power tuning
performance em dados

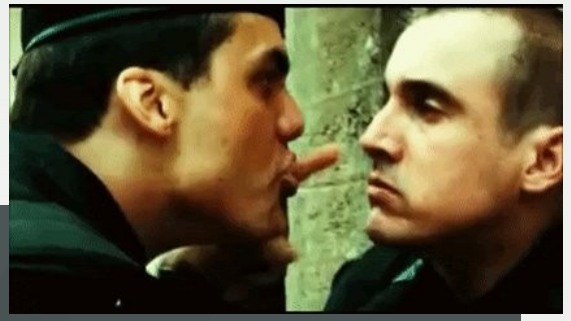
Oracle
Lendo Planos de Execução

Ricardo Portilho Proni
ricardo.portilho@powertuning.com.br

Quando ir para o Plano de Execução?

- Sistema Operacional;
  - Instância / Banco;
    - Sessão;
      - SQL.

Claro, RTFM...



Database / Oracle / Oracle Database / Release 19

SQL Tuning Guide

Expand

6 Explaining and Displaying Execution Plans

Knowledge of how to explain a statement and display its plan is essential to SQL tuning.

6.1 Introduction to Execution Plans

An **execution plan** is the sequence of operations that the database performs to run a SQL statement.

6.1.1 Contents of an Execution Plan

The execution plan operation alone cannot differentiate between well-tuned statements and those that perform suboptimally.

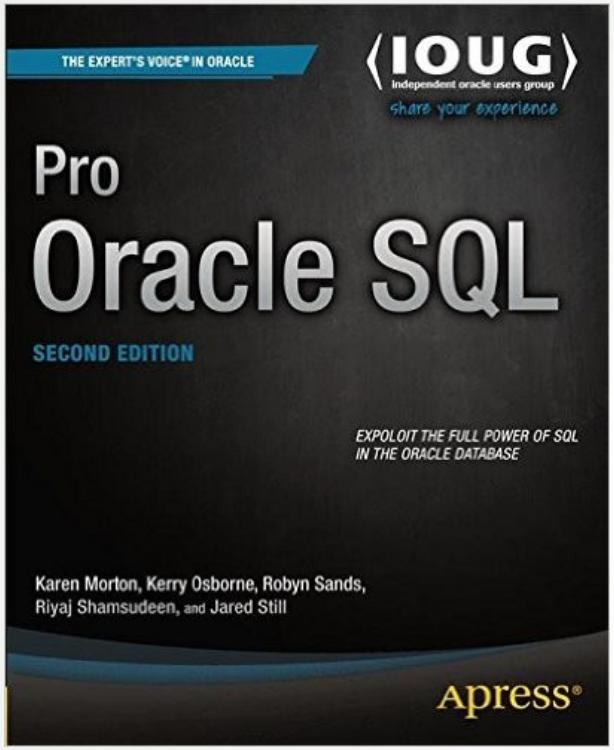
The plan consists of a series of steps. Every step either retrieves rows of data physically from the database or prepares them for the user issuing the statement. The following plan shows a join of the `employees` and `departments` tables:

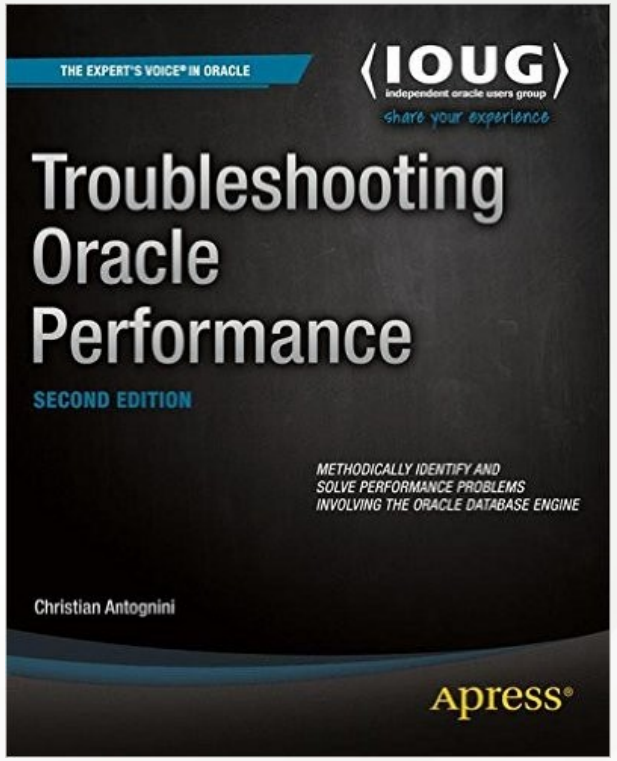
```
SQL_ID  g9xaqjktdhbcd, child number 0
-------------------------------------
```

Copy

https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/generating-and-displaying-execution-plans.html#GUID-60E30B1C-342B-4D71-B154-C26623D6A3B1

Livros





TikTok

Dancinhas

?

Redgate / Jonathan Lewis

Jonathan Lewis
26 MARCH 2015

Execution Plans Part 14: SQL Monitoring

This is the last part of my series on Execution plans, and features an option which is only available if you have licensed the Diagnostic and Performance Packs. It's a feature that allows you to watch the flow of data through an execution plan as the query is running, typically through the graphic interface supplied by Enterprise Manager (or Grid... Read more

Jonathan Lewis
11 MARCH 2015

Execution Plans Part 13: SQL Trace

In parts 11 and 12 of this series I described the "rowsource execution statistics" that we can collect as Oracle runs a query, then described a strategy for generating and accessing these statistics in a way that was particularly convenient if you could use your own session to run the SQL you wanted to analyze. In this article we're going... Read more

Jonathan Lewis
13 JANUARY 2015

Execution Plans Part 12: Cardinality Feedback

In the previous instalment of this series I introduced three ways of accessing the run-time statistics for a query and described, for one of the methods, the basics of the information we can get and how we can use it. In this article I want to expand on the use of one method to show it can help use identify... Read more

Jonathan Lewis
17 DECEMBER 2014

Execution Plans Part 11: Actuals

So far in this series we've talked about interpreting the shape of an execution plan and understanding the meaning of the predictions that the optimizer has made about cost and cardinality. It's finally time to see how Oracle gives us execution plans that show us how well the optimizer's estimates match the actual work done as the query ran. Ther... Read more

https://www.red-gate.com/simple-talk/author/jonathan-lewis-3/page/3/

?

Fontes de Planos de Execução

- PLAN_TABLE, carregada por EXPLAIN PLAN / DBMS_XPLAN.DISPLAY ou AUTOTRACE (e SQL Developer, Toad, etc.). Não use se tiver Binds!
- *Arquivos Trace (10046, 10053, etc.) / tkprof.*
- VIEWs de planos compilados e armazenados na Library Cache;
  - V$SQL_PLAN
  - V$SQL_PLAN_STATISTICS
  - V$SQL_WORKAREA
  - V$SQL_PLAN_STATISTICS_ALL (V$SQL_PLAN_STATISTICS + V$SQL_WORKAREA)
- Tabelas de AWR / STATSPACK;

```
@$ORACLE_HOME/rdbms/admin/awrsqrpt.sql
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_AWR('a10jnjwd22gs8'));
 - DBA_HIST_SQL_PLAN (AWR)
 - DBA_HIST_SQLTEXT (AWR)
 - DBA_HIST_SQLSTAT (AWR)
 - DBA_HIST_SQLBIND (AWR)
@$ORACLE_HOME/rdbms/admin/sprepsql.sql
 - STATS$SQL_PLAN (STATSPACK)
```

Recuperação de "Ambiente de Execução":

```
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('gkxxkghxubh1a', NULL,'OUTLINE
PEEKED_BINDS'));
SQL> SELECT NAME, VALUE_STRING, VALUE_ANYDATA, CHILD_NUMBER, LAST_CAPTURED FROM
V$SQL_BIND_CAPTURE WHERE SQL_ID = 'gkxxkghxubh1a' ORDER BY LAST_CAPTURED;
SQL> SELECT NAME, VALUE FROM V$SQL_OPTIMIZER_ENV WHERE SQL_ID = 'gkxxkghxubh1a' AND CHILD_NUMBER
= 0 AND ISDEFAULT = 'NO';
```

AWR requer Enterpise Edition + Option Diagnostics Pack

Views / DISPLAY_CURSOR

Outra Sessão:

```
SQL> ALTER SYSTEM SET STATISTICS_LEVEL=ALL;
SQL> SELECT SQL_ID, FIRST_LOAD_TIME, CHILD_NUMBER, SQL_TEXT FROM V$SQL WHERE SQL_TEXT LIKE
'%FROM T WHERE OBJECT_TYPE%' ORDER BY FIRST_LOAD_TIME;
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('5srht3gdqzz3c', 0, 'ALLSTATS LAST'));
```

Minha Sessão:

```
SQL> ALTER SESSION SET STATISTICS_LEVEL=ALL;
SQL> SELECT COUNT(OBJECT_NAME) FROM T WHERE OBJECT_TYPE = 'T';
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST'));
```

OU

```
SQL> SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(OBJECT_NAME) FROM T WHERE
OBJECT_TYPE = 'T';
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  | Writes |  OMem |  1Mem | Used-Mem | Used-Tmp|
-----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |     40 |00:06:50.58 |     384K|    429K|  45508 |       |       |          |         |
|   1 |  SORT ORDER BY      |      |      1 |     40 |     40 |00:06:50.58 |     384K|    429K|  45508 |  2048 |  2048 | 2048   (0)|         |
|   2 |   HASH GROUP BY     |      |      1 |     40 |     40 |00:06:50.58 |     384K|    429K|  45508 |  1200K|  1200K| 1379K  (0)|         |
|*  3 |    HASH JOIN        |      |      1 |   1443M|   1463M|00:04:48.25 |     384K|    429K|  45508 |   465M|    23M|   93M  (1)|    368K|
|   4 |     TABLE ACCESS FULL| T2   |      1 |     11M|     11M|00:00:13.86 |     192K|    192K|      0 |       |       |          |         |
|   5 |     TABLE ACCESS FULL| T1   |      1 |     11M|     11M|00:00:11.06 |     192K|    192K|      0 |       |       |          |         |
-----------------------------------------------------------------------------------------------------------------------------------------------------
```

FORMAT

- TYPICAL = DEFAULT
- ALL = TYPICAL + QB + PROJECTION + ALIAS + REMOTE
- ADVANCED = ALL + OUTLINE + BINDS
- ALLSTATS = IOSTATS + MEMSTATS (all executions)
- ALLSTATS LAST (last execution)
- ADAPTIVE (12c)

Ordem

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	139	348 (4)	00:00:01
1	SORT AGGREGATE		1	139		
* 2	FILTER					
* 3	HASH JOIN		73959	9M	348 (4)	00:00:01
4	INDEX FULL SCAN	I_USER2	130	520	1 (0)	00:00:01
* 5	HASH JOIN		73959	9750K	347 (4)	00:00:01
6	INDEX FULL SCAN	I_USER2	130	3250	1 (0)	00:00:01
* 7	TABLE ACCESS FULL	OBJ$	73959	7944K	345 (3)	00:00:01
8	NESTED LOOPS		1	32	4 (0)	00:00:01
9	NESTED LOOPS		1	23	3 (0)	00:00:01
10	TABLE ACCESS BY INDEX ROWID	IND$	1	10	2 (0)	00:00:01
* 11	INDEX UNIQUE SCAN	I_IND1	1		1 (0)	00:00:01
* 12	TABLE ACCESS CLUSTER	TAB$	1	13	1 (0)	00:00:01
* 13	INDEX RANGE SCAN	I_OBJ1	1	9	1 (0)	00:00:01
* 14	TABLE ACCESS CLUSTER	TAB$	1	13	2 (0)	00:00:01
* 15	INDEX UNIQUE SCAN	I_OBJ#	1		1 (0)	00:00:01
* 16	TABLE ACCESS BY INDEX ROWID	SEQ$	1	8	1 (0)	00:00:01
* 17	INDEX UNIQUE SCAN	I_SEQ1	1		0 (0)	00:00:01
* 18	TABLE ACCESS BY INDEX ROWID	IND$	1	8	2 (0)	00:00:01
* 19	INDEX UNIQUE SCAN	I_IND1	1		1 (0)	00:00:01
20	NESTED LOOPS		1	15	2 (0)	00:00:01
21	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
* 22	INDEX RANGE SCAN	I_OBJAUTH1	1	12	1 (0)	00:00:01
* 23	HASH JOIN		1	22	3 (0)	00:00:01
24	NESTED LOOPS		1	19	3 (0)	00:00:01
25	TABLE ACCESS BY INDEX ROWID	IND$	1	10	2 (0)	00:00:01
* 26	INDEX UNIQUE SCAN	I_IND1	1		1 (0)	00:00:01
* 27	INDEX RANGE SCAN	I_OBJAUTH1	1	9	1 (0)	00:00:01
28	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
* 29	HASH JOIN		1	25	3 (0)	00:00:01
30	NESTED LOOPS		1	22	3 (0)	00:00:01
31	TABLE ACCESS BY INDEX ROWID	TABPART$	1	10	2 (0)	00:00:01
* 32	INDEX UNIQUE SCAN	I_TABPART_OBJ$	1		1 (0)	00:00:01
* 33	INDEX RANGE SCAN	I_OBJAUTH1	1	12	1 (0)	00:00:01
34	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
* 35	HASH JOIN		1	25	2 (0)	00:00:01
36	NESTED LOOPS		1	22	2 (0)	00:00:01
37	TABLE ACCESS BY INDEX ROWID	TABCOMPART$	1	10	1 (0)	00:00:01
* 38	INDEX UNIQUE SCAN	I_TABCOMPART$	1		0 (0)	00:00:01
* 39	INDEX RANGE SCAN	I_OBJAUTH1	1	12	1 (0)	00:00:01
40	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
41	NESTED LOOPS		1	15	2 (0)	00:00:01
42	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
* 43	INDEX RANGE SCAN	I_OBJAUTH1	1	12	1 (0)	00:00:01
44	NESTED LOOPS		1	15	2 (0)	00:00:01
45	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
* 46	INDEX RANGE SCAN	I_OBJAUTH1	1	12	1 (0)	00:00:01
47	NESTED LOOPS		1	79	8 (0)	00:00:01
48	NEST					
49	NE					
50	NE					
51						
* 52						

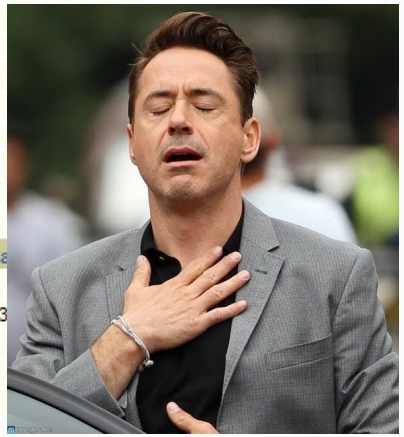
95	SORT GROUP BY		1	35	10 (0)	00:00:01
96	NESTED LOOPS		1	35	8 (0)	00:00:01
97	NESTED LOOPS		5	115	3 (0)	00:00:01
* 98	INDEX UNIQUE SCAN	I_OLAP_CUBES$	1	13	0 (0)	00:00:01
99	TABLE ACCESS BY INDEX ROWID BATCHED	DEPENDENCY$	5	50	3 (0)	00:00:01
*100	INDEX RANGE SCAN	I_DEPENDENCY1	5		2 (0)	00:00:01
*101	INDEX RANGE SCAN	I_OBJ1	1	12	1 (0)	00:00:01
102	NESTED LOOPS		1	12	2 (0)	00:00:01
103	FIXED TABLE FULL	X$KZSRO	2	6	0 (0)	00:00:01
*104	INDEX RANGE SCAN	I_OBJAUTH2	1	9	1 (0)	00:00:01
*105	TABLE ACCESS BY INDEX ROWID BATCHED	USER_EDITIONING$	1	6	2 (0)	00:00:01
*106	INDEX RANGE SCAN	I_USER_EDITIONING	2		1 (0)	00:00:01
*107	TABLE ACCESS BY INDEX ROWID BATCHED	USER_EDITIONING$	1	6	2 (0)	00:00:01
*108	INDEX RANGE SCAN	I_USER_EDITIONING	2		1 (0)	00:00:01
109	NESTED LOOPS		1	29	2 (0)	00:00:01
*110	INDEX SKIP SCAN	I_USER2	1	20	1 (0)	00:00:01
*111	INDEX RANGE SCAN	I_OBJ4	1	9	1 (0)	00:00:01

Ordem – SQLT (Doc ID 215187.1)

Execution Plan phv:3867221913 [B] [W] sqlt_phv:78420 sqlt_phv2:16894 source:GV$SQL_PLAN inst:1 child:0(000000008FB227B8)

SQL Text: [+]
SQL: [+]

ID	Exec Ord	Operation	Go To	More	Peek Bind	Capt Bind	Cost[2]	Estim Card	Work Area
0	195	SELECT STATEMENT					240	6	
1	194	SORT UNIQUE		[+]			239	6	[+]
2	193	. UNION-ALL		[+]			208		
3	29	.. FILTER		[+]	[+]	[+]	33		
4	28	... NESTED LOOPS SEMI		[+]			33	1	
5	25	.... NESTED LOOPS SEMI		[+]			31	1	
6	22	....+ ~~HASH JOIN~~		[+]			29	1	[+]
7	20	....+. NESTED LOOPS		[+]			29	1	
8	17	....+.. ~~STATISTICS COLLECTOR~~		[+]			28		
9	16	....+... ~~HASH JOIN~~		[+]			28	1	[+]
10	14	....+.... NESTED LOOPS		[+]			28	1	
11	11	....+....+ ~~STATISTICS COLLECTOR~~		[+]			27		
12	10	....+....+. NESTED LOOPS		[+]			27	1	
13	2	....+....+.. TABLE ACCESS BY INDEX ROWID BATCHED HST_MSG_SPB	[+]	[+]	[+]	[+]	13	7	
14	1	....+....+... INDEX SKIP SCAN IX9_HST_MSG_SPB	[+]	[+]	[+]	[+]	12	17	
15	9	....+....+.. TABLE ACCESS BY INDEX ROWID BATCHED HST_EVENTO_SPB	[+]	[+]	[+]	[+]	2	1	
16	8	....+....+... INDEX RANGE SCAN IX9_HST_EVENTO_SPB	[+]	[+]	[+]	[+]	1	1	
17	7	....+....+.... SORT AGGREGATE		[+]			6	1	
18	6	....+....+....+ NESTED LOOPS SEMI		[+]			6	1	
19	4	....+....+....+. TABLE ACCESS BY INDEX ROWID BATCHED HST_MSG_SPB	[+]	[+]	[+]	[+]	4	1	
20	3	....+....+....+.. INDEX RANGE SCAN IX1_HST_MSG_SPB	[+]	[+]			3	2	
21	5	....+....+....+. INDEX RANGE SCAN IX10_HST_EVENTO_SPB	[+]	[+]			2	1	
22	13	....+....+ TABLE ACCESS BY INDEX ROWID SISTEMA_ORIGEM	[+]	[+]			1		
23	12	....+....+. INDEX UNIQUE SCAN PK_SISTEMA_ORIGEM	[+]	[+]			0		
24	15	....+.... ~~TABLE ACCESS FULL~~ ~~SISTEMA_ORIGEM~~	[+]	[+]			1		
25	19	....+.. TABLE ACCESS BY INDEX ROWID AREA	[+]	[+]			1		
26	18	....+... INDEX UNIQUE SCAN PK_AREA	[+]	[+]			0		
27	21	....+. ~~TABLE ACCESS FULL~~ ~~AREA~~	[+]	[+]			1		
28	24	....+ INLIST ITERATOR					2		
29	23	....+. INDEX RANGE SCAN IX3_SEGR_DADO	[+]	[+]	[+]	[+]	2		
ID	Exec Ord	Operation	Go To	More	Peek Bind	Capt Bind	Cost[2]	Estim Ca	
30	27	.... INLIST ITERATOR					2		
31	26	....+ INDEX RANGE SCAN IX3_SEGR_DADO	[+]	[+]	[+]	[+]	2	13	
32	64	.. FILTER		[+]	[+]	[+]	40		
33	63	... NESTED LOOPS SEMI		[+]			40		
34	60	.... ~~HASH JOIN~~		[+]			38		



Access Paths

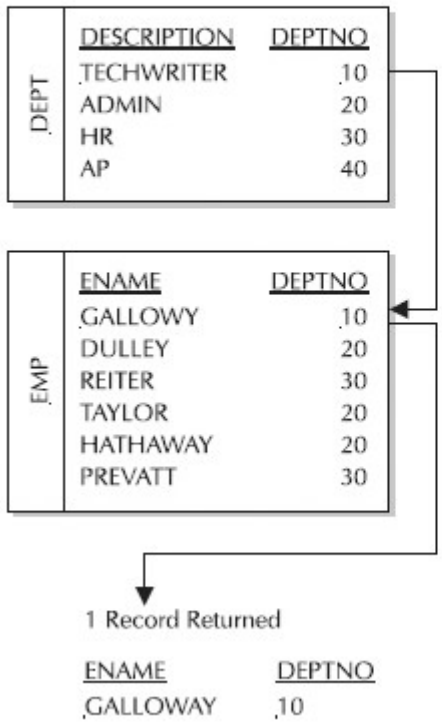
- Table Access by ROWID
- Index Unique Scan
- Index Range Scan
- Index Range Scan descending
- Index Skip Scan
- Full Index Scan (FIS)
- Fast Full Index Scan (FFIS) *
- Full Table Scan (FTS) *

** Leitura não ordenada, utiliza Multi Block Read / db file scattered read.*

Join Methods – Nested Loops

- É mais eficiente com pequenos Result Sets;
- Geralmente ocorre quando há índices nas colunas utilizadas pelo Join;
- Utiliza pouca memória, pois o Result Set é construído uma linha por vez.

```
select  /*+ ordered */ ename, dept.deptno
from    dept, emp
where   dept.deptno = emp.deptno;
```



NESTED LOOPS Join

The loop would continue until each of the DEPTNOs in the DEPT table have been checked against those in the EMP table. The subseqent loop results are (for DEPTNOs 20, 30, 40):

The second loop:
3 Records Returned

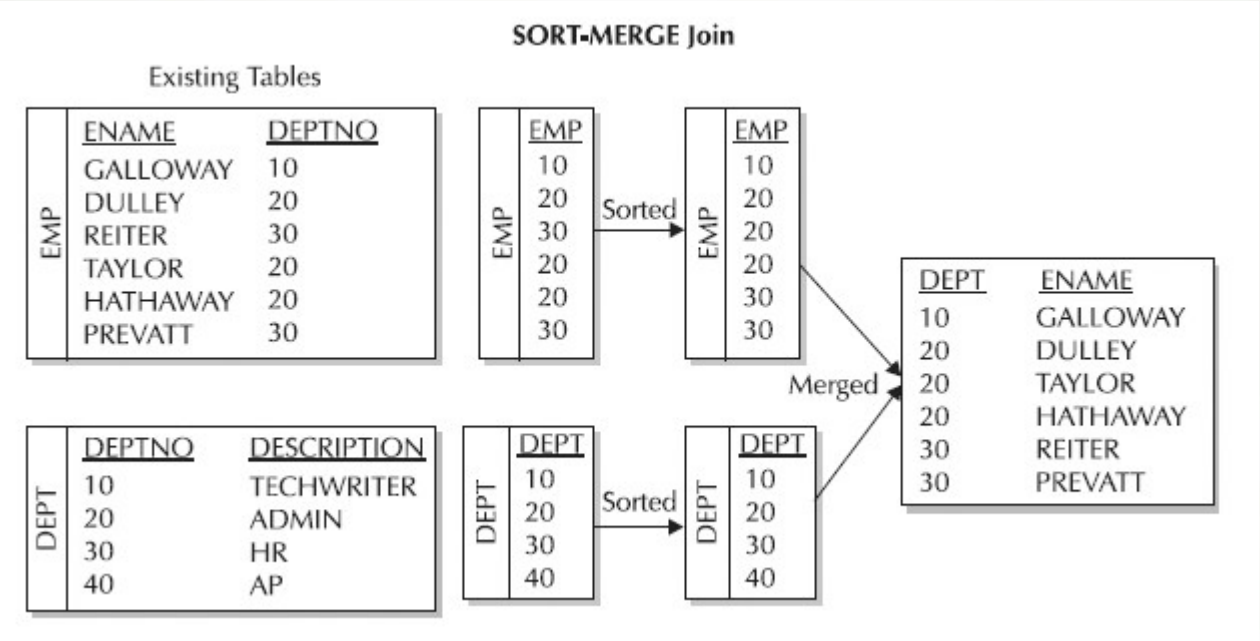
ENAME	DEPTNO
DULLEY	20
TAYLOR	20
HATHAWAY	20

The third loop:
2 Records Returned

ENAME	DEPTNO
REITER	30
PREVATT	30

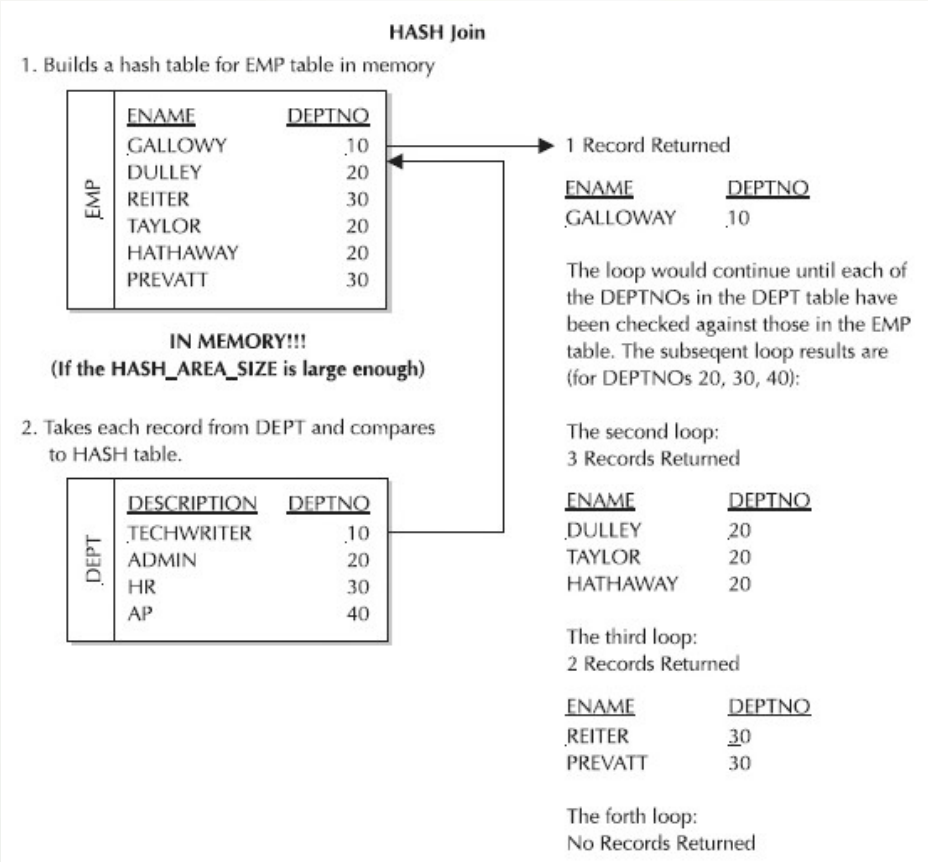
Join Methods – Sort Merge Join

- Geralmente é utilizado para Result Sets maiores, e quando não há índices;
- Geralmente é utilizado quando é uma operação de desigualdade;
- O maior custo é a ordenação, portanto quando não há Índice em algum “Lado”;
- Poderá ser utilizada apenas PGA, ou pode ser necessário TEMP.



Join Methods – Hash Join

- Só ocorre em equi-joins;
- Geralmente é utilizado para grandes Result Sets;
- Geralmente é utilizado se o menor Result Set cabe em memória;
- Poderá ser utilizada apenas PGA, ou pode ser necessário TEMP.



O que procurar?

- Ponto de aumento de Cost / Rows / Bytes;
- Diferença entre A-Rows e E-Rows (com DISPLAY_CURSOR + STATISTICS_LEVEL);
- Nested Loops com grande quantidade de Starts;
- Sort Merge / Hash Join com pequena quanitade de Rows;
- FTS / FIS / FFIS em “*Filter”;
- FIS / FFIS / Index Skip Scan em “*Access”;
- Desperdício:

```
-------------------------------------------------------------------------------
| Id  | Operation                     | Name  | Starts | E-Rows | A-Rows |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |       |      1 |        |      0 |
|   1 |  MERGE JOIN                   |       |      1 |     10 |      0 |
|*  2 |   TABLE ACCESS BY INDEX ROWID | T1    |      1 |     10 |     10 |
|   3 |    INDEX FULL SCAN            | T1_PK |      1 |  10000 |  10000 |
|*  4 |   SORT JOIN                   |       |     10 |     10 |      0 |
|*  5 |    TABLE ACCESS BY INDEX ROWID| T2    |      1 |     10 |     10 |
|   6 |     INDEX FULL SCAN           | T2_PK |      1 |  10000 |  10000 |
-------------------------------------------------------------------------------
```

O que procurar?

Tip – What A-Rows and E-Rows Mean

```
select e.ename as "Employee Name",
       t.tname as "Task Name"
from   employees        e
       join tasks t on (t.emp_id = e.id)
where  e.etype <= 5;
```

- E-Rows is *per start*
- E-Rows is an *estimate*
- A-Rows is the *actual* number of rows produced *in total*

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Starts | E-Rows |E-Bytes| Cost (%CPU)| E-Time   | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |      1 |        |       |   710 (100)|          |    200 |00:00:00.04 |    1812 |
|   1 |  NESTED LOOPS                |             |      1 |    200 |  9400 |   710   (1)| 00:00:01 |    200 |00:00:00.04 |    1812 |
|*  2 |   TABLE ACCESS FULL          | EMPLOYEES   |      1 |    100 |  2700 |   410   (1)| 00:00:01 |    100 |00:00:00.01 |    1465 |
|   3 |   TABLE ACCESS BY INDEX ROWID| TASKS       |    100 |      2 |    40 |     3   (0)| 00:00:01 |    200 |00:00:00.01 |     347 |
|*  4 |    INDEX RANGE SCAN          | TASK_EMP_FK |    100 |      2 |       |     1   (0)| 00:00:01 |    200 |00:00:00.01 |     147 |
-------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("E"."ETYPE"<=5)
   4 - access("T"."EMP_ID"="E"."ID")
```

A-Rows = Starts * E-Rows

Caveat: Partitioning changes this

ORACLE

Índices – Guidelines

- Utilize tipos de dados corretos, nos objetos e nos SQLs.
- **Crie índices em colunas utilizadas na cláusula WHERE;**
- **Crie índices em colunas utilizadas em JOINs;**
- **Crie índices em colunas de alta seletividade;**
- **Crie índices em colunas de baixa seletividade mas que contenham dados com seletividades muito distintas;**
- **Crie índices compostos em colunas utilizadas frequentemente na mesma cláusula WHERE;**
- **Em índices compostos, utilize as colunas com maior seletividade à esquerda;**
- Crie índices em colunas utilizadas em SORTs de alto custo.
- Se um valor de uma coluna indexada não for utilizado em uma cláusula WHERE, verifique se este valor pode ser trocado para NULL;
- Se um valor de uma coluna indexada utilizado em uma cláusula WHERE for raro, considere um Function Based Index:

```
CREATE INDEX IDX_ORDER_NEW ON ORDERS(CASE STATUS WHEN 'N' THEN 'N' ELSE NULL END);
```

- Prefira índices PRIMARY KEY, se o modelo permitir;
- Prefira índices UNIQUE, se o modelo permitir, mas PRIMARY KEY não é possível;
- Adicione NOT NULL, se possível;
- Prefira índices BTREE em colunas de alta seletividade (CPF, NF);
- Prefira índices BITMAP em colunas de baixa seletividade (ESTADO, CIDADE);
- Evite índices em colunas utilizadas em cláusula WHERE apenas com funções;
- **Utilize Function Based Index em colunas utilizadas em cláusula WHERE mais frequentemente com funções;**
- Prefira índices BITMAP para grandes tabelas;
- Evite índices BITMAP em colunas que sofrem muito DML, principalmente de forma concorrente;
- Utilize IOTs em PKs frequentemente utilizadas na cláusula WHERE;
- Utilize índices REVERSE em colunas que sofrem DML em alta concorrência;
- Utilize índices Partitioned Partial em Partições se apenas algumas partições sofrerão SELECT desta coluna indexada.
- Utilize índices e funções TEXT ao invés de LIKE.
- Busque sempre minimizar a quantidade de índices de uma tabela;
- Considere o espaço utilizado por índices (60% - 40%).
- Crie índices em Foreign Keys (FKs) que sofrem DML de forma concorrente (Enq TM);
- Evite índices em colunas que sofrem muitos UPDATEs;
- Evite índices em tabelas que sofrem muitos INSERTs ou DELETEs.
- Em alterações em lote, remova os índices (e CONSTRAINTs), se possível.

Índices – Guidelines

Índice = WHERE

Índice = WHERE

Índice = WHERE

Índice = WHERE

Índices – Guidelines

Verificando Seletividade:

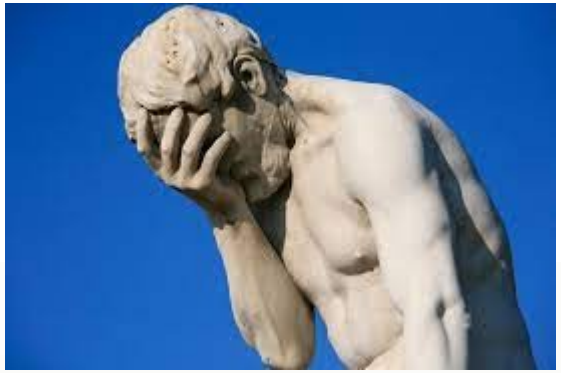
```
SQL> SELECT COUNT(*) FROM T;
SQL> SELECT COUNT(DISTINCT(OBJECT_ID)) FROM T;
SQL> SELECT COUNT(DISTINCT(OWNER)) FROM T;
SQL> SELECT COUNT(DISTINCT(OBJECT_TYPE)) FROM T;
SQL> SELECT COUNT(OWNER), OWNER FROM T GROUP BY OWNER ORDER BY 1;
SQL> SELECT COUNT(OBJECT_TYPE), OBJECT_TYPE FROM T GROUP BY OBJECT_TYPE ORDER BY 1;
```

Verificando Índices Existentes:

```
SQL> SELECT
     B.TABLE_NAME,
     A.INDEX_NAME,
     A.COLUMN_POSITION,
     B.COLUMN_NAME,
     C.BLEVEL,
     C.CLUSTERING_FACTOR,
     C.NUM_ROWS,
     C.LEAF_BLOCKS
FROM
     DBA_IND_COLUMNS A, DBA_TAB_COLUMNS B, DBA_INDEXES C
WHERE
     A.TABLE_OWNER = B.OWNER AND
     A.TABLE_OWNER = C.OWNER AND
     A.TABLE_NAME = B.TABLE_NAME AND
     A.INDEX_NAME = C.INDEX_NAME AND
     A.COLUMN_NAME = B.COLUMN_NAME AND
     A.TABLE_OWNER = 'SCOTT' AND
     A.TABLE_NAME IN ('T')
ORDER BY
     B.TABLE_NAME, A.INDEX_NAME, A.COLUMN_POSITION,B.COLUMN_NAME;
```

Índices – “AntiPatterns”

```
SQL> SELECT A.COLUMN_POSITION, B.COLUMN_NAME, C.BLEVEL, C.CLUSTERING_FACTOR, C.NUM_ROWS, C.LEAF_BLOCKS FROM DBA_IND_COLUMNS A, DBA_TAB_COL
R = C.OWNER AND A.TABLE_NAME = B.TABLE_NAME AND A.INDEX_NAME = C.INDEX_NAME AND A.COLUMN_NAME = B.COLUMN_NAME AND A.TABLE_OWNER = 'TOTVS'
 B.TABLE_NAME, A.INDEX_NAME, A.COLUMN_POSITION, B.COLUMN_NAME;

COLUMN_POSITION COLUMN_NAME                    BLEVEL CLUSTERING_FACTOR   NUM_ROWS LEAF_BLOCKS
--------------- ------------------------------ ------ ----------------- ---------- -----------
              1 CT2_FILIAL                          4         210946116  353127769     6630167
              2 CT2_ORIGEM                          4         210946116  353127769     6630167
              3 R_E_C_N_O_                          4         210946116  353127769     6630167
              4 D_E_L_E_T_                          4         210946116  353127769     6630167

SQL> SELECT COUNT(*) FROM TOTVS.CT2010;

  COUNT(*)
----------
 353138648

SQL> SELECT COUNT(DISTINCT(CT2_FILIAL)) FROM TOTVS.CT2010;

COUNT(DISTINCT(CT2_FILIAL))
---------------------------
                          1

SQL> SELECT COUNT(DISTINCT(CT2_ORIGEM)) FROM TOTVS.CT2010;

COUNT(DISTINCT(CT2_ORIGEM))
---------------------------
                        371

SQL> SELECT COUNT(DISTINCT(R_E_C_N_O_)) FROM TOTVS.CT2010;

COUNT(DISTINCT(R_E_C_N_O_))
---------------------------
                  353138648

SQL> SELECT COUNT(DISTINCT(D_E_L_E_T_)) FROM TOTVS.CT2010;

COUNT(DISTINCT(D_E_L_E_T_))
---------------------------
                          2

SQL>
```

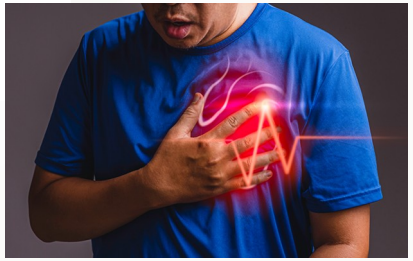


Índices – “AntiPatterns”

```
SQL> SELECT A.COLUMN_POSITION, B.COLUMN_NAME, C.BLEVEL, C.CLUSTERING_FACTOR, C.NUM_ROWS, C.LEAF_BLOCKS FROM DBA_IND_COLUMNS A, DBA_TAB_COLUMNS B,
R = C.OWNER AND A.TABLE_NAME = B.TABLE_NAME AND A.INDEX_NAME = C.INDEX_NAME AND A.COLUMN_NAME = B.COLUMN_NAME AND A.TABLE_OWNER = 'TOTVS' AND A.T
 B.TABLE_NAME, A.INDEX_NAME, A.COLUMN_POSITION, B.COLUMN_NAME;

COLUMN_POSITION COLUMN_NAME                      BLEVEL CLUSTERING_FACTOR   NUM_ROWS LEAF_BLOCKS
--------------- ------------------------------ ---------- ----------------- ---------- -----------
              1 CV3_FILIAL                              3         445888352  561758779     5224775
              2 CV3_TABORI                              3         445888352  561758779     5224775
              3 CV3_RECORI                              3         445888352  561758779     5224775
              4 CV3_RECDES                              3         445888352  561758779     5224775
              5 R_E_C_N_O_                              3         445888352  561758779     5224775
              6 D_E_L_E_T_                              3         445888352  561758779     5224775

6 linhas selecionadas.

SQL> SELECT COUNT(DISTINCT(CV3_FILIAL)) FROM TOTVS.CV3010;

COUNT(DISTINCT(CV3_FILIAL))
---------------------------
                          1

SQL> SELECT COUNT(DISTINCT(CV3_TABORI)) FROM TOTVS.CV3010;

COUNT(DISTINCT(CV3_TABORI))
---------------------------
                         15

SQL> SELECT COUNT(DISTINCT(CV3_RECORI)) FROM TOTVS.CV3010;

COUNT(DISTINCT(CV3_RECORI))
---------------------------
                   98542550

SQL> SELECT COUNT(DISTINCT(CV3_RECDES)) FROM TOTVS.CV3010;

COUNT(DISTINCT(CV3_RECDES))
---------------------------
                  247594697

SQL> SELECT COUNT(DISTINCT(R_E_C_N_O_)) FROM TOTVS.CV3010;

COUNT(DISTINCT(R_E_C_N_O_))
---------------------------
                  561769933

SQL> SELECT COUNT(DISTINCT(D_E_L_E_T_)) FROM TOTVS.CV3010;

COUNT(DISTINCT(D_E_L_E_T_))
---------------------------
                          1

SQL>
```

Índices – “AntiPatterns”

```
SQL> SELECT A.COLUMN_POSITION, B.COLUMN_NAME, C.BLEVEL, C.CLUSTERING_FACTOR, C.NUM_ROWS, C.LEAF_BLOCKS FROM DBA_
ME = B.TABLE_NAME AND A.INDEX_NAME = C.INDEX_NAME AND A.COLUMN_NAME = B.COLUMN_NAME AND A.TABLE_OWNER = 'TOTVS'
LUMN_NAME;

COLUMN_POSITION COLUMN_NAME                      BLEVEL CLUSTERING_FACTOR   NUM_ROWS LEAF_BLOCKS
--------------- ------------------------------ ---------- ----------------- ---------- -----------
              1 CD2_FILIAL                              4          63731796  276374541     4598991
              2 CD2_TPMOV                               4          63731796  276374541     4598991
              3 CD2_SERIE                               4          63731796  276374541     4598991
              4 CD2_DOC                                 4          63731796  276374541     4598991
              5 CD2_CODFOR                              4          63731796  276374541     4598991
              6 CD2_LOJFOR                              4          63731796  276374541     4598991
              7 CD2_ITEM                                4          63731796  276374541     4598991
              8 CD2_CODPRO                              4          63731796  276374541     4598991
              9 CD2_IMP                                 4          63731796  276374541     4598991
             10 R_E_C_N_O_                              4          63731796  276374541     4598991
             11 D_E_L_E_T_                              4          63731796  276374541     4598991

11 linhas selecionadas.

SQL> SELECT COUNT(DISTINCT(CD2_FILIAL)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_FILIAL))
---------------------------
                        258

SQL> SELECT COUNT(DISTINCT(CD2_TPMOV)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_TPMOV))
--------------------------
                         2

SQL> SELECT COUNT(DISTINCT(CD2_SERIE)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_SERIE))
--------------------------
                       413

SQL> SELECT COUNT(DISTINCT(CD2_DOC)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_DOC))
------------------------
                 1108099

SQL> SELECT COUNT(DISTINCT(CD2_CODFOR)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_CODFOR))
---------------------------
                      10500

SQL> SELECT COUNT(DISTINCT(CD2_LOJFOR)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_LOJFOR))
---------------------------
                        322

SQL> SELECT COUNT(DISTINCT(CD2_ITEM)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_ITEM))
-------------------------
                      687

SQL> SELECT COUNT(DISTINCT(CD2_CODPRO)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_CODPRO))
---------------------------
                      24218

SQL> SELECT COUNT(DISTINCT(CD2_IMP)) FROM TOTVS.CD2010;

COUNT(DISTINCT(CD2_IMP))
------------------------
                       7

SQL> SELECT COUNT(DISTINCT(R_E_C_N_O_)) FROM TOTVS.CD2010;

COUNT(DISTINCT(R_E_C_N_O_))
---------------------------
                  277218707

SQL> SELECT COUNT(DISTINCT(D_E_L_E_T_)) FROM TOTVS.CD2010;

COUNT(DISTINCT(D_E_L_E_T_))
---------------------------
                          2

SQL>
```



Vamos para a Prática...

Diamante

AGGRANDIZE

COMMVAULT

TD SYNNEX

Platina

GOLDENGATEBR
Consultoria & Treinamentos

DISCOVER

Ouro

scansource

VERTICA
by opentext™

Prata

TRACES

CAFÉ COM CLOUD

ROX
We take care of your data

Apoio

FIAP

GRUPO POSEIDON DIGITAL