec'

, 'Executions Per Txn'

, 'Response Time Per Txn'

, 'SQL Service Response Time'

, 'User Transaction Per Sec'

)

Order by 1

;

The query above contains more response-time metrics than simply the Database CPU and Wait Time Ratios (we'll cover those later), but you can see the benefit in being able to acquire this information. For this particular instance, the average Database CPU Time Ratio is 94, which is well within our acceptable limits.

The next question DBAs pose at the system level involves the average level of response time that their user community is experiencing. (Prior to Oracle Database 10*g* this type of data was difficult to capture, but not anymore.) The query shown above that interrogates the V$SYSMETRIC\_SUMMARY view tells us what we need to know. If complaints of unacceptable response times are mounting from users, the DBA can check the Response Time Per Txn and SQL Service Response Time metrics to see if a database issue exists. For example, the statistics shown above report that the maximum response time per user transaction has been only .28 second, with the average response time being a blazing .08 second. Oracle certainly wouldn't be to blame in this case.

If, however, response times are longer than desired, the DBA will then want to know what types of user activities are responsible for making the database work so hard. Again, before Oracle Database 10*g*, this information was more difficult to acquire, but now the answer is only a query away:

select case db\_stat\_name

when 'parse time elapsed' then

'soft parse time'

else db\_stat\_name

end db\_stat\_name

, case db\_stat\_name

when 'sql execute elapsed time' then

time\_secs - plsql\_time

when 'parse time elapsed' then

time\_secs - hard\_parse\_time

else time\_secs

end time\_secs

, case db\_stat\_name

when 'sql execute elapsed time' then

round(100 \* (time\_secs - plsql\_time) / db\_time,2)

when 'parse time elapsed' then

round(100 \* (time\_secs - hard\_parse\_time) / db\_time,2)

else round(100 \* time\_secs / db\_time,2)

end pct\_time

from (select stat\_name db\_stat\_name

, round((value / 1000000),3) time\_secs

from sys.v\_$sys\_time\_model

where stat\_name not in ( 'DB time','background elapsed time'

, 'background cpu time','DB CPU')

)

, (select round((value / 1000000),3) db\_time

from sys.v\_$sys\_time\_model

where stat\_name = 'DB time'

)

, (select round((value / 1000000),3) plsql\_time

from sys.v\_$sys\_time\_model

where stat\_name = 'PL/SQL execution elapsed time'

)

, (select round((value / 1000000),3) hard\_parse\_time

from sys.v\_$sys\_time\_model

where stat\_name = 'hard parse elapsed time'

)

order by 2 desc

;

In addition to active time, a DBA will want to know the global wait times as well. Prior to Oracle Database 10g, a DBA had to view individual wait events to understand waits and bottlenecks, but now Oracle provides a summary/rollup mechanism for waits via wait classes:

select wait\_class

, total\_waits

, round(100 \* (total\_waits / sum\_waits),2) pct\_waits

, round((time\_waited / 100),2) time\_waited\_secs

, round(100 \* (time\_waited / sum\_time),2) pct\_time

from (select wait\_class

, total\_waits

, time\_waited

from v$system\_wait\_class

where wait\_class != 'Idle'

)

, (select sum(total\_waits) sum\_waits

, sum(time\_waited) sum\_time

from v$system\_wait\_class

where wait\_class != 'Idle'

)

order by 5 desc

;

It's much easier to tell now that the bulk of overall wait time is due, for example, to user I/O waits than to try to tally individual wait events to get a global picture.

As with response-time metrics, you can also look back in time over the last hour with a query like this one:

select to\_char(a.end\_time,'DD-MON-YYYY HH:MI:SS') end\_time

, b.wait\_class

, round((a.time\_waited / 100),2) time\_waited

from sys.v\_$waitclassmetric\_history a

, sys.v\_$system\_wait\_class b

where a.wait\_class# = b.wait\_class#

and b.wait\_class != 'Idle'

order by 1,2

;

You can, of course, just focus on a single SID with the V$SESS\_TIME\_MODEL view and obtain data for all statistical areas of a session. You can also view current session wait activity using the new wait classes using the following query:

select a.sid

, b.username

, a.wait\_class

, a.total\_waits

, round((a.time\_waited / 100),2) time\_waited\_secs

from sys.v\_$session\_wait\_class a

, sys.v\_$session b

where b.sid = a.sid

and b.username is not null

and a.wait\_class != 'Idle'

order by 5 desc

;

After this stage, you can check the standard individual wait events as you've been able to do in earlier versions of Oracle with V$SESSION\_WAIT and V$SESSION\_EVENT. You'll also find the new wait classes in these two modified views with Oracle Database 10*g*.

If you need to look back in time to discover what sessions were logged on and consuming the most resources, you can use the following query. In the example below, we're looking at activity from midnight to 5 a.m. on November 21, 2004, that involved user I/O waits:

select sess\_id

, username

, program

, wait\_event

, sess\_time

, round(100 \* (sess\_time / total\_time),2) pct\_time\_waited

from (select a.session\_id sess\_id

, decode(session\_type, 'background', session\_type, c.username) username

, a.program program

, b.name wait\_event

, sum(a.time\_waited) sess\_time

from sys.v\_$active\_session\_history a

, sys.v\_$event\_name b

, sys.dba\_users c

where a.event# = b.event#

and a.user\_id = c.user\_id

and sample\_time between to\_date ('201710201200' , 'yyyymmddhh24mi')

and to\_date ('201710201200' , 'yyyymmddhh24mi')

and b.wait\_class = 'User I/O'

group by a.session\_id

, decode(session\_type, 'background', session\_type, c.username)

, a.program

, b.name

),

(select sum(a.time\_waited) total\_time

from sys.v\_$active\_session\_history a

, sys.v\_$event\_name b

where a.event# = b.event#

and sample\_time between to\_date ('201710201200' , 'yyyymmddhh24mi')

and to\_date ('201710201200' , 'yyyymmddhh24mi')

and b.wait\_class = 'User I/O'

)

order by 6 desc

;

The Oracle Database 10*g* V$ACTIVE\_SESSION\_HISTORY view comes into play here to provide an insightful look back in time at session experiences for a given time period. This view gives you a lot of excellent information without the need for laborious tracing functions. We'll see more use of it in the next section, which deals with analyzing the response times of SQL statements.

SQL Response-Time Analysis

Examining the response time of SQL statements became easier in Oracle9*i*, and with Oracle Database 10*g*, DBAs have many tools at their disposal to help them track inefficient database code.

Historically the applicable V$ view here has been V$SQLAREA. Starting with Oracle9*i*, Oracle added the ELAPSED\_TIME and CPU\_TIME columns, which have been a huge help in determining the actual end user experience of a SQL statement execution (at least, when dividing them by the EXECUTIONS column, which produces the average amount of time per execution).

In Oracle Database 10*g*, six new wait-related and timing columns have been added to V$SQLAREA:

* APPLICATION\_WAIT\_TIME
* CONCURRENCY\_WAIT\_TIME
* CLUSTER\_WAIT\_TIME
* USER\_IO\_WAIT\_TIME
* PLSQL\_EXEC\_TIME
* JAVA\_EXEC\_TIME

The new columns are helpful in determining, for example, the amount of time that a procedure spends in PL/SQL code vs. standard SQL execution, and if a SQL statement has experienced any particular user I/O waits. For example, a query you could use to find the top five SQL statements with the highest user I/O waits would be:

select \*

from (select sql\_text

, sql\_id

, elapsed\_time

, cpu\_time

, user\_io\_wait\_time

from sys.v\_$sqlarea

order by 5 desc

)

where rownum < 6;

Of course, getting the SQL statements with the highest elapsed time or wait time is good, but you need more detail to get to the heart of the matterwhich is where the V$ACTIVE\_SESSION\_HISTORY view again comes into play. With this view, you can find out what actual wait events delayed the SQL statement along with the actual files, objects, and object blocks that caused the waits (where applicable).

For example, let's say you've found a particular SQL statement that appears to be extremely deficient in terms of user I/O wait time. You can issue the following query to get the individual wait events associated with the query along with the corresponding wait times, files, and objects that were the source of those waits:

select event,

time\_waited,

owner,

object\_name,

current\_file#,

current\_block#

from sys.v\_$active\_session\_history a,

sys.dba\_objects b

where sql\_id = '6gvch1xu9ca3g' and

a.current\_obj# = b.object\_id and

time\_waited <> 0

;

## Analysing Statspack –

Notes on <https://jonathanlewis.wordpress.com/2008/02/18/analysing-statspack/>

### How do you interpret Statspack data ?

Why do you want to interpret Statspack data ?

## Stored Outline

How to force Oracle to execute SQL statements the same way no matter what occurs in the database or the OS level.

SQL> GRANT QUERY REWRITE TO SYSTEM;  
Grant succeeded.  
SQL> CONNECT SYSTEM/MANAGER

The following command creates a stored outline named EMPLOYEES with a category of SALARY.

SQL> CREATE OR REPLACE OUTLINE EMPLOYEES  
 2   FOR CATEGORY SALARY ON  
 3   SELECT  ENAME, SAL, LOC  
 4   FROM    EMP, DEPT  
 5   WHERE   EMP.DEPTNO = DEPT.DEPTNO;  
Outline created.

Several data dictionary views get updated when you create stored outlines. The view *user\_outline\_hints* is actually looking at *outln.ol$\_hints*. The following query shows that Oracle stores ten hints for the SQL statement joining the EMP and DEPT tables in category SALARY.

 SQL> SELECT  HINT  
 2   FROM    USER\_OUTLINE\_HINTS  
 3   WHERE   NAME = 'EMPLOYEES'  
 4\*  ORDER   BY HINT;

HINT  
------------------------------  
FULL(DEPT)  
FULL(EMP)  
NOREWRITE  
NOREWRITE  
NO\_EXPAND  
NO\_FACT(DEPT)  
NO\_FACT(EMP)  
ORDERED  
PQ\_DISTRIBUTE(EMP NONE NONE)  
USE\_HASH(EMP)  
10 rows selected.

Another data dictionary view, *user\_outlines*, is actually looking at *outln.ol$*. This view is updated when you create a stored outline. Notice that the outline has not been used.

SQL> SELECT NAME,  
 2          CATEGORY,  
 3          USED  
 4\*  FROM   USER\_OUTLINES;

NAME         CATEGORY     USED  
------------ ------------ ---------  
EMPLOYEES    SALARY       UNUSED

One method of insuring that Oracle uses the cost-based optimizer is to analyze one or more tables used in a query. The following two statements analyze the EMP and DEPT tables.

SQL> ANALYZE TABLE EMP COMPUTE STATISTICS;

Table analyzed.

SQL> ANALYZE TABLE DEPT COMPUTE STATISTICS;

Table analyzed.

If *query\_rewrite\_enabled*is not set to true, Oracle will not use a stored outline or materialized view.

SQL> ALTER SESSION SET QUERY\_REWRITE\_ENABLED = TRUE;

Session altered.

To cause the cost-based optimizer to rewrite your query and use the stored outline for CATEGORY SALARY, enter the following command.

SQL> ALTER SESSION SET

     USE\_STORED\_OUT LINES = SALARY;

Session altered.

Now, the moment of truth we have all been waiting for. Write the query stored in the outline EMPLOYEES category SALARY, and query the data dictionary to see if Oracle used your outline to execute the query. Because the used column shows "USED," Oracle did use your stored outline.

SQL> SELECT ENAME, SAL, LOC

 2 FROM EMP, DEPT

 3\* WHERE EMP.DEPTNO = DEPT.DEPTNO;

...... Output Omitted Here ...

SQL> SELECT NAME,

 2 CATEGORY,

 3 USED

 4\* FROM USER\_OUTLINES;

NAME CATEGORY USED

--------- -------- -----

EMPLOYEES SALARY USED

To see the EXPLAIN PLAN output for this query, use AUTOTRACE. Notice that Oracle uses a HASH JOINand a full table scan on each of the two tables, which are hints in *user\_outline\_hints*.

SQL> SET AUTOTRACE TRACEONLY EXPLAIN  
SQL> SELECT ENAME, SAL, LOC  
 2   FROM   EMP,   DEPT  
 3\*  WHERE  EMP.DEPTNO = DEPT.DEPTNO;  
Execution Plan  
-----------------------------------------  
0   SELECT STATEMENT Optimizer=CHOOSE  
       (Cost=3 Card=14 Bytes=266)  
1 0 HASH JOIN (Cost=3 Card=14 Bytes=266)  
2 1   TABLE ACCESS (FULL) OF 'DEPT'  
       (Cost=1 Card=4 Bytes=36)  
3 1 TABLE ACCESS (FULL) OF 'EMP'  
       (Cost=1 Card=14 Bytes=140)

### Stored Outlines

Stored outlinescan be very useful, especially when the SQL can't be changed (packaged applications).

In order to use them, you need to capture the SQL statement, and then create an outline with exactly the same SQL statement that uses an alternative access path (perhaps via a hint).

Initially, set up security to create the outlines:

connect 'sys/xxx as sysdba'

grant create any outline to scott;

grant execute on dbms\_outln to scott;

grant execute on dbms\_outln\_edit to scott;

grant select on dba\_outlines to scott;

To begin with, set up the session to allow stored outlines, or to ensure that the *init.ora* parameter *create\_stored\_outlines*is set accordingly.

connect scott/tiger

alter session set create\_stored\_outlines =  TRUE

create or replace outline first\_outline

on select a.emp\_id, a.name, b.dependent\_id, b.name

from emp a,

dependent b

where a.empid\_id = b.emp\_id

Now a public outline has been created, but in order to test it, make a private outline (personal copy).  In order to do this, you need to create the tables to hold the private outline first:

execute dbms\_outln\_edit.create\_edit\_tables;

create private outline first\_private\_outline from first\_outline;

--- check out what is in the outline table by default,

--- just from creating the private outline

select \* from ol$hints

where ol\_name = 'first\_private\_outline';

--- change which ever access path we are interested in

--- for example, could change a nested\_lop join to

--- a hash join

update ol$hints set hint\_text='USE\_HASH(B)'

where hint#=5;

commit;

--- check the result, and be sure the update worked

select \* from ol$hints

where ol\_name = 'first\_private\_outline';

--- resync the outline

execute dbms\_outline\_edit.refresh\_private\_outline ('first\_private\_outline')

--- test it out

alter session set use\_private\_outlines=true;

set autotrace on explain

select a.emp\_id, a.name, b.dependent\_id, b.name

from emp a,

dependent b

where a.empid\_id = b.emp\_id

;

--- if new access plan is correct, make the outline public

--- for use by everyone

create or replace second\_outline from private first\_private\_outline;

Now, whenever the original SQL is executed, it will use the revised access plan instead of the original (assuming the *init.ora* parameter *use\_stored\_outlines*is set to true).

Note that the SQL used to create the stored outline and the SQL that is run after the outline is created, must be IDENTICAL, or it won't use the outline.  Identical means the text must be in the same case, same amount of white space, etc.  Oracle 9i and 10g are more forgiving on this requirement than Oracle 8i.

The catalog tables *dba\_outlines*and *dba\_outline\_hints*contain relevant information about the outlines.  They are based on catalog tables OUTLN.OL$ and OUTLN.OL$HINTS.

Stored Outlines have been greatly enhanced with the new Oracle 10g features of SQL Profiles.

**Swapping stored Outlines**

MOSC note 92202 .1 describes a procedure to tune SQL that you cannot touch by performing these steps:

1. Identify the sub-optimal SQL and create a stored outline
2. Tune an equivalent query with a faster execution plan and create a stored outline
3. Swap the bad stored outline for the tuned stored outline

Oracle provides this example for swapping the outlines:

UPDATE   
OUTLN.OL$HINTS  
SET   
OL\_NAME=DECODE   
(OL\_NAME,'HINTSQL','ORIGINALSQL','ORIGINALSQL','HINTSQL')  
WHERE   
OL\_NAME IN ('HINTSQL','ORIGINALSQL');  
Commit;

## SQL Profiles to replace stored outlines

 A procedure to tune SQL that you cannot touch by performing these steps:

1. Identify the sub-optimal SQL and create a stored outline
2. Tune an equivalent query with a faster execution plan and create a stored outline
3. Swap the bad stored outline for the tuned stored outline

Oracle provides this example for swapping the outlines:

UPDATE   
   OUTLN.OL$HINTS  
SET   
   OL\_NAME=DECODE    
     (OL\_NAME,'HINTSQL','ORIGINALSQL','ORIGINALSQL','HINTSQL')  
WHERE   
   OL\_NAME IN ('HINTSQL','ORIGINALSQL');  
Commit;

**SQL Profiles**

The Oracle 10g automatic tuning advisor allowed us to implement tuning suggestions in the form of SQL profiles that will improve performance, and SQL profiles can be used the same way as stored outlines (optimizer plan stability).  The SQL Profile is a collection of the historical information of prior runs of the SQL statement, comparison details of the actual and estimated cardinality and predicate selectivity, etc.

A SQL Profile is stored persistently in the data dictionary, so it does not require any application code changes.

A SQL profile helps generate a better execution plan than the normal optimization because it is tested against a real-world workload in the SQL Tuning Set (STS). Additional tasks like checking for advanced predicate selectivity, correlation between columns, join skews, and complex predicates such as functions, help in profiling the SQL statement. Once a SQL statement is profiled and stored, differing execution plans can be invoked at will.

## Take a look at this too

<http://www.dba-oracle.com/art_orafaq_oracle_sql_tune_table_scan.htm>

<https://oracle-base.com/articles/misc/outlines>

<http://www.dba-oracle.com/oracle11g/oracle11g_sql_plan_management.htm>

<http://www.dba-oracle.com/art_otn_cbo_p3.htm>

<http://www.dba-oracle.com/art_sql_philosophy.htm>

<http://www.oraclemagician.com/index.php?p=1_2_Oracle-Performance>