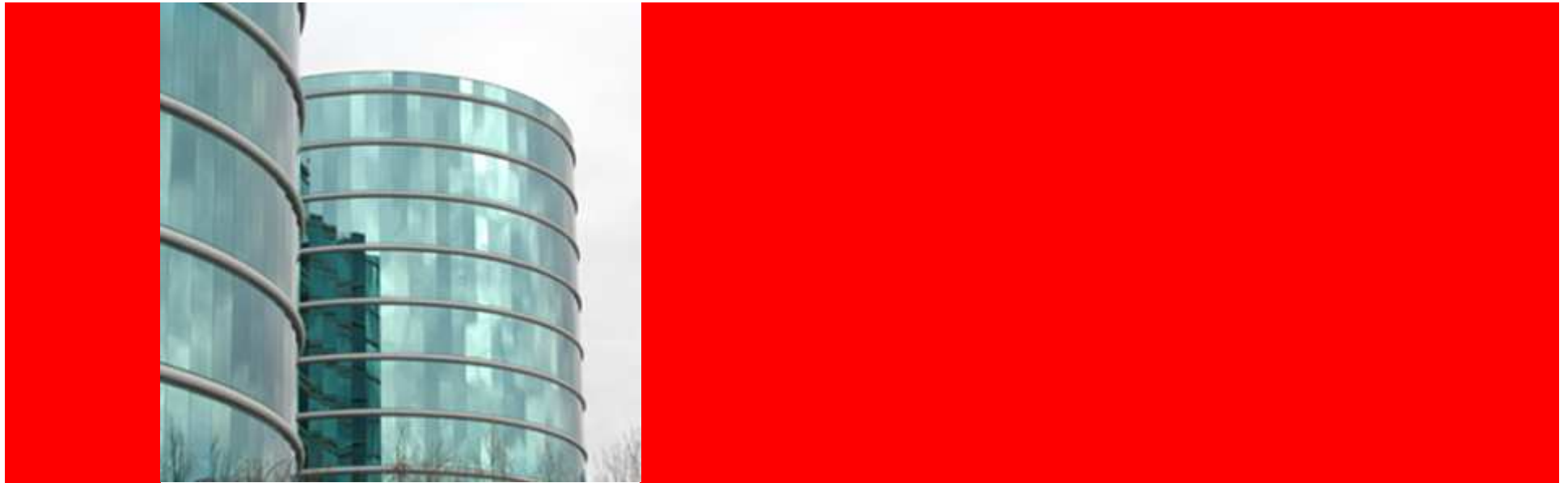


ORACLE®

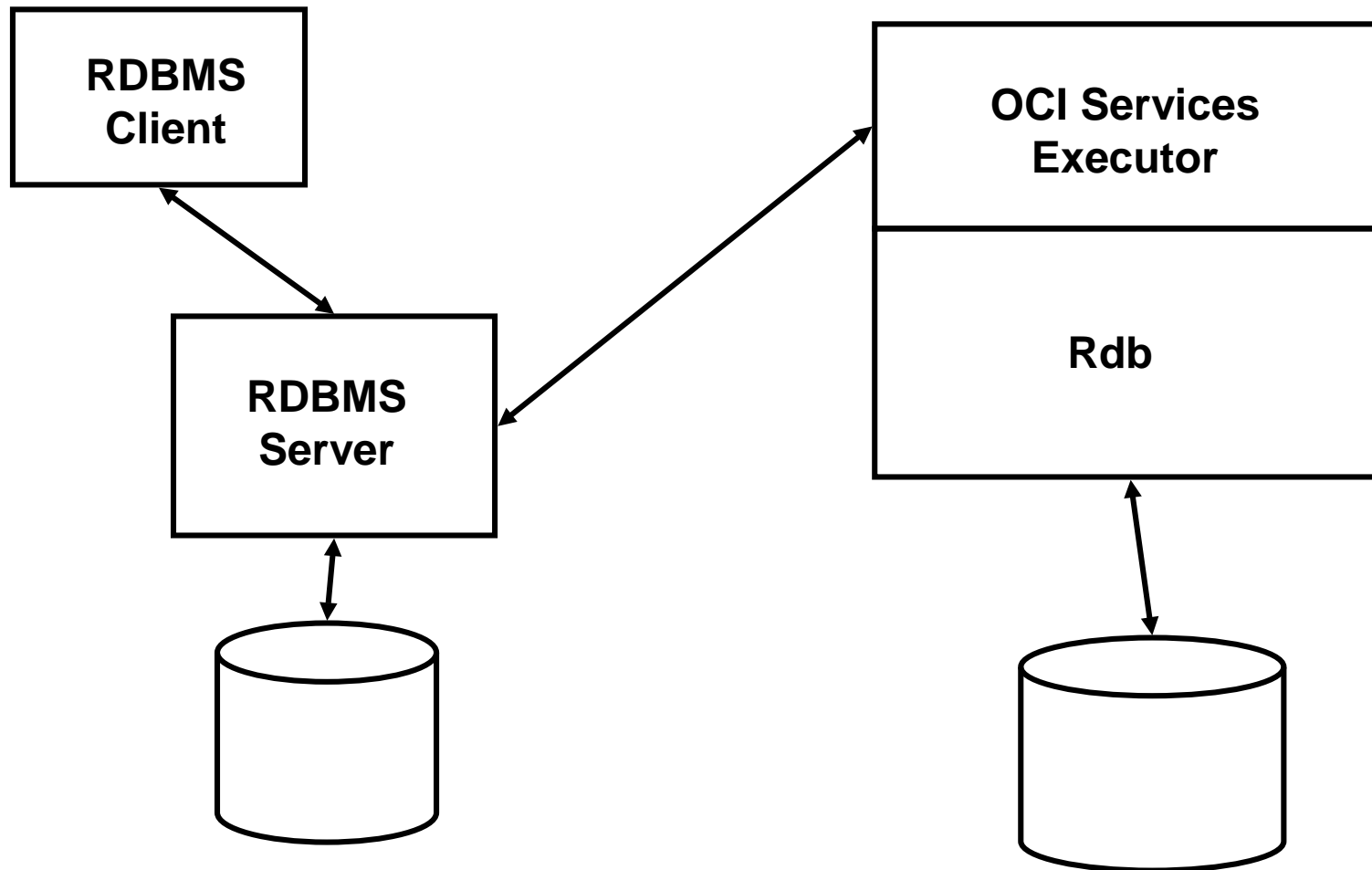


ORACLE[®]

OCI Services Statistics (Enhancements in V7.3.0.2)

Author: Peter Jackson

Overview





A Simple Query

```
SQL> CREATE DATABASE LINK OCI_TEST  
      CONNECT TO SCOTT IDENTIFIED BY TIGER  
      USING OCI_TEST;
```

```
SQL> SELECT COUNT(*)  
      FROM EMPLOYEES@OCI_TEST, DUAL  
      WHERE LAST_NAME = 'Smith';
```

COUNT(*)

2



A Complex Query

```
select distinct p.part_srvc_stat_cd, p.part_25_no,  
               v.vartn_cd, mf.mdl_fmly_cd, b.chass_cnt, p.part_desc  
from cdt400_part_mstr p, cdt350_vartn v,  
     cdt475_mdlfm_part_des mf,  
     cdt010_model a, cdt250_fetr_model b  
where mf.mdl_fmly_cd in ( 'J650', 'J750', 'JCF5', 'JCF6' ) AND  
      mf.rec_from_type = '1' AND mf.vartn_id = v.vartn_id AND  
      mf.part_id = p.part_id AND p.part_srvc_stat_cd = '426' AND  
      mf.mdl_fmly_cd=a.mdl_fmly_cd AND  
      b.mdl_7_cd = a.mdl_7_cd AND v.vartn_cd=b.vartn_cd;
```



What Happens

- Query is sent from the client to the RDBMS server
- That server chooses a plan
- As the plan requires queries are sent to remote databases
- The remote databases will optimize the queries they get and execute them
- The data is assembled by the RDBMS server and returned to the client



How to Look at the Remote Queries

- The most reliable way is to use the OCI Services executor log file
- ALTER SESSION LOG FULL;
- Search for the table name(s)
- At least two queries per table
- SELECT * FROM "EMPLOYEES"
- SELECT "LAST_NAME" FROM "EMPLOYEES"
WHERE "LAST_NAME"='Smith'



What Can Go Wrong

- cdt250_fetr_model b has two predicates
- b.mdl_7_cd = a.mdl_7_cd
- v.vartn_cd = b.vartn_cd
- Depending on the order in which the tables are accessed by the plan none, one or both of these may be available for the remote query
- In this case both were needed for good performance
- With OCI Services 7.3.0.0 RDBMS was choosing to use neither



RDBMS Optimizer

- RDBMS like Rdb uses a Cost Based Optimizer
- To calculate the costs it uses statistics about the tables
- It would ask OCI Services for such statistics
- But it was not getting good data back
- So it would sometimes choose poor plans



How to Look at the Statistics

```
SQL> ALTER SESSION SET EVENTS  
      '10053 trace name context forever, level 1';
```

NOTE:225598.1 How to Obtain Tracing of Optimizer
Computations (EVENT 10053)



Statistics – 7.3.0.0

BASE STATISTICAL INFORMATION

Table Stats::

Table: DUAL Alias: DUAL

#Rows: 1 #Blks: 1 AvgRowLen: 2.00

Table Stats::

Table: EMPLOYEES Alias: EMPLOYEES (NOT ANALYZED)

#Rows: 0 #Blks: 100 AvgRowLen: 100.00



TableScan Cost - 7.3.0.0

For Employees under SINGLE TABLE ACCESS PATH

Access Path: TableScan

Cost: 0.11 Resp: 0.11 Degree: 0

Cost_io: 0.00 Cost_cpu: 712144

Resp_io: 0.00 Resp_cpu: 712144



Statistics 7.3.0.2

Table Stats::

Table: EMPLOYEES Alias: EMPLOYEES

#Rows: 100 #Blks: 0 AvgRowLen: 100.00

Column (#2): LAST_NAME(CHARACTER)

AvgLen: 14.00 NDV: 83 Nulls: 0 Density: 0.012048 Min: 0 Max: 0

Index Stats::

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 2 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 83 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00



Statistics 7.3.0.2

Table Stats::

Table: EMPLOYEES Alias: EMPLOYEES

#Rows: 100 #Blks: 0 AvgRowLen: 100.00

Column (#2): LAST_NAME(CHARACTER)

AvgLen: 14.00 NDV: 83 Nulls: 0 Density: 0.012048 Min: 0 Max: 0

Index Stats::

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 2 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 83 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00



Statistics 7.3.0.2

Table Stats::

Table: EMPLOYEES Alias: EMPLOYEES

#Rows: 100 #Blks: 0 AvgRowLen: 100.00

Column (#2): LAST_NAME(CHARACTER)

AvgLen: 14.00 NDV: 83 Nulls: 0 Density: 0.012048 Min: 0 Max: 0

Index Stats::

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 2 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 83 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00



Statistics 7.3.0.2

Table Stats::

Table: EMPLOYEES Alias: EMPLOYEES

#Rows: 100 #Blks: 0 AvgRowLen: 100.00

Column (#2): LAST_NAME(CHARACTER)

AvgLen: 14.00 **NDV: 83** Nulls: 0 **Density: 0.012048** Min: 0 Max: 0

Index Stats::

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 0.00

Index: 0 **Col#: 2** (NOT ANALYZED)

LVLS: 1 #LB: 25 **#DK: 83** LB/K: 1.00 DB/K: 1.00 CLUF: 0.00



TableScan cost – 7.3.0.2

Access Path: TableScan

Cost: 52.01 Resp: 52.01 Degree: 0

Cost_io: 52.00 Cost_cpu: 42000

Resp_io: 52.00 Resp_cpu: 42000



Still Not Good Enough

- Now TableScan is not chosen for cdt250_fetr_model
- But usually only one predicate is chosen
- Irrelevant factors can cause the plan to vary
- But it is usually still too slow



Index Costing

- The cost for an index retrieval using a predicate will be:

Levels + IO to Scan Index (#LB) * Selectivity

With the OCI Services defaults that is

1 + 25 * Selectivity



Selectivities

- The selectivity of the individual predicates will be the column densities
- We have fairly accurate values for the columns involved in the complex query
- MDL_7_CD - NDV: 175 - Density: 0.0057143
- VARTN_CD - NDV: 223392 - Density: 4.4764e-06



Index Costs

Using MDL_7_CD only	1.14
Using VARTN_CD only	1.00
Using both	1.00



RMU/COLLECT

- RMU/COLLECT OPT/STAT=STORAGE



Statistics – After Collection

Table Stats::

Table: EMPLOYEES Alias: EMPLOYEES

#Rows: 100 #Blks: 51 AvgRowLen: 100.00

Column (#2): LAST_NAME(CHARACTER)

AvgLen: 14.00 NDV: 83 Nulls: 0 Density: 0.012048 Min: 0 Max: 0

Index Stats::

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 1 #LB: 100 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 51.00

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 2 #LB: 1 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 51.00

Index: 0 Col#: 2 (NOT ANALYZED)

LVLS: 2 #LB: 4 #DK: 83 LB/K: 1.00 DB/K: 1.00 CLUF: 51.00



TableScan After Collection

Access Path: TableScan

Cost: 51.06 Resp: 51.06 Degree: 0

Cost_io: 51.00 Cost_cpu: 405193

Resp_io: 51.00 Resp_cpu: 405193



Real Statistics

Table Stats::

Table: CDTP250_FETR_MODEL Alias: B

#Rows: 2431449 #Blks: 27705 AvgRowLen: 100.00

Column (#8): CHASS_CNT(NUMBER)

AvgLen: 22.00 NDV: 75982 Nulls: 0 Density: 1.3161e-05 Min: 0
Max: 0

Column (#3): MDL_7_CD(CHARACTER)

AvgLen: 7.00 NDV: 175 Nulls: 0 Density: 0.0057143 Min: 0 Max: 0

Column (#1): VARTN_CD(CHARACTER)

AvgLen: 5.00 NDV: 223392 Nulls: 0 Density: 4.4764e-06 Min: 0
Max: 0



Real Statistics

Index Stats::

Index: 0 Col#: 3 4 1 2 19 (NOT ANALYZED)

LVLS: 4 #LB: 19412 #DK: 2431449 LB/K: 1.00 DB/K: 1.00 CLUF: 27705.00

Index: 0 Col#: 1 2 4 3 19 (NOT ANALYZED)

LVLS: 4 #LB: 22861 #DK: 2431449 LB/K: 1.00 DB/K: 1.00 CLUF: 27705.00

Index: 0 Col#: 4 3 1 2 19 (NOT ANALYZED)

LVLS: 4 #LB: 23218 #DK: 2431449 LB/K: 1.00 DB/K: 1.00 CLUF: 27705.00

Index: 0 Col#: 19 3 4 1 2 (NOT ANALYZED)

LVLS: 4 #LB: 15514 #DK: 2431449 LB/K: 1.00 DB/K: 1.00 CLUF: 27705.00

Index: 0 Col#: 3 6 (NOT ANALYZED)

LVLS: 4 #LB: 6540 #DK: 60954 LB/K: 1.00 DB/K: 1.00 CLUF: 27705.00

Index: 0 Col#: 2 9 3 (NOT ANALYZED)

LVLS: 4 #LB: 185319 #DK: 144700 LB/K: 1.00 DB/K: 1.00 CLUF: 27705.00



Real Statistics

Table Stats::

Table: CDTP010_MODEL Alias: A

#Rows: 471 #Blks: 5 AvgRowLen: 100.00

Column (#2): MDL_FMLY_CD(CHARACTER)

AvgLen: 4.00 NDV: 89 Nulls: 0 Density: 0.011236 Min: 0 Max: 0

Column (#1): MDL_7_CD(CHARACTER)

AvgLen: 7.00 NDV: 471 Nulls: 0 Density: 0.0021231 Min: 0 Max: 0


Index Stats::

Index: 0 Col#: 1 (NOT ANALYZED)

LVLS: 2 #LB: 1 #DK: 471 LB/K: 1.00 DB/K: 1.00 CLUF: 5.00

Index: 0 Col#: 2 (NOT ANALYZED)

LVLS: 2 #LB: 16 #DK: 89 LB/K: 1.00 DB/K: 1.00 CLUF: 5.00



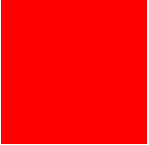
TableScan – Large Table

Access Path: TableScan

Cost: 27845.22 Resp: 27845.22 Degree: 0

Cost_io: 27705.00 Cost_cpu: 902419705

Resp_io: 27705.00 Resp_cpu: 902419705



TableScan – Small Table

Access Path: TableScan

Cost: 5.09 Resp: 5.09 Degree: 0

Cost_io: 5.00 Cost_cpu: 578798

Resp_io: 5.00 Resp_cpu: 578798



Index Costs After Collection

Using MDL_7_CD only Index 3 4 1 2 19	114.93
Using MDL_7_CD only Index 3 6	41.37
Using VARTN_CD only Index 1 2 4 3 19	14.88
Using both	Not possible?



Better Index Costing

- The formula for calculating index costs is closer to

$$\begin{aligned} &\text{Levels} + \#LB * \text{Index Selectivity} \\ &+ CLUF * \text{Selectivity with filters} \end{aligned}$$

- CLUF stands for Clustering Factor
- It is supposed to be the number of IOs to read through all the rows in the index order
- Note:39836.1 Clustering Factor



Actual Index Costs

Access Path: index (IndexOnly)

Index: 0

resc_io: 114.00 resc_cpu: 812894

ix_sel: 0.0057143 ix_sel_with_filters: 2.5580e-08

NL Join: Cost: 1014.31 Resp: 1014.31 Degree: 1

Cost_io: **162.02** Cost_cpu: 5485054491

Resp_io: 162.02 Resp_cpu: 5485054491

Access Path: index (IndexOnly)

Index: 0

resc_io: 4.00 resc_cpu: 29536

ix_sel: 4.4764e-06 ix_sel_with_filters: 2.5580e-08

NL Join: Cost: 904.19 Resp: 904.19 Degree: 1

Cost_io: **52.02** Cost_cpu: 5484271133

Resp_io: 52.02 Resp_cpu: 5484271133



Summary

If you have performance problems with queries that use OCI Services and access multiple databases:

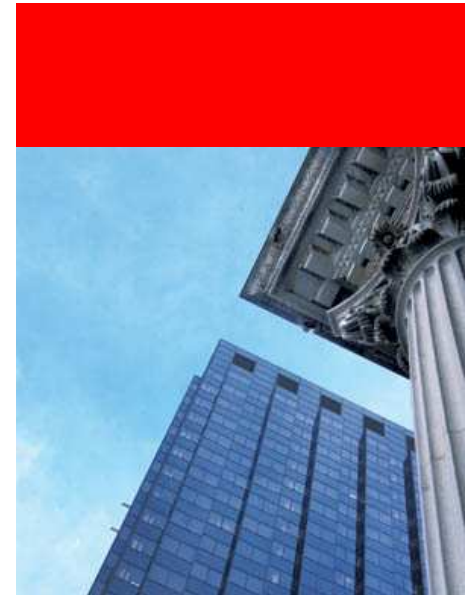
- 1) Upgrade to 7.3.0.2
- 2) Do a RMU/COLLECT OPT/STAT=STOR
- 3) Use event 10053 to create an alert log, and get a matching executor log and then contact support

Note:867679.1 How to Troubleshoot Performance Problems Involving Access to a Rdb Database Via a DBlink





Appendix





Notes for the CD

- Comments and details that would get in the way of the presentation but you might like for reference follow
- Matched by slide name



Overview

- One RDBMS and one Rdb database is probably the most common configuration in which the enhanced statistics are useful
- All that is needed is a Rdb database and another database – Rdb, RDBMS or some other database accessed through RDBMS
- Even when a single Rdb database is referenced in the query they may be useful if the query can not be passed straight to Rdb



A Simple Query

- If only one database is involved, even if several remote tables are referenced the query should be passed over untouched.
- Unfortunately it won't be. This is being worked on as Bug:8811747
- When multiple databases are involved then the RDBMS server has to produce a plan for executing it.
- It will normally use the Cost Based Optimizer to do this.
- It needs statistics to about the tables to calculate the costs, and will ask OCI Service for those statistics.
- These are the ones that have been enhanced in 7.3.0.2
- For a simple query like this they don't matter, but for more complex queries an incorrect plan might lead to a table with millions of rows being sequentially scanned repeatedly.



A Complex Query

- This is a real query, used with permission.
- Synonyms are used to hide the DB links.
- The last two tables are in Rdb.
- One did have millions of rows.
- Without the statistics enhancements this query takes a very, very long time, longer than we have ever cared to wait.



How to Look at the Remote Queries

- The `SELECT * FROM <table>` query is prepared but not executed. It is only used to get a description of the table from Rdb.
- `SET AUTOTRACE ON` in SQL*Plus will get remote queries reported, but may not be accurate. This is because there is an additional step which can combine queries on the same remote database. This feature is called “colocated inline joins”.



Statistics – 7.3.0.0

- 7.3.0.0 would return a full set of statistics, but they would mostly be zero.
- The #Rows should have been passed but was not
- This number is probably the most important for calculating the cost of a strategy
- Since values were passed RDBMS would not use defaults
- A list of columns with their types and sizes is passed, but only the ones RDBMS thinks are interesting are listed. None in this case.
- The index keys were returned, even though they are not reported in the example.



TableScan Cost - 7.3.0.0

- The IO cost of a full table scan is passed over as zero by OCI Services prior to 7.3.0.2.
- So RDBMS would think that a TableScan was efficient
- TableScans do not need predicates so they might not be passed
- When no predicates are passed Rdb is not able to less than a full scan.



Statistics 7.3.0.2

- The table is no longer flagged as NOT ANALYZED
- The estimate of the table cardinality stored by Rdb is passed by OCI services as the number of rows.
- In 7.3.0.2 the stored estimated index cardinality from Rdb is passed as the number of distinct keys #DK.
- Rdb does not store column statistics, but for this column there is an index using the column as the key so OCI Service uses the number of distinct keys from the index as the number of distinct values (NDV) for the column.
- The density is the inverse of the NDV.
- The other non-zero entries shown now default to same values as RDBMS would default to.
- The index names are not passed as the optimizer does not need them.



TableScan cost – 7.3.0.2

- The IO cost of a table scan is now defaulted to 52.
- The cost of doing an index rangescan will be roughly the number of levels plus the number of logical blocks (#LB)
- With the default level of 1 and default #LB of 25 than gives an IO cost of 26 for doing a full index scan.
- We set the default TableScan cost to twice that to discourage the RDBMS optimizer from choosing plans involving tablescans.



Statistics – After Collection

- #Blks is set to what we calculate to be the number of I/Os to do a table scan.
- The number of levels is the collected value.
- The #LB is the calculated I/O cost to do an index scan.
- The first index is a hashed one which is why the #LB is so high.
- CLUF is set equal to the TableScan IO cost.



Real Statistics

- We can get good estimates for the NDV for columns which are the first (or only) part of an index key. Columns 1 and 3 in this case.
- Column 8 has the NDV defaulted.
- The default is 3.125% of the table cardinality. This value is used as a default by RDBMS and also within Rdb. It comes from an old paper on relational databases.



Actual Index Costs

- Costs shown are for the two cheapest indexes from where the table is joined in the best cost plan
- IndexOnly does not mean the same as in Rdb
- Here is the next cheapest index which has no filter

Access Path: index (RangeScan)

Index: 0

resc_io: 200.00 resc_cpu: 1425338

ix_sel: 0.0057143 ix_sel_with_filters: 0.0057143

NL Join: Cost: 1100.41 Resp: 1100.41 Degree: 1

Cost_io: 248.02 Cost_cpu: 5485666935

Resp_io: 248.02 Resp_cpu: 5485666935



ORACLE®