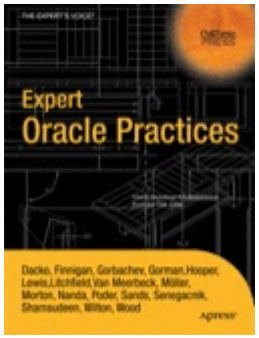

Utilizing new CBO features after upgrading to Oracle 12c – A practical example
Jože Senegačnik
Oracle ACE Director
joze.senegacnik@dbprof.com

About the Speaker

Jože Senegačnik

- First experience with Oracle Version 4 in 1988
- 26 years of experience with Oracle RDBMS.
- Proud member of the OakTable Network www.oaktable.net
- Oracle ACE Director
- Co-author of the OakTable book “Expert Oracle Practices” by Apress (Jan 2010)
- VP of Slovenian OUG (SIOUG) board
- CISA – Certified IS auditor
- Blog about Oracle: http://joze-senegacnik.blogspot.com



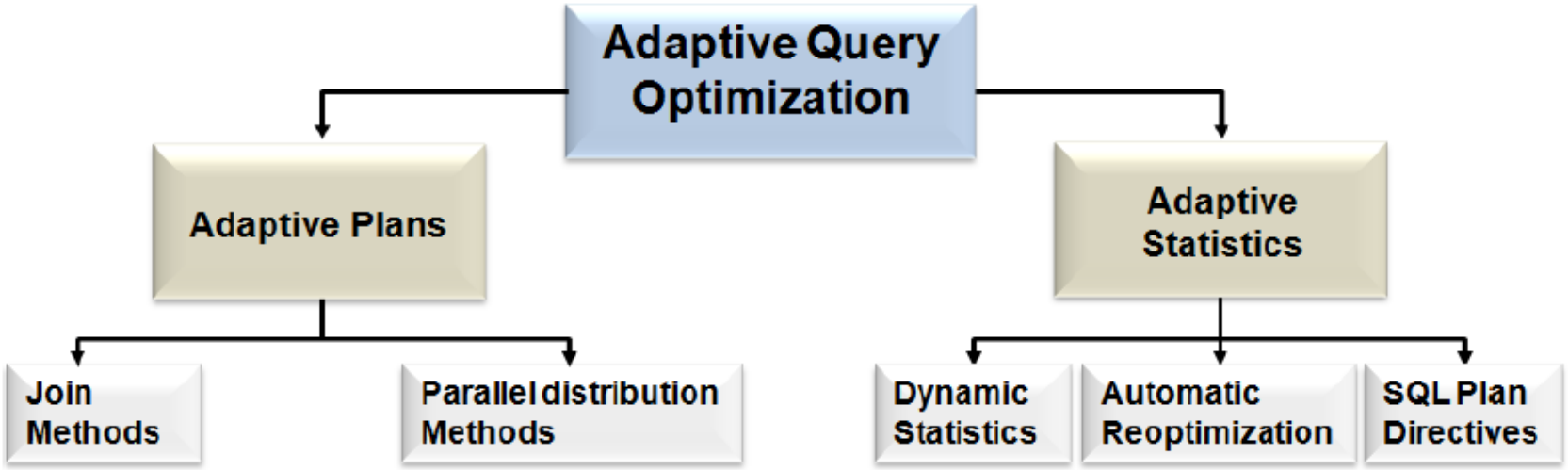
- PPL(A) – private pilot license PPL(A) / instrument rated IR/SE
- Blog about flying: http://jsenegacnik.blogspot.com
- Blog about Building Ovens, Baking and Cooking: http://senegacnik.blogspot.com

Agenda

- Adaptive Query Optimization
- Online statistics gathering
- Dynamic Statistics (a.k.a dynamic sampling)
- SQL plan directives
- New types of histograms
  - **Top-Frequency histograms**
  - **Hybrid histograms**
- Upgrade process

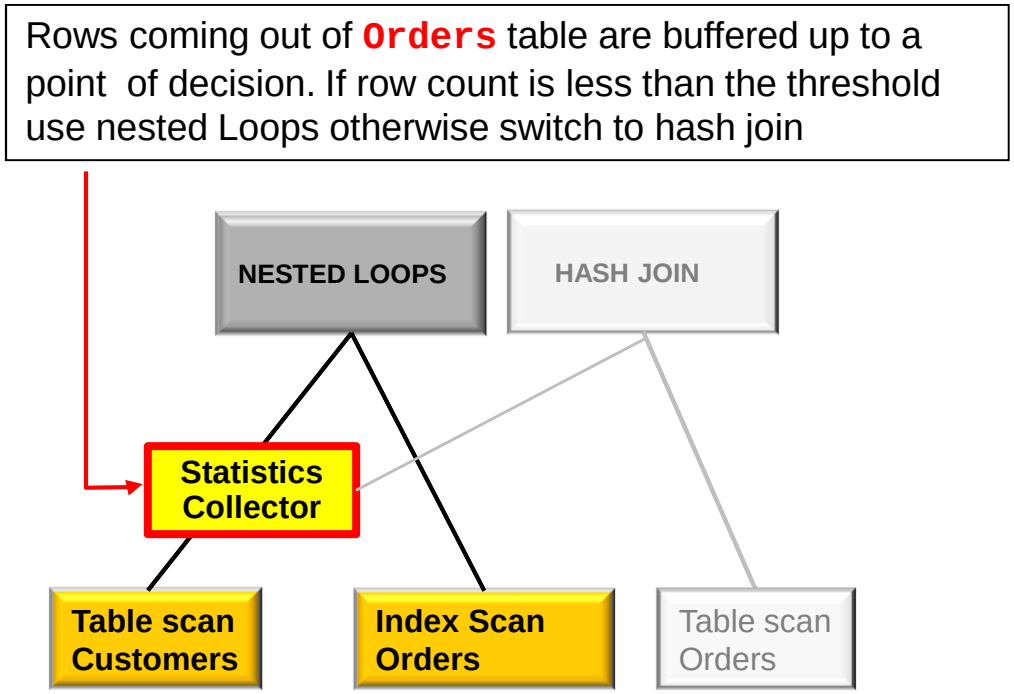
Adaptive Query Optimization

- Optimizer learning from actual execution and „own mistakes“.



Adaptive (Execution) Plan

- Alternative sub-plans are prepared
- Sub-plans are stored in the cursor
- Stats collect is inserted before join
- Rows are buffered until final decision is made



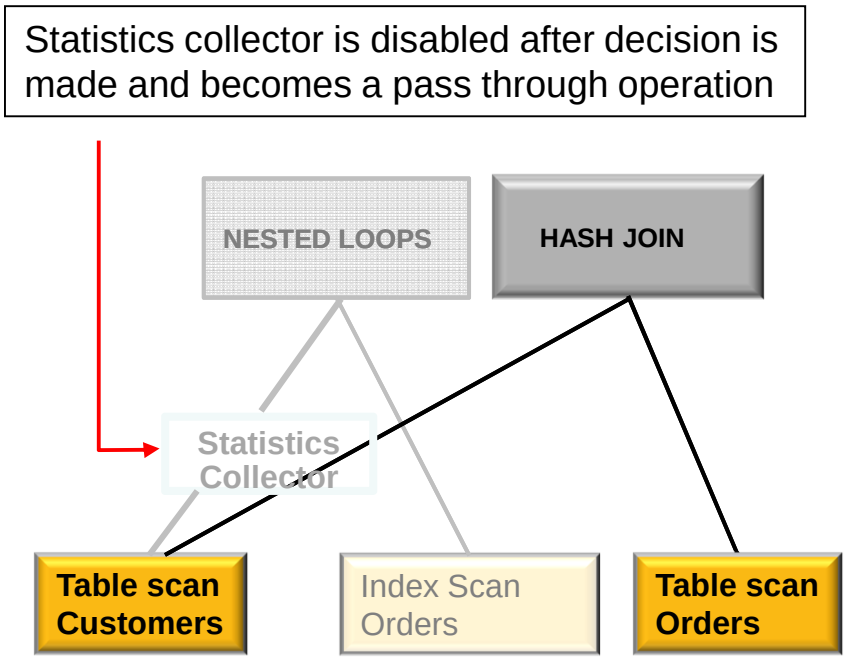
Default plan is a NESTED LOOP (NL) join

Adaptive Plan

– If number of rows seen in statistics collector exceeds threshold

- Plan switches to hash join
- Statistics collect becomes disabled

– Plan is resolved on first execution and remains the same for subsequent executions



Final Plan is a HASH JOIN (HJ)

Adaptive Plan - Demo

- Random SQL statement with adaptive plan selected from V$SQL for demonstration

```
SELECT /*+ CONNECT_BY_FILTERING */ s.privilege# FROM sys.sysauth$ s
 CONNECT BY s.grantee# = PRIOR s.privilege#
 AND (s.privilege# > 0 OR s.privilege# = -352)
 START WITH (s.privilege# > 0 OR s.privilege# = -352)
 AND s.grantee# IN
   (SELECT c1.privilege# FROM sys.codeauth$ c1 WHERE c1.obj# = :1)
UNION
SELECT c2.privilege# FROM sys.codeauth$ c2 WHERE c2.obj# = :2
ORDER BY 1 ASC
```

```
SQL> select * from table(dbms_xplan.display_cursor('gngtvs38t0060',format=>'+adaptive'));
-------------------------------------------------------------------------------------------------
|   Id  | Operation                                | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT                         |              |       |       |     3 (100)|          |
|     1 |  SORT UNIQUE                             |              |     5 |   130 |     3   (0)| 00:00:01 |
|     2 |   UNION-ALL                              |              |       |       |            |          |
|  *  3 |    CONNECT BY WITH FILTERING (UNIQUE)    |              |       |       |            |          |
|- *  4 |     HASH JOIN                            |              |     1 |    35 |     1   (0)| 00:00:01 |
|     5 |      NESTED LOOPS                        |              |     1 |    35 |     1   (0)| 00:00:01 |
|-    6 |       STATISTICS COLLECTOR               |              |       |       |            |          |
|  *  7 |        INDEX RANGE SCAN                  | I_CODEAUTH1  |     1 |    26 |     0   (0)|          |
|  *  8 |       INDEX RANGE SCAN                   | I_SYSAUTH1   |     1 |     9 |     1   (0)| 00:00:01 |
|- *  9 |      INDEX FAST FULL SCAN                | I_SYSAUTH1   |     1 |     9 |     1   (0)| 00:00:01 |
|    10 |     NESTED LOOPS                         |              |     3 |    66 |     2   (0)| 00:00:01 |
|    11 |      CONNECT BY PUMP                     |              |       |       |            |          |
|  * 12 |      INDEX RANGE SCAN                    | I_SYSAUTH1   |     3 |    27 |     1   (0)| 00:00:01 |
|  * 13 |    INDEX RANGE SCAN                      | I_CODEAUTH1  |     1 |    26 |     0   (0)|          |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("S"."GRANTEE#"=PRIOR NULL)
   4 - access("S"."GRANTEE#"="C1"."PRIVILEGE#")
   7 - access("C1"."OBJ#"=:1)
   8 - access("S"."GRANTEE#"="C1"."PRIVILEGE#")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
   9 - filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
  12 - access("S"."GRANTEE#"="connect$_by$_pump$_003"."PRIOR s.privilege#      $POS99")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
  13 - access("C2"."OBJ#"=:2)
Note
-----
   - this is an adaptive plan (rows marked '-' are inactive)
```

```
SQL> select * from table(dbms_xplan.display_cursor('gngtvs38t0060'));

-------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |              |       |       |     3 (100)|          |
|   1 |  SORT UNIQUE                           |              |     5 |   130 |     3   (0)| 00:00:01 |
|   2 |   UNION-ALL                            |              |       |       |            |          |
|*  3 |    CONNECT BY WITH FILTERING (UNIQUE)  |              |       |       |            |          |
|   4 |     NESTED LOOPS                       |              |     1 |    35 |     1   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                  | I_CODEAUTH1  |     1 |    26 |     0   (0)|          |
|*  6 |      INDEX RANGE SCAN                  | I_SYSAUTH1   |     1 |     9 |     1   (0)| 00:00:01 |
|   7 |     NESTED LOOPS                       |              |     3 |    66 |     2   (0)| 00:00:01 |
|   8 |      CONNECT BY PUMP                   |              |       |       |            |          |
|*  9 |      INDEX RANGE SCAN                  | I_SYSAUTH1   |     3 |    27 |     1   (0)| 00:00:01 |
|* 10 |    INDEX RANGE SCAN                    | I_CODEAUTH1  |     1 |    26 |     0   (0)|          |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("S"."GRANTEE#"=PRIOR NULL)
   5 - access("C1"."OBJ#"=:1)
   6 - access("S"."GRANTEE#"="C1"."PRIVILEGE#")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
   9 - access("S"."GRANTEE#"="connect$_by$_pump$_003"."PRIOR s.privilege#      $POS99")
       filter(("S"."PRIVILEGE#">0 OR "S"."PRIVILEGE#"=(-352)))
  10 - access("C2"."OBJ#"=:2)

Note
-----
   - this is an adaptive plan
```

Determining Adaptive Plans

- New column added V$SQL.IS_RESOLVED_ADAPTIVE_PLAN
- Values
  - Y – final plan was determined
  - N – final plan was not determined yet (not executed yet or executing but final plan was not determined yet)
  - NULL – non-adaptive plan

```
SQL> select nvl(IS_RESOLVED_ADAPTIVE_PLAN,'NULL') as val, count(*)
     from V$SQL
     group by nvl(IS_RESOLVED_ADAPTIVE_PLAN,'NULL');

VAL    COUNT(*)
---- ----------
Y          1153
NULL       7971
N            22 -> all SQLs were not executed yet (executions = 0)
```

Displaying Adaptive Plans

- FINAL Execution Plan

```
select * from
table(dbms_xplan.display_cursor('gngtvs38t0060'));

Note
-----
 - this is an adaptive plan
```

- ADAPTIVE Execution Plan

```
select * from table(dbms_xplan.display_cursor('gngtvs38t0060',
       format=>'+adaptive'));

Note
-----
  - this is an adaptive plan (rows marked '-' are inactive)
```

Disabling Adaptive Plans

- OPTIMIZER_ADAPTIVE_REPORTING_ONLY parameter
  - FALSE (default) – enable adaptive plan
  - TRUE – just report possibilities, but not change the plan. The default plan will always be used.
  - Information is collected on how the plan would have adapted in a non-reporting mode.
- Display execution plan in reporting only mode

```
select *
from table(dbms_xplan.display_cursor('gngtvs38t0060',
        format=>'+report'));
```

Dynamic Statistics

- New level 11 for DYNAMIC SAMPLING added
- CBO automatically decide to use dynamic statistics for a SQL statement
- Existing DBMS_STATS statistic can be improved for:
  - Situations where optimizer was making a guess (LIKE predicates, wildcards, ..)
- The dynamic sampling results will be persisted in the cache, as dynamic statistics, allowing other SQL statements to share these statistics.

```
SQL> show parameter dynamic

NAME                                 TYPE        VALUE
------------------------------------ ----------- -----
optimizer_dynamic_sampling           integer     2

SQL> alter system set optimizer_dynamic_sampling = 11;
System altered.
```

Dynamic Statistics

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
SQL_ID  4ua5q8cgupu22, child number 0
-------------------------------------
select * from sh.sales where prod_id in (113,114,115)
------------------------------------------------------------------------------------------------
| Id  | Operation           | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |       |       |   517 (100)|          |       |       |
|   1 |  PARTITION RANGE ALL|       | 40222 |  1139K|   517   (2)| 00:00:01 |     1 |    28 |
|*  2 |   TABLE ACCESS FULL | SALES | 40222 |  1139K|   517   (2)| 00:00:01 |     1 |    28 |
------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("PROD_ID"=113 OR "PROD_ID"=114 OR "PROD_ID"=115))

Note
-----
   - dynamic statistics used: dynamic sampling (level=AUTO)
```

- Actual number of rows is 40222

Automatic Reoptimization

- Known also as Automatic Cardinality Feedback in 11g
- 12c supports multiple forms of reoptimization.
  - Statistics feedback (cardinality feedback)
  - Performance feedback

```
SQL> select IS_REOPTIMIZABLE, count(*) as COUNT
  2  from v$sql group by IS_REOPTIMIZABLE;

I      COUNT
- ----------
Y        911
N       8927
```

Statistics Feedback

- CBO enables statistics feedback in the following cases:
  - tables with no statistics,
  - multiple conjunctive or disjunctive filter predicates on a table,
  - predicates containing complex operators for which the optimizer cannot accurately compute cardinality estimates.
- After execution CBO compares original cardinality estimates to the actual cardinalities from the execution.
- If estimates differ significantly from actual cardinalities, CBO stores the correct estimates for subsequent use.
- CBO also creates SQL plan directive so other SQL statements can benefit as well (see later demo).
- If the original estimates are found o.k. then the reoptimization flag is reset and no further monitoring is performed.

Statistics Feedback

```
select /*+ gather_plan_statistics */
     c.cust_first_name, c.cust_last_name,
sum(s.amount_sold)
from sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_city = 'Los Angeles'
and c.cust_state_province = 'CA'
and c.country_id = 52790
and s.time_id = '19-FEB-00'
group by  c.cust_first_name, c.cust_last_name;
```

Dependent columns – break CBO rule of independency

First Execution

```
select * from table(dbms_xplan.display_cursor(format=>'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name           | Starts | E-Rows | A-Rows |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                |      1 |        |      3 |
|   1 |  HASH GROUP BY                               |                |      1 |      1 |      3 |
|   2 |   NESTED LOOPS                               |                |      1 |        |     10 |
|   3 |    NESTED LOOPS                              |                |      1 |      1 |    601 |
|   4 |     PARTITION RANGE SINGLE                   |                |      1 |    601 |    601 |
|   5 |      TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| SALES          |      1 |    601 |    601 |
|   6 |       BITMAP CONVERSION TO ROWIDS            |                |      1 |        |    601 |
|*  7 |        BITMAP INDEX SINGLE VALUE             | SALES_TIME_BIX |      1 |        |      1 |
|*  8 |     INDEX UNIQUE SCAN                        | CUSTOMERS_PK   |    601 |      1 |    601 |
|*  9 |    TABLE ACCESS BY INDEX ROWID               | CUSTOMERS      |    601 |      1 |     10 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   7 - access("S"."TIME_ID"='19-FEB-00')
   8 - access("C"."CUST_ID"="S"."CUST_ID")
   9 - filter(("C"."CUST_CITY"='Los Angeles' AND "C"."CUST_STATE_PROVINCE"='CA' AND
              "C"."COUNTRY_ID"=52790))
```

Second Execution

```
select * from table(dbms_xplan.display_cursor(format=>'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name            | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                 |      1 |        |      3 |
|   1 |  HASH GROUP BY                               |                 |      1 |      3 |      3 |
|   2 |   NESTED LOOPS                               |                 |      1 |        |     10 |
|   3 |    NESTED LOOPS                              |                 |      1 |     10 |    601 |
|   4 |     PARTITION RANGE SINGLE                   |                 |      1 |    601 |    601 |
|   5 |      TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| SALES           |      1 |    601 |    601 |
|   6 |       BITMAP CONVERSION TO ROWIDS            |                 |      1 |        |    601 |
|*  7 |        BITMAP INDEX SINGLE VALUE             | SALES_TIME_BIX  |      1 |        |      1 |
|*  8 |     INDEX UNIQUE SCAN                        | CUSTOMERS_PK    |    601 |      1 |    601 |
|*  9 |    TABLE ACCESS BY INDEX ROWID               | CUSTOMERS       |    601 |      1 |     10 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   7 - access("S"."TIME_ID"='19-FEB-00')
   8 - access("C"."CUST_ID"="S"."CUST_ID")
   9 - filter(("C"."CUST_CITY"='Los Angeles' AND "C"."CUST_STATE_PROVINCE"='CA' AND
              "C"."COUNTRY_ID"=52790))
Note
-----
   - statistics feedback used for this statement
```

SQL Plan Directives

- SQL plan directive is additional information that the optimizer uses to generate a more optimal execution plan.
- They are created for:
  - Data skew in join column, force using dynamic statistics
  - Query expressions rather than at a statement or object level in order to be sharable among more SQL statements
- Automatically created by CBO via Automatic Reoptimization.

See Created SQL Plan Directives

- First flush directives from shared pool to see them with the above query

```
SQL> exec dbms_spd.FLUSH_SQL_PLAN_DIRECTIVE

SQL> select to_char(d.directive_id) as dir_id, o.owner, o.object_name,
           o.subobject_name as col_name, o.object_type, d.type,d.state,d.reason
     from dba_sql_plan_directives d, dba_sql_plan_dir_objects o
     where d.directive_id = o.directive_id
     and o.owner='SH'
     order by 1,2,3,4,5;
```

DIR_ID	OWNER	OBJECT_NA	COL_NAME	OBJECT	TYPE	STATE	REASON
18037153769499555617	SH	CUSTOMERS	COUNTRY_ID	COLUMN	DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
18037153769499555617	SH	CUSTOMERS	CUST_CITY	COLUMN	DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
18037153769499555617	SH	CUSTOMERS	CUST_STATE_PROVINCE	COLUMN	DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
18037153769499555617	SH	CUSTOMERS		TABLE	DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
3240241301808666714	SH	CUSTOMERS		TABLE	DYNAMIC_SAMPLING	NEW	JOIN CARDINALITY MISESTIMATE
3240241301808666714	SH	SALES		TABLE	DYNAMIC_SAMPLING	NEW	JOIN CARDINALITY MISESTIMATE
7939480357619414784	SH	CUSTOMERS	CUST_FIRST_NAME	COLUMN	DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE
7939480357619414784	SH	CUSTOMERS	CUST_LAST_NAME	COLUMN	DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE
7939480357619414784	SH	CUSTOMERS		TABLE	DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE
7939480357619414784	SH	SALES		TABLE	DYNAMIC_SAMPLING	NEW	GROUP BY CARDINALITY MISESTIMATE

SQL Plan Directives versus Extended Statistics

```
SQL> exec dbms_stats.gather_table_stats('SH','CUSTOMERS')

SQL> select table_name, extension_name, extension
  2  from dba_stat_extensions
  3  where owner='SH'
  4  and table_name='CUSTOMERS';
```

```
TABLE_NAME      EXTENSION_NAME                 EXTENSION
--------------- ------------------------------ --------------------------------------------------
CUSTOMERS       SYS_STSMZ$C3AIHLPBROI#SKA58H_N ("CUST_CITY","CUST_STATE_PROVINCE","COUNTRY_ID")
CUSTOMERS       SYS_STS_RXGTQQIMWO0T8EDCN5TKOK ("CUST_FIRST_NAME","CUST_LAST_NAME")
```

- Extended statistics is automatically created/gathered from directives when statistics is gathered.

New Types of Histograms

- Before 12c there are two types of histograms available:
  - Frequency Histogram
  - Height-Balanced Histogram
- In 12c we got two new:
  - Top Frequency
  - Hybrid
- Frequency histogram in 12c can have more then 254 buckets (up to 2048).

Gathering Histograms

```
SQL> exec dbms_stats.gather_table_stats('SH','SALES',
          estimate_percent=>dbms_stats.AUTO_SAMPLE_SIZE,
          method_opt=>'for all columns size 254')

PL/SQL procedure successfully completed.

SQL> select column_name,num_distinct,num_nulls,histogram, notes
  2  from dba_tab_col_statistics where table_name='SALES';

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS HISTOGRAM
------------------------------ ------------ ---------- ---------------
PROD_ID                                  72          0 FREQUENCY
CUST_ID                                7059          0 HYBRID
TIME_ID                                1460          0 HYBRID
CHANNEL_ID                                4          0 FREQUENCY
PROMO_ID                                  4          0 FREQUENCY
QUANTITY_SOLD                             1          0 FREQUENCY
AMOUNT_SOLD                            3586          0 HYBRID
```

Gathering Histograms

```
SQL> exec dbms_stats.gather_table_stats('SH','SALES',
          estimate_percent=>dbms_stats.AUTO_SAMPLE_SIZE,
          method_opt=>'for all columns size 2048')

PL/SQL procedure successfully completed.

SQL> select column_name,num_distinct,num_nulls,histogram, notes
  2  from dba_tab_col_statistics where table_name='SALES';

COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS HISTOGRAM
------------------------------ ------------ ---------- ---------------
PROD_ID                                  72          0 FREQUENCY
CUST_ID                                7059          0 HYBRID
TIME_ID                                1460          0 FREQUENCY
CHANNEL_ID                                4          0 FREQUENCY
PROMO_ID                                  4          0 FREQUENCY
QUANTITY_SOLD                             1          0 FREQUENCY
AMOUNT_SOLD                            3586          0 HYBRID
```

Top-Frequency Histogram

- Frequency histogram is created when NDV <= 254
- But if a relatively small number of values occupies most of the rows (>99% rows) a frequency histogram on that small set of values is very useful.
- Un-popular values are simply ignored.
- Gathering mechanism behind is the same as for frequency histograms
- Top-Frequency Histograms are only created with AUTO_SAMPLE_SIZE

Top-Frequency Histogram

```
SQL> select ATTR,count(*) from SYSMAN.EM_CCS_PARSED_DATA group by attr order by 2;

ATTR                                 COUNT(*)

@STARTMODE                                1
domain-version                            1
SSLSessionCacheTimeout                    1
DumpIOLogLevel                            1
...
...
...
staging-mode                             86
scope                                    86
security-dd-model                        93
source-path                              93
target                                  110
action                                  113
type                                    230
value                                   281
name                                    961
text_value                             1418

261 rows selected.
```

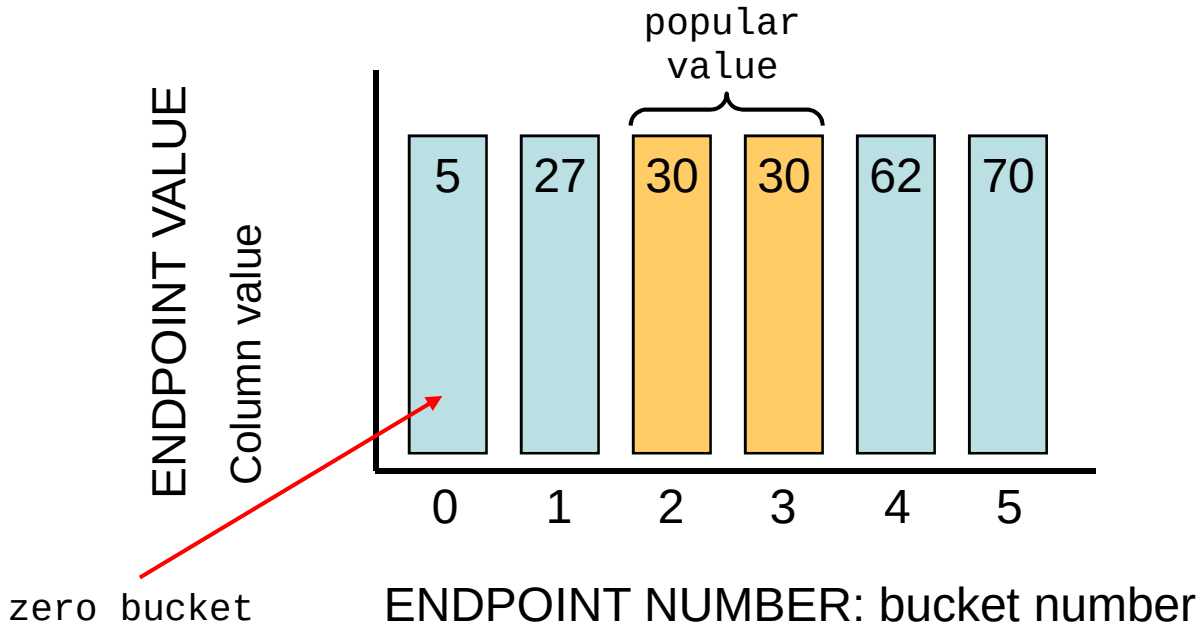
Un-popular values are ignored

Hybrid Histogram

- Like old fashion height balanced histogram when NDV > 254, but also contains elements of a frequency histogram as well as the “bucket” approach of the height-balanced.
- Store the actual frequencies of bucket endpoints in the histogram.
- No values are allowed to spill over multiple buckets
- More endpoint values can be squeezed in a histogram
- Achieves the same effect as increasing the # of buckets
- Only created with AUTO_SAMPLE_SIZE
- Frequency of endpoint values recorded in new column called ENDPOINT_REPEAT_COUNT.

Existing Height Balanced Histograms

5 buckets, 10 distinct values
(2000 rows per bucket)



Hybrid Histogram

```
SQL> select endpoint_number,endpoint_value,endpoint_repeat_count
  2  from DBA_TAB_HISTOGRAMS
  3  where table_name='SALES'
  4  and column_name='TIME_ID'
  5  order by endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE ENDPOINT_REPEAT_COUNT
--------------- -------------- --------------------
              2        2450815                    2
             24        2450829                    1
             47        2450837                    5
...
...
           5542        2452272                    5
           5543        2452273                    1
           5550        2452275                    3

254 rows selected.
```

Online Statistics Gathering

- Oracle automatically gathers optimizer statistics during the index creation.
- Same functionality now available for direct path operations:
  - create table as select (CTAS)
  - insert as select (IAS)
- Online statistics gathering does not gather histograms or index statistics due to too big overhead.
- To gather the necessary histogram and index statistics without re-gathering the base column statistics use the DBMS_STATS.GATHER_TABLE_STATS procedure with the new options parameter set to 'GATHER AUTO'.
- Prior gathering statistics fire some queries to populate COL_USAGE$ table which keep track of used columns in where clause in queries which will kick creating historam in case of skew distribution.

```
exec dbms_stats.gather_table_stats('SH','SALES3',options=>'GATHER AUTO')
```

Upgrade Process

Upgrade Strategy

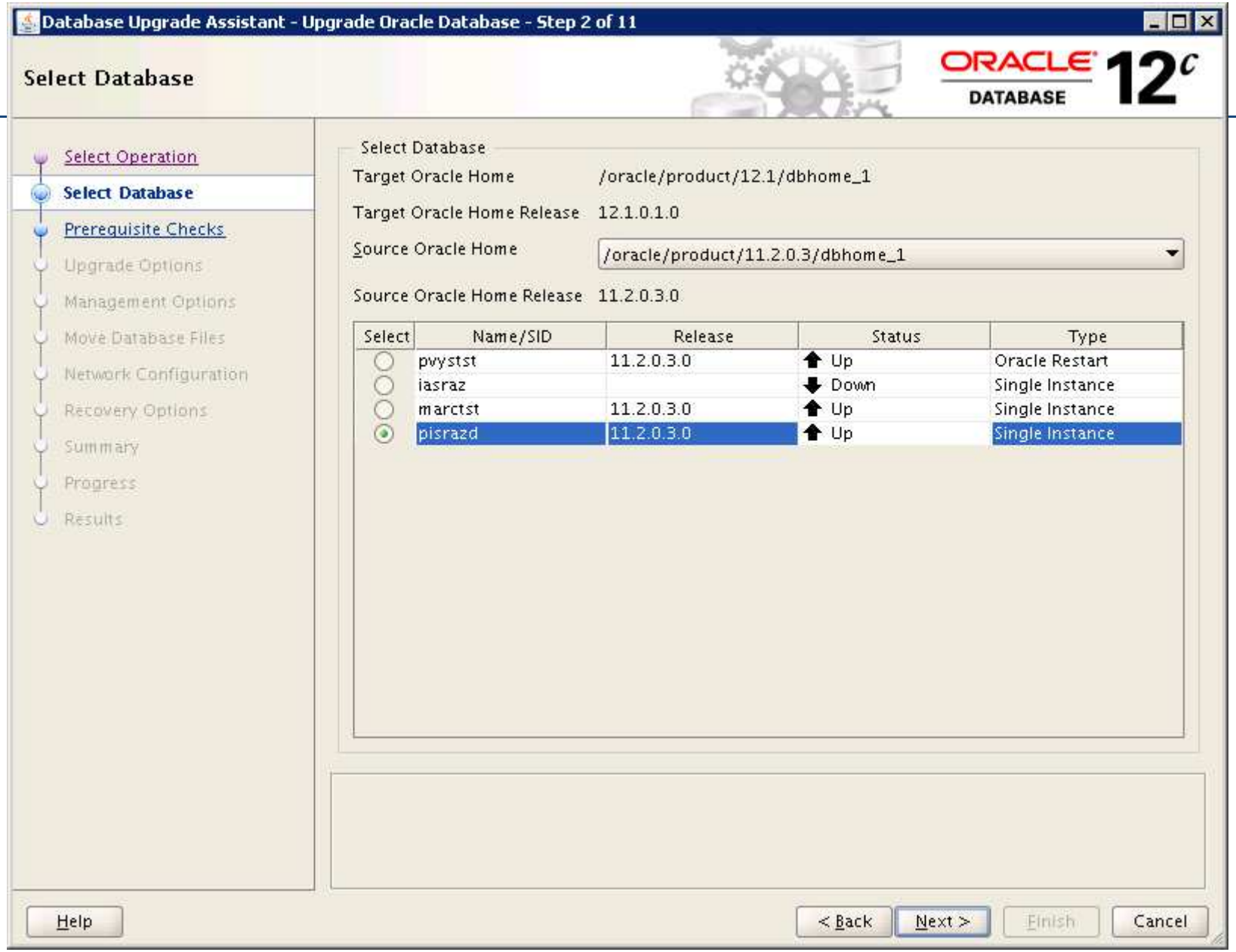
- In-place upgrade from 11g to 12.1
- Migration from 11g (or previous versions) to 12.1 multitenant
- Options for migration:
  - Transporatable tablespaces (min 10.2.0.4)
  - DataPump
  - GoldenGate
  - DbVisit

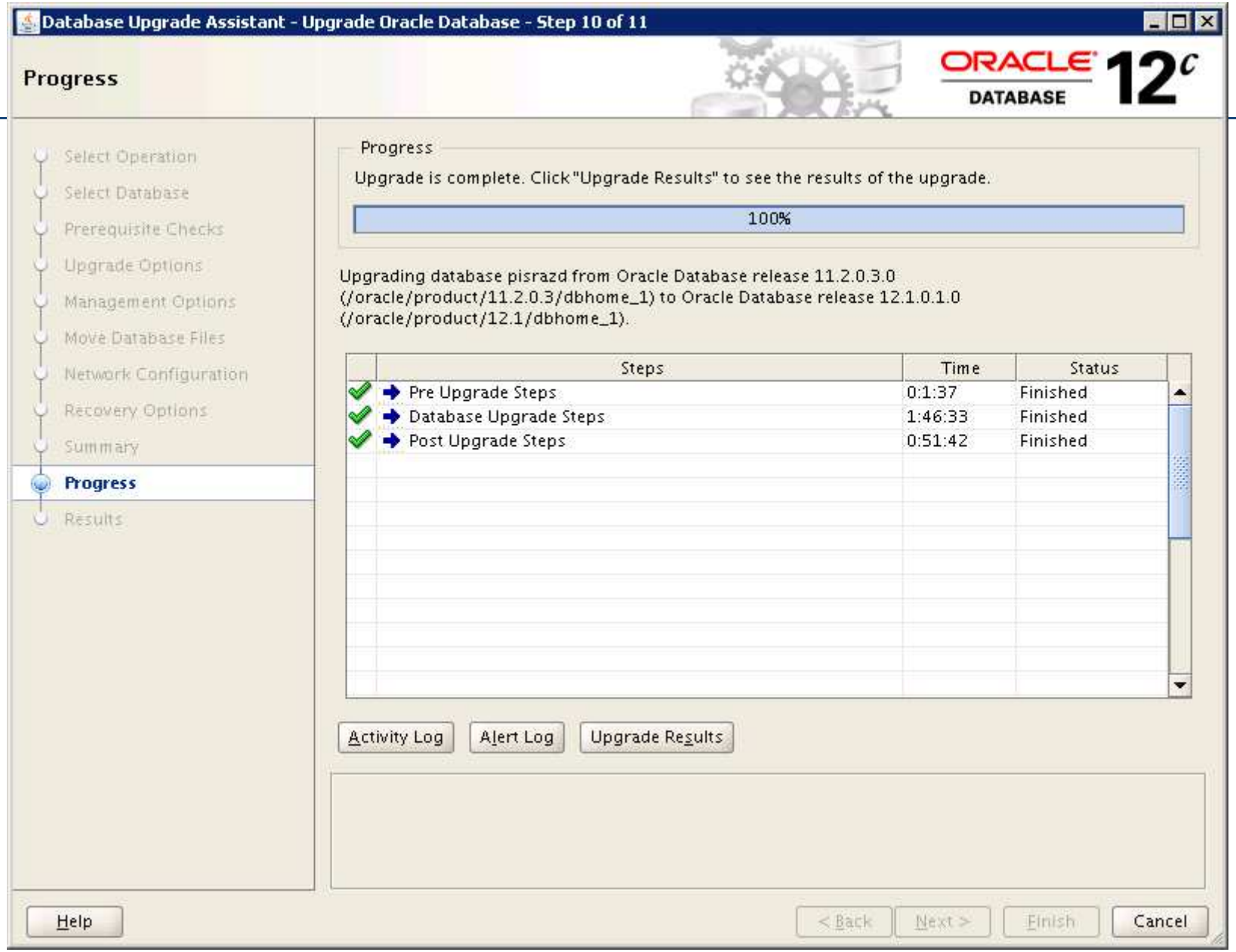
Pre upgrade tasks for in-place upgrd.

- Fix some parameters
- Empty Recyclebin
  - In my case after purging there were still 2 objects which were not displayed in dba_recyclebin, but only in sys.recyclebin$
- job_queue_processes > 0
  - If you are upgrading production database then you should probably disable all batch jobs

Problems reported by DBUA

- Inactive DBIDs found in AWR. The inactive DBIDs in AWR may need additional update after upgrade.
- Database contains schemas with objects dependent on DBMS_LDAP package. Refer to the Oracle Database Upgrade Guide for instructions to configure Network ACLs.
- Database contains INVALID objects prior to upgrade. The list of invalid SYS/SYSTEM objects was written to registry$sys_inv_objs. The list of non-SYS/SYSTEM objects was written to registry$nonsys_inv_objs unless there were over 5000. Use utluiobj.sql after the upgrade to identify any new invalid objects due to the upgrade.
- Database is using an old time zone file version. After the upgrade, patch the database time zone file version using DBMS_DST package to record latest time zone file version.


Database Upgrade Assistant - Upgrade Oracle Database - Step 2 of 11
Select Database
ORACLE DATABASE 12c
Select Operation
Select Database
Prerequisite Checks
Upgrade Options
Management Options
Move Database Files
Network Configuration
Recovery Options
Summary
Progress
Results
Select Database
Target Oracle Home /oracle/product/12.1/dbhome_1
Target Oracle Home Release 12.1.0.1.0
Source Oracle Home /oracle/product/11.2.0.3/dbhome_1
Source Oracle Home Release 11.2.0.3.0
Select Name/SID Release Status Type
pvystst 11.2.0.3.0 Up Oracle Restart
iasraz Down Single Instance
marctst 11.2.0.3.0 Up Single Instance
pisrazd 11.2.0.3.0 Up Single Instance
Help
< Back
Next >
Finish
Cancel


Database Upgrade Assistant - Upgrade Oracle Database - Step 10 of 11
Progress
ORACLE DATABASE 12c
Select Operation
Select Database
Prerequisite Checks
Upgrade Options
Management Options
Move Database Files
Network Configuration
Recovery Options
Summary
Progress
Results
Progress
Upgrade is complete. Click "Upgrade Results" to see the results of the upgrade.
100%
Upgrading database pisrazd from Oracle Database release 11.2.0.3.0 (/oracle/product/11.2.0.3/dbhome_1) to Oracle Database release 12.1.0.1.0 (/oracle/product/12.1/dbhome_1).
Steps
Time
Status
Pre Upgrade Steps
0:1:37
Finished
Database Upgrade Steps
1:46:33
Finished
Post Upgrade Steps
0:51:42
Finished
Activity Log
Alert Log
Upgrade Results
Help
< Back
Next >
Finish
Cancel

Post-Upgrade Problems

- Connectivity problems after upgrade
  - Can‘t connect to the upgraded database with old versions of sqlplus
  - ORA-28040. No matching authentication protocol
  - Related to SQLNET.ALLOWED_LOGON_VERSION
  - If changed to less than 12 (default) ASM instance crashes and does not start.

Repeated Upgrade

- If database is part of Oracle Restart you must remove it from OR to be able to perform upgrade of the same database again otherwise you don‘t see it in DBUA

```
srvctl remove database -db mydb
```

Added SQL Plan Directives

- During batch runs there were some SQL Plan Directives added

```
SQL> select trunc(created),count(*)
     from dba_sql_plan_directives
     group by trunc(created) order by 1;

TRUNC(CREATED)        COUNT(*)
------------------- ----------
08.06.2014 00:00:00         31
09.06.2014 00:00:00         15
13.06.2014 00:00:00         12
14.06.2014 00:00:00         10
```

See Created SQL Plan Directives

- First flush directives from shared pool to see them with the above query

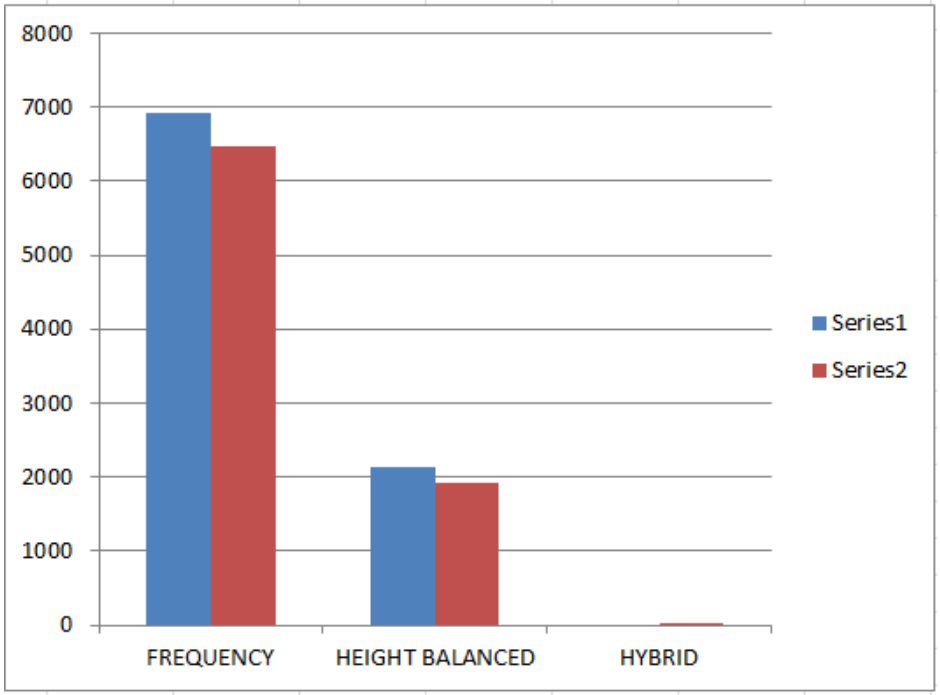
```
SQL> exec dbms_spd.FLUSH_SQL_PLAN_DIRECTIVE

SQL> select to_char(d.directive_id) as dir_id, o.owner, o.object_name,
           o.subobject_name as col_name, o.object_type, d.type,d.state,d.reason
     from dba_sql_plan_directives d, dba_sql_plan_dir_objects o
     where d.directive_id = o.directive_id
     and o.owner= 'XXX'
     order by 1,2,3,4,5;
```

DIR_ID	OBJECT_NAME	COL_NAME	OBJECT TYPE	STATE	REASON
10390878210327062227	PRV_UPORABNIKI	ID_UPO	COLUMN DYNAMIC_SAMPLING	HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
10390878210327062227	PRV_UPORABNIKI	VELJA_DO_UPO	COLUMN DYNAMIC_SAMPLING	HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
10390878210327062227	PRV_UPORABNIKI	VELJA_OD_UPO	COLUMN DYNAMIC_SAMPLING	HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
10390878210327062227	PRV_UPORABNIKI		TABLE DYNAMIC_SAMPLING	HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
10832336984164660415	MW_ZSAV_ZAMUDNE_OBREST	NAR_SIF_ZAV	COLUMN DYNAMIC_SAMPLING	MISSING_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
10832336984164660415	MW_ZSAV_ZAMUDNE_OBREST		TABLE DYNAMIC_SAMPLING	MISSING_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
11672637092389086966	STORITVE	VELJA_DO_STV	COLUMN DYNAMIC_SAMPLING	MISSING_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
11672637092389086966	STORITVE	VELJA_OD_STV	COLUMN DYNAMIC_SAMPLING	MISSING_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
11672637092389086966	STORITVE		TABLE DYNAMIC_SAMPLING	MISSING_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
11818724840910478262	CENIKI	VELJA_DO_CNK	COLUMN DYNAMIC_SAMPLING	PERMANENT	SINGLE TABLE CARDINALITY MISESTIMATE
11818724840910478262	CENIKI	VELJA_OD_CNK	COLUMN DYNAMIC_SAMPLING	PERMANENT	SINGLE TABLE CARDINALITY MISESTIMATE
11818724840910478262	CENIKI		TABLE DYNAMIC_SAMPLING	PERMANENT	SINGLE TABLE CARDINALITY MISESTIMATE
11866936856556346023	FAK_FAKTURE	STATUS_OBDELAVE	COLUMN DYNAMIC_SAMPLING	PERMANENT	SINGLE TABLE CARDINALITY MISESTIMATE

Histograms Database Level

	11g	12c
FREQUENCY	6928	6472
HEIGHT BAL	2145	1932
HYBRID		21
TOTAL	9073	8425

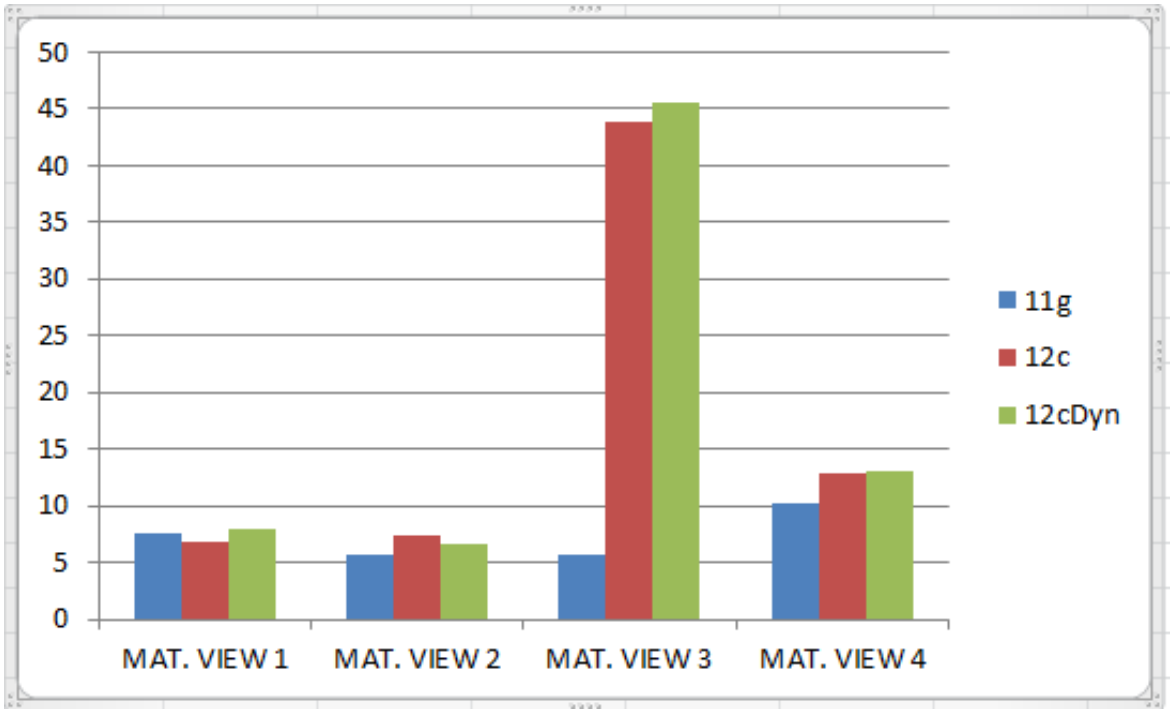


Performance Comparison

- Scenario
  - Enable flashback database
  - Run once and export SQL Plan Directives, SQL Plan Baselines
  - Flashback database
  - Import baselines and directives
  - Run again and monitor performance
  - Repeat this and change parameters (dynamic_sampling=11), drop baselines, profiles

Performance Comparison

- Majority of the most critical operations run in almost same time
- Huge degradation only observed in one MW.



Problematic SQL statement

- Observations
  - INSERT part of mat. view refresh operation
  - Was problematic at upgrade from 10.2 to 11.2 already
  - Has a valid SQL Profile which was used for optimization
  - The execution plan was prepared as adaptive and switched from NL to HJ in some of the steps

```
INSERT /*+ BYPASS_RECURSIVE_CHECK */ INTO MW_XXX SELECT *
FROM W_YYY
```

- SQL statement not monitored
  - 301 steps in execution plan – thus not monitored by default
  - **_sqlmon_max_planlines** shoud be set to higher value (default 300)

Was the Statement Reoptimized?

- Was this statement reoptimized?

```
SQL> select child_number, is_reoptimizable from
     v$sql where sql_id='057ny41c1s61y';

CHILD_NUMBER I
------------ -
           2 N
```

Cardinalities And New Features

```
SQL> select count(*) from W_XXX; //mat.view

  COUNT(*)
----------
    619014
```

- CBO Cardinality estimate was only 58
- Huge mismatch between estimated and actual number of rows

```
-------------------------------------------------------------------------------------------------------------------------------
|   Id  | Operation                                          | Name                 | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------------
|     0 | INSERT STATEMENT                                   |                      |       |       |       | 41291 (100)|          |
|     1 |  LOAD TABLE CONVENTIONAL                           |                      |       |       |       |            |          |
|     2 |   VIEW                                             | W_XXX                |    58 | 68788 |       | 41291   (3)| 00:00:02 |
```

Adaptive Plan

```
|  *  4 |     HASH JOIN                          |                    |     1 |  1057 |       |  4563   (3)| 00:00:01 |
|-    5 |      NESTED LOOPS                      |                    |       |       |       |            |          |
|-    6 |       NESTED LOOPS                     |                    |     1 |  1057 |       |  4563   (3)| 00:00:01 |
|-    7 |        STATISTICS COLLECTOR            |                    |       |       |       |            |          |
|- *  8 |         HASH JOIN                      |                    |     1 |  1018 |       |  4562   (3)| 00:00:01 |
|     9 |          NESTED LOOPS                  |                    |     1 |  1018 |       |  4562   (3)| 00:00:01 |
|-   10 |           STATISTICS COLLECTOR         |                    |       |       |       |            |          |
|  * 11 |            FILTER                      |                    |       |       |       |            |          |
|    12 |             NESTED LOOPS OUTER         |                    |     1 |   966 |       |  4559   (3)| 00:00:01 |
|    13 |              NESTED LOOPS              |                    |     1 |   669 |       |  4552   (3)| 00:00:01 |
|    14 |               NESTED LOOPS             |                    |     1 |   626 |       |  4549   (3)| 00:00:01 |
|  * 15 |                HASH JOIN               |                    |     1 |   621 |       |  4548   (3)| 00:00:01 |
|-   16 |                 NESTED LOOPS           |                    |     1 |   621 |       |  4548   (3)| 00:00:01 |
|-   17 |                  STATISTICS COLLECTOR  |                    |       |       |       |            |          |
|  * 18 |                   FILTER               |                    |       |       |       |            |          |
```

- Observations
  - Some of NL switched to HJ, some not
  - As final cardinality estimate was only 58 there is a potential that thresholds for adaptive plan were miscalculated
  - HJ operations use FTS instead of index access

Further Observations

- Conclusions
  - Some estimates for adaptive plan were completely wrong and thus switching from NL to HJ was not appropriate, because the execution plan in 11g, which performs well, uses NL.
  - Actually this feature can bring a lot of instability in the system when things go wrong.

- This SQL runs only once per month so first execution should be optimal!
  - SQL Plan should be fixed either by SQL Plan Baseline or SQL Profile
  - Plan should be made non-adaptive (probably)

Conclusions (1)

- Most of SQL statements performed as before.
- Expect some statements that might have worse execution plan.
- Most parts of the code perform better after several runs
  - 11g production ~ 27 min
  - 12c test ~ 21 min (like 10.2.0.3)

Conclusions

- Optimizer is now learning from own mistakes.
- Wrong Statistics = Wrong Plan
- Big efforts to utilize every oportunity for statistics improvement.
- New optimization paths.
- Let us see what will the real practice show.

Thank you for your interest!

Q&A