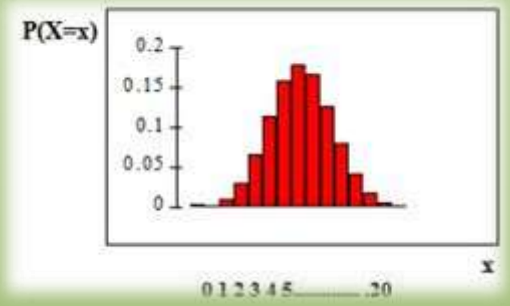
SQL That (Almost) Tunes Itself: Oracle 12c's Built-In Tuning Features



Jim Czuprynski
Zero Defect Computing, Inc.

My Credentials

ORACLE Certified Professional

ORACLE ACE Director

- 30+ years of database-centric IT experience
- Oracle DBA since 2001
- Oracle 9i, 10g, 11g OCP and Oracle ACE Director
- > 100 articles on databasejournal.com and ioug.org
- Teach core Oracle DBA courses (Grid + RAC, Exadata, Performance Tuning, Data Guard)
- Regular speaker at Oracle OpenWorld, IOUG COLLABORATE, OUG Norway, and Hotsos
- Oracle-centric blog (*Generally, It Depends*)

Our Agenda

- Statistics: "*It's Vegas, Baby!*"
- Cardinality Feedback
- Adaptive Plans
- Automatic Re-Optimization (ARO)
- SQL Plan Directives
- Automatic Extended Statistics Gathering
- Adaptive SQL Plan Management
- SPM Evolve Advisor
- Q+A

It's Vegas, Baby!

Oracle Optimizer = Vegas Oddsmaker

Get
Execution
Plan *Right:*
Easy Street

Get Execution Plan *Wrong*: Meet "Tony the Ant"

How do we tilt the odds in the Optimizer's favor?

We teach the Optimizer to "count cards"

Adaptive Execution Plans (AEP)

Statistics Feedback

- Originally introduced as *Cardinality Feedback* in Oracle 11gR2 as part of *Adaptive Cursor Sharing*
- Captures *actual* execution statistics during *query execution*
- Compares *expected* vs. *actual* cardinality during first execution of query
- During second execution, optimizer uses *actual* execution statistics to *reparse* statement's plan
- Works best for *non-skewed* row sources with *limited volatility*

Adaptive Execution Plans (AEP)

The optimizer can now adaptively recognize and capture *multiple* potential execution sub-plans within an *existing* execution plan:

- AEP constructs *dynamic plans* automatically
- AEP dynamic *statistics collector* buffers each row set
  - If new row count exceeds prior counts during statement execution, the optimizer will choose *alternative* favored subplan (e.g. HASH JOIN instead of NESTED LOOP)
  - Otherwise, AEP will utilize the *original* sub-plan
- Largest AEP benefit: Sub-plans whose row sets contain *dramatically-skewed* data

Adaptive Execution Plans In Action

```
SELECT /*+ MONITOR DYN_1 */
   C.cust_last_name
  FROM
   ap.invoices I
  ,sh.customers C
 WHERE I.custome
   AND C.cust_ci
 ORDER BY C.cust

SELECT PLAN_TABL
  FROM TABLE(DBM
```

```
Plan hash value: 608225413

-------------------------------------------------------------------
|   Id  | Operation                | Name               | E-Rows |
-------------------------------------------------------------------
|     0 | SELECT STATEMENT         |                    |    497 |

|     1 |  SORT ORDER BY           |                    |    497 |
|-  *  2 |   HASH JOIN              |                    |    497 |
|     3 |    NESTED LOOPS          |                    |    497 |
|-    4 |     STATISTICS COLLECTOR |                    |        |
|   *  5 |      TABLE ACCESS FULL   | CUSTOMERS          |    179 |
|   *  6 |     INDEX RANGE SCAN     | INVOICES_CUST_IDX  |      3 |
|-    7 |    INDEX FULL SCAN       | INVOICES_CUST_IDX  |      3 |
-------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("I"."CUSTOMER_ID"="C"."CUST_ID")
   5 - filter("C"."CUST_CITY"='Chicago' OR "C"."CUST_CITY"='Los Angeles')
   6 - access("I"."CUSTOMER_ID"="C"."CUST_ID")

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

Automatic Re-Optimization (ARO)

Automatic Re-Optimization (ARO)

For some statements, ARO features may help to overcome intrinsic limitations of AEP *dynamic plans*:

- The optimizer discovers an inefficiency during a statement's first execution that AEP *cannot resolve* (e.g. order in which row sets are joined)
- During the next execution, the optimizer *gathers additional statistics* to improve the join order
- All subsequent executions of the same statement improve as more execution statistics and optimizer statistics are gathered

Automatic Re-Optimization: 1st Execution

```
SELECT /*+ MONITOR GATHER_PLAN_STATISTICS ARO_1 */
     INV.cust_id
    ,C.cust_last_name cust_name
    ,SUM(INV.xtd_amt) xtd_amt
    ,SUM(S.quantity_sold) qty_sold
    ,SUM(s.amount_sold) amt_sold
```

```
SQL> GRANT SELECT ANY DICTIONARY TO ap;
```

First Execution:

```
SQL_ID  d0g29dw05u1jv, child number 0
Plan hash value: 1879733942
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1	00:00:00.02	400
1	SORT GROUP BY		1	521	1	00:00:00.02	400
2	NESTED LOOPS		1	521	238	00:00:00.01	400
3	NESTED LOOPS		1	521	238	00:00:00.01	396
4	NESTED LOOPS		1	4	1	00:00:00.01	230
5	VIEW		1	4	1	00:00:00.01	227
6	HASH GROUP BY		1	4	1	00:00:00.01	227
* 7	HASH JOIN		1	1250	500	00:00:00.01	227
* 8	TABLE ACCESS FULL	INVOICES	1	500	500	00:00:00.01	7
* 9	TABLE ACCESS FULL	INVOICE_ITEMS	1	1250	500	00:00:00.01	220
10	TABLE ACCESS BY INDEX ROWID	CUSTOMERS	1	1	1	00:00:00.01	3
* 11	INDEX UNIQUE SCAN	CUSTOMERS_PK	1	1	1	00:00:00.01	2
12	PARTITION RANGE ALL		1	130	238	00:00:00.01	166
13	TABLE ACCESS BY LOCAL INDEX ROWID BATCHED	SALES	28	130	238	00:00:00.01	166
14	BITMAP CONVERSION TO ROWIDS		16		238	00:00:00.01	32
* 15	BITMAP INDEX SINGLE VALUE	SALES_CUST_BIX	16		5	00:00:00.01	32
* 16	INDEX UNIQUE SCAN	PRODUCTS_PK	238	1	238	00:00:00.01	4

```
   AND S.cust_id = C.cust_id
   AND S.prod_id = P.prod_id
 GROUP BY
    INV.cust_id
   ,C.cust_last_name
 ORDER BY
    INV.cust_id
   ,C.cust_last_name;
```

Automatic Re-Optimization: 2nd Execution

Second Execution:

```
SQL_ID  d0g29dw05u1jv, child number 1
Plan hash value: 1879733942
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1	00:00:00.01	400
1	SORT GROUP BY		1	1	1	00:00:00.01	400
2	NESTED LOOPS		1	238	238	00:00:00.01	400
3	NESTED LOOPS		1	238	238	00:00:00.01	396
4	NESTED LOOPS		1	1	1	00:00:00.01	230
5	VIEW		1	1	1	00:00:00.01	227
6	HASH GROUP BY		1	1	1	00:00:00.01	227
* 7	HASH JOIN		1	500	500	00:00:00.01	227
* 8	TABLE ACCESS FULL	INVOICES	1	500	500	00:00:00.01	7
* 9	TABLE ACCESS FULL	INVOICE_ITEMS	1	500	500	00:00:00.01	220
10	TABLE ACCESS BY INDEX ROWID	CUSTOMERS	1	1	1	00:00:00.01	3
* 11	INDEX UNIQUE SCAN	CUSTOMERS_PK	1	1	1	00:00:00.01	2
12	PARTITION RANGE ALL		1	238	238	00:00:00.01	166
13	TABLE ACCESS BY LOCAL INDEX ROWID BATCHED	SALES	28	238	238	00:00:00.01	166
14	BITMAP CONVERSION TO ROWIDS		16		238	00:00:00.01	32
* 15	BITMAP INDEX SINGLE VALUE	SALES_CUST_BIX	16		5	00:00:00.01	32
* 16	INDEX UNIQUE SCAN	PRODUCTS_PK	238	1	238	00:00:00.01	4

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

   7 - access("I"."INVOICE_ID"="LI"."INVOICE_ID")
   8 - filter("I"."TAXABLE_AMT">1000)
   9 - filter(("LI"."EXTENDED_AMT"<100 AND "LI"."TAXABLE_IND"<'Y'))
  11 - access("INV"."CUST_ID"="C"."CUST_ID")
  15 - access("S"."CUST_ID"="C"."CUST_ID")
  16 - access("S"."PROD_ID"="P"."PROD_ID")
-----
Note
-----
   - statistics feedback used for this statement
```

SQL Plan Directives (SPD)

SQL Plan Directives

Oracle 12*c*R1 offers the capability to capture and retain compilation and execution statistics within the data dictionary:

- Before, a statement's compilation and execution statistics were retained *only within the Shared Pool*
- Now these statistics will be retained *within the data dictionary instead* as SQL Plan Directives (SPDs)
- SPDs are not SQL statement specific!
  - They pertain to *best methods* to process *row sets*
  - Therefore, *multiple* future queries may benefit
- DBMS_XPLAN.DISPLAY … +NOTES tells if an SPD has been used against an existing SQL statement
- New *data dictionary views* capture SPD metadata

SPD: First Pass

```
-- Increase dynamic sampling rate for SPD creation
```

First Execution:

```
Plan hash value: 929742582

-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     26 |00:00:00.28 |    8551 |
|*  1 |  FILTER                       |               |      1 |        |     26 |00:00:00.28 |    8551 |
|   2 |   SORT GROUP BY               |               |      1 |      1 |    389 |00:00:00.28 |    8551 |
```

SPD Metadata:

```
Available SQL Plan Directives for Selected Schemas (from DBA_SQL_PLAN_DIR* Views)

Abbrev   Last                                                                    SPD
SPD ID   Used On             Owner    Object Name     Object Type  SubObject Name  State          SQL Plan Directive Reason
-------- ------------------- -------- --------------- ------------ --------------- -------------- ------------------------------------
61430811                     SH       SALES           TABLE                        NEW            SINGLE TABLE CARDINALITY MISESTIMATE
61430811                     SH       SALES           COLUMN       QUANTITY_SOLD   NEW            SINGLE TABLE CARDINALITY MISESTIMATE
61430811                     SH       SALES           COLUMN       AMOUNT_SOLD     NEW            SINGLE TABLE CARDINALITY MISESTIMATE
```

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

   1 - filter(SUM("S"."QUANTITY_SOLD")>30)
   7 - filter(("S"."AMOUNT_SOLD">1599.99 AND "S"."QUANTITY_SOLD"<3))
   8 - access("S"."PROD_ID"="P"."PROD_ID")
   9 - access("S"."CUST_ID"="C"."CUST_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=4)
```

SPD: Subsequent Passes

Second Execution:

```
Plan hash value: 929742582

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              |      1 |        |     26 |00:00:00.17 |    8551 |
|*  1 |  FILTER                        |              |      1 |        |     26 |00:00:00.17 |    8551 |
|   2 |   SORT GROUP BY                |              |      1 |      1 |    389 |00:00:00.17 |    8551 |
|   3 |    NESTED LOOPS                |              |      1 |        |   3459 |00:00:00.04 |    8551 |
|   4 |     NESTED LOOPS               |              |      1 |    325 |   3459 |00:00:00.03 |    5092 |
|   5 |      NESTED LOOPS              |              |      1 |    325 |   3459 |00:00:00.01 |    1624 |
|   6 |       PARTITION RANGE ALL      |              |      1 |    325 |   3459 |00:00:00.01 |    1620 |
|*  7 |        TABLE ACCESS FULL       | SALES        |     28 |    325 |   3459 |00:00:00.06 |    1620 |
|*  8 |       INDEX UNIQUE SCAN        | PRODUCTS_PK  |   3459 |      1 |   3459 |00:00:00.01 |       4 |
|*  9 |      INDEX UNIQUE SCAN         | CUSTOMERS_PK |   3459 |      1 |   3459 |00:00:00.02 |    3468 |
|  10 |     TABLE ACCESS BY INDEX ROWID| CUSTOMERS    |   3459 |      1 |   3459 |00:00:00.01 |    3459 |
-------------------------------------------------------------------------------------------------------------

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

   1 - filter(SUM("S"."QUANTITY_SOLD")>30)
   7 - filter(("S"."AMOUNT_SOLD">1599.99 AND "S"."QUANTITY_SOLD"<3))
   8 - access("S"."PROD_ID"="P"."PROD_ID")
   9 - access("S"."CUST_ID"="C"."CUST_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=4)
   - 1 SQL Plan Directive used for this statement
```

SPD Metadata:

Available SQL Plan Directives for Selected Schemas (from DBA_SQL_PLAN_DIR* Views)

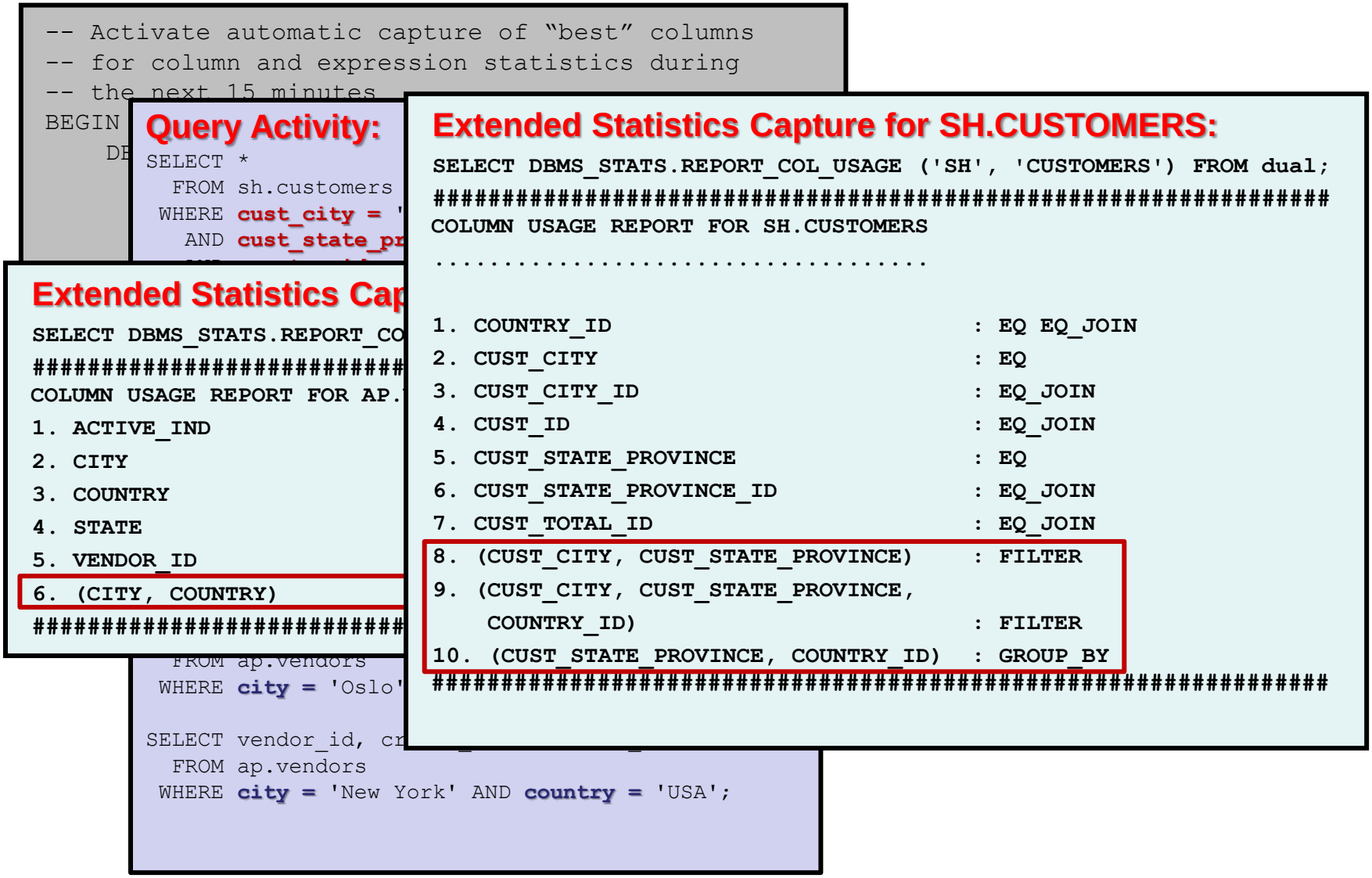
Abbrev SPD ID	Last Used On	Owner	Object Name	Object Type	SubObject Name	SPD State	SQL Plan Directive Reason
61430811	2013-11-25.14:10:43	SH	SALES	TABLE		HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
61430811	2013-11-25.14:10:43	SH	SALES	COLUMN	AMOUNT_SOLD	HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE
61430811	2013-11-25.14:10:43	SH	SALES	COLUMN	QUANTITY_SOLD	HAS_STATS	SINGLE TABLE CARDINALITY MISESTIMATE

Optimizer Statistics Enhancements

Upgrades to Optimizer Statistics Gathering

- Table statistics are now *automatically gathered* during *bulk load operations*:
  - CREATE TABLE <table_name> … AS SELECT … (CTAS)
  - INSERT INTO TABLE <table_name> … SELECT …
  - *Materialized view refreshes* via CTAS
- Statistics can now be gathered for *Global Temporary Tables*
  - Statistics are available for each *individual* session using a GTT
  - One session's statistics can also be *shared with other sessions* when little divergence between sessions
- Statistics refreshed via the regularly scheduled maintenance window are captured *concurrently* for *tables, indexes,* and *table partitions*
- Best targets for *column group* and *expression* statistics can now be *automatically identified*

Get “Best” Columns for Extended Statistics



Creating “Best” Columns’ Extended Statistics

Identify “best” columns for target of statistics:

Before:

```
Plan hash value: 4112917622
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |  1949 | 31184 |   424   (1)| 00:00:01 |
|   1 |  SORT GROUP BY     |           |  1949 | 31184 |   424   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| CUSTOMERS | 55500 |   867K|   423   (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

After:

```
Plan hash value: 4112917622
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |   145 |  2320 |   424   (1)| 00:00:01 |
|   1 |  SORT GROUP BY     |           |   145 |  2320 |   424   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| CUSTOMERS | 55500 |   867K|   423   (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

Adaptive SPM

Adaptive SQL Plan Management

Automatic Plan Evolution (APE) now available via package **DBMS_SPM**

- By default, a new automatic task runs during regular maintenance window
- *Non-accepted plans* (NAPs) are re-evaluated for automatic evolution:
  - Most *recently added plans* get precedence
  - NAPs that still perform *poorly*: Wait 30 days
  - Any NAPs that perform *better* *are* automatically enabled
- New SPM report procedure shows results of Automatic Plan Evolution

SPM Evolve Advisor

In prior releases:

- All SQL Plan evolution had to be performed *manually*
- Gathering SPM advice on whether a SQL Plan *could* evolve required DBA intervention

In Oracle 12*c*R1:

- Automatic SQL Plan Evolution tasks *included* as part of regularly-scheduled maintenance tasks
- *Manual* advice and implementation also supported via new DBMS_SPM procedures
- Warning! Tuning Pack licensing is required