Correlation, Nocorrelation and extended stats

I blogged about extended stats in my earlier blog, extended stats, and also documented that in Investigations: extended stats and multi-column correlation. I was testing extended stats further and ran in to some interesting situations.

Extended stats can be used to store correlation between columns. Correlation between two columns needs to detect at least, two properties of the column values:

1. Correlated column values
2. Uncorrelated column values

Let's explore this further.

Test case

Creating a table and populating with 10,000 rows.

```
create table y1 (a number, b number,  c number);
begin
  for i in 1..1000 loop
    for j in 1..10 loop
      insert into y1 values (j,mod(j,5), mod(j,2) );
    end loop;
  end loop;
end;
/
REM Distribution of these colum values given below.
select a, b, count(*) from y1 group by a,b order by a,b
/
         A          B   COUNT(*)
---------- ---------- ----------
         1          1       1000
         2          2       1000
         3          3       1000
         4          4       1000
         5          0       1000
         6          1       1000
         7          2       1000
         8          3       1000
         9          4       1000
        10          0       1000

10 rows selected.

REM Let's also add an index to this table
create index y1_i1 on y1 (a, b);
```

Let's collect stats on this table and populate histogram information with 254 buckets. Even though, there is no skew in data, collecting histograms is needed for proper comparison with extended stats

```
begin
   dbms_stats.gather_table_stats (
   ownname =>'rs',
   tabname=>'y1',
   estimate_percent=>null,
   cascade=>true, method_opt =>' for all columns size 254');
end;
/
```

Case #1 : Predicates (a=1 and b=1) with an index

Let us review cardinality information calculated by CBO for predicates a=1 and b=1.

```
explain plan for select c from y1 where a=1 and b=1;
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |   200 |  1600 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| Y1    |   200 |  1600 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | Y1_I1 |   200 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
Predicate Information:
----------------------
2 - access("A"=1 AND "B"=1)

CBO 10053 trace lines:
=====================
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for Y1[Y1]
  Column (#1):
    NewDensity:0.050000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:10,
NDV:10
  Column (#2):
    NewDensity:0.100000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:5,
NDV:5
   ColGroup (#1, Index) Y1_I1 ----------------------------------------------(1)
    Col#: 1 2    CorStregth: 5.00 -----------------------------------------(2)
   ColGroup Usage:: PredCnt: 2  Matches Full:  Partial:
  Table: Y1  Alias: Y1
    Card: Original: 10000.000000  Rounded: 200  Computed: 200.00  Non Adjusted: 200.00
  Access Path: TableScan
    Cost:  7.11  Resp: 7.11  Degree: 0
      Cost_io: 7.00  Cost_cpu: 2396429
      Resp_io: 7.00  Resp_cpu: 2396429
  ColGroup Usage:: PredCnt: 2  Matches Full:  Partial:
  ColGroup Usage:: PredCnt: 2  Matches Full:  Partial:
  Access Path: index (AllEqRange)
    Index: Y1_I1
    resc_io: 5.00  resc_cpu: 114847
  ix_sel: 0.020000  ix_sel_with_filters: 0.020000 -------------------------(3)
```

Lines #1 and #2 above indicate that correlation between these two columns can be calculated using the index Y1_I1. Line #3 shows that index selectivity is 0.02. As you will see shortly, there are striking similarities between line (#1) above and calculations based on extended statistics stored explicitly described in case #3 below.

In earlier releases, Cardinality for (a=1 and b=1) predicates are estimated with the formula (1/10)*(1/5)*10000=200 and this cardinality estimate is also matches with index based stats. Truly speaking correlation between these two columns are not captured correctly in index based selectivity calculations as there are 1000 distinct rows for predicates (a=1 and b=1).

Case #2 : Predicates (a=6 and b=6) with an index

Now, Let's review what happens in case of predicates (a=6 and b=6). There are no rows satisfying these predicates and so, a cardinality estimate closer to zero is preferred.

```
explain plan for select c from y1 where a=6 and b=6;

-------------------------------------------+-----------------------------------+
| Id  | Operation                   | Name    | Rows  | Bytes | Cost  | Time      |
-------------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT            |         |       |       |     2 |           |
| 1   |  TABLE ACCESS BY INDEX ROWID | Y1      |    50 |   400 |     2 |  00:00:01 |
| 2   |   INDEX RANGE SCAN          | Y1_I1   |    50 |       |     1 |  00:00:01 |
-------------------------------------------+-----------------------------------+
Predicate Information:
----------------------
2 - access("A"=6 AND "B"=6)

CBO 10053 trace lines:
======================
  Single Table Cardinality Estimation for Y1[Y1]
  Column (#1):
    NewDensity:0.050000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:10,
NDV:10
  Column (#2):
    NewDensity:0.100000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:5,
NDV:5
  ColGroup (#1, Index) Y1_I1 ----------------------------------------------(1)
    Col#: 1 2    CorStregth: 5.00 ----------------------------------------(2)
  ColGroup Usage:: PredCnt: 2  Matches Full:  Partial:
  Using prorated density: 0.050000 of col #2 as selectvity of out-of-range/non-existent
value pred
  Table: Y1  Alias: Y1
    Card: Original: 10000.000000  Rounded: 50  Computed: 50.00  Non Adjusted: 50.00
  Access Path: TableScan
    Cost:  7.11  Resp: 7.11  Degree: 0
      Cost_io: 7.00  Cost_cpu: 2368429
      Resp_io: 7.00  Resp_cpu: 2368429
  ColGroup Usage:: PredCnt: 2  Matches Full:  Partial:
  ColGroup Usage:: PredCnt: 2  Matches Full:  Partial:
  Using prorated density: 0.050000 of col #2 as selectvity of out-of-range/non-existent
value pred
  Access Path: index (AllEqRange)
    Index: Y1_I1
    resc_io: 2.00  resc_cpu: 34983
    ix_sel: 0.005000  ix_sel_with_filters: 0.005000 ----------------------(3)
    Cost: 2.00  Resp: 2.00  Degree: 1
  Best:: AccessPath: IndexRange
  Index: Y1_I1
         Cost: 2.00  Degree: 1  Resp: 2.00  Card: 50.00  Bytes: 0
```

From lines #1 and Line #2 above, index selectivity is used in cardinality calculations. Predicate b=6 is outside the range of values and so prorated density of 0.005 is used. Cardinality for (a=6 and b=6) predicate is 10000*ix_selectivity=10000*0.005=50. 50 is much closer to zero then 1000 and thus within an acceptable range.

Case #3: Predicates (a=1 and b=1) with extended stats

Now, let's add extended statistics to this mix.

```
REM Adding extended statistics to educate CBO about correlation between
columns a and b.

SELECT
        dbms_stats.create_extended_stats(
                ownname=>user,
                tabname => 'Y1',
```

```
                extension => '(a, b)' ) AS a_b_correlation
FROM dual;

REM Extended stats depicted as above, adds a virtual column to the
table with an internal function call.
REM Collecting stats on the table with histograms.

exec dbms_stats.gather_Table_stats( user, 'Y1', estimate_percent =>
null, method_opt => 'for all columns size 254');
```

Since we have added extended stats for these columns, the optimizer should be able to accurately estimate cardinality for (a=1 and b=1) predicates.

```
explain plan for select c from y1 where a=1 and b=1;

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |  1000 |  9000 |     7   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| Y1   |  1000 |  9000 |     7   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("A"=1 AND "B"=1)

CBO 10053 trace lines
=====================
SINGLE_TABLE_ACCESS_PATH
  Single Table Cardinality Estimation for Y1[Y1]
  Column (#1):
    NewDensity:0.050000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:10,
NDV:10
  Column (#2):
    NewDensity:0.100000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:5,
NDV:5
  Column (#4):
    NewDensity:0.050000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:10,
NDV:10 -------(1)
  ColGroup (#1, VC) SYS_STUNA$6DVXJXTP05EH56DTIR0X --------------------------------(2)
    Col#: 1 2    CorStregth: 5.00
  ColGroup Usage:: PredCnt: 2  Matches Full: #0  Partial:  Sel: 0.1000 ------------(3)
  Table: Y1  Alias: Y1
    Card: Original: 10000.000000  Rounded: 1000  Computed: 1000.00  Non Adjusted: 1000.00
```

Line 1 shows that there is a new virtual column. This virtual column was added for extended stats call we made earlier. Line #2 is strikingly similar to 10053 trace lines printed in earlier section, but instead of index name, VC (presumably Virtual Column) is used. [reprinted here from case #1: ColGroup (#1, Index) Y1_I1]. Line #3 shows that selectivity is set to 0.1. Cardinality estimate is 1000 and that estimation follows the calculation: 10000* VC selectivity=10000 * 0.1=1000. There are exactly 1000 rows with a=1 and b=1. That is an excellent estimate.

Case #4 : Predicates (a=6 and b=6) with extended stats

In theory, extended stats should be able to detect uncorrelated predicate values too. Let's see what happens if we use predicates (a=6 and b=6). Remember that, better estimate should be closer to zero.

```
explain plan for select c from y1 where a=6 and b=6;

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |   499 |  4491 |     7   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| Y1   |   499 |  4491 |     7   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("B"=6 AND "A"=6)
```

Yikes! Cardinality estimate is 499 with extended stats when there are nocorrelation between those column values. It is worse than cardinality estimates for the case #2 with just index. Why?

```
CBO 10053 trace lines:
======================

SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for Y1[Y1]
  Column (#1):
    NewDensity:0.050000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:10,
NDV:10
  Column (#2):
    NewDensity:0.100000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:5,
NDV:5
  Column (#4):
    NewDensity:0.050000, OldDensity:0.000050 BktCnt:10000, PopBktCnt:10000, PopValCnt:10,
NDV:10
  ColGroup (#1, VC) SYS_STUNA$6DVXJXTP05EH56DTIR0X
     Col#: 1 2    CorStregth: 5.00
  ColGroup Usage:: PredCnt: 2  Matches Full:   Using prorated density: 0.049909 of col #4
as selectvity of out-of-range/non-existent value pred
#0  Partial:  Sel: 0.0499 ----------------------------------------(1)
  Using prorated density: 0.050000 of col #2 as selectvity of out-of-range/non-existent
value pred
  Table: Y1  Alias: Y1
    Card: Original: 10000.000000  Rounded: 499  Computed: 499.09  Non Adjusted: 499.09
```

Line #1 indicates that a prorated density of 0.0499 is used. In essence, if there are no histogram buckets for that range then a default selectivity of 0.0499 (4.99%) used! This leads to an over estimation of cardinality.

Zodiac signs and skewed data

But, what if the column has skewed data? I was trying to create a test case and fortunately Greg Rahn has created a script to populate zodiac sign and birth date in his Script. [You can read his blog entry about extended statistics here]. I used that script to create tables, populate data etc and I will quickly jump to the point.

```
REM Every Taurus is born in may.

explain plan for select count(*) from calendar c where zodiac = 'TAURUS' and month =
'MAY';
-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |     1 |    16 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |          |     1 |    16 |            |          |
```

```
|*  2 |   TABLE ACCESS FULL| CALENDAR |    21 |   336 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------


CBO 10053 trace lines:
======================

  ColGroup (#1, VC) SYS_STUWHPY_ZSVI_W3#C$I3EUUYB4
    Col#: 2 3    CorStregth: 6.00
  ColGroup Usage:: PredCnt: 2  Matches Full: #0  Partial:  Sel: 0.0575 ------(1)
  Table: CALENDAR  Alias: C
    Card: Original: 365.000000  Rounded: 21  Computed: 21.00  Non Adjusted: 21.00
```

Above lines shows that correct selectivity of 0.0575 is used in cardinality calculations and there are exactly 21 rows in that table too. Let's see what happens in the case of uncorrelated values.

```
REM There are no rows with zodiac sign as Taurus and Jan

explain plan for select count(*) from calendar c where zodiac = 'TAURUS' and month =
'JAN';
---------------------------------------+-----------------------------------+
| Id  | Operation           | Name    | Rows  | Bytes | Cost  | Time      |
---------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT    |         |       |       |     3 |           |
| 1   |  SORT AGGREGATE     |         |     1 |    16 |       |           |
| 2   |   TABLE ACCESS FULL | CALENDAR|     4 |    64 |     3 |  00:00:01 |
---------------------------------------+-----------------------------------+
Predicate Information:
----------------------
2 - filter(("MONTH"='JAN' AND "ZODIAC"='TAURUS'))

SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for CALENDAR[C]
  Column (#3):
    NewDensity:0.039726, OldDensity:0.001370 BktCnt:365, PopBktCnt:365, PopValCnt:12,
NDV:12
  Column (#2):
    NewDensity:0.038356, OldDensity:0.001370 BktCnt:365, PopBktCnt:365, PopValCnt:12,
NDV:12
  Column (#4):
    NewDensity:0.009589, OldDensity:0.001370 BktCnt:365, PopBktCnt:365, PopValCnt:24,
NDV:24
  ColGroup (#1, VC) SYS_STUWHPY_ZSVI_W3#C$I3EUUYB4
    Col#: 2 3    CorStregth: 6.00
  ColGroup Usage:: PredCnt: 2  Matches Full:   Using density: 0.009589 of col #4 as
selectivity of unpopular value pred
#0  Partial:  Sel: 0.0096 ------------------------(1)
  Using density: 0.038356 of col #2 as selectivity of unpopular value pred
  Table: CALENDAR  Alias: C
    Card: Original: 365.000000  Rounded: 4  Computed: 3.50  Non Adjusted: 3.50
  Access Path: TableScan
    Cost:  3.01  Resp: 3.01  Degree: 0
      Cost_io: 3.00  Cost_cpu: 123907
      Resp_io: 3.00  Resp_cpu: 123907
  Best:: AccessPath: TableScan
         Cost: 3.01  Degree: 1  Resp: 3.01  Card: 3.50  Bytes: 0
```

From Line #1 above, we can see that selectivity is reduced as that virtual column value is unpopular. This results in a cardinality of 4, which is closer to the ideal estimate of zero. But, if number of rows is higher in the table, this can result in higher cardinality estimates too.

Summary

In summary, I was hoping to use extended stats for both correlated and uncorrelated values. But, that doesn't seem to work well for uncorrelated or non-existent values, as

default selectivity estimation of 0.0499 is on the higher side. This can lead to poor optimizer execution plans. But, if CBO is able to decipher that virtual column values are unpopular, then cardinality estimate seems to be better

Thus, I believe, that CBO should choose lower default selectivity for virtual columns created for extended stats (correlation).

In CBO development group's defense though, this feature is implemented as a virtual column and it is possible that there may be some other conditions that I may not have considered (or overlooked). There is also an interesting parameter _optimizer_extended_stats_usage_control with a default value of 240. I don't know what that parameter is for and unable to decipher it either (yet). Perhaps, there is a way to reduce this default selectivity improving usability of this new and valuable feature.