Top 10 Features in Oracle 12C for Developers and DBA's

Gary Bhandarkar
Merck & Co., Inc., Rahway, NJ USA

MERCK
Be well

Agenda

- Background
- ORACLE 12c FEATURES
- CONCLUSION

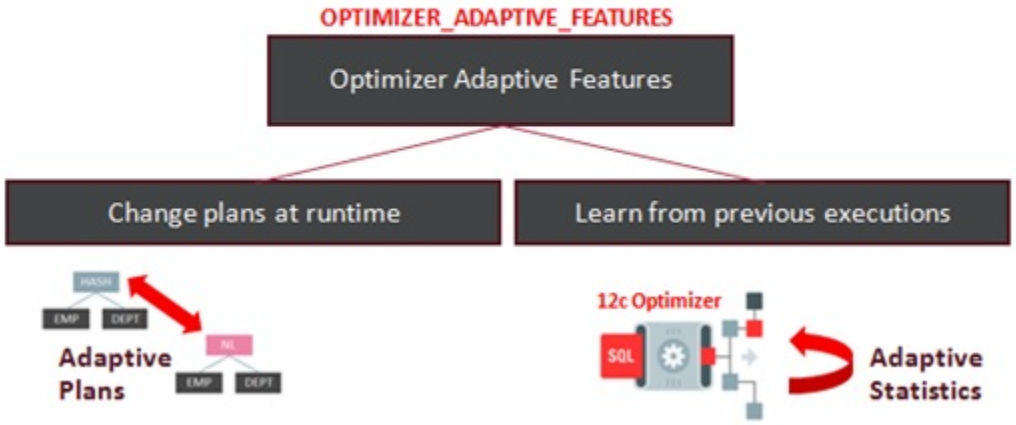
Top 10 Oracle 12c Features

- Feature 1: ADAPTIVE QUERY OPTIMIZATION

- **Upgrading from Oracle Database 11g (or an earlier release)**
- Once you've upgraded the database to Oracle Database 12c Release 2, use the *default* adaptive feature settings. To do this, simply don't include any adaptive feature parameters in your database's initialization parameter file. In other words, there's no need to set *optimizer_adaptive_plans* or *optimizer_adaptive_statistics*.

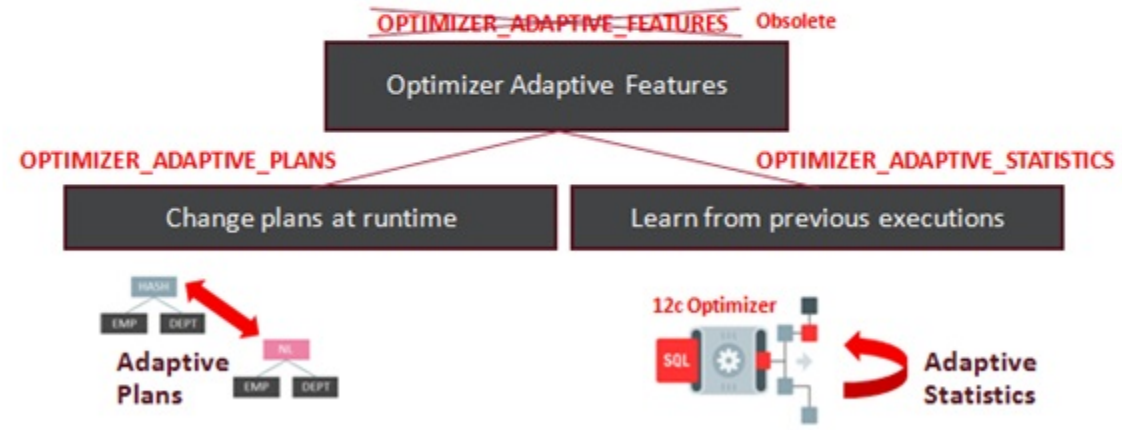
ADAPTIVE QUERY OPTIMIZATION

In Oracle Database 12c Release 1, the database parameter *optimizer_adaptive_features* controls all of the adaptive features like this:



ADAPTIVE QUERY OPTIMIZATION

In Oracle Database 12c Release 2, this parameter has been made obsolete and replaced with two new parameters that control adaptive plans and adaptive statistics separately, like this:



ADAPTIVE QUERY OPTIMIZATION

These features are enabled by default:

optimizer_adaptive_plans default **TRUE**	Description
Nested loop join/Hash join selection	The optimizer chooses between nested loops or hash joins at query runtime.
Adaptive parallel distribution method	The parallel distribution method is determined at runtime.
Star transformation bitmap pruning	Certain bitmap indexes may be removed from the SQL execution plan at runtime if selectivity is significantly poorer than the estimate.

ADAPTIVE QUERY OPTIMIZATION

These features are disabled by default:

optimizer_adaptive_statistics default **FALSE**	Description
SQL plan directives	SQL plan directives are created and used to adapt SQL execution plans.
Statistics feedback for joins	Cardinality from table joins is used to improve SQL execution plans.
Performance feedback	Improves the degree of parallelism chosen when PARALLEL_DEGREE_POLICY is set to ADAPTIVE
Adaptive dynamic sampling for parallel execution	Dynamic statistics are gathered for certain parallel queries to improve cardinality estimates. The sample size is determined automatically.

ADAPTIVE QUERY OPTIMIZATION

- **Auto Creation of Column Group Statistics**
  - **Disabled in Oracle 12c Release 2**
  - **If you need it on**
    - EXEC DBMS_STATS.SET_GLOBAL_PREFS('AUTO_STAT_EXTENSIONS','ON')

ADAPTIVE QUERY OPTIMIZATION

- explain plan for
- select /*+ gather_plan_statistics */ p.product_name from oe.order_items o,oe.product_information p
- where o.unit_price = 15
- and o.quantity > 1
- and p.product_id= o.product_id;
- select * from dba_tables where table_name='ORDER_ITEMS';
- select * from table(dbms_xplan.display());

ADAPTIVE QUERY OPTIMIZATION

```
SQL>
SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 983807676

------------------------------------------------------------------------------
| Id  | Operation                          | Name                 |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                      |
|   1 |  NESTED LOOPS                      |                      |
|   2 |   NESTED LOOPS                     |                      |
|*  3 |    TABLE ACCESS FULL               | ORDER_ITEMS2         |
|*  4 |    INDEX UNIQUE SCAN               | PRODUCT_INFORMATION_PK |
|   5 |   TABLE ACCESS BY INDEX ROWID      | PRODUCT_INFORMATION  |
------------------------------------------------------------------------------

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

   3 - filter("O"."UNIT_PRICE"=15 AND "O"."QUANTITY">1)
   4 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")

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

ADAPTIVE QUERY OPTIMIZATION

- select /*+ gather_plan_statistics */ p.product_name from oe.order_items o,oe.product_information p
- where o.unit_price = 15
- and o.quantity > 1
- and p.product_id= o.product_id;
- select * from table(dbms_xplan.display_cursor());

ADAPTIVE QUERY OPTIMIZATION

```
Plan hash value: 1553478007

-------------------------------------------------------------------------------------
| Id  | Operation          | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                     |       |       |     8 (100)|          |
|*  1 |  HASH JOIN         |                     |    13 |   416 |     8   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| ORDER_ITEMS         |    13 |   156 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| PRODUCT_INFORMATION |   288 |  5760 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")
   2 - filter(("O"."UNIT_PRICE"=15 AND "O"."QUANTITY">1))

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

ADAPTIVE QUERY OPTIMIZATION

- 12.1 –OPTIMIZER_ADAPTIVE_FEATURES
  - Enables or disables adaptive query optimization features
  - Adaptive plans
  - SQL plan directives
  - Automatic reoptimization
- – It isn't the case in 12.1.0.1 (bug 16824474)

  - The default value is TRUE
  - OPTIMIZER_DYNAMIC_SAMPLING controls dynamic statistics

ADAPTIVE QUERY OPTIMIZATION

- 12.2 OPTIMIZER_ADAPTIVE_PLANS
  - Enables or disables adaptive plans
  - The default value is TRUE

ADAPTIVE QUERY OPTIMIZATION

- 12.2 OPTIMIZER_ADAPTIVE_STATISTICS
  - Enables or disables adaptive statistics
  - SQL plan directives
- – Their creation is always enabled, only their use is managed
  - Automatic reoptimization
- – Statistics feedback as implemented in 11.2 is always enabled

  - The default value is FALSE
  - OPTIMIZER_DYNAMIC_SAMPLING controls dynamic statistics

ADAPTIVE QUERY OPTIMIZATION

- Backport of 12.2 Configuration to 12.1.0.2

Patch to backport the 12.2 initialization parameters to 12.1.0.2:

22652097: PROVIDE SEPARATE CONTROLS FOR ADAPTIV
AND ADAPTIVE STATISTICS FEATURES

When installed, OPTIMIZER_ADAPTIVE_FEATURES can no lor

ADAPTIVE QUERY OPTIMIZATION

Minimal Adaptability (11.2 Default)

OPTIMIZER_ADAPTIVE_PLANS = FALSE
OPTIMIZER_ADAPTIVE_STATISTICS = FALSE
AUTO_STAT_EXTENSIONS = OFF

Medium Adaptability (12.2 Default)

OPTIMIZER_ADAPTIVE_PLANS = TRUE
OPTIMIZER_ADAPTIVE_STATISTICS = FALSE
AUTO_STAT_EXTENSIONS = OFF

Significant Adaptability

OPTIMIZER_ADAPTIVE_PLANS = TRUE
OPTIMIZER_ADAPTIVE_STATISTICS = TRUE
AUTO_STAT_EXTENSIONS = OFF

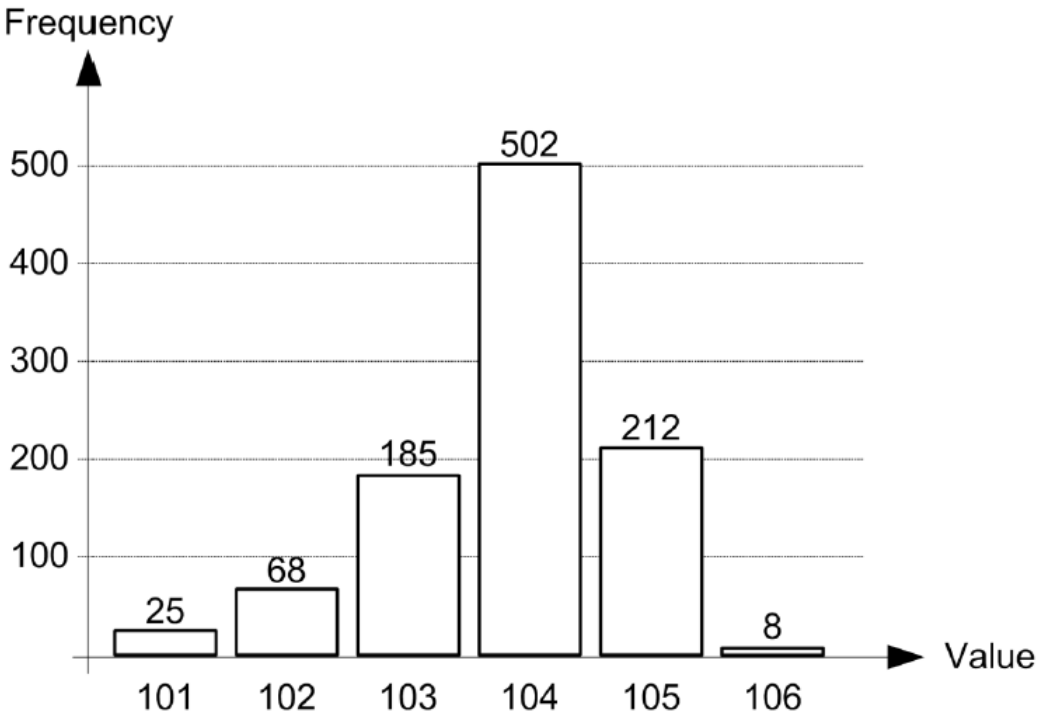
Maximum Adaptability (12.1 Default)

OPTIMIZER_ADAPTIVE_PLANS = TRUE
OPTIMIZER_ADAPTIVE_STATISTICS = TRUE
AUTO_STAT_EXTENSIONS = ON

- FEATURE 2: OBJECT STATISTICS

FREQUENCY HISTOGRAMS

- Precise estimations
- Limited number of distinct values (254)



HEIGHT BALANCED HISTOGRAMS

- Only used when a frequency histogram cannot be build because of the limited number of buckets
- Precision highly dependent on data skewing
- It is all about popular values (i.e. values associated to several buckets)

- Sometimes misleading
- When histograms are re-gathered, they can lead to instability in estimations

HISTOGRAMS 12C

- New maximum number of buckets: 2048
- More efficient way to gather histograms (AUTO_SAMPLE_SIZE only)
- New types of histograms
  - Top-frequency histograms
  - Hybrid histograms

- Top frequency histograms and hybrid histograms are supposed to replace height-balanced histograms

Top Frequency Histograms

- Similar to frequency histograms, but only the top-n values are stored
- Requirements:
- Number of distinct values larger than *n*
- Top-n values account for at least *x* percent of the rows
- –x = 100 –100 / n
- ESTIMATE_PERCENT must be set to AUTO_SAMPLE_SIZE

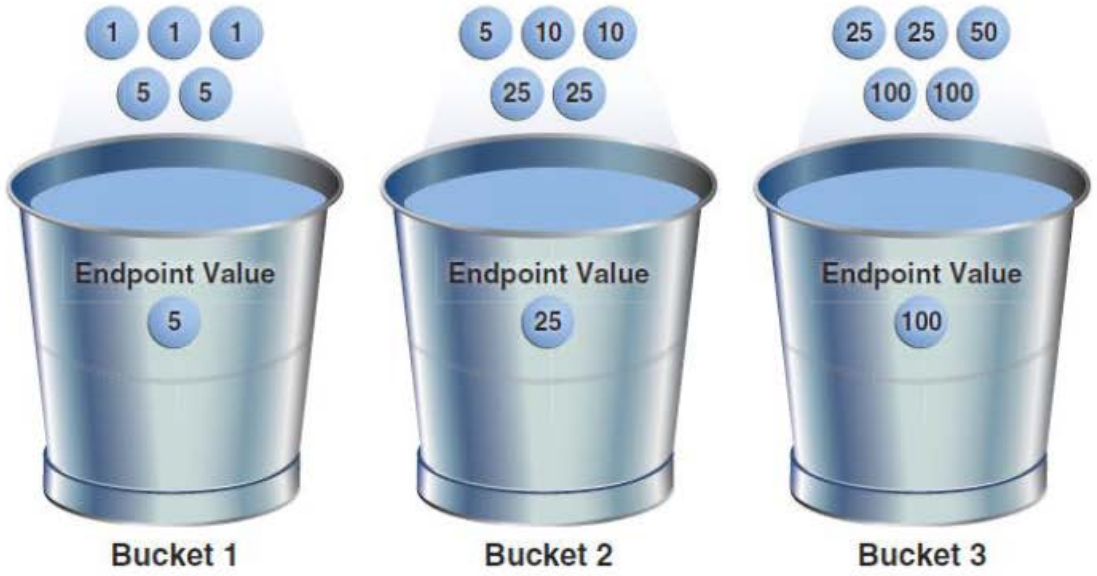
- Minimum and maximum values are always part of the histogram

HYBRID HISTOGRAMS

- Combination of height-balanced histograms with frequency histograms
- Improvements compared to height-balanced histograms
- One value is stored in a single bucket
- Frequency information added to the endpoint values to recognize almost popular values

HYBRID HISTOGRAMS

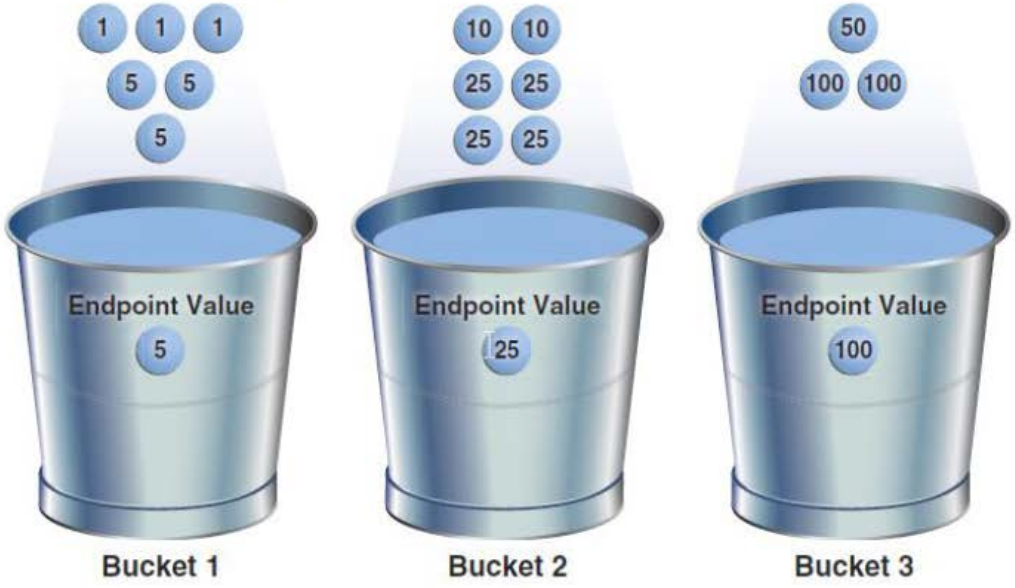
Step 1: build a height-balanced histogram



Source: Oracle Database 12c – SQL Tuning Guide, Oracle Corporation, Mai 2013

HYBRID HISTOGRAMS

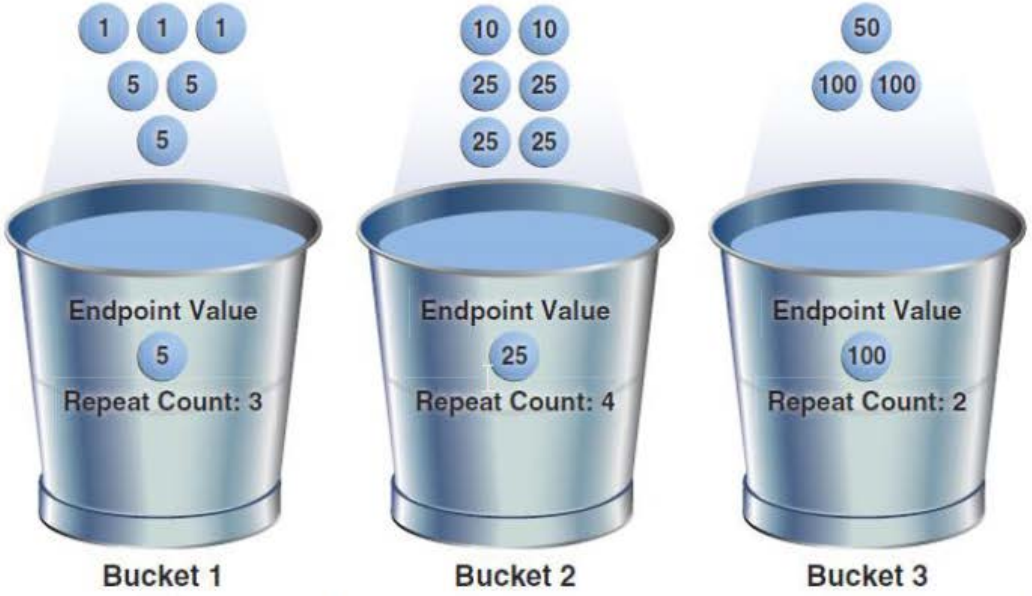
Step 2: group together endpoint values



Source: Oracle Database 12c – SQL Tuning Guide, Oracle Corporation, Mai 2013

HYBRID HISTOGRAMS

Step 3: add frequency of the endpoint values



Source: Oracle Database 12c – SQL Tuning Guide, Oracle Corporation, Mai 2013

OBJECT STATISTICS

- To simulate Oracle Database 11*g* behavior, which is necessary to create a height-based histogram, set estimate_percent to a nondefault value. If you specify a nondefault percentage, then the database creates frequency or height-balanced histograms.

- BEGIN DBMS_STATS.GATHER_TABLE_STATS (ownname => 'SH', tabname => 'COUNTRIES', method_opt => 'FOR COLUMNS COUNTRY_SUBREGION_ID SIZE 7', estimate_percent => 100);END;

- FEATURE 3: PLAN STABILITY

SQL PLAN MGMT EVOLVE ADVISOR

- 12c introduces a new advisor: SPM Evolve Advisor
- Task name = SYS_AUTO_SPM_EVOLVE_TASK
- Not covered by any option

- It runs during the maintenance window
- Its purpose is to execute an evolution for the non-accepted execution plans associated to SQL plan baselines

SQL PLAN BASELINES AUTOMATIC CAPTURE

- It's enabled when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = TRUE
- Through 12.1 a SQL plan baselines is created for every SQL statement that is executed repeatedly
- As of 12.2 several include/exclude filter types are available
- SQL text
- Parsing schema name
- Module
- Action

SQL PLAN BASELINES AUTOMATIC CAPTURE

Include only SQL statements whose text start with "SELECT"

```
dbms_spm.configure(parameter_name => 'AUTO_CAPTURE_SQL_TEXT',
                   parameter_value => 'SELECT%',
                   allow => TRUE);
```

Exclude SQL statements executed by the "Order Processing" module

```
dbms_spm.configure(parameter_name => 'AUTO_CAPTURE_MODULE',
                   parameter_value => 'Order Processing',
                   allow => FALSE);
```

SQL PLAN BASELINES MANUAL CAPTURE

Through 12.1 capture from library cache and SQL tuning sets are possible

As of 12.2 capture from AWR is also available:

```
ret := dbms_spm.load_plans_from_awr(
  begin_snap => 1234,
  end_snap => 1235,
  basic_filter => 'sql_id=''48vuyqjwpf9wg'''
);
```

COMMAND_TYPE
SQL_ID
SQL_TEXT
PLAN_HASH_VALUE
PARSING_SCHEMA_NAME
…

The capture takes place also in case the objects on which the execution plan is based doesn't exist

SQL PLAN MGMT EVOLVE ADVISOR

- To know what the SPM Evolve Advisor did:
- Find the name associated to an execution

- Generate a report about the performed activities

- **SELECT execution_name, execution_start**
- **FROM dba_advisor_executions**
- **WHERE task_name= 'SYS_AUTO_SPM_EVOLVE_TASK'**
- **ORDER BY execution_start DESC**
- **SELECT dbms_spm.report_auto_evolve_task() FROM dual**

- FEATURE 4: OPTIMIZATION TECHNIQUES

UNION[ALL]

Through 11g branches of a UNION and UNION ALL queries are processed sequentially

Only one branch at a time is executed

Individual branches can be processed in serial or parallel

CONCURRENT UNION[ALL]

- As of 12.1, with the concurrent execution branches can be executed in parallel
- and concurrently
- It is automatically used when PX is considered for at least one branch
- NO_PQ_CONCURRENT_UNION disables it
- It can be enabled for serial branches with PQ_CONCURRENT_UNION
- It also work for remote tables
- Since PX is required, it's not available in Standard Edition

- The concurrent execution isn't explicitly shown in the execution plan
- PX SELECTOR is used for serial branches
- Look for the PQ_CONCURRENT_UNION hint in the outline data section

```
-------------------------------------------------------------------------
| Id  | Operation            | Name     |    TQ  |IN-OUT| PQ Distrib|
-------------------------------------------------------------------------
|   1 |  PX COORDINATOR      |          |        |      |           |
|   2 |   PX SEND QC (RANDOM)  | :TQ10000 |  Q1,00 | P->S | QC (RAND)  |
|   3 |    VIEW              |          |  Q1,00 | PCWP |           |
|   4 |     UNION-ALL        |          |  Q1,00 | PCWP |           |
|   5 |      PX SELECTOR|          |  Q1,00 | PCWP |           |
|   6 |       TABLE ACCESS  FULL| T1       |  Q1,00 | PCWP |           |
|   7 |      PX SELECTOR|          |  Q1,00 | PCWP |           |
|   8 |       TABLE ACCESS  FULL| T2       |  Q1,00 | PCWP |           |
-------------------------------------------------------------------------
```

- The hint can’t be used to enable concurrent execution if the query optimizer doesn’t consider it
- In 12.1, because of bug 19565803, the execution involving remote tables can be inefficient
- In 12.1.0.1, because of bug 15851422, hints don’t work in case no query block is specified

- FEATURE 5: INDEXES

MULTIPLE INDEXES ON SAME COLUMNS

Through 11.2 it's *not* possible to create multiple indexes on the same set of columns

```
SQL> CREATE INDEX i_i ON t (n1);

SQL> CREATE UNIQUE INDEX i_ui ON t (n1);
CREATE UNIQUE INDEX i_ui ON t (n1)
                            *
ERROR at line 1:
ORA-01408: such column list already indexed
```

A maintenance window is required to change the uniqueness, type, or partitioning of an index

SUPPORT FOR MULTIPLE INDEXES

Only one of them can be visible at a time

They have to differ by uniqueness, type, or partitioning

```
SQL> CREATE INDEX i_i ON t (n1);

SQL> CREATE UNIQUE INDEX i_ui ON t (n1) INVISIBLE;

SQL> CREATE BITMAP INDEX i_bi ON t (n1) INVISIBLE;

SQL> CREATE INDEX i_pi ON t (n1) INVISIBLE
  2  GLOBAL PARTITION BY HASH (n1) PARTITIONS 4;
```

PARTIAL INDEXES

- For performance purposes it's sometimes *not* necessary to index all data stored in a table
- E.g. it might be enough to index only the data of the last day or week

- Through 11.2 partial indexes can be created by making partitions of local indexes unusable
- 12.1 onward provides a specific syntax for partial indexes
- Supported for partitioned tables only
- Supported for non-partitioned, local and global indexes

PARTIAL INDEXES FOR PARTITIONED TABLES

- When a table is created, indexing for (sub)partitions can be turned on or off
- The default value is specified at the table level
- The default value can be overridden at the (sub)partition level

- When an index is created, it's possible to specify whether the indexing property has to be observed or not (INDEXING PARTIAL|FULL)
- The query optimizer takes advantage of the table expansion query transformation to make sure that data is accessed in the optimal way

```
CREATE TABLE t (…)
INDEXING OFF
PARTITION BY RANGE (d) (
  PARTITION t_jan_2016 VALUES LESS THAN (…),
  PARTITION t_feb_2016 VALUES LESS THAN (…),
  …
  PARTITION t_nov_2016 VALUES LESS THAN (…),
  PARTITION t_dec_2016 VALUES LESS THAN (…) INDEXING ON
)
```

```
CREATE INDEX i ON t (d) INDEXING PARTIAL
```

```
SELECT * FROM t
WHERE d BETWEEN to_date('2016-11-30 23:00:00','yyyy-mm-dd hh24:mi:ss')
            AND to_date('2016-12-01 01:00:00','yyyy-mm-dd hh24:mi:ss')

-------------------------------------------------------------------------------
| Id  | Operation                                | Name    | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |         |       |       |
|   1 |  VIEW                                    | VW_TE_2 |       |       |
|   2 |   UNION-ALL                              |         |       |       |
|   3 |    TABLE ACCESS BY GLOBAL INDEX ROWID    | T       |    12 |    12 |
|   4 |     INDEX RANGE SCAN                     | I       |       |       |
|   5 |    PARTITION RANGE SINGLE                |         |    11 |    11 |
|   6 |     TABLE ACCESS FULL                    | T       |    11 |    11 |
-------------------------------------------------------------------------------
```

- FEATURE 6: PARTITIONING

INTERVAL REFERENCE PARTITIONING 12.1

```
CREATE TABLE p (id NUMBER CONSTRAINT p_pk PRIMARY KEY,
                pk DATE)
PARTITION BY RANGE (pk) INTERVAL (NUMTOYMINTERVAL(1, 'MONTH'))
(PARTITION p0 VALUES LESS THAN (to_date('20170101','YYYYMMDD')))
```

```
CREATE TABLE c (id NUMBER CONSTRAINT c_pk PRIMARY KEY,
                pid NUMBER NOT NULL,
                CONSTRAINT c_p_fk FOREIGN KEY (pid) REFERENCES p)
PARTITION BY REFERENCE (c_p_fk)
```

MULTI COLUMN LIST PARTITIONING 12.2

```
CREATE TABLE sales (country_iso_code VARCHAR2(2),
                    channel_id NUMBER,
                    ...)
PARTITION BY LIST (country_iso_code, channel_id)(
  PARTITION p0 VALUES (('DE',2),('DE',3),('GB',2),('GB',3)),
  PARTITION p1 VALUES (('DE',4),('DE',5),('GB',4),('GB',5)),
  PARTITION p2 VALUES (('JP',2),('JP',4)),
  PARTITION p3 VALUES (('US',1),('US',2),('US',3)),
  PARTITION p4 VALUES (DEFAULT)
)
```

Only a single default is supported

AUTOMATIC LIST PARTITIONING 12.2

```
CREATE TABLE sales (country_iso_code VARCHAR2(2),
                    ...)
PARTITION BY LIST (country_iso_code) AUTOMATIC
(PARTITION p0 VALUES ('CH'))
```

Restriction: (obviously) default partition not supported

TABLE CREATION FOR PARTITION EXCHANGE 12.2

- Creates a clone of the existing partitioned table including same columns ordering and properties

- **CREATE TABLE sales_exchange FOR EXCHANGE WITH TABLE sales;**

- Restrictions:
- Partitioned IOTs aren't supported
- Constraints (except for NOT NULL) and indexes aren't cloned

PARTITIONED EXTERNAL TABLE 12.2

```
CREATE TABLE t_ext (pkey NUMBER, content VARCHAR2(30))
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY data_pump_dir
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    FIELDS CSV WITHOUT EMBEDDED RECORD TERMINATORS
  )
)
PARTITION BY LIST (pkey) (
  PARTITION t_ext_01 VALUES (1) LOCATION ('t_ext_01.dat'),
  PARTITION t_ext_02 VALUES (2) LOCATION ('t_ext_02.dat')
)
```

- FEATURE 7: MATERIALIZED VIEWS

- Refreshes are performed with the help of another table
- A new container table is created
- The up-to-date data is inserted into it through a direct-path insert
- The new container table is switched with the old container table
- The old container table is dropped

- This method makes sure that the impact on concurrent queries accessing the materialized is minimized
- The drawback is that during the refresh, twice as much space is needed

REAL TIME MATERIALIZED VIEWS 12.2

Using materialized view for query rewrite even though not fully synchronized

Possible while using materialized view logs on queried tables

Requirement: out-of-place refresh must be possible

Not possible when using ON COMMIT refresh logic

SELECT statement needs hint FRESH_MV to take account of materialized view logs

REAL TIME MATERIALIZED VIEWS 12.2

```
CREATE MATERIALIZED VIEW mv_RT_sales
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE
ENABLE ON QUERY COMPUTATION
AS
SELECT sa.prod_id,
       dt.year_id,
       sum(sa.qty_sold) as tot_qty_sold
FROM sales sa
     INNER JOIN time_calendar_dim dt
     ON sa.date_fk = dt.date_id
GROUP BY sa.prod_id, dt.year_id
```

- FEATURE 8: ZONE MAPS

- A zone map is a redundant access structure associated to a table
- At most one zone map per table can be created

- Zone maps are intended to reduce the number of I/O during table scans
- Zone pruning
- Partition pruning

- Requirements to use zone maps:
- Oracle Partitioning option
- Exadata or Supercluster

HOW ZONE MAP BUILT AND WHAT IT CONTAINS

The target table is divided in zones

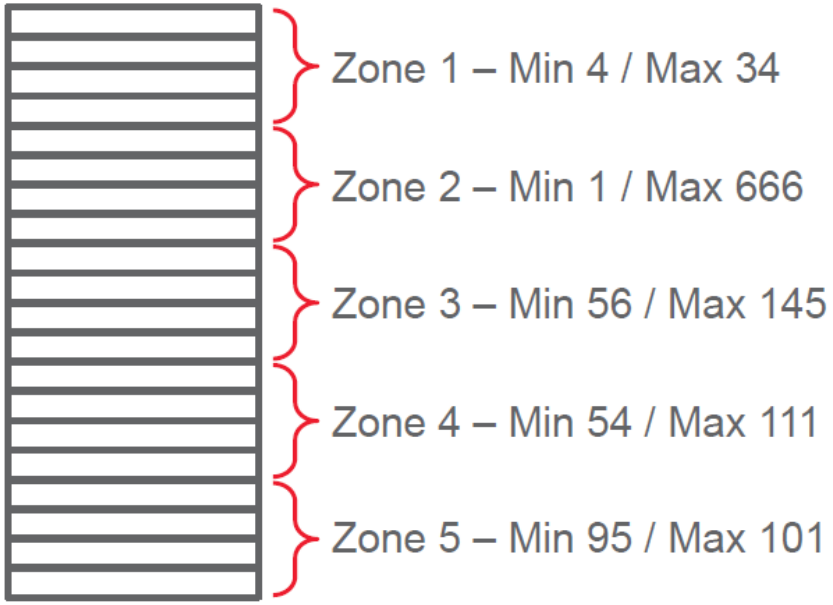
- SYS_OP_ZONE_ID(ROWID,SCALE)

A zone consists of a specific number of blocks

- The SCALE parameter controls it

$$\#blocks = 2^{scale}$$

For every zone, the zone map stores the minimum and maximum values for a number of columns



BASIC ZONE MAP

A basic zone map stores information about the columns of a single table

```
CREATE MATERIALIZED ZONEMAP p_bzm ON p (id, n1, n2)
```

A zone map is a materialized view with some particular properties

```
SQL> SELECT object_type, object_name
  2  FROM user_objects
  3  WHERE object_name LIKE '%P_BZM';

OBJECT_TYPE         OBJECT_NAME
------------------- -------------
MATERIALIZED VIEW P_BZM
TABLE               P_BZM
INDEX               I_ZMAP$_P_BZM
```

BASIC ZONE MAP EXECUTION PLAN

```
SELECT count(*) FROM p WHERE n1 = 42

-----------------------------------------------------
| Id  | Operation                                | Name |
-----------------------------------------------------
|   0 | SELECT STATEMENT                         |      |
|   1 |  SORT AGGREGATE                          |      |
|*  2 |   TABLE ACCESS STORAGE FULL WITH ZONEMAP | P    |
-----------------------------------------------------

2 - storage("N1"=42)
    filter((SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$",
           CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE
           WHEN (zm."MIN_2_N1" > :1 OR zm."MAX_2_N1" < :2) THEN 3
           ELSE 2 END END FROM "CHRIS"."P_ZM" zm
           WHERE zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$"',
           SYS_OP_ZONE_ID(ROWID),42,42)<3 AND "N1"=42))
```

JOIN ZONE MAPS

A join zone map stores information about the columns of several tables

```
CREATE MATERIALIZED ZONEMAP p_jzm AS
SELECT sys_op_zone_id(p.rowid) AS zone_id$,
       min(p.n1) AS min_p_n1, max(p.n1) AS max_p_n1,
       min(p.n2) AS min_p_n2, max(p.n2) AS max_p_n2,
       min(c.n1) AS min_c_n1, max(c.n1) AS max_c_n1,
       min(c.n2) AS min_c_n2, max(c.n2) AS max_c_n2
FROM p LEFT OUTER JOIN c ON p.id = c.p_id
GROUP BY sys_op_zone_id(p.rowid)
```

JOIN ZONE MAPS EXECUTION PLAN

```
SELECT count(*) FROM p JOIN c ON p.id = c.p_id WHERE c.n2 = 42

-----------------------------------------------------------
| Id  | Operation                                 | Name  |
-----------------------------------------------------------
|   0 | SELECT STATEMENT                          |       |
|   1 |  SORT AGGREGATE                           |       |
|*  2 |   HASH JOIN                               |       |
|*  3 |    TABLE ACCESS STORAGE FULL              | C     |
|*  4 |    TABLE ACCESS STORAGE FULL WITH ZONEMAP | P     |
-----------------------------------------------------------

2 - access("P"."ID"="C"."P_ID")
3 - storage("C"."N2"=42)
    filter("C"."N2"=42)
4 - filter(SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$",
           CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE … )
```

- FEATURE 9: IN MEMORY

ROW STORE

Also known as *n-ary storage models* (NSM)

Data is stored row-by-row

Classical storage for relational databases

block

PK	COLA	COLB	COLC
PK1	VALA1	VALB1	VALC1
PK2	VALA2	VALB2	VALC2
PK3	VALA3	VALB3	VALC3
PK4	VALA4	VALB4	VALC4
PK5	VALA5	VALB5	VALC5
PK6	VALA6	VALB6	VALC6
PK7	VALA7	VALB7	VALC7

COLUMN STORE

Also known as *decomposed storage models* (DSM)

Data is stored column-by-column

- Logically they are key-value pairs

Different columns are stored in different blocks

block

ID	PK
1	PK1
2	PK2
3	PK3
4	PK4
5	PK5
6	PK6
7	PK7

block

ID	COLA
1	VALA1
2	VALA2
3	VALA3
4	VALA4
5	VALA5
6	VALA6
7	VALA7

block

ID	COLB
1	VALB1
2	VALB2
3	VALB3
4	VALB4
5	VALB5
6	VALB6
7	VALB7

block

ID	COLC
1	VALC1
2	VALC2
3	VALC3
4	VALC4
5	VALC5
6	VALC6
7	VALC7

ORACLE IM

- Simultaneously a row store and a column store
- On-disk row store
- In-memory column store

- The content of the two stores is transactionally consistent
- Transparent for applications

IM ADVISOR

With 53MB, Data Concerning SQL Statements With Performance Benefit Is Not Available

Some SQL statements may have performance benefit, but data concerning these SQL statements is not available. You may wish to use a longer statistics capture window or add a SQL tuning set. If this does not help, consult with an Oracle Database In-Memory expert.

With 53MB, All 3 Objects Recommended To Place In-Memory For Analytics Processing

Object Type	Object	Compression Type	Estimated In-Memory Size	Analytics Processing Seconds	Estimated Reduced Analytics Processing Seconds	Estimated Analytics Processing Performance Improvement Factor	Benefit / Cost Ratio (Reduced Analytics Processing / In-Memory Size)
TABLE	.W_RXI_STUDY_SITE_DX	No memory compression	1MB	129200	116280	10.0X	91 : 1
TABLE	.W_RXI_STUDY_SITE_CON_DHL	No memory compression	31MB	129201	116281	10.0X	4 : 1
TABLE	. .W_PARTY_PER_DX	No memory compression	13MB	43067	38760	10.0X	3 : 1

- FEATURE 10: APPROXIMATE QUERY PROCESSING

APPROXIMATE QUERY PROCESSING

Computing the number of distinct values can be resource intensive

In some situation performance is more important than precision

Many algorithms that *estimate* the number of distinct values have been developed

Oracle implemented one of this algorithms (HyperLogLog)

In case an estimated number of distinct values is acceptable, you can replace COUNT(DISTINCT *expr*) with APPROX_COUNT_DISTINCT(*expr*)

APPROXIMATE QUERY PROCESSING

- 100’000’000 rows stored in a 12.1.0.2 database
- 12.7 GB
- 24 columns
- Number of distinct values is 2m, where $m \in \mathbb{N}* \; m < 25$

- Test queries:
- **SELECT count(DISTINCT n_m) FROM t**
- **SELECT approx_count_distinct(n_m) FROM t**

APPROXIMATE QUERY PROCESSING

Base functions

- APPROX_MEDIAN
- APPROX_PERCENTILE

Approximate aggregate transformation and related parameters

Support in materialized views

- Including query rewrite

Helper functions

- APPROX_COUNT_DISTINCT_DETAIL
- APPROX_COUNT_DISTINCT_AGG
- TO_APPROX_COUNT_DISTINCT
- APPROX_PERCENTILE_DETAIL
- APPROX_PERCENTILE_AGG
- TO_APPROX_PERCENTILE

APPROXIMATE QUERY PROCESSING

If APPROX_FOR_COUNT_DISTINCT is set to TRUE (default is FALSE), the optimizer applies the following transformation:

COUNT(DISTINCT <expr>) → APPROX_COUNT_DISTINCT(<expr>)

APPROXIMATE QUERY PROCESSING

If APPROX_FOR_PERCENTILE is set to PERCENTILE_CONT or ALL (default is NONE), the optimizer applies the following transformation:

PERCENTILE_CONT(<expr>) WITHIN GROUP (ORDER BY <expr>)
→ APPROX_PERCENTILE(<expr>) WITHIN GROUP (ORDER BY <expr>)

APPROXIMATE QUERY PROCESSING

If APPROX_FOR_PERCENTILE is set to PERCENTILE_DISC or ALL (default is NONE), the optimizer applies the following transformation:

PERCENTILE_DISC(<expr>) WITHIN GROUP (ORDER BY <expr>)
→ APPROX_PERCENTILE(<expr>) WITHIN GROUP (ORDER BY <expr>)

APPROXIMATE QUERY PROCESSING

If APPROX_FOR_PERCENTILE is *not* set to the default value (NONE), the optimizer applies the following transformation:

MEDIAN(<expr>)
→ APPROX_PERCENTILE(0.5) WITHIN GROUP (ORDER BY <expr>)

APPROXIMATE QUERY PROCESSING

The optimizer isn't able to apply the query transformation in case the OVER clause is specified

If APPROX_FOR_AGGREGATION is set to TRUE (default is FALSE) and provided the other parameters aren't set, all query transformations are enabled

HELPER FUNCTIONS

- The result of APPROX_COUNT_DISTINCT, APPROX_MEDIAN and
- APPROX_PERCENTILE can be stored in tables and materialized views
- However, further aggregations are not possible!
- The helper function has to be used to store approximations that has to be
- post-processed
- APPROX_*_DETAIL generates data that allows post-processing
- APPROX_*_AGG aggregates data generated by APPROX_*_DETAIL
- TO_APPROX_* converts in readable form the output of APPROX_*_AGG

Thank-You