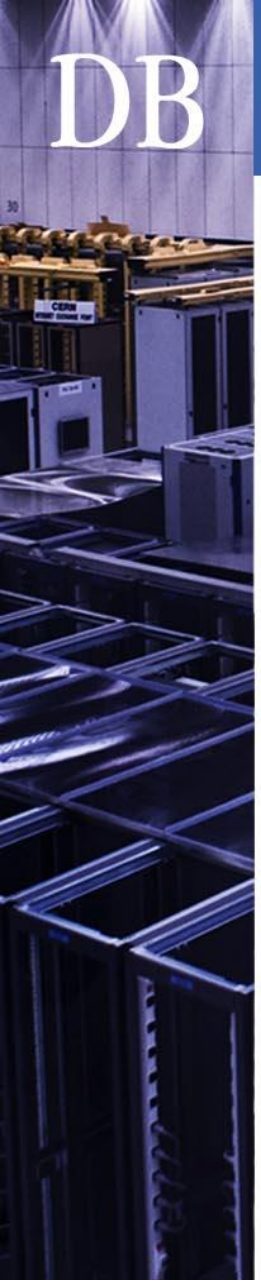
Execution plans and tools

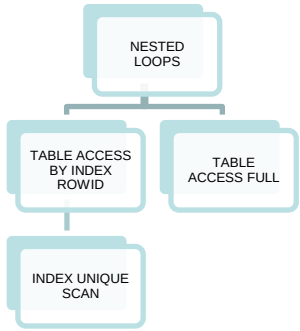
Dawid Wojcik

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



Execution plans

- Execution plan
  - Text or graphical representation of steps Oracle server takes to execute specific SQL
  - Execution plan is a tree in which every node is a DB server operation
  - Prepared during hard parsing of a statement and kept inside library cache
  - There can be multiple execution plans for the same query
    - Depending on bind variables
    - Depending on statistics
    - Depending on hints
    - Plans may change when they age out of library cache (new hard parse required)
    - An explain plan might be different than actual execution plan



Viewing execution plans

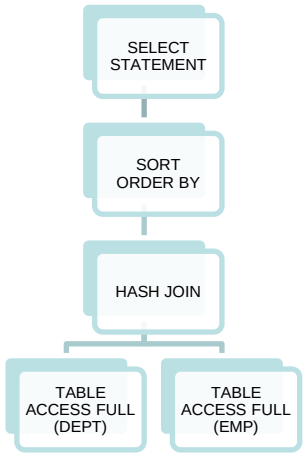
- When designing a query
  - `explain plan for select …`
  - `select * from table(dbms_xplan.display());`
- Viewing an existing cursor's plan (sql_id is known)
  - `select * from table(dbms_xplan.display_cursor('sql_id',cursor_id,'all'));`
- Viewing all plans from AWR (Automatic Workload Repository)
  - `select * from table(dbms_xplan.display_awr('sql_id',null,null,'all'));`
- If you suspect statistics or cardinality estimation problem
  - `select /*+ gather_plan_statistics */ …`
  - `select * from table(dbms_xplan.display_cursor(null, null, 'ALLSTATS LAST'));`
  - see (Google ;)) Cardinality Feedback Tuning by Wolfgang Breitling
- See Oracle documentation for details

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

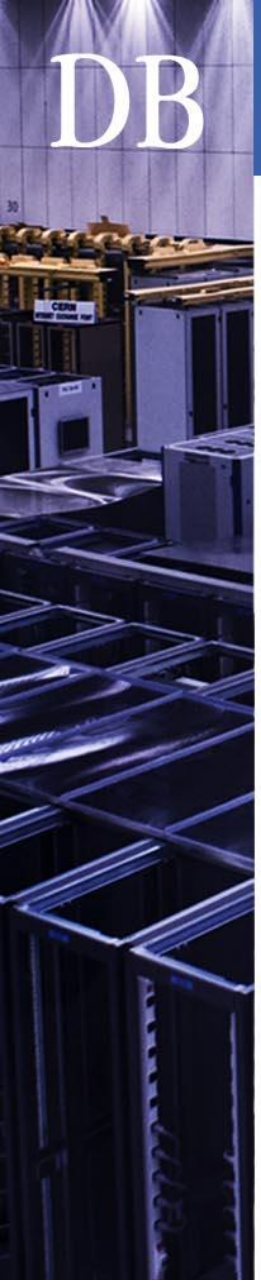
Reading execution plans

- Few simple rules of reading execution plans
  - Parent operations get input only from their children (data sources)
  - Data access starts from the first line without children
  - Rows are "sent" upwards to parent data sources in cascading fashion

```
select d.dname, d.loc, e.empno, e.ename from emp e, dept d where e.deptno = d.deptno
and d.dname = 'SALES' and e.ename between 'A%' and 'X%' order by e.deptno;

-------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     5 |   315 |     8  (25)| 00:00:01 |
|   1 |  SORT ORDER BY      |      |     5 |   315 |     8  (25)| 00:00:01 |
|*  2 |   HASH JOIN         |      |     5 |   315 |     7  (15)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| DEPT |     1 |    30 |     3   (0)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| EMP  |    14 |   462 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - access("E"."DEPTNO"="D"."DEPTNO")
   3 - filter("D"."DNAME"='SALES')
   4 - filter("E"."ENAME">='A%' AND "E"."ENAME"<='X%')
```



CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



Reading execution plans

CERN IT Department

```
select name as "name", coalesce(sum("pending"),0) "pending", coalesce(sum("running"),0) "running", coalesce(sum("unknown"),0) "unknown",
coalesce(sum("terminated"),0) "terminated", coalesce(sum("done"),0) "done", coalesce(sum("canc"),0) "cancelled", coalesce(sum("abort"),0)
"aborted",coalesce(sum("apps"),0) "app-succeeded" , coalesce(sum("applic-failed"),0) "applic-failed",coalesce(sum("site-failed"),0) "site-
failed",coalesce(sum("user-failed"),0) "user-failed",coalesce(sum("unk-failed"),0) "unk-failed", coalesce(sum("site-calc-failed"),0) "site-calc-
failed", coalesce(sum("NEvProc"),0) "events", coalesce(sum("ExeCPU"),0) "cpu", coalesce(sum("WrapWC"),0) "wc", coalesce(sum("allunk"),0) as "allunk",
coalesce(sum("UnSuccess"),0) as "unsuccess" from ( select short_ce."ShortCEName" as name, decode ("DboardStatusId", 'T',
decode(JOB."DboardGridEndId",'D',1,0)) "done", decode(JOB."DboardStatusId",'R',1,0) "running", decode(JOB."DboardStatusId",'T',1,0) "terminated",
decode(JOB."DboardStatusId",'P',1,0) "pending", decode("DboardStatusId", 'U', 1, 0) as "unknown", decode ("DboardStatusId", 'T',
decode(JOB."DboardGridEndId",'C',1,0)) as "canc", decode ("DboardStatusId", 'T', decode(JOB."DboardGridEndId",'A',1,0)) as "abort", decode
("DboardStatusId", 'T', decode(JOB."DboardJobEndId",'S',1,0)) as "apps", decode ("DboardStatusId", 'T', decode ("DboardJobEndId", 'F',
decode("SiteUserFlag", 'application', 1, 0))) as "applic-failed", decode ("DboardStatusId", 'T', decode ("DboardJobEndId", 'F',
decode("SiteUserFlag", 'site', 1, 0))) as "site-failed", decode ("DboardStatusId", 'T', decode ("DboardJobEndId", 'F', decode("SiteUserFlag", 'user',
1, 0))) as "user-failed", decode ("DboardStatusId", 'T', decode ("DboardJobEndId", 'F', decode("SiteUserFlag", 'unknown', 1, 0))) as "unk-failed",
decode ("DboardStatusId", 'T', decode ("DboardJobEndId", 'F', decode("SiteUserFlag", 'site', 1, 0))) as "site-calc-failed", decode ("DboardStatusId",
'T', decode (JOB."DboardGridEndId", 'U', decode(JOB."DboardJobEndId", 'U', 1, 0))) as "allunk", decode ("DboardStatusId", 'T', coalesce("NEvProc",0))
as "NEvProc", decode ("DboardStatusId", 'T', decode("ExeCPU",0,(decode(sign("WrapCPU"),1,"WrapCPU",0)),"ExeCPU")) as "ExeCPU", decode
("DboardStatusId", 'T', coalesce("WrapWC",0)) as "WrapWC", decode(JOB."DboardJobEndId",'S',(decode(JOB."DboardGridEndId",'C',1,'A',1,0)),0) as
"UnSuccess" from JOB,TASK, TASK_TYPE ,short_ce,site, APP_GENERIC_STATUS_REASON where JOB."TaskId"=TASK."TaskId" and TASK."TaskTypeId" =
TASK_TYPE."TaskTypeId" and JOB."ShortCEId" = short_ce."ShortCEId" and job."SiteId" = site."SiteId" and JOB."JobExecExitCode" =
APP_GENERIC_STATUS_REASON."AppGenericErrorCode" (+) and (("FinishedTimeStamp" <= :bv_date2 and "FinishedTimeStamp" >= :bv_date1 AND "DboardStatusId"
= 'T' AND "DboardFirstInfoTimeStamp" >= cast(:bv_date1 AS TIMESTAMP) - interval '14' day) OR ("DboardStatusId" in ('P','R') AND
"DboardFirstInfoTimeStamp" >= cast(:bv_date1 AS TIMESTAMP) - interval '14' day)) and task_type."NewType" = :bv_activity and site."VOName" = :bv_site
order by short_ce."ShortCEName") group by name order by "pending"+"running"+"unknown"+"terminated" desc;
```

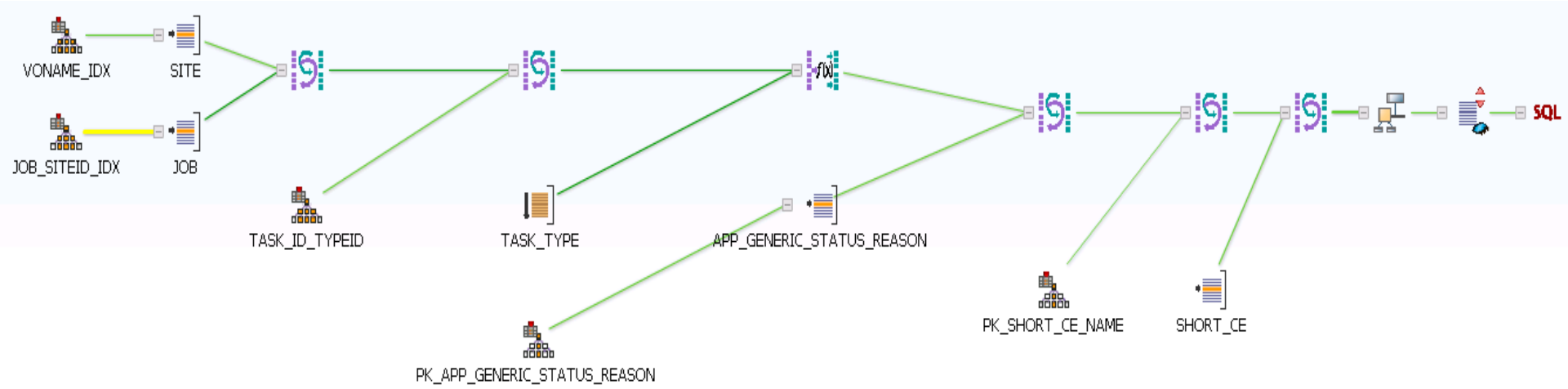
```
-----------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                          | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                               |       |       | 36975 (100)|          |       |       |
|   1 |  SORT ORDER BY                           |                               |     1 |   142 | 36975   (1)| 00:05:51 |       |       |
|   2 |   HASH GROUP BY                          |                               |     1 |   142 | 36975   (1)| 00:05:51 |       |       |
|   3 |    NESTED LOOPS                          |                               |       |       |            |          |       |       |
|   4 |     NESTED LOOPS                         |                               |     1 |   142 | 36973   (1)| 00:05:51 |       |       |
|   5 |      NESTED LOOPS OUTER                  |                               |     1 |   115 | 36972   (1)| 00:05:51 |       |       |
|*  6 |       HASH JOIN                          |                               |     1 |   100 | 36971   (1)| 00:05:51 |       |       |
|   7 |        NESTED LOOPS                      |                               |     4 |   344 | 36969   (1)| 00:05:51 |       |       |
|   8 |         NESTED LOOPS                     |                               |     4 |   304 | 36961   (1)| 00:05:51 |       |       |
|   9 |          TABLE ACCESS BY INDEX ROWID     | SITE                          |     1 |    16 |     2   (0)| 00:00:01 |       |       |
|* 10 |           INDEX RANGE SCAN               | VONAME_IDX                    |     1 |       |     1   (0)| 00:00:01 |       |       |
|* 11 |          TABLE ACCESS BY GLOBAL INDEX ROWID| JOB                         |     4 |   240 | 36959   (1)| 00:05:51 | ROWID | ROWID |
|* 12 |           INDEX RANGE SCAN               | JOB_SITEID_IDX                | 224K|       |  1810   (1)| 00:00:18 |       |       |
|* 13 |         INDEX RANGE SCAN                 | TASK_ID_TYPEID                |     1 |    10 |     2   (0)| 00:00:01 |       |       |
|* 14 |        TABLE ACCESS FULL                 | TASK_TYPE                     |     2 |    28 |     2   (0)| 00:00:01 |       |       |
|  15 |       TABLE ACCESS BY INDEX ROWID        | APP_GENERIC_STATUS_REASON     |     1 |    15 |     1   (0)| 00:00:01 |       |       |
|* 16 |        INDEX UNIQUE SCAN                 | PK_APP_GENERIC_STATUS_REASON  |     1 |       |     0   (0)|          |       |       |
|* 17 |      INDEX UNIQUE SCAN                   | PK_SHORT_CE_NAME              |     1 |       |     0   (0)|          |       |       |
|  18 |     TABLE ACCESS BY INDEX ROWID          | SHORT_CE                      |     1 |    27 |     1   (0)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------------------------------------
```

… 100 more lines with predicates …

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

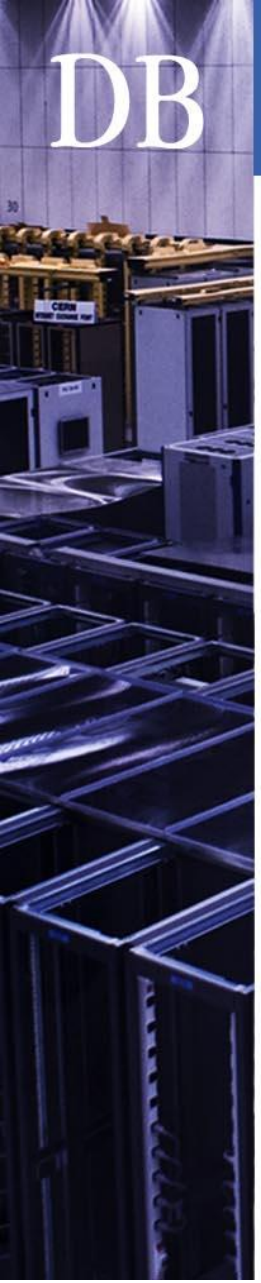
CERN

Reading execution plans



Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	HASH GROUP BY	
3	NESTED LOOPS	
4	NESTED LOOPS	
5	NESTED LOOPS OUTER	
* 6	HASH JOIN	
7	NESTED LOOPS	
8	NESTED LOOPS	
9	TABLE ACCESS BY INDEX ROWID	SITE
* 10	INDEX RANGE SCAN	VONAME_IDX
* 11	TABLE ACCESS BY GLOBAL INDEX ROWID	JOB
* 12	INDEX RANGE SCAN	JOB_SITEID_IDX
* 13	INDEX RANGE SCAN	TASK_ID_TYPEID
* 14	TABLE ACCESS FULL	TASK_TYPE
15	TABLE ACCESS BY INDEX ROWID	APP_GENERIC_STATUS_REASON
* 16	INDEX UNIQUE SCAN	PK_APP_GENERIC_STATUS_REASON
* 17	INDEX UNIQUE SCAN	PK_SHORT_CE_NAME
18	TABLE ACCESS BY INDEX ROWID	SHORT_CE

- Parent operations get input only from their children (data sources)
- Data access starts from the first line without children
- Rows are “sent” upwards to parent data sources in cascading fashion

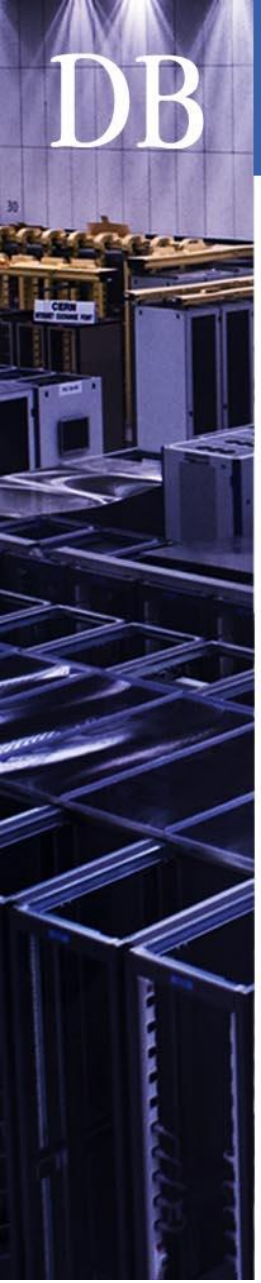


Execution plan – interpreting

- Oracle tries to estimate cardinality of each execution phase (row in the plan)
  - It uses statistics (on tables and indexes)
  - It applies certain heuristics for complex clauses
  - It can use dynamic sampling, if no statistics available
  - if the estimate is orders of magnitude wrong – the execution plan will not be optimal (hours vs. minutes)!
  - Use **`/*+ gather_plan_statistics */`** hint

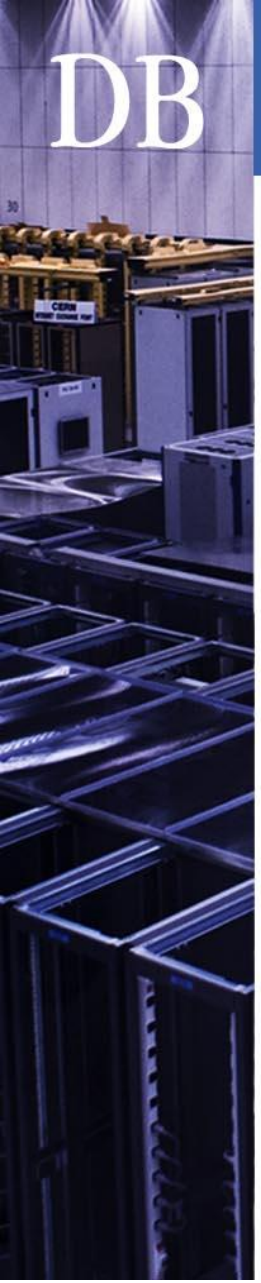
```
-------------------------------------------------------------------------------------------
|  Id | Operation                              | Name        | Starts | E-Rows | A-Rows |
-------------------------------------------------------------------------------------------
|   1 | SORT GROUP BY                          |             |      1 |      1 | 1      |
|*  2 |  FILTER                                |             |      1 |        | 1314K  |
|   3 |   NESTED LOOPS                         |             |      1 |      1 | 1314K  |
|*  4 |    HASH JOIN                           |             |      1 |      1 | 1314K  |
|*  5 |     INDEX RANGE SCAN                   | T2_IND_3    |      1 |   2841 | 2022   |
|*  6 |     TABLE ACCESS BY LOCAL INDEX ROWID  | TEST        |      1 |   3879 | 4771K  |
|*  7 |      INDEX SKIP SCAN                   | TEST_IND_2  |      1 |   3567 | 4771K  |
|*  8 |    INDEX RANGE SCAN                    | T6_IND_4    |  1314K |      1 | 1314K  |
-------------------------------------------------------------------------------------------
```

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it



SQL Monitoring

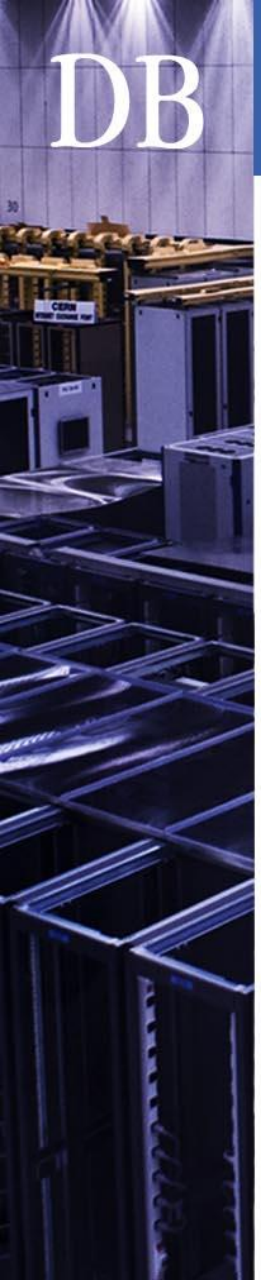
- Oracle 11g Real-Time SQL Monitoring
  - Allows you to monitor the performance of SQL statements while they are being executed and the breakdown of time and resources used during execution
  - Monitors statements that consume more than 5 seconds of CPU or IO time (and samples the execution every second)
  - One can override it by using the `MONITOR` or `NO_MONITOR` hints.
  - Reports can be viewed in Oracle Enterprise Manager or generated directly in a database using package dbms_sqltune.report_sql_monitor



SQL trace

- SQL Trace
  - The only way to capture all the SQL being executed and all the execution steps (and waits) in a session is to switch on SQL trace.
    - `ALTER SESSION SET tracefile_identifier = my_trace1;`
    - `ALTER SESSION SET sql_trace = true;`
    - … run your SQL or PL/SQL …
    - `ALTER SESSION SET sql_trace = false;`
  - Beware – SQL tracing may impact performance of your application, if the tracing is activated for long time
  - Trace files are stored on the DB server and you can ask DBA to send them to you (they can be very big)
  - Trace files can be read in their raw state or translated using the tkprof utility

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

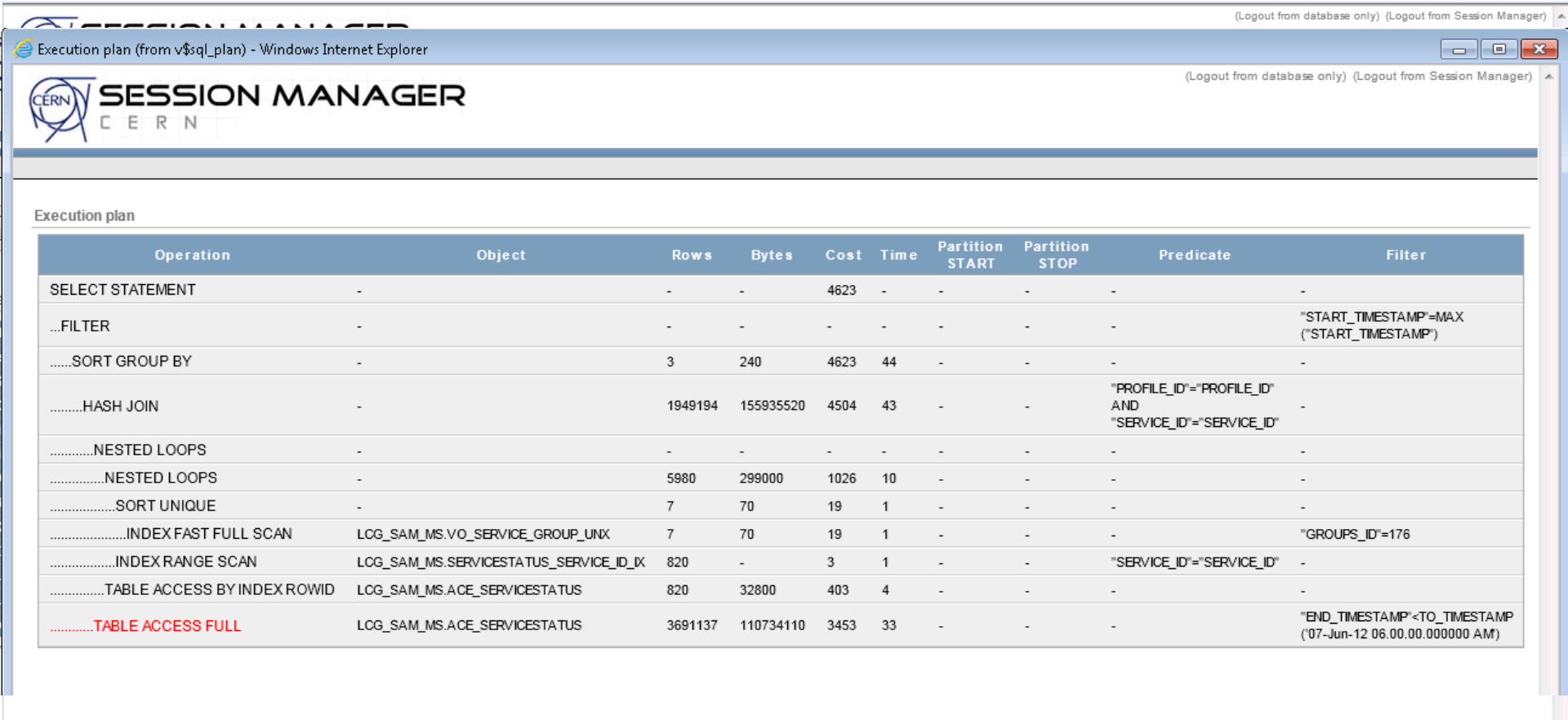


Snapper

- Snapper tool by Tanel Poder
  - An easy to use Oracle session-level performance snapshot utility
  - Comes as a PL/SQL script that does not require creation of any database objects
  - Very useful for ad-hoc performance diagnosis, especially in environments with restrictive change management
  - Example will be presented by Eric

Session Manager

- Available for many production and DBs at CERN
  - https://phydb.web.cern.ch/phydb/SessionManager.html



(Logout from database only) (Logout from Session Manager)

Execution plan (from v$sql_plan) - Windows Internet Explorer

SESSION MANAGER
CERN

(Logout from database only) (Logout from Session Manager)

Execution plan

Operation	Object	Rows	Bytes	Cost	Time	Partition START	Partition STOP	Predicate	Filter
SELECT STATEMENT	-	-	-	4623	-	-	-	-	-
...FILTER	-	-	-	-	-	-	-	-	"START_TIMESTAMP"=MAX ("START_TIMESTAMP")
.....SORT GROUP BY	-	3	240	4623	44	-	-	-	-
.....HASH JOIN	-	1949194	155935520	4504	43	-	-	"PROFILE_ID"="PROFILE_ID" AND "SERVICE_ID"="SERVICE_ID"	-
.....NESTED LOOPS	-	-	-	-	-	-	-	-	-
.....NESTED LOOPS	-	5980	299000	1026	10	-	-	-	-
.....SORT UNIQUE	-	7	70	19	1	-	-	-	-
.....INDEX FAST FULL SCAN	LCG_SAM_MS.VO_SERVICE_GROUP_UNX	7	70	19	1	-	-	-	"GROUPS_ID"=176
.....INDEX RANGE SCAN	LCG_SAM_MS.SERVICESTATUS_SERVICE_ID_IX	820	-	3	1	-	-	"SERVICE_ID"="SERVICE_ID"	-
.....TABLE ACCESS BY INDEX ROWID	LCG_SAM_MS.ACE_SERVICESTATUS	820	32800	403	4	-	-	-	-
.....TABLE ACCESS FULL	LCG_SAM_MS.ACE_SERVICESTATUS	3691137	110734110	3453	33	-	-	-	"END_TIMESTAMP"<TO_TIMESTAMP ('07-Jun-12 06.00.00.000000 AM')

Oracle Enterprise Manager

- Available for many production and DBs at CERN
  - https://oms.cern.ch/em