Red Stack
eArticle

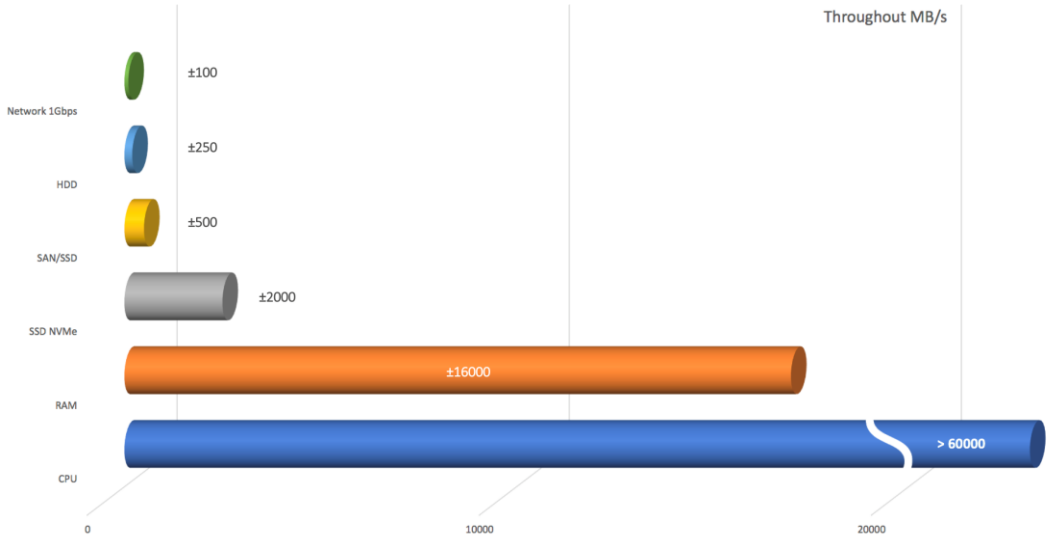
Understanding Oracle Database Performance

Jérôme Dubar, dbi services

You can find numerous sources on Oracle Database Performance and Tuning on the internet and also a lot of books focusing on that. There is a lot of things to talk about, and you can spend days and days to learn how it works and what can be done. But it's a hard work and when performance troubles occur, you need to be fast and effective. This is my approach to understand why an Oracle Database is not so performant and elaborate an action plan to improve performance.

Introduction – System architecture

The fastest place on your database server is the CPU: high speed, multiple cores, ultra-fast cache… Nowadays CPUs can process a huge amount of data first read from memory in a fraction of time. Random Access Memory (RAM) is just below in terms of speed. If your database could reside entirely in RAM, it would be very fast! And Oracle, as some others, can run a database completely in memory if you have the costly option. But for most of the Oracle databases, database files are stored on disk and RAM is only used as a cache for data. Datafiles can be stored on local disks of different kinds, Hard Disk Drives (HDD), Solid State Disks (SSD) and NVMe SSD (SSD directly attached to PCIe bus). Disks can also be on a Storage Area Network (SAN) or on a Network Attached Storage (NAS). But all these technologies are (far) beyond RAM speed.



The fastest components of a system are CPU and RAM

Most activity of your database is reading data (SELECT). When you need blocks of data for executing a statement, Oracle first reads them from disk and put them in a memory area called the SGA. Later, if you need to access these same data blocks, Oracle reads them directly from SGA, so reads them faster. Data read speed should be very fast if you have enough SGA for the most frequent statements and if your instance is running since a long time (as a freshly started instance has an empty SGA).

Other work of your database is writing data blocks on disk in the datafiles (INSERT/UPDATE/DELETE). But as Oracle is good engineered software, modified blocks are first written to SGA and writes on disks are made by an asynchronous process. Actually, Oracle writes changed blocks to datafiles when he decides to. You have no guarantee that a committed transaction is written to the datafiles unless you perform a manual checkpoint or you shut down your database properly.

The only synchronous disk operation needed when inserting/updating/deleting data is sequential write of changed blocks in redolog files (on commit). As you know, redologs are the crash recovery guarantee, and these files are continuously updated on disk at least as soon as a commit is done. Redologs speed is critical for highly transactional databases. It's why Oracle recommends using dedicated and fast disks for these files. Datafiles can be on slower disks.

Step 1 – Generate a performance report on your database

First step for performance analysis of your database: you'll have to generate a performance report on it. It's the easiest way to collect and consolidate all the performance metrics (even metrics you'll never use) in a single file you will analyse later.

Performance reports can be generated with embedded Oracle tools: AWR or Statspack. You don't need any other extra tool. AWR requires Enterprise Edition and the Diagnostic Pack option. If you got them, AWR is immediately usable.

If you don't have the option or if your database is running on Standard Edition you'll have to use Statspack instead. Statspack doesn't come pre-installed: you have to install it. On Standard Edition as on Enterprise Edition databases. Statspack is free of charge and can be installed straight from the database home invoking a script:

```
sqlplus / as sysdba
@?/rdbms/admin/spcreate
```

You'll easily find how to install it with your favourite search engine. Note that Statspack needs tweaks on 12c, please refer to this note from Franck PACHOT: https://blog.dbi-services.com/statspack-idle-events/

These 2 performance tools (never use both of them on the same database) are working the same way: every hour the tool takes a "picture" (known as a snapshot) of the current performance metrics and flush these metrics into dedicated tables. A performance report is a consolidation of these metrics in a time window between two snapshots.

With AWR, generating a performance report is done by:

```
sqlplus / as sysdba
@?/rdbms/admin/awrrpt
```

With Statspack, generating a performance report is done by:

```
sqlplus / as sysdba
@?/rdbms/admin/spreport
```

You'll have to answer 3 to 5 questions depending on the tool:

- File type for the output, mainly html or text (Statspack output is text only)
- Number of days of snapshot history to choose from (Statspack displays all the snapshots available)
- Snapshot number for the beginning period
- Snapshot number for the ending period
- File name of the report file

Performance of a database is fluctuating, depending on the day, the time of the day, the load, the size of your data, etc. You can start to generate a performance report on the peak period of your database and focus on it. If you don't know the peak period, another approach is to study the average performance of your database for a long period, a complete week for example. Don't hesitate to generate multiple performance reports at different time periods. It will improve your understanding of your database behaviour.

In a few words: Oracle tools are available to generate performance reports. You don't need third party software. Performance reports are readable files with a lot of metrics for a given time window. You should generate reports for multiple time windows.

Step 2 – Check the DB Time of your database

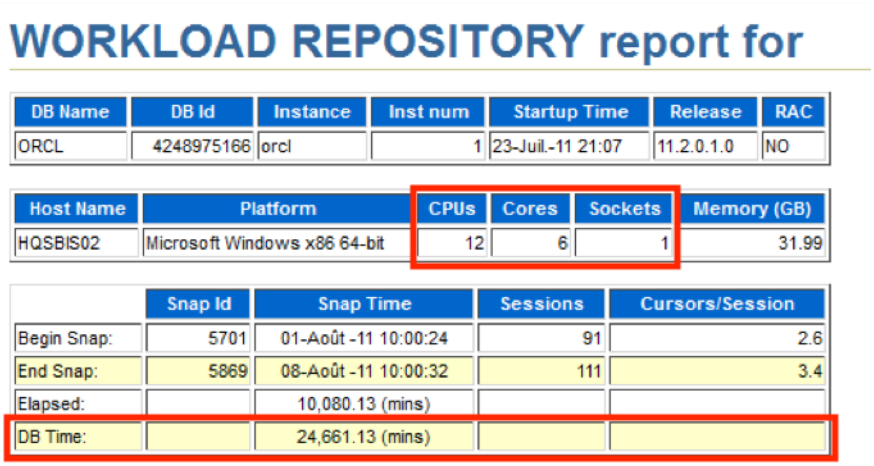
The DB Time represents the amount of time consumed on your system by your database in the elapsed time between your beginning snapshot and ending snapshot. It's a key number defining your database load. A big DB Time means a big load: probably a lot of users, running a lot of statements on a quite big database. Or a really unoptimized database!

Actually, you cannot tell if a DB Time is really big or not: it mainly depends on your system. If your system is a big dedicated server with dozens of CPUs, big DB Time is probably not so big to handle compared to a single core server running the same database with the same activity. DB Time should always be compared to CPU time available during the elapsed time of your performance report. And CPU time is not elapsed time. The formula is the following:

```
Available CPU time = elapsed time X number of CPU cores on your server
```

As an example, if your DB Time is 20 minutes during an elapsed time of 1 hour, it's quite a big DB Time for your single core server. A third of your CPU time available is consumed by your database. But if your server has 2 CPUs with 18 cores each, 36 hours of CPU time are available, and 20 minutes is just about 1% of your server computing capacity...

It's quite easy to check the DB Time of your database in your AWR or Statspack report, you can find it at the beginning of the report.



WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
ORCL	4248975166	orcl	1	23-Juil.-11 21:07	11.2.0.1.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
HQSBIS02	Microsoft Windows x86 64-bit	12	6	1	31.99

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	5701	01-Août -11 10:00:24	91	2.6
End Snap:	5869	08-Août -11 10:00:32	111	3.4
Elapsed:		10,080.13 (mins)		
DB Time:		24,661.13 (mins)		

In this example, the server is a single socket 6 cores CPU (it's in the report but you can also check that directly from the system). You should be aware of the number of CPUs shown here. Actually, this is the number of threads, and a thread is not really a dedicated core but a split of a single core in 2 execution channels (for Intel processors). On this kind of processor, 2 threads can be considered as a boosted core of about 130% the power of one core. But you should consider these 30% as boost power only.

Total CPU time available on this server is here 6 x 10'080 minutes (1 week) = 60'480 minutes. DB Time is quite big, 24'661 minutes represent more than 40% of the system computing power, permanently used by this database. If you consider that the activity is fluctuating during the day, this database probably encounters performance troubles during peak periods.

What happens if your database is trying to consume more power than available on your system? Well, nothing. Everything will continue working, but the database will be slower to return the results to the users or to the application. Even if your system is highly loaded by your database, everything will continue working. The users will probably alert you far before the crash of your system. Oracle Database is quite stable on a highly loaded system.

Don't forget that your system may host more than one database, and maybe other programs are sharing the server resources. In case of a system dedicated to multiple Oracle databases, you'll have to sum the DB Time of all the databases (by running performance report on each database on the same period) to see if overall DB Time is suitable for your system.

In a few words: In the performance report, check the DB Time of your database and compare it to the available CPU time on your system. If it's a few %, your database activity is low for your system and you shouldn't encounter performance troubles. If the DB Time is quite big, go to step 3 on performance profile analysis. If your server is hosting multiple database, sum the DB Time of each database to know the total system load.

Step 3 – Study the performance profile of your database

The DB Time is just the total time spent by the database on your system. But this time can be spent on efficient tasks, and less efficient tasks… In your AWR report or Statspack report, there is something called the Top X timed events (name depends on the version). Actually, this is where your database is spending time and more precisely how the DB Time is divided into.

Let's take a first example:

Top 5 Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		1,651,293		62.8	
db file sequential read	300,139,356	657,763	2	25.0	User I/O
PX Deq Credit: send blkd	2,751,058	243,931	89	9.3	Other
db file scattered read	136,888,796	48,752	0	1.9	User I/O
log file sync	9,174,074	13,958	2	.5	Commit

In the introduction, we saw that CPU and memory are the fastest place in your server. For this database, more than 62% of the DB Time is spent as CPU time (understand CPU and memory time). And it's quite good news! Other events have a Wait Class, indicating that the database is waiting for something. Actually, everything that is not CPU time is waiting for something. Here the main wait class is user I/O, mainly for db file (datafiles) sequential read. Sequential reads are (most of the time) considered as "good reads" as sequential means blocks dispatched on disk, understand indexed reads. So, these are low cost reads on disk. On the opposite, scattered (datafiles) reads are "bad reads" as it mainly consists of contiguous blocks reads, understand full table scan for example (the need for Oracle to read an entire table to -probably- return a limited number of rows).

You know that everything cannot be stored in database memory structure (SGA), especially if your database is big, so occasionally reading datafiles is normal. But in this example, 25% of sequential read is quite a big part of your database activity. How to simply decrease these sequential reads? You probably need to increase your SGA size. If your SGA is bigger, Oracle will probably read less blocks on disk because more blocks are available in memory, in the buffer cache part of your SGA. You should be aware that increasing SGA is only possible if your system has enough RAM. If not, keep in mind that memory is cheap and easily upgradable.

Another performance profile from another database:

Top 5 Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
db file scattered read	57,378,623	182,501	3	28.6	User I/O
db file sequential read	40,890,463	164,466	4	25.8	User I/O
CPU time		91,136		14.3	
RMAN backup & recovery I/O	1,063,783	21,432	20	3.4	System I/O
control file sequential read	2,834,991	18,036	6	2.8	System I/O

If previous database had a not so bad performance profile, this one is really poor. Efficient work on the database is only 14% of the total DB Time. Most of the time is spent waiting, and here waiting for I/O. Oracle needs to read blocks on disk and disks are not able to give them quickly enough. It could be a self-destructive behaviour: waiting for blocks on disk is spending CPU time without computing anything, and it can lower the computing cycles available for pure CPU. And if your redologs are sharing the same storage as your datafiles (if you didn't respect the design of an Oracle database) the performance could be even worse.

This database probably needs two tuning actions. First of all, an increase of the SGA could decrease the I/O needed. And fine tuning of few SQL statements could remove the need of scattered reads ("bad reads"). We now need to dig into SQL statements to identify those consuming most of the resources (step 4).

Last example from another database:

```
Top 5 Timed Events                                                    Avg %Total
~~~~~~~~~~~~~~~~~~                                                   wait   Call
Event                                            Waits    Time (s)   (ms)   Time
---------------------------------------- ------------ ----------- ------ ------
log file switch (checkpoint incomplete)            301         315   1048   70.7
CPU time                                                       116          26.0
db file async I/O submit                           315           5     16    1.2
log file parallel write                          1,771           4      2     .9
control file parallel write                      2,631           2      1     .4
          -------------------------------------------------------------
```

It's another problem here. Database is not able to make redolog switches fast enough, 70% of the DB Time is lost. You should here examine the configuration of your redologs: maybe they are too small, or you don't have enough groups?

There is plenty of other events that can cause waits, if a wait event is significant, study what it refers to.

In a few words: DB Time has to be composed of mainly CPU Time, target is 70% and more. The lower the CPU Time is, the worse the database is working. Db file events are access to the datafiles, scattered means unoptimized reads, sequential means optimized reads. Sequential reads can be lowered by increasing SGA settings. Redologs configuration is also important for performance. There are plenty of other possible wait events.

Step 4 – Study the main SQL statements

There's probably thousands of statements running on your database every day, but you don't have to care about that. Oracle is running most of these statements as fast as it can process them, and you'll never heard about the vast majority of them. Very often the performance troubles are related to very few statements, and if you or the developers can optimize these 2 or 3 statements, it could dramatically decrease the DB Time and improve the overall performance.

Main SQL statements are at the bottom of the AWR or Statspack report. You'll find several tables but focus on SQL statements ordered by elapsed time. Here is an example:

SQL ordered by Elapsed Time

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
286,377	282,928	1	286376.56	10.89	dsv0v7u028vqm		DECLARE job BINARY_INTEGER := ...
284,731	281,336	0		10.83	8ru089sd1frx4		INSERT /*+ BYPASS_RECURSIVE_CH...
253,330	252,482	11	23030.02	9.64	487c0771ad7wz	SQL*Plus	BEGIN p_vad_delai_stk(1000); E...
238,339	237,948	847,982	0.28	9.06	d3fqcs9588q8r	SQL*Plus	SELECT DATLIV, SUM (QTERAL) Q...
230,375	227,205	33,377	6.90	8.76	g2wzmv1ckp6cp		SELECT SUM(EVL.QTECDE-LEAST(EV...
177,448	174,443	4	44361.91	6.75	bzwu8tavqm0rk	SQL*Plus	BEGIN PKG_PFN_OPTIMRAO.PFN_OPT...
175,122	173,965	5	35024.39	6.66	9m0jt4zczyyk7	SQL*Plus	CREATE TABLE PFN_OPTIMRAO_PRO_...
150,800	8,340	6	25133.37	5.74	820aw69dqttq9	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...
106,144	105,859	1,799,504,320	0.00	4.04	bathukprjfwau		SELECT SUM(LEAST(EVP.QTECDE, E...
89,887	89,455	5,497,174	0.02	3.42	q8f9qctabbj2c		SELECT GREATEST(MAX(v.P12), M...
86,820	86,056	48	1808.74	3.30	2qs1bth05bhnz		SELECT decode(count(1), 1, m...
51,361	51,321	1,886,352,880	0.00	1.95	2zw0cnjfcrzk5		SELECT MIN(REC.NUMEVE) FROM EV...
50,565	2,064	187	270.40	1.92	8zwqshkraujah	SQL*Plus	BEGIN P_PFN_VTIC_NONAME; END; ...
48,040	1,974	182	263.96	1.83	4tzd11tdbr17f	SQL*Plus	SELECT EVE.CODSOC, EVE.ACHVTE...
40,487	972	1	40487.25	1.54	d6624jt44yz65	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...
32,801	9	1	32801.34	1.25	1jp2a66xbumkx	TOAD background query session	select numeve, count(1) from ...
28,869	2,271	7	4124.08	1.10	9k2814ab24khs	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...

This table is quite interesting because it shows the most consuming statements indifferently for big statements run once, or small statements run thousands of time. The % Total DB Time indicates the part of DB Time the statement is responsible for. In my opinion, a single statement shouldn't take more than 10% of the total DB Time. If it's the case, the statement should probably be reviewed or rewritten.

On this database given as an example, the DB Time seems to be well balanced between the statements. You should be aware that not all the statements are captured during the snapshots, so trust this table with a grain of salt.

Let's take another database. In this example, we chose to use statements ordered by reads, meaning disk reads.

SQL ordered by Reads

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
307,446,396	445	690,890.78	21.90	4252.36	11185.38	af3q17f9fscpk	SQL*Plus	BEGIN p_pfn_maj_codeta_er(1); ...
286,208,661	425	673,432.14	20.39	3964.13	10319.55	aqxc3tbt7hjbb	SQL*Plus	SELECT MSK.* FROM MSK, UT_SPL...
65,048,020	10	6,504,802.00	4.63	1380.24	4258.35	fm6gpks4k24fm	SQL*Plus	BEGIN p_vad_fin_fact(1000, 'FI...
65,041,544	10	6,504,154.40	4.63	1372.98	4232.72	2w6yq16h5cn6k	SQL*Plus	SELECT EVP.CODSOC, EVP.ACHVTE...
40,011,973	6	6,668,662.17	2.85	8339.57	150800.22	820aw69dqttq9	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...
34,298,285	9	3,810,920.56	2.44	1799.39	4637.62	79j290sm1qxp6	SQL*Plus	BEGIN p_vad_decisi_liv(1000, '...
33,542,189	5	6,708,437.80	2.39	2065.05	4380.30	8ptj1yrphkmth	SQL*Plus	Begin p_pfn_export_upd_datmo...
28,599,963	6	4,766,660.50	2.04	1085.52	2368.88	71ur4w5tdb1s7	wireportserver.exe	SELECT TIE_MAG.SIGTIE, TIE_...
28,261,882	7	4,037,411.71	2.01	625.52	1934.34	cbudavn2n1ff1v	SQL*Plus	BEGIN p_vad_reservation(1000, ...
24,405,594	8	3,050,699.25	1.74	2154.38	18535.51	c94zqvqmzrtqj		SELECT MevMsk.CODSOC, ...
22,259,295	187	119,033.66	1.59	2064.42	50565.50	8zwqshkraujah	SQL*Plus	BEGIN P_PFN_VTIC_NONAME; END; ...
21,299,663	182	117,031.12	1.52	1973.84	48040.24	4tzd11tdbr17f	SQL*Plus	SELECT EVE.CODSOC, EVE.ACHVTE...
19,014,844	10	1,901,484.40	1.35	404.42	1608.05	3wcah505p9xcm	SQL*Plus	BEGIN p_fej_csk_differes(1000)...
19,001,065	10	1,900,106.50	1.35	383.81	1570.02	qtnum77ba11fz	SQL*Plus	SELECT EVL.CODPRO, SUM (QTECD...
18,803,749	710	26,484.15	1.34	1040.21	4582.49	cyjq8qypxtuaf		SELECT DISTINCT NUMEVE FROM EV...
18,679,963	904	20,663.68	1.33	1005.85	4618.35	qpqu7q5zq8p2u		UPDATE EVT JEvt SET DATMOD=:1,...
18,456,776	9	2,050,752.89	1.31	462.70	1502.79	130mwwtxqx2dq	SQL*Plus	BEGIN p_vad_maj_stcde(1000, 'M...
18,166,529	9	2,018,503.22	1.29	559.05	5116.02	06pmwypsjzv0r	SQL*Plus	SELECT EVE.CODSOC, EVE.ACHVTE...
16,316,638	9	1,812,959.78	1.16	512.02	1259.92	5f90z6s8m8htb	SQL*Plus	UPDATE EVT SET CODZN18 = ' ', ...
16,105,086	9	1,789,454.00	1.15	379.51	1157.53	8zf3mz5ryf8y6	SQL*Plus	UPDATE EVT SET CODZN14 = 'N', ...
15,256,508	7	2,179,501.14	1.09	341.61	1058.24	801hhhpdpz426	SQL*Plus	SELECT EVP.CODSOC, EVP.ACHVTE...
14,351,606	6	2,391,934.33	1.02	2316.23	13080.74	6q5n7kkzhxubu	SQL*Plus	begin w_fej_maj_adr_cmp(1); e...

First of all, there is a big statement consuming 20% of the DB Time (second line). And you can see that PL/SQL DB Time includes recursive SQL call's DB Time (first line). Physical reads column is here interesting: it's displaying the number or blocks read from disk. Suppose your database has 8KB block size as it's the standard now: the identified statement here reads more than 2TB of data during the period… It's really, really huge.

In a few words: unique SQL statements should never consume more than 10% of the DB Time. You should analyse first statements and try to find if there is something to improve on them. Improving 2 or 3 statements in the top SQL can decrease your DB Time and increase the overall performance of the database.

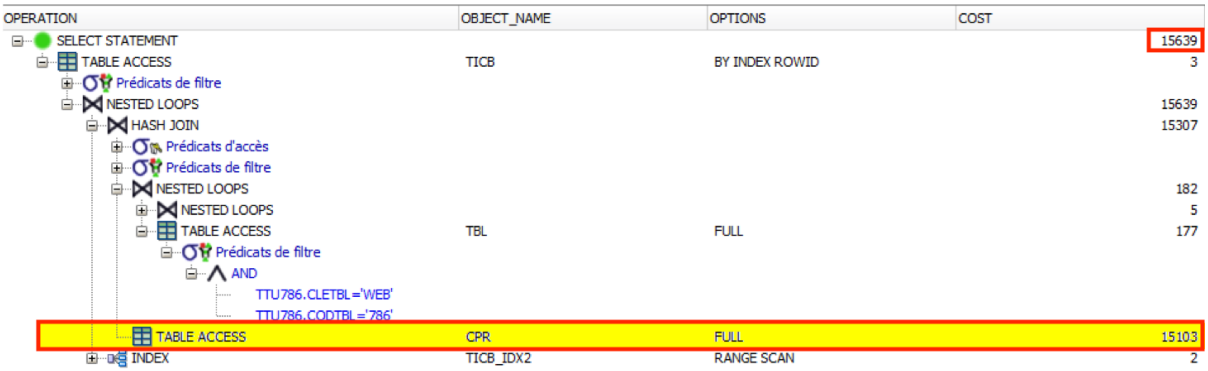
Step 5 – Study indexation of statements

As a DBA, you probably don't have access to SQL code embedded in the application. And you probably don't want to examine hundreds of lines of SQL!

But sometimes improving a statement is just about implementing a new index to avoid scattered reads. Let's take an example in real life. This statement was responsible for 20% of the DB Time of a database:

```
SELECT ticb.sigtie, cpr.datmod, cpr.utimod
      FROM soc1.mev a, soc1.ticb, soc1.mev b,
           soc1.cpr, soc1.tbl ttu786
     WHERE b.codsoc=:p_codsoc and b.codent='CPR' and b.segment=' '
       and ttu786.codtbl='786' and ttu786.cletbl='WEB'
       and cpr.codsoc=b.codsoc_phy and cpr.datmod>=ttu786.lir
       and a.codsoc=:p_codsoc and a.codent='TIE'
       and a.segment='CLI'
       and ticb.codsoc=a.codsoc_phy and ticb.typtie='CLI'
       and ticb.typcpr='PRV' and ticb.codcb=cpr.codcb;
```

At first sight, this statement is not very complex. As a DBA, you can review the execution plan (with SQL developer for example) to see how this statement is executed by Oracle:



The (estimated) cost of executing this statement is 15639 blocks to read. This cost is mainly produced by a single operation responsible for nearly all the necessary reads: a full table access. Actually, all the lines of this table need to be read here. But if you look back at the SQL statement, there are filters on columns codsoc and datmod on this table. There's probably no reason to read the entire table!

Maybe there is no index on these columns? You can check that with:

```
select index_name, count(*) from dba_ind_columns
where table_name = 'CPR'
  and column_name in ('CODSOC', 'DATMOD')
  group by index_name having count(*)>=2;
```

If there is no index, it's easy to understand why a full table scan is needed.

A simple optimization could consist of creating an index on these two columns:

```
create index soc1.CPR_IDX2 on SOC1.CPR (codsoc,datmod) tablespace IDX;
```

And then look at the new execution plan:

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			4104
TABLE ACCESS	TICB	BY INDEX ROWID	3
Prédicats de filtre			
NESTED LOOPS			4104
NESTED LOOPS			3773
NESTED LOOPS			182
TABLE ACCESS	CPR	BY INDEX ROWID	3591
INDEX	CPR_IDX2	RANGE SCAN	589
Prédicats d'accès			
AND			
CPR.DATMOD>=TTU786.LIR			
CPR.CODSOC=B.CODSOC_PHY			
CPR.DATMOD IS NOT NULL			
Prédicats de filtre			
INDEX	TICB_IDX2	RANGE SCAN	2

Blocks needed to execute this statement are now 4 times less than before: from 20% of the DB Time this statement now only spends 5% of the DB Time. In other words, with this single index you just decreased the DB Time of your database by 15%...

Beware of the increased cost when inserting and updating data with a new index. And keep in mind that this new index can cause other statements to run slower.

In a few words: Indexing if part of the database administrator's job. Good indexes can reduce the need for full table scans and waits for I/O. And can increase the overall performance of the database. Don't forget to check again the performance of your database after creating new indexes.

Conclusion

For me, this approach is working for 90% of the performance troubles I have to fix. But sometimes tuning can be much more complex stuff! To increase your knowledge and understanding, don't hesitate to generate multiple performance reports on multiple databases, even on databases without performance issues.