Multi column correlation

I have encountered this situation many times before: Cost based optimizer assumes no correlation between two columns (until 10g) and this has the effect of reducing cardinality of a row source erraneously. Consider following example.

```
create table t_vc as select  mod(n, 100) n1,  mod(n, 100) n2 , mod(n, 50) n3 , mod(n, 20) n4
from  (select level n from dual connect by level <= 10001);
```

There is a strong correlation between n1 and n2 above. N1 =N2. There is some correlation amoung other columns. Let's collect statistics with histograms on all columns.

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

```
explain plan for select count(*) from t_vc where n1=10 and n2=10;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     6 |     9   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |      |     1 |     6 |            |          |
|*  2 |   TABLE ACCESS FULL| T_VC |     1 |     6 |     9   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Notice the # of rows column in the table above. It is just 1. But, there are 100 rows returned for that SQL.

CBO estimate is simple (example without considering histograms):

# of rows~= total # of rows * (1/NDV for n1) * (1/NDV for n2)

= 10000 * (1/100) * (1/100) =1 row.

Far from the truth!

Extended stats

Oracle 11g introduces extended stats to relieve some pain. In 11g, an extended stats can be added between columns, enabling CBO to consider correlation between these column values.

```
SELECT dbms_stats.create_extended_stats(
                ownname   => user, tabname   => 'T_VC',
                extension => '(n1, n2)'   ) AS n1_n2_correlation
FROM dual;
```

```
N1_n2_correlation
-------------------------------------------------
SYS_STUBZH0IHA7K$KEBJVXO5LOHAS
```

Let's collect stats again on this table and check the SQL plan.

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

explain plan for select count(*) from t_vc where n1=10 and n2=10;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |   100 |  1200 |     9   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T_VC |   100 |  1200 |     9   (0)| 00:00:01 |
```

Estimated # of rows correctly at 100.

Under the wrap

Adding an extended stats adds a new virtual column to the table. Here is the line from sqltrace. Virtual column name is cryptic and seems to have been derived from table_name, column name combinations.

```
alter table "CBQT"."T_VC" add (SYS_STUBZH0IHA7K$KEBJVXO5LOHAS as
(sys_op_combined_hash(n1, n2)) virtual BY USER for statistics);
```

A new deterministic hash function sys_op_combined_hash introduced here. Function returns unique value for each combination of values passed.

```
col h1 format 99999999999999999999999999
select sys_op_combined_hash  (1,1) h1 from dual;
                                                H1
                         ----------------------------
                         7026129190895635777
select sys_op_combined_hash  (1,2) h1 from dual;
                                                H1
                          ----------------------------
                          298332787864732998
```

Collecting histograms on all columns collects histograms on this virtual column also.

Trace lines

Using the above histogram, CBO seems to have been able to find that there is a strong correlation between these two columns.

```
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for T_VC[T_VC]
  Column (#1):
    NewDensity:0.005000, OldDensity:0.000050 BktCnt:10001,
PopBktCnt:10001, PopValCnt:100, NDV:100
  Column (#2):
    NewDensity:0.005000, OldDensity:0.000050 BktCnt:10001,
PopBktCnt:10001, PopValCnt:100, NDV:100
  Column (#5):
    NewDensity:0.005000, OldDensity:0.000050 BktCnt:10001,
PopBktCnt:10001, PopValCnt:100, NDV:100
  ColGroup (#1, VC) SYS_STUBZH0IHA7K$KEBJVXO5LOHAS
    Col#: 1 2    CorStregth: 100.00
  ColGroup Usage:: PredCnt: 2  Matches Full: #0  Partial:  Sel: 0.0100
  Table: T_VC  Alias: T_VC
    Card: Original: 10001.000000  Rounded: 100  Computed: 100.00  Non
Adjusted: 100.00
  Access Path: TableScan
    Cost:  9.11  Resp: 9.11  Degree: 0
      Cost_io: 9.00  Cost_cpu: 2404620
      Resp_io: 9.00  Resp_cpu: 2404620
  Best:: AccessPath: TableScan
         Cost: 9.11  Degree: 1  Resp: 9.11  Card: 100.00  Bytes: 0
```

Notice the colgroup line and corStrength field. It is set to 100

# of rows~= total # of rows * (1/NDV for n1) * (1/NDV for n2)*corStrength

= 10000 * (1/100) * (1/100)*100 =100 rows.

Cardinality estimates are exactly matching with reality. Cardinality calculations are quite important for performance.

In the next section, we will discuss this further ;-)