TECHNICAL NOTE

GATHERING AGGREGATED COST-BASED OPTIMISER STATISTICS ON PARTITIONED OBJECTS IN ORACLE 10G

Prepared By David Kurtz, Go-Faster Consultancy Ltd.
Technical Note
Version 0.03
Thursday 10 May 2012
(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)
File: Partition.Statistics.10g.v1.00.docx, 10 May 2012

Contents

Introduction

This document discusses an alternative approach for collecting statistics on partitioned objects in Oracle 10g.[1]

The objective is to maintain statistics on partitioned objects that are at least as good as the statistics generated by DBMS_STATS without further intervention, but taking less time and using fewer resources. This method has been piloted on the two largest result tables in PeopleSoft Global Payroll (PS_GP_RSLT_ACUM and PS_GP_RSLT_PIN) which have both been partitioned and sub-partitioned[2], but it could, and probably should, be extended to other partitioned tables.

Instead of collecting statistics on logical segments (the table in a partitioned table does not physically exist as a single segment, nor do the partitions in a sub-partitioned table). Oracle calls these Global Statistics. Instead, we will only collect statistics on the partitions or sub-partitions that physically exist and will let Oracle calculate the statistics on the logical segments by aggregating the statistics on the physical segments.

This works fairly well in many cases, but some column statistics are not always calculated correctly. Specifically the number of distinct values (NDV) and the density (which is the reciprocal of NDV on a column without histograms[3]). Where this value is not accurate it can be corrected after the statistics are generated.

---

[1]This approach is unnecessary in 11g because of various changes to the DBMS_STATS package including the introduction of synopses in incremental statistics – see http://structureddata.org/2008/07/16/oracle-11g-incremental-global-statistics-on-partitioned-tables/

[2] Oracle usually uses the term 'composite partitioning' to mean that a table is partitioned and sub-partitioned.

[3] NB: Aggregated statistics do not include histograms even if histograms exist on the statistics gathered on the physical segments.

Acknowledgements

Most of the ideas for this document come from Doug Burns' series of blog posts Statistics on Partitioned Tables at http://oracledoug.com/serendipity/index.php?/archives/1590-Statistics-on-Partitioned-Tables-Contents.html.

See also

- How to Display HIGH_VALUE/LOW_VALUE Columns from USER_TAB_COL_STATISTICS by Greg Rahn at http://structureddata.org/2007/10/16/how-to-display-high_valuelow_value-columns-from-user_tab_col_statistics/.
- Tuning With Statistics By Wolfgang Brietling at http://www.centrexcc.com/SQL%20Tuning%20with%20Statistics.ppt.pdf.
- Exchange partition and aggregated statistics - 10g, 11g documentation bugs by Randolf Geist at http://oracle-randolf.blogspot.com/2008/04/exchange-partition-and-aggregated.html.

My thanks to Doug for also reviewing this document. However, any mistakes or misunderstandings remain entirely my own!

Collecting Statistics

This document demonstrates the proposed technique with a practical example[4]. All the necessary scripts are included so that the test can be reproduced.

1. I'll start by creating a subset of PS_GP_RSLT_ACUM called GFC_GP_RSLT_ACUM. This will have only three of the 32 range partitions, each of which will only have 3 list partitions[5]

```
DROP TABLE sysadm.gfc_gp_rslt_acum
/
CREATE TABLE sysadm.gfc_gp_rslt_acum
(emplid VARCHAR2(11) NOT NULL
,cal_run_id VARCHAR2(18) NOT NULL
,empl_rcd SMALLINT NOT NULL
,gp_paygroup VARCHAR2(10) NOT NULL
,cal_id VARCHAR2(18) NOT NULL
,orig_cal_run_id VARCHAR2(18) NOT NULL
,rslt_seg_num SMALLINT NOT NULL
,pin_num INTEGER NOT NULL
,empl_rcd_acum SMALLINT NOT NULL
,acm_from_dt DATE
,acm_thru_dt DATE
,slice_bgn_dt DATE
,slice_end_dt DATE
,seq_num8 INTEGER NOT NULL
,user_key1 VARCHAR2(25) NOT NULL
,user_key2 VARCHAR2(25) NOT NULL
,user_key3 VARCHAR2(25) NOT NULL
,user_key4 VARCHAR2(25) NOT NULL
,user_key5 VARCHAR2(25) NOT NULL
,user_key6 VARCHAR2(25) NOT NULL
,country VARCHAR2(3) NOT NULL
,acm_type VARCHAR2(1) NOT NULL
,acm_prd_optn VARCHAR2(1) NOT NULL
,calc_rslt_val DECIMAL(18,6) NOT NULL
,calc_val DECIMAL(18,6) NOT NULL
,user_adj_val DECIMAL(18,6) NOT NULL
,pin_parent_num INTEGER NOT NULL
,corr_rto_ind VARCHAR2(1) NOT NULL
,valid_in_seg_ind VARCHAR2(1) NOT NULL
,called_in_seg_ind VARCHAR2(1) NOT NULL)
TABLESPACE GPAPP PCTUSED 95 PCTFREE 1
PARTITION BY RANGE(EMPLID)
SUBPARTITION BY LIST (CAL_RUN_ID)
SUBPARTITION TEMPLATE
(SUBPARTITION x2010M08 VALUES
('2010UM08','2010UW29','2010UW30','2010UW31','2010UW32','2010AM08','2010AW29','2010AW30','2010AW31','2010AW32')
```

[4] The data values in this example have been obfuscated to maintain client confidentiality.

[5] There are actually 53 list partitions in the production data that I based this on.

```
 TABLESPACE GP2010M08TAB
,SUBPARTITION x2010M09 VALUES
('2010UM09','2010UW33','2010UW34','2010UW35','2010UW36','2010AM09','2010AW33','2010AW34','2010AW35','2010AW36')
 TABLESPACE GP2010M09TAB
,SUBPARTITION x2010M10 VALUES
('2010UM10','2010UW37','2010UW38','2010UW39','2010UW40','2010AM10','2010AW37','2010AW38','2010AW39','2010AW40')
TABLESPACE GP2010M10TAB)
(PARTITION gp_rslt_acum_001 VALUES LESS THAN ('12942100') TABLESPACE GPSTRM01TAB
,PARTITION gp_rslt_acum_007 VALUES LESS THAN ('30331100') TABLESPACE GPSTRM07TAB
,PARTITION gp_rslt_acum_032 VALUES LESS THAN (MAXVALUE)   TABLESPACE GPSTRM32TAB)
/
truncate table gfc_gp_rslt_acum;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_001_2010M08);
commit;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_007_2010M08);
commit;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_032_2010M08);
commit;

INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_001_2010M09);
commit;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_007_2010M09);
commit;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_032_2010M09);
commit;

INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_001_2010M10);
commit;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_007_2010M10);
commit;
INSERT /*+APPEND NOLOGGING*/ INTO gfc_gp_rslt_acum SELECT * FROM ps_gp_rslt_acum SUBPARTITION (GP_RSLT_ACUM_032_2010M10);
commit;
```

2. Create a 'statistics table' that I have called GFC_STAT_TABLE into which we can export statistics. Note that I have compressed the primary key index that is automatically created with the table. I have created an additional index which improves the performance of some the queries that correct the statistics.

```
EXECUTE sys.dbms_stats.create_stat_table('SYSADM','GFC_STAT_TABLE');
ALTER INDEX sysadm.gfc_stat_table REBUILD COMPRESS PARALLEL;
CREATE INDEX sysadm.gfc_stat_table2 ON sysadm.gfc_stat_table (type, c5, c1, c2, c3, c4) COMPRESS PARALLEL;
```

3. First I will gather the statistics on my new table the way it is currently being collected with global statistics at all levels, and then export them to the table with the ID of 'ALL'.

    a. Note that I have only collected histograms on certain columns.

```
--create statistics the wrong way
begin
    sys.dbms_stats.delete_table_Stats
      (ownname => 'SYSADM'
      ,tabname => 'GFC_GP_RSLT_ACUM'
      );
    sys.dbms_stats.gather_table_Stats
      (ownname => 'SYSADM'
      ,tabname => 'GFC_GP_RSLT_ACUM'
      ,cascade => TRUE
      ,granularity => 'ALL'
      ,method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
      );
    DELETE FROM sysadm.gfc_stat_table WHERE statid = 'ALL';
    sys.dbms_stats.export_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'GFC_GP_RSLT_ACUM'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'ALL'
      ,cascade=>TRUE
      );
end;
/
```

4. Next I will clear all the statistics, at all levels, on the table and then only gather statistics as I propose we should be doing, i.e. at sub-partition level. I will also export them to the table with the ID of 'SUBPART'.

```
--create statistics the right way
begin
    sys.dbms_stats.delete_table_Stats
      (ownname => 'SYSADM'
      ,tabname => 'GFC_GP_RSLT_ACUM'
      );
    sys.dbms_stats.gather_table_Stats
      (ownname => 'SYSADM'
      ,tabname => 'GFC_GP_RSLT_ACUM'
      ,cascade => TRUE
      ,granularity => 'SUBPARTITION'
      ,method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
      );
    DELETE FROM sysadm.gfc_stat_table WHERE statid = 'SUBPART';
    sys.dbms_stats.export_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'GFC_GP_RSLT_ACUM'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'SUBPART'
      ,cascade=>TRUE
      );
end;
/
```

Now, I can compare the two sets of statistics.

Collected Statistics

5. Lets take a look at the stats we collected and exported using the following query

```
set lines 140
break on segment skip 1 on statid skip 1
column seq format 999
column statid format a8
column segment format a25
column c1 format a18 Heading 'C1:Table'
column c2 format a16 heading 'C2:Partition'
column c3 format a25 heading 'C3:Subpartition'
column c4 format a20 heading 'C4:Column'
column n1 format 999,999 heading 'NDV'
column n2 heading 'Density'
column r1 format a26 heading 'High Value'
column r2 format a26 heading 'Low Value'
clear screen

select statid, /*c2, c4, */
  COALESCE(s.c3, s.c2, s.c1) segment
, row_number() over (partition by statid, c1, c2, c3, c4, c5 order by n10) as seq
, s.n1, 1/s.n2, s.n3
, s.n10
, s.n11
, agg_stats.display_raw(s.r1, c.data_type) r1
, agg_stats.display_raw(s.r2, c.data_type) r2
FROM gfc_stat_table s
        inner join all_tab_columns c
        on c.owner=s.c5
        and c.table_name = s.c1
        and c.column_name = s.c4
WHERE s.c1 = 'GFC_GP_RSLT_ACUM'
--And s.c2 = 'GP_RSLT_ACUM_001'
and s.c5 = 'SYSADM'
--and s.statid = 'SUBPART'
and s.c4 = 'EMPLID'
order by c4, statid, c2, c3, n10
/
```

I have used footnotes to indicate to exactly what the comment refers.

a. Column: EMPLID.

STATID	SEGMENT	SEQ	NDV	1/S.N2	N3	High Value	Low Value
ALL	GP_RSLT_ACUM_001_X2010M08	1	3,590	3590	2808	10000100	12942000
	GP_RSLT_ACUM_001_X2010M09	1	3,529	3529	2801	10001000	12942000
	GP_RSLT_ACUM_001_X2010M10	1	3,610	3610	2824	10000100	12942000
	GP_RSLT_ACUM_001	1	3,645[6]	3645	2820	10000100	12941500[7]
	GP_RSLT_ACUM_007_X2010M08	1	3,326	3326	2658	27522500	30331000
	GP_RSLT_ACUM_007_X2010M09	1	3,183	3183	2443	27522100	30330000
	GP_RSLT_ACUM_007_X2010M10	1	3,272	3272	2620	27522100	30330300
	GP_RSLT_ACUM_007	1	3,293[8]	3293	2541	27522100	30330300
	GP_RSLT_ACUM_032_X2010M08	1	3,136	3136	2498	97300500	99999000
	GP_RSLT_ACUM_032_X2010M09	1	3,102	3102	2586	97300500	99999000
	GP_RSLT_ACUM_032_X2010M10	1	3,145	3145	2495	97300500	99999000
	GP_RSLT_ACUM_032	1	3,257	3257	2656	97300500	99999000
	GFC_GP_RSLT_ACUM	1	8,394	8394	3981	10000100	99998000
SUBPART	GP_RSLT_ACUM_001_X2010M08	1	3,568	3568	2808	10001000	12942000
	GP_RSLT_ACUM_001_X2010M09	1	3,479	3479	2762	10000100	12942000
	GP_RSLT_ACUM_001_X2010M10	1	3,408	3408	2505	10000100	12939200
	GP_RSLT_ACUM_001	1	3,488[9]	3488	0	10000100	12942000
	GP_RSLT_ACUM_007_X2010M08	1	3,274	3274	2696	27522100	30331000
	GP_RSLT_ACUM_007_X2010M09	1	3,268	3268	2661	27522100	30331000
	GP_RSLT_ACUM_007_X2010M10	1	3,231	3231	2534	27522100	30330300
	GP_RSLT_ACUM_007	1	3,261	3261	0	27522100	30331000
	GP_RSLT_ACUM_032_X2010M08	1	3,286	3286	2707	97301000	99999000
	GP_RSLT_ACUM_032_X2010M09	1	2,946	2946	2336	97301000	99999000
	GP_RSLT_ACUM_032_X2010M10	1	3,175	3175	2548	97300500	99999000
	GP_RSLT_ACUM_032	1	3,138	3138	0	97300500	99999000
	GFC_GP_RSLT_ACUM	1	9,887[10]	9887	0	10000100	99999000

---

[6] The number of distinct employee IDs on the partition is greater than that on any sub-partition, and this is as expected.

[7] The high value on the global statistics on partition GP_RSLT_ACUM_001 is less than the maximum high value of the constituent sub-partitions. I think this is caused by sampling. This problem does not occur on the aggregated statistics (statid SUBPART).

[8] Perversely the number of distinct values on partition GP_RSLT_ACUM_007 is less than on sub-partition GP_RSLT_ACUM_007_X2010M08. This is clearly impossible, but is another effect of sampling.

[9] The number of distinct values on the partition is less that the largest number of distinct values of the constituent sub-partitions. This cannot be right, it must be at the least the maximum value of NDV on the sub-partitions, and may, as we see in the global statistics, be higher. This is an example of a statistics that needs to be corrected, though not by much.

[10] *dbms_stats* correctly adds up the number of distinct values in the range partitions to product the number of distinct values in the table. It knows it can do this because the low and high values do not overlap.

b. CAL_RUN_ID. This table is list sub-partitoned on CAL_RUN_ID andwe can see that while the global statistics are reasonable, both are wrong, but in different ways.

Each of the list sub-partitions within a range partition contains a distinct list of CAL_RUN_IDs. So the number of distinct CAL_RUN_IDs for the partition should be the sum of the number of distinct values on each constituent sub-partitions.

STATID	SEGMENT	SEQ	NDV	1/S.N2	N3	High Value	Low Value
ALL	GP_RSLT_ACUM_001_X2010M08	1	6	6	6	2010AM08	2010UM08
	GP_RSLT_ACUM_001_X2010M09	1	6	6	6	2010AM09	2010UM09
	GP_RSLT_ACUM_001_X2010M10	1	5	5	5	2010AW37	2010UM10
	GP_RSLT_ACUM_001	1	14[11]	14	14	2010AM09	2010UM10
	GP_RSLT_ACUM_007_X2010M08	1	6	6	6	2010AM08	2010UM08
	GP_RSLT_ACUM_007_X2010M09	1	6	6	6	2010AM09	2010UM09
	GP_RSLT_ACUM_007_X2010M10	1	5	5	5	2010AM10	2010UM10
	GP_RSLT_ACUM_007	1	14	14	14	2010AM08	2010UM10
	GP_RSLT_ACUM_032_X2010M08	1	5	5	5	2010AW29	2010UM08
	GP_RSLT_ACUM_032_X2010M09	1	6	6	6	2010AM09	2010UM09
	GP_RSLT_ACUM_032_X2010M10	1	6	6	6	2010AM10	2010UM10
	GP_RSLT_ACUM_032	1	17[12]	17	17	2010AM08	2010UM10
	GFC_GP_RSLT_ACUM	1	16	16	16	2010AM09	2010UM10
SUBPART	GP_RSLT_ACUM_001_X2010M08	1	5	5	5	2010AW29	2010UM08
	GP_RSLT_ACUM_001_X2010M09	1	6	6	6	2010AM09	2010UM09
	GP_RSLT_ACUM_001_X2010M10	1	5	5	5	2010AW37	2010UM10
	GP_RSLT_ACUM_001	1	6[13]	6	0	2010AM09	2010UM10
	GP_RSLT_ACUM_007_X2010M08	1	6	6	6	2010AM08	2010UM08
	GP_RSLT_ACUM_007_X2010M09	1	6	6	6	2010AM09	2010UM09
	GP_RSLT_ACUM_007_X2010M10	1	6	6	6	2010AM10	2010UM10
	GP_RSLT_ACUM_007	1	6	6	0	2010AM08	2010UM10
	GP_RSLT_ACUM_032_X2010M08	1	5	5	5	2010AW29	2010UM08
	GP_RSLT_ACUM_032_X2010M09	1	6	6	6	2010AM09	2010UM09
	GP_RSLT_ACUM_032_X2010M10	1	6	6	6	2010AM10	2010UM10
	GP_RSLT_ACUM_032	1	6	6	0	2010AM09	2010UM10
	GFC_GP_RSLT_ACUM	1	6	6	0	2010AM08	2010UM10

[11] In this case I can just add the NDV on each of the partitions together because this is the partitioning column. The number of distinct values should be 17 (6+6+5). However, sampling only found 14.

[12] On this partition, *dbms_stats* sampling did find 17 distinct values, so it does get it right sometimes!

[13] However, when *dbms_stats* aggregates NDV for the sub-partitions it gets it wrong – I think this is because the minimum and maximum values on the column overlap. We know this table is sub-partitioned on this column, so we can add up the number of distinct values on each column.

c. CAL_ID. This column has histograms, but I have only shown a few rows for each partition.

STATID	SEGMENT	SEQ	NDV	1/S.N2	N3	High Value	Low Value
ALL	GP_RSLT_ACUM_001_X2010M08	1	55	2695494.61	55	ABCDEF 2010AW07 UKPAY 2010M08	
	GP_RSLT_ACUM_001_X2010M08	2	55	2695494.61	55	ABCDEF 2010AW07 UKPAY 2010M08	
…							
	GP_RSLT_ACUM_001_X2010M08	54	55	2695494.61	55	ABCDEF 2010AW07 UKPAY 2010M08	
	GP_RSLT_ACUM_001_X2010M08	55	55	2695494.61	55	ABCDEF 2010AW07 UKPAY 2010M08	
	GP_RSLT_ACUM_001_X2010M09	1	106	2930950.97	106	ABCDEF 2010AW01 UKPAY 2010M09	
	GP_RSLT_ACUM_001_X2010M09	2	106	2930950.97	106	ABCDEF 2010AW01 UKPAY 2010M09	
…							
	GP_RSLT_ACUM_001_X2010M09	105	106	2930950.97	106	ABCDEF 2010AW01 UKPAY 2010M09	
	GP_RSLT_ACUM_001_X2010M09	106	106	2930950.97	106	ABCDEF 2010AW01 UKPAY 2010M09	
	GP_RSLT_ACUM_001_X2010M10	1	56	3000015.91	56	ABCDEF 2010AW02 UKPAY 2010M10	
	GP_RSLT_ACUM_001_X2010M10	2	56	3000015.91	56	ABCDEF 2010AW02 UKPAY 2010M10	
…							
	GP_RSLT_ACUM_001_X2010M10	55	56	3000015.91	56	ABCDEF 2010AW02 UKPAY 2010M10	
	GP_RSLT_ACUM_001_X2010M10	56	56	3000015.91	56	ABCDEF 2010AW02 UKPAY 2010M10	
	GP_RSLT_ACUM_001	1	99	8500645.64	99	ABCDEF 2010AW01 UKPAY 2010M10	
	GP_RSLT_ACUM_001	2	99	8500645.64	99	ABCDEF 2010AW01 UKPAY 2010M10	
…							
	GP_RSLT_ACUM_001	98	99	8500645.64	99	ABCDEF 2010AW01 UKPAY 2010M10	
	GP_RSLT_ACUM_001	99[14]	99	8500645.64	99	ABCDEF 2010AW01 UKPAY 2010M10	
	GP_RSLT_ACUM_007_X2010M08	1	50	2480438.48	50	ABCDEF 2010AW24 UKPAY 2010M08	
	GP_RSLT_ACUM_007_X2010M08	2	50	2480438.48	50	ABCDEF 2010AW24 UKPAY 2010M08	
…							
	GP_RSLT_ACUM_007_X2010M08	49	50	2480438.48	50	ABCDEF 2010AW24 UKPAY 2010M08	
	GP_RSLT_ACUM_007_X2010M08	50	50	2480438.48	50	ABCDEF 2010AW24 UKPAY 2010M08	
	GP_RSLT_ACUM_007_X2010M09	1	111	2654297.1	111	ABCDEF 2009GW52 UKPAY 2010M09	
	GP_RSLT_ACUM_007_X2010M09	2	111	2654297.1	111	ABCDEF 2009GW52 UKPAY 2010M09	
…							
	GP_RSLT_ACUM_007_X2010M09	110	111	2654297.1	111	ABCDEF 2009AW52 UKPAY 2010M09	
	GP_RSLT_ACUM_007_X2010M09	111	111	2654297.1	111	ABCDEF 2009AW52 UKPAY 2010M09	
	GP_RSLT_ACUM_007_X2010M10	1	50	2773876.76	50	ABCDEF 2010AW37 UKPAY 2010UW25	
	GP_RSLT_ACUM_007_X2010M10	2	50	2773876.76	50	ABCDEF 2010AW37 UKPAY 2010UW25	
…							
	GP_RSLT_ACUM_007_X2010M10	49	50	2773876.76	50	ABCDEF 2010AW37 UKPAY 2010UW25	
	GP_RSLT_ACUM_007_X2010M10	50	50	2773876.76	50	ABCDEF 2010AW37 UKPAY 2010UW25	
	GP_RSLT_ACUM_007	1	84	7971673.33	83	ABCDEF 2010AW01 UKPAY 2010M10	
	GP_RSLT_ACUM_007	2	84	7971673.33	83	ABCDEF 2010AW01 UKPAY 2010M10	
…							
	GP_RSLT_ACUM_007	82	84	7971673.33	83	ABCDEF 2010AW01 UKPAY 2010M10	
	GP_RSLT_ACUM_007	83	84	7971673.33	83	ABCDEF 2010AW01 UKPAY 2010M10	
	GP_RSLT_ACUM_032_X2010M08	1	75	2601649.93	75	ABCDEF 2010AW04 UKPAY 2010M08	
	GP_RSLT_ACUM_032_X2010M08	2	75	2601649.93	75	ABCDEF 2010AW04 UKPAY 2010M08	
…							
	GP_RSLT_ACUM_032_X2010M08	74	75	2601649.93	75	ABCDEF 2010AW04 UKPAY 2010M08	
	GP_RSLT_ACUM_032_X2010M08	75	75	2601649.93	75	ABCDEF 2010AW04 UKPAY 2010M08	
	GP_RSLT_ACUM_032_X2010M09	1	121	2732563.26	121	ABCDEF 2009AW47 UKPAY 2010M09	
	GP_RSLT_ACUM_032_X2010M09	2	121	2732563.26	121	ABCDEF 2009AW47 UKPAY 2010M09	

[14] Only 99 buckets while one of the sub-partitions has 106! A mistake caused by sampling.

```
…
         GP_RSLT_ACUM_032_X2010M09  120      121 2732563.26        121 ABCDEF 2009AW47 UKPAY 2010M09
         GP_RSLT_ACUM_032_X2010M09  121      121 2732563.26        121 ABCDEF 2009AW47 UKPAY 2010M09
         GP_RSLT_ACUM_032_X2010M10    1       91  2785654.2         91 ABCDEF 2010AW02 UKPAY 2010M10
         GP_RSLT_ACUM_032_X2010M10    2       91  2785654.2         91 ABCDEF 2010AW02 UKPAY 2010M10
…
         GP_RSLT_ACUM_032_X2010M10   90       91  2785654.2         91 ABCDEF 2010AW02 UKPAY 2010M10
         GP_RSLT_ACUM_032_X2010M10   91       91  2785654.2         91 ABCDEF 2010AW02 UKPAY 2010M10
         GP_RSLT_ACUM_032             1      129 8367154.18        128 ABCDEF 2009AW51 UKPAY 2010M10
         GP_RSLT_ACUM_032             2      129 8367154.18        128 ABCDEF 2009AW51 UKPAY 2010M10
…
         GP_RSLT_ACUM_032           127      129 8367154.18        128 ABCDEF 2009AW51 UKPAY 2010M10
         GP_RSLT_ACUM_032           128      129 8367154.18        128 ABCDEF 2009AW51 UKPAY 2010M10
         GFC_GP_RSLT_ACUM             1      111 25088351.9        111 ABCDEF 2010AW01 UKPAY 2010M10
         GFC_GP_RSLT_ACUM             2      111 25088351.9        111 ABCDEF 2010AW01 UKPAY 2010M10
…
         GFC_GP_RSLT_ACUM           110      111 25088351.9        111 ABCDEF 2010AW01 UKPAY 2010M10
         GFC_GP_RSLT_ACUM           111      111 25088351.9        111 ABCDEF 2010AW01 UKPAY 2010M10
```

[15]

```
STATID   SEGMENT                    SEQ      NDV     1/S.N2         N3 High Value      Low Value
-------- ------------------------- ---- -------- ---------- ---------- --------------- ---------------
SUBPART  GP_RSLT_ACUM_001_X2010M08    1       52 2754194.91         52 ABCDEF 2010AW05 UKPAY 2010M08
         GP_RSLT_ACUM_001_X2010M08    2       52 2754194.91         52 ABCDEF 2010AW05 UKPAY 2010M08
…
         GP_RSLT_ACUM_001_X2010M08   51       52 2754194.91         52 ABCDEF 2010AW05 UKPAY 2010M08
         GP_RSLT_ACUM_001_X2010M08   52       52 2754194.91         52 ABCDEF 2010AW05 UKPAY 2010M08
         GP_RSLT_ACUM_001_X2010M09    1      101 2906275.45        101 ABCDEF 2010AW02 UKPAY 2010M09
         GP_RSLT_ACUM_001_X2010M09    2      101 2906275.45        101 ABCDEF 2010AW02 UKPAY 2010M09
…
         GP_RSLT_ACUM_001_X2010M09  100      101 2906275.45        101 ABCDEF 2010AW02 UKPAY 2010M09
         GP_RSLT_ACUM_001_X2010M09  101      101 2906275.45        101 ABCDEF 2010AW02 UKPAY 2010M09
         GP_RSLT_ACUM_001_X2010M10    1       59 3025938.17         59 ABCDEF 2010AW01 UKPAY 2010M11
         GP_RSLT_ACUM_001_X2010M10    2       59 3025938.17         59 ABCDEF 2010AW01 UKPAY 2010M11
…
         GP_RSLT_ACUM_001_X2010M10   58       59 3025938.17         59 ABCDEF 2010AW01 UKPAY 2010M11
         GP_RSLT_ACUM_001_X2010M10   59       59 3025938.17         59 ABCDEF 2010AW01 UKPAY 2010M11

         GP_RSLT_ACUM_001             1      101        101          0 ABCDEF 2010AW01 UKPAY 2010M11

         GP_RSLT_ACUM_007_X2010M08    1       48 2539061.72         48 ABCDEF 2010AW28 UKPAY 2010M08
         GP_RSLT_ACUM_007_X2010M08    2       48 2539061.72         48 ABCDEF 2010AW28 UKPAY 2010M08
…
         GP_RSLT_ACUM_007_X2010M08   47       48 2539061.72         48 ABCDEF 2010AW28 UKPAY 2010M08
         GP_RSLT_ACUM_007_X2010M08   48       48 2539061.72         48 ABCDEF 2010AW28 UKPAY 2010M08
         GP_RSLT_ACUM_007_X2010M09    1      106 2665680.49        106 ABCDEF 2010AW01 UKPAY 2010M09
         GP_RSLT_ACUM_007_X2010M09    2      106 2665680.49        106 ABCDEF 2010AW01 UKPAY 2010M09
```

[16]

---

[15] Histograms are still collected as a part of global statistics at table and partition level on a sub-partitioned table.

[16] Note that there are no histograms on aggregated statistics. The number of distinct values is the maximum number of buckets on any of the sub-partitions.


```
…
        GP_RSLT_ACUM_007_X2010M09  105      106 2665680.49        106 ABCDEF 2010AW01 UKPAY 2010M09
        GP_RSLT_ACUM_007_X2010M09  106      106 2665680.49        106 ABCDEF 2010AW01 UKPAY 2010M09
        GP_RSLT_ACUM_007_X2010M10    1       52 2778500.23         52 ABCDEF 2010AW37 UKPAY 2010UW25
        GP_RSLT_ACUM_007_X2010M10    2       52 2778500.23         52 ABCDEF 2010AW37 UKPAY 2010UW25
…
        GP_RSLT_ACUM_007_X2010M10   51       52 2778500.23         52 ABCDEF 2010AW37 UKPAY 2010UW25
        GP_RSLT_ACUM_007_X2010M10   52       52 2778500.23         52 ABCDEF 2010AW37 UKPAY 2010UW25

        GP_RSLT_ACUM_007             1      106        106          0 ABCDEF 2010AW01 UKPAY 2010UW25

        GP_RSLT_ACUM_032_X2010M08    1       76  2599399.2         76 ABCDEF 2010AW06 UKPAY 2010M08
        GP_RSLT_ACUM_032_X2010M08    2       76  2599399.2         76 ABCDEF 2010AW06 UKPAY 2010M08
…
        GP_RSLT_ACUM_032_X2010M08   75       76  2599399.2         76 ABCDEF 2010AW06 UKPAY 2010M08
        GP_RSLT_ACUM_032_X2010M08   76       76  2599399.2         76 ABCDEF 2010AW06 UKPAY 2010M08
        GP_RSLT_ACUM_032_X2010M09    1      122 2843965.19        122 ABCDEF 2009AW50 UKPAY 2010M09
        GP_RSLT_ACUM_032_X2010M09    2      122 2843965.19        122 ABCDEF 2009AW50 UKPAY 2010M09
…
        GP_RSLT_ACUM_032_X2010M09  121      122 2843965.19        122 ABCDEF 2009AW50 UKPAY 2010M09
        GP_RSLT_ACUM_032_X2010M09  122      122 2843965.19        122 ABCDEF 2009AW50 UKPAY 2010M09
        GP_RSLT_ACUM_032_X2010M10    1       81  2839327.4         81 ABCDEF 2010AW04 UKPAY 2010M10
        GP_RSLT_ACUM_032_X2010M10    2       81  2839327.4         81 ABCDEF 2010AW04 UKPAY 2010M10
…
        GP_RSLT_ACUM_032_X2010M10   80       81  2839327.4         81 ABCDEF 2010AW04 UKPAY 2010M10
        GP_RSLT_ACUM_032_X2010M10   81       81  2839327.4         81 ABCDEF 2010AW04 UKPAY 2010M10

        GP_RSLT_ACUM_032             1      122        122          0 ABCDEF 2009AW50 UKPAY 2010M10

        GFC_GP_RSLT_ACUM             1      122        122          0 ABCDEF 2009AW50 UKPAY 2010UW25[17]
```

[17] The number of distinct values at table level is the maximum number of buckets on any of the partitions. Not always correct, but not far off global statistics.

Correcting Statistics

We have seen that there can be problems in NDV in both global and aggregated statistics. All the other statistics seem to be correctly aggregated by *dbms_stats*. What can we do about NDV?

Firstly, I can recalculate a minimum value of NDV myself and update the exported statistics directly. Then the statistics can be imported using the Oracle supplied API.

In general, the minimum value of NDV can be calculated as the maximum of NDV of constituent partitions or sub-partition, but if we are calculating a value for NDV for the partitioning column, then we can sum the NDV across the partitions[18].

[18] I haven't looked at the case where a table is (range) partitioned on a combination of columns. I would guess that this rule only works for the first partitioning column.

Implementation of New Approach

The aggregated statistics will be created and managed with the *agg_stats* packaged procedure. The demonstration in this section uses the script on page 6, that simulates the way the statistics are currently collected, to set up the statistics as a starting point.

6. Gathering, fixing and reimporting the statistics in one step.

   The *agg_stats* package contains a single procedure, *gather_fix_stats*, which will collect and fix stats on a named table. This procedure can then be called from a scheduled database job.

```
BEGIN
 agg_stats.gather_fix_stats
 (p_tabname=>'GFC_GP_RSLT_ACUM'
 ,p_method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
,p_statid=>'SUBPART',p_force=>TRUE
 );
END;
/
```

However, the constituent operations can also be run one by one, and I will demonstrate that in the following sections.

a. Delete old Global Statistics on any logical segments. Previously global statistics had been collected on all segments in the table, but we need to get rid of them to make *dbms_stats* do the aggregation for us.

```
DECLARE
 l_stats_changed BOOLEAN;
BEGIN
 agg_stats.delete_global_stats(p_tabname=>'GFC_GP_RSLT_ACUM' ,p_force=>TRUE
                               ,p_stats_changed=>l_stats_changed);
END;
/

18:44:02 05.01.2012:Deleting Table Level Statistics on SYSADM.GFC_GP_RSLT_ACUM
18:44:02 05.01.2012:Deleting Partition Statistics on SYSADM.GFC_GP_RSLT_ACUM(GP_RSLT_ACUM_001)
18:44:02 05.01.2012:Deleting Partition Statistics on SYSADM.GFC_GP_RSLT_ACUM(GP_RSLT_ACUM_007)
18:44:02 05.01.2012:Deleting Partition Statistics on SYSADM.GFC_GP_RSLT_ACUM(GP_RSLT_ACUM_032)
18:44:02 05.01.2012:Deleted Statistics on 4 logical partitons

PL/SQL procedure successfully completed.

Elapsed: 00:00:01.59
```

b. Now we need to collect statistics to create aggregated statistics, so this procedure will collects stats on any physical partitions whose stats are missing or stale. Then it will collect stats on the last sub-partition in each partition where the partition doesn't have stats, and the last partition in the table where the table doesn't have stats. This will make *dbms_stats* do the aggregation.

```
EXECUTE agg_stats.gather_table_stats(p_tabname=>'GFC_GP_RSLT_ACUM',p_force=>TRUE);

18:44:04 05.01.2012:Gathering statistics on SYSADM.GFC_GP_RSLT_ACUM physical subpartition
(GP_RSLT_ACUM_001_X2010M10) to generate merged statistics
18:44:05 05.01.2012:Gathering statistics on SYSADM.GFC_GP_RSLT_ACUM physical subpartition
(GP_RSLT_ACUM_007_X2010M10) to generate merged statistics
18:44:12 05.01.2012:Gathering statistics on SYSADM.GFC_GP_RSLT_ACUM physical subpartition
(GP_RSLT_ACUM_032_X2010M10) to generate merged statistics
18:44:14 05.01.2012:Gathered Statistics on 3 physical partitons

PL/SQL procedure successfully completed.

Elapsed: 00:00:17.21
```

c. Now we need to fix stats the generated stats when NDV is obviously too low. First we export the statistics to the statistics table, and do sub-partition aggregation to partition first, and that rolls up into the partition to table aggregation.

```
EXECUTE agg_stats.fix_stats(p_tabname=>'GFC_GP_RSLT_ACUM',p_statid=>'SUBPART');
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_001:col CALC_RSLT_VAL:NDV 5765->5893
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_007:col CALC_RSLT_VAL:NDV 5737->5902
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_032:col CALC_RSLT_VAL:NDV 5318->5643
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_001:col CALC_VAL:NDV 5743->5861
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_007:col CALC_VAL:NDV 5697->5857
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_032:col CALC_VAL:NDV 5287->5608
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_001:col CAL_RUN_ID:NDV 6->16
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_007:col CAL_RUN_ID:NDV 6->18
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_032:col CAL_RUN_ID:NDV 6->17
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_001:col EMPLID:NDV 3515->3529
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_007:col EMPLID:NDV 3308->3336
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_032:col EMPLID:NDV 3133->3164
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_001:col USER_KEY1:NDV 1913->1924
18:45:33 05.01.2012:C:partition:SYSADM.GFC_GP_RSLT_ACUM.GP_RSLT_ACUM_032:col USER_KEY1:NDV 1777->1804
18:45:33 05.01.2012:C:table SYSADM.GFC_GP_RSLT_ACUM:col EMPLID:NDV 9956->10029
18:45:33 05.01.2012:C:table SYSADM.GFC_GP_RSLT_ACUM:col CAL_RUN_ID:NDV 6->18
18:45:33 05.01.2012:C:table SYSADM.GFC_GP_RSLT_ACUM:col USER_KEY1:NDV 1913->1924

PL/SQL procedure successfully completed.

Elapsed: 00:00:17.70
```

So far, only the contents of the statistics table has been updated.

d. Finally, the corrected statistics are reimported from the statistics table into the data dictionary.

```
BEGIN
    sys.dbms_stats.import_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'GFC_GP_RSLT_ACUM'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'SUBPART'
      ,cascade=>TRUE
      );
END;
/
```

Installation Instructions

The agg_stats package should be installed as follows:

7. I recommend increasing the JOB_QUEUE_PROCESSES parameter from the default setting of 10 to be equal to the value of CPU_COUNT.

8. The agg_stats package requires certain privileges to be directly granted by SYS (not via a role)

```
GRANT EXECUTE ON dbms_lock TO sysadm;
GRANT EXECUTE ON dbms_scheduler TO sysadm;
GRANT CREATE JOB TO sysadm;
```

9. The script agg_stats.sql (see page 22) creates the statistics table GFC_STAT_TABLE and a package of procedures

Implementation Instructions

10. As a one-off action for each table to use the new approach, the existing statistics shoud be exported to the statistics table so that they can be restored later if necessary. This will also be useful during testing to switch back and forth at will between the two sets of statistics as necessary.

```
begin
    sys.dbms_stats.export_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'PS_GP_RSLT_ACUM'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'ORIGINAL'
      ,cascade=>TRUE
      );
end;
/
begin
    sys.dbms_stats.export_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'PS_GP_RSLT_PIN'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'ORIGINAL'
      ,cascade=>TRUE
      );
end;
/
```

11. I am taking this oppotunity to change the way histograms are collected on the sub-partitioned payroll result table. However, collecting sub-partition statistics on a whole table does not use database parallelism well. Therefore, I have enhanced the package to optionally collect statistics for all physical partitions or sub-partitions whether they are stale or otherwise, and also to optionally submit requests to collect statistics to the database scheduler, so that multiple requests can run in parallel. The first time

    a. The very first time that statistics are collected on the partitioned tables the histograms need to be regenerated differently, so they need to be generated on all sub-partitions not just stale ones. Therefore:

        i. *p_allparts* parameters is set to true to force collection on all sub-partitins.

        ii. *p_sched_job* parameter is set to true so that statistics on each sub-partitions are collected by a different job on the Oracle job schedule. *agg_stats.gather_fix_stats* will wait for all the jobs to complete before it returns.

```
Spool dmk
Set serveroutput on lines 200 pages 0
BEGIN
 agg_stats.gather_fix_stats
 (p_tabname=>'PS_GP_RSLT_ACUM'
 ,p_method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
 ,p_allparts=>TRUE /*force collection on all partitions regardless*/
 ,p_statid=>'SUBPART'
 ,p_force=>TRUE
 ,p_sched_job=>TRUE /*collect stats in scheduled jobs*/
 );

agg_stats.gather_fix_stats
 (p_tabname=>'PS_GP_RSLT_PIN'
 ,p_method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
 ,p_allparts=>TRUE /*force collection on all partitions regardless*/
 ,p_statid=>'SUBPART'
 ,p_force=>TRUE
 ,p_sched_job=>TRUE /*collect stats in scheduled jobs*/
 );
END;
/
Spool off
```

b. Subsequently, it is only necessary to gather stale statistics, so the following commands should be run from database job. Currently we only collect statistics once per week on Saturday mornings. However, with the new approach stale statistics could be refreshed daily. I suggest the following commands be run daily before the payroll batch processes.

```
BEGIN
 agg_stats.gather_fix_stats
 (p_tabname=>'PS_GP_RSLT_ACUM'
 ,p_method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
 ,p_allparts=>FALSE /*force collection on all partitions regardless*/
 ,p_statid=>'SUBPART'
 ,p_force=>TRUE
 ,p_sched_job=>TRUE /*collect stats in scheduled jobs*/
 );
END;
/
BEGIN
 agg_stats.gather_fix_stats
 (p_tabname=>'PS_GP_RSLT_PIN'
 ,p_method_opt=>'FOR ALL COLUMNS SIZE 1, FOR COLUMNS SIZE 254 CAL_RUN_ID, CAL_ID, ORIG_CAL_RUN_ID'
 ,p_allparts=>FALSE /*force collection on all partitions regardless*/
 ,p_statid=>'SUBPART'
 ,p_force=>TRUE
 ,p_sched_job=>TRUE /*collect stats in scheduled jobs*/
 );
END;
/
```

12. Done. So what have we got in the statistics table?

```
Column owner format a8
Column table_name format a18
Column statid format a10
break on owner skip 1 on table_name skip 1
select c5 owner
, c1 table_name
, statid
, count(*)
from gfc_stat_table
group by statid, c5, c1
order by 1,2,3
/
```

Note the significant reduction in the number of rows in the statistics table for the actual application tables. This is mainly because I have chosen to limit the columns for which histograms are collected.

```
OWNER    TABLE_NAME          STATID       COUNT(*)
-------- ------------------- ---------- ----------
SYSADM   GFC_GP_RSLT_ACUM    ALL              1987
                             SUBPART          1438

         PS_GP_RSLT_ACUM     ORIGINAL      2580366
                             SUBPART        177612

         PS_GP_RSLT_PIN      ORIGINAL      1118718
                             SUBPART        154898
```

13. Now all we have to do is test it! To restore the original statistics we merely need to reimport them from the statistics table.

```
BEGIN
    sys.dbms_stats.import_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'PS_GP_RSLT_ACUM'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'ORIGINAL'
      ,cascade=>TRUE
      ,force=>TRUE
      );
END;
/
BEGIN
    sys.dbms_stats.import_table_stats
      (ownname=>'SYSADM'
      ,tabname=>'PS_GP_RSLT_PIN'
      ,statown=>'SYSADM'
      ,stattab=>'GFC_STAT_TABLE'
      ,statid=>'ORIGINAL'
      ,cascade=>TRUE
      ,force=>TRUE
      );
END;
/
```

APPENDIX

Source Code

```
REM agg_stats.sql
REM see also http://www.centrexcc.com/SQL%20Tuning%20with%20Statistics.ppt.pdf
REM see also http://skdba.blogspot.com/2006/06/keeping-history-of-cbo-stats.html
REM see also http://oracledoug.com/serendipity/index.php?/archives/1590-Statistics-on-Partitioned-Tables-Contents.html
spool agg_stats
set serveroutput on timi on echo on

REM This package can submit requests to job scheduler, consider increasing JOB_PROCESSES to the same value as CPU_COUNT

REM requires the following grants to be made explicitly by SYS
GRANT EXECUTE ON dbms_lock TO sysadm;
GRANT EXECUTE ON dbms_scheduler TO sysadm;
GRANT CREATE JOB TO sysadm;

ROLLBACK;

--NB this package does not handle global partitioned indexes

--The Stat Table must be created before the package which references it
--EXECUTE sys.dbms_stats.drop_stat_table('SYSADM', 'GFC_STAT_TABLE');
EXECUTE sys.dbms_stats.create_stat_table('SYSADM','GFC_STAT_TABLE');
ALTER INDEX sysadm.gfc_stat_table REBUILD COMPRESS PARALLEL;
CREATE INDEX sysadm.gfc_stat_table2 ON sysadm.gfc_stat_table (type, c5, c1, c2, c3, c4) COMPRESS PARALLEL;

CREATE OR REPLACE PACKAGE sysadm.agg_stats AS
--------------------------------------------------------------------------------
--This procedure deletes statistics on any logical segments that have global statistics
--------------------------------------------------------------------------------
  PROCEDURE delete_global_stats
    (p_ownname       VARCHAR2 DEFAULT 'SYSADM'
    ,p_tabname       VARCHAR2
    ,p_force         BOOLEAN DEFAULT FALSE
    ,p_stats_changed IN OUT BOOLEAN
    ,p_testmode      BOOLEAN DEFAULT FALSE
    );
--------------------------------------------------------------------------------
--Function to decode raw values in R1 and R2 statistics columns.  Not used elsehwere in this package
--stolen from http://structureddata.org/2007/10/16/how-to-display-high_valuelow_value-columns-from-user_tab_col_statistics/
--------------------------------------------------------------------------------
  FUNCTION display_raw
    (p_rawval RAW
    ,p_type   VARCHAR2
    ) RETURN VARCHAR2;
--------------------------------------------------------------------------------
```

```
--This procedure corrects the NDV and density column statistics that have been
--exported to the stat table when the value is lower than theoretically possible.
--------------------------------------------------------------------------------
  PROCEDURE fix_stats
    (p_ownname       VARCHAR2 DEFAULT 'SYSADM'
    ,p_tabname       VARCHAR2
    ,p_statid        VARCHAR2
    ,p_stats_changed IN OUT BOOLEAN
    ,p_testmode      BOOLEAN DEFAULT FALSE
    );
--------------------------------------------------------------------------------
--This procedure gathers statistics on physical partitions or sub-partitions whose statistics
--are either missing  or stale, or on the last partition or sub-partition in a table or partition
--that does not have aggregated statistics
--------------------------------------------------------------------------------
  PROCEDURE gather_table_stats
    (p_ownname       VARCHAR2 DEFAULT 'SYSADM'
    ,p_tabname       VARCHAR2
    ,p_method_opt    VARCHAR2 DEFAULT sys.dbms_stats.get_param('METHOD_OPT')
    ,p_allparts      BOOLEAN DEFAULT FALSE
    ,p_force         BOOLEAN DEFAULT FALSE
    ,p_sched_job     BOOLEAN DEFAULT FALSE
    ,p_stats_changed IN OUT BOOLEAN
    ,p_testmode      BOOLEAN DEFAULT FALSE
    );
--------------------------------------------------------------------------------
--This procedure provides performs all the steps involved in gathering, exporting,
--correcting and reimporting statistics on a partitioned table
--------------------------------------------------------------------------------
  PROCEDURE gather_fix_stats
    (p_ownname    VARCHAR2 DEFAULT 'SYSADM'
    ,p_tabname    VARCHAR2
    ,p_method_opt VARCHAR2 DEFAULT sys.dbms_stats.get_param('METHOD_OPT')
    ,p_allparts   BOOLEAN DEFAULT FALSE
    ,p_statid     VARCHAR2
    ,p_force      BOOLEAN DEFAULT FALSE
    ,p_sched_job  BOOLEAN DEFAULT FALSE
    ,p_testmode   BOOLEAN DEFAULT FALSE
    );
--------------------------------------------------------------------------------
--This procedure calls gather_fix_stats for all partitioned tables whose stats are locked.
--------------------------------------------------------------------------------
  PROCEDURE gather_all_locked_part_tables
    (p_ownname    VARCHAR2 DEFAULT 'SYSADM'
    ,p_method_opt VARCHAR2 DEFAULT sys.dbms_stats.get_param('METHOD_OPT')
    ,p_allparts   BOOLEAN DEFAULT FALSE
    ,p_statid     VARCHAR2
    ,p_testmode   BOOLEAN DEFAULT FALSE
    );
--------------------------------------------------------------------------------
END agg_stats;
/
show errors
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
CREATE OR REPLACE PACKAGE BODY sysadm.agg_stats AS
--------------------------------------------------------------------------------
k_ownname CONSTANT VARCHAR2(8)  := 'SYSADM';
k_stattab CONSTANT VARCHAR2(16) := 'GFC_STAT_TABLE';
k_module  CONSTANT VARCHAR2(48) := 'AGG_STATS.';
--------------------------------------------------------------------------------
--Procedure to emit messages
--------------------------------------------------------------------------------
FUNCTION display_bool
(p_bool BOOLEAN
) RETURN VARCHAR2 IS
BEGIN
  IF p_bool THEN
    RETURN 'TRUE';
  ELSE
    RETURN 'FALSE';
  END IF;
END display_bool;
--------------------------------------------------------------------------------
PROCEDURE msg
(p_msg VARCHAR2
,p_testmode BOOLEAN DEFAULT FALSE
) IS
BEGIN
  IF p_testmode THEN
    dbms_output.put_line('Test: '||TO_CHAR(SYSDATE,'hh24:mi:ss dd.mm.yyyy')||':'||p_msg);
  ELSE
    dbms_output.put_line(TO_CHAR(SYSDATE,'hh24:mi:ss dd.mm.yyyy')||':'||p_msg);
  END IF;
END msg;
--------------------------------------------------------------------------------
PROCEDURE exec_sql
(p_sql VARCHAR2
,p_testmode BOOLEAN DEFAULT FALSE) IS
BEGIN
  IF p_testmode THEN NULL;
    msg('Test SQL: '||p_sql);
  ELSE
    msg(p_sql);
    EXECUTE IMMEDIATE p_sql;
  END IF;
END exec_sql;
--------------------------------------------------------------------------------
--This procedure deletes global statistics on logical segments
--------------------------------------------------------------------------------
PROCEDURE delete_global_stats
(p_ownname       VARCHAR2 DEFAULT 'SYSADM'
,p_tabname       VARCHAR2
,p_force         BOOLEAN DEFAULT FALSE
,p_stats_changed IN OUT BOOLEAN
,p_testmode      BOOLEAN DEFAULT FALSE
) IS
  l_counter INTEGER := 0;
  l_module VARCHAR2(48);
  l_action VARCHAR2(32);
```

```
BEGIN
  dbms_application_info.read_module(module_name=>l_module, action_name=>l_action);
  dbms_application_info.set_module(module_name=>k_module||'DELETE_GLOBAL_STATS',
action_name=>p_ownname||'.'||p_tabname);
  msg('delete_global_stats(p_ownname=>'||p_ownname||', p_tabname=>'||p_tabname||',
p_force=>'||display_bool(p_force)||', p_stats_changed=>'||display_bool(p_stats_changed)||')',p_testmode);
  FOR i IN ( --delete global table level stats where partition exists
        SELECT   t.*
        FROM     all_tab_statistics t
        WHERE    t.owner = p_ownname
        AND      t.table_name = p_tabname
        AND      t.object_type = 'TABLE'
        AND      t.global_stats = 'YES'
        AND EXISTS(
                 SELECT 'x'
                 FROM     all_tab_statistics p
                 WHERE    p.owner = t.owner
                AND     p.table_name = t.table_name
                 AND     p.object_type = 'PARTITION')
        ) LOOP
    msg('Deleting Table Level Statistics on '||i.owner||'.'||i.table_name,p_testmode);
    IF NOT p_testmode THEN
      sys.dbms_stats.delete_table_stats
        (ownname=>i.owner
        ,tabname=>i.table_name
        ,cascade_parts=>FALSE
        ,force=>p_force
        );
    END IF;
    l_counter := l_counter + 1;
  END LOOP;

  FOR i IN ( --delete partition level global stats where subpartition exists
        SELECT   t.*
        FROM     all_tab_statistics t
        WHERE    t.owner = p_ownname
        AND      t.table_name = p_tabname
        AND      t.object_type = 'PARTITION'
        AND      t.global_stats = 'YES'
        AND EXISTS(
                 SELECT 'x'
                 FROM     all_tab_statistics p
                 WHERE    t.owner = p_ownname
                 AND      p.table_name = t.table_name
                 AND      p.partition_name = t.partition_name
                 AND      p.object_type = 'SUBPARTITION')
        ) LOOP
    msg('Deleting Partition Statistics on
'||i.owner||'.'||i.table_name||'('||i.partition_name||')',p_testmode);
    IF NOT p_testmode THEN
      sys.dbms_stats.delete_table_stats
        (ownname=>i.owner
        ,tabname=>i.table_name
        ,partname=>i.partition_name
        ,cascade_parts=>FALSE --do not cascade to child partiton
```

```
        ,force=>p_force
        );
      l_counter := l_counter + 1;
    END IF;
  END LOOP;
  msg('Deleted Statistics on '||l_counter||' logical partitons',p_testmode);
  IF l_counter > 0 THEN
    p_stats_changed := TRUE;
  END IF;
  dbms_application_info.set_module(module_name=>l_module, action_name=>l_action);

END delete_global_stats;
--------------------------------------------------------------------------------
--Function to decode raw values in R1 and R2 statistics columns.  Not used elsehwere in this package
--stolen from http://structureddata.org/2007/10/16/how-to-display-high_valuelow_value-columns-from-
user_tab_col_statistics/
--------------------------------------------------------------------------------
FUNCTION display_raw
(p_rawval RAW
,p_type   VARCHAR2
) RETURN VARCHAR2 IS
  cn  NUMBER;
  cv  VARCHAR2(32);
  cd  DATE;
  cnv NVARCHAR2(32);
  cr  ROWID;
  cc  CHAR(32);
BEGIN
  IF (p_type = 'NUMBER') THEN
    dbms_stats.convert_raw_value(p_rawval, cn);
    RETURN TO_CHAR(cn);
  ELSIF (p_type = 'VARCHAR2') THEN
    dbms_stats.convert_raw_value(p_rawval, cv);
    RETURN TO_CHAR(cv);
  ELSIF (p_type = 'DATE') THEN
    dbms_stats.convert_raw_value(p_rawval, cd);
    RETURN TO_CHAR(cd);
  ELSIF (p_type = 'NVARCHAR2') THEN
    dbms_stats.convert_raw_value(p_rawval, cnv);
    RETURN TO_CHAR(cnv);
  ELSIF (p_type = 'ROWID') THEN
    dbms_stats.convert_raw_value(p_rawval, cr);
    RETURN TO_CHAR(cnv);
  ELSIF (p_type = 'CHAR') THEN
    dbms_stats.convert_raw_value(p_rawval, cc);
    RETURN TO_CHAR(cc);
  ELSE
    RETURN 'UNKNOWN DATATYPE';
  END IF;
END display_raw;
--------------------------------------------------------------------------------
--This procedure corrects some column statistics that have been exported to the stat table
--------------------------------------------------------------------------------
PROCEDURE fix_stats
(p_ownname       VARCHAR2 DEFAULT 'SYSADM'
```

```
,p_tabname       VARCHAR2
,p_statid        VARCHAR2
,p_stats_changed IN OUT BOOLEAN
,p_testmode      BOOLEAN DEFAULT FALSE
) IS
  l_srec DBMS_STATS.STATREC;
  l_module VARCHAR2(48);
  l_action VARCHAR2(32);
BEGIN
  dbms_application_info.read_module(module_name=>l_module, action_name=>l_action);
  dbms_application_info.set_module(module_name=>k_module||'FIX_STATS',
action_name=>p_ownname||'.'||p_tabname);
  msg('fix_stats(p_ownname=>'||p_ownname||', p_tabname=>'||p_tabname||', p_statid=>'||p_statid
           ||', p_stats_changed=>'||display_bool(p_stats_changed)||')',p_testmode);
  FOR i IN ( --this query corrects ndv and density statistics on partitions from subpartitions
with subpart as ( --aggregates stats on columns with histograms so only have one row per segment per column
        SELECT   statid, type, c5, c1, c4, c2, c3
        ,        MAX(n1) n1
        ,        COUNT(*) num_histograms
--      ,        MIN(r1) r1
--      ,        MAX(r2) r2
        FROM     sysadm.gfc_stat_table
        WHERE    c2 IS NOT NULL --partition
        AND      c3 IS NOT NULL --subpartition
        GROUP BY statid, type, c5, c1, c4, c2, c3
), agg as ( --for each column we calculate max and sum of NDV
        SELECT   statid, type, c5, c1, c4, c2
        ,        MAX(n1) max_n1
        ,        SUM(n1) sum_n1
        ,        COUNT(*) num_rows
--      ,        MIN(r1) r1
--      ,        MAX(r2) r2
        FROM     subpart
        GROUP BY statid, type, c5, c1, c4, c2
), part as ( --get current stats for partition
        SELECT   *
        FROM     sysadm.gfc_stat_table
        WHERE    c2 IS NOT NULL
        AND      c3 IS NULL
), cf as ( --compare aggregated stats with actual stats
SELECT   agg.statid, agg.type, agg.c5, agg.c4, agg.c1, agg.c2
,        part.n1
,        CASE WHEN k.column_position IS NOT NULL THEN sum_n1 --sum NDV across partitions on partitioning
column
             ELSE max_n1 --otherwise take maximum NDV across columns
        END as new_n1
,        part.n2
,        part.n3
--,      n1*n2 x
--,      part.r1, part.r2
FROM     agg
        LEFT OUTER JOIN all_subpart_key_columns k --list of subpartition key columns
        ON k.owner = agg.c5
        AND k.object_type = 'TABLE'
        AND k.name = agg.c1
```

```
        AND k.column_name = agg.c4
,       part
WHERE   agg.statid = part.statid
AND     agg.type = part.type
and     agg.c5 = part.c5
and     agg.c1 = part.c1
and     agg.c4 = part.c4
and     agg.c2 = part.c2
ORDER BY agg.c5, agg.c1, agg.c2
)
SELECT  *
FROM    cf
WHERE   statid = p_statid
AND     c5 = p_ownname
AND     c1 = p_tabname
AND     new_n1 > n1 --only if calculated NDV is higher
AND     type = 'C'
ORDER BY c4, c1, c2
  ) LOOP

    --update stats via Oracle API
    msg(i.type||':partition:'||i.c5||'.'||i.c1||'.'||i.c2||':col '||i.c4||':NDV '||i.n1||'-
>'||i.new_n1,p_testmode);
    p_stats_changed := TRUE;

    /*not updating stats via Oracle API because it blanks values not set.  Easier to update table
    sys.dbms_stats.set_column_stats
    (ownname  => p_ownname
    ,tabname  => p_tabname
    ,colname  => i.c4
    ,partname => i.c2
    ,statown  => k_ownname
    ,stattab  => k_stattab
    ,statid   => p_statid
    ,distcnt  => i.new_n1
    ,density  => 1/NULLIF(i.new_n1,0) --density always reciprocal of NDV because never have stats
    );
    */

    IF NOT p_testmode THEN
      UPDATE sysadm.gfc_stat_table
      SET    d1 = SYSDATE --update last analysed time because set_column_stats does not
      ,      n1 = i.new_n1 --number of distinct values
      ,      n2 = 1/NULLIF(i.new_n1,0) --density
--    ,      r1 = i.r1  --set min column value
--    ,      r2 = i.r2  --set max column value
      WHERE  c5 = i.c5  --owner
      AND    c1 = i.c1  --table
      AND    c2 = i.c2  --partition
      AND    c3 IS NULL --subpartition
      AND    c4 = i.c4  --column
      AND    type = i.type
      AND    statid = p_statid
      ;
    END IF;
```

```
  END LOOP;

  FOR i IN ( --this query corrects ndv and density statistics on tables from partitions
with part as ( --this query aggregates stats on columns with histograms so we only have one row per segment
per column
        SELECT  statid, type, c5, c1, c4, c2
        ,       MAX(n1) n1
        ,       COUNT(*) num_histograms
--      ,       MIN(r1) r1
--      ,       MAX(r2) r2
        FROM    sysadm.gfc_stat_table
        WHERE   c2 IS NOT NULL --partition
        AND     c3 IS NULL --subpartition
        GROUP BY statid, type, c5, c1, c4, c2
), agg as ( --for each column we calculate max and sum of NDV
        SELECT  statid, type, c5, c1, c4
        ,       MAX(n1) max_n1
        ,       SUM(n1) sum_n1
        ,       COUNT(*) num_rows
--      ,       MIN(r1) r1
--      ,       MAX(r2) r2
        FROM    part
        GROUP BY statid, type, c5, c1, c4
) , tab as ( --get current stats for table
        SELECT  *
        FROM    sysadm.gfc_stat_table
        WHERE   c2 IS NULL
        AND     c3 IS NULL
) , cf as ( --compare aggregated stats with actual stats
SELECT  agg.statid, agg.type, agg.c5, agg.c4, agg.c1
,       tab.n1
,       CASE WHEN k.column_position IS NOT NULL THEN sum_n1 --sum NDV across partitions on partitioning
column
             ELSE max_n1 --otherwise take maximum NDV across columns
        END as new_n1
,       tab.n2
,       tab.n3
--,     n1*n2 x
--,     tab.r1, tab.r2
FROM    agg
        LEFT OUTER JOIN all_part_key_columns k --list of partition key columns
        ON k.owner = agg.c5
        AND k.object_type = 'TABLE'
        AND k.name = agg.c1
        AND k.column_name = agg.c4
,       tab
WHERE   agg.c5 = tab.c5
and     agg.c1 = tab.c1
and     agg.c4 = tab.c4
AND     agg.type = tab.type
AND     agg.statid = tab.statid
ORDER BY agg.c5, agg.c1
)
SELECT  *
```

```
FROM     cf
WHERE    statid = p_statid
AND      c5 = p_ownname
AND      c1 = p_tabname
AND      new_n1 > n1 --only if calculated NDV is higher
AND     type = 'C'
  ) LOOP

    msg(i.type||':table '||i.c5||'.'||i.c1||':col '||i.c4||':NDV '||i.n1||'->'||i.new_n1,p_testmode);
    p_stats_changed := TRUE;

    /*not updating stats via Oracle API because it blanks values not set.  Easier to update table
    sys.dbms_stats.set_column_stats
    (ownname  => p_ownname
    ,tabname  => p_tabname
    ,colname  => i.c4
    ,partname => NULL
    ,statown  => k_ownname
    ,stattab  => k_stattab
    ,statid   => p_statid
    ,distcnt  => i.new_n1
    ,density  => 1/NULLIF(i.new_n1,0) --density always reciprocal of NDV because never have stats
    );*/

    IF NOT p_testmode THEN
      UPDATE sysadm.gfc_stat_table
      SET    d1 = SYSDATE --update last analysed time because set_column_stats does not
      ,      n1 = i.new_n1 --number of distinct values
      ,      n2 = 1/NULLIF(i.new_n1,0) --density
--    ,      r1 = i.r1  --set min column value
--    ,      r2 = i.r2  --set max column value
      WHERE  c5 = i.c5  --owner
      AND    c1 = i.c1  --table
      AND    c2 IS NULL --partition
      AND    c3 IS NULL --subpartition
      AND    c4 = i.c4  --column
      AND    type = i.type
      AND    statid = p_statid
      ;
    END IF;

  END LOOP;
  dbms_application_info.set_module(module_name=>l_module, action_name=>l_action);

END fix_stats;
--------------------------------------------------------------------------------
--This procedure waits for database jobs submitted to collect statistics.
--The time it waits is determined by the estimate to complete, but at least every minute
--------------------------------------------------------------------------------
PROCEDURE wait_for_jobs
(p_ownname       VARCHAR2 DEFAULT 'SYSADM'
,p_tabname       VARCHAR2
,p_job_submitted INTEGER
) IS
  l_rindex     BINARY_INTEGER; --longops index
```

```
  l_slno       BINARY_INTEGER;
  l_obj        BINARY_INTEGER;

  l_sleep      INTEGER := 60; --seconds to sleep in wait loop
  l_startdttm  DATE; --timestamp when start to submit jobs

  l_num_jsched NUMBER; --number of scheduled jobs
  l_num_jrun   NUMBER; --number of running jobs
  l_num_jobs   NUMBER; --total number of jobs
BEGIN
  l_rindex := dbms_application_info.set_session_longops_nohint;
  l_startdttm := SYSDATE;

  WHILE TRUE LOOP --wait here if stats jobs scheduled
    with X as (
              select t.owner, t.table_name, p.subpartition_name segment_name
              from   all_tab_subpartitions p
              ,      all_part_tables t
              where  t.owner = p.table_owner
              and    t.table_name = p.table_name
              and    t.subpartitioning_type != 'NONE'
              union all
              select t.owner, t.table_name, p.partition_name segment_name
              from   all_tab_partitions p
              ,      all_part_tables t
              where  t.owner = p.table_owner
              and    t.table_name = p.table_name
              and    t.partitioning_type != 'NONE'
              and    t.subpartitioning_type = 'NONE'
              )
    select COUNT(*) jobs
    ,      NVL(SUM(CASE WHEN j.state = 'RUNNING'   THEN 1 END),0) running
    ,      NVL(SUM(CASE WHEN j.state = 'SCHEDULED' THEN 1 END),0) scheduled
    INTO   l_num_jobs, l_num_jrun, l_num_jsched
    from   x
    ,      ALL_SCHEDULER_JOBS j
    where  x.owner = j.owner
    and    x.segment_name = j.job_name
    and    x.table_name = p_tabname
    and    x.owner = p_ownname
    ;

    msg(''||l_num_jobs||' jobs outstanding ('||l_num_jrun||' running, '||l_num_jsched||' scheduled)');
    dbms_application_info.set_session_longops(l_rindex, l_slno,
          p_ownname||'.'||p_tabname, l_obj, 0, p_job_submitted-l_num_jobs, p_job_submitted, 'partition',
'partitions');
    IF l_num_jobs = 0 THEN --break out of loop if no jobs left
      EXIT;
    ELSE
      l_sleep := LEAST(60,(SYSDATE-l_startdttm)*86400*l_num_jobs/NULLIF(p_job_submitted-l_num_jobs,0)/2); -
-sleep for 1 minute or half estimated time to complete
      sys.dbms_lock.sleep(NVL(l_sleep,1));
    END IF;
  END LOOP;
END wait_for_jobs;
```

```
--------------------------------------------------------------------------------
--This procedure gathers statistics on physical partitions or sub-partitions whose statistics
--are either missing  or stale, or on the last partition or sub-partition in a table or partition
--that does not have aggregated statistics
--------------------------------------------------------------------------------
PROCEDURE gather_table_stats
(p_ownname       VARCHAR2 DEFAULT 'SYSADM'
,p_tabname       VARCHAR2
,p_method_opt    VARCHAR2 DEFAULT sys.dbms_stats.get_param('METHOD_OPT')
,p_allparts      BOOLEAN DEFAULT FALSE
,p_force         BOOLEAN DEFAULT FALSE
,p_sched_job     BOOLEAN DEFAULT FALSE
,p_stats_changed IN OUT BOOLEAN
,p_testmode      BOOLEAN DEFAULT FALSE
) IS
  l_counter    INTEGER := 0; --number of partitions stats updated
  l_module     VARCHAR2(48);
  l_action     VARCHAR2(32);
  l_allparts   VARCHAR2(1) := 'N'; --if y then updated all partitions even if not stale states
  l_message    VARCHAR2(100); --string to hold part of message

BEGIN
  dbms_application_info.read_module(module_name=>l_module, action_name=>l_action);
  dbms_application_info.set_module(module_name=>k_module||'GATHER_TABLE_STATS',
action_name=>p_ownname||'.'||p_tabname);
  msg('gather_table_stats(p_ownname=>'||p_ownname||', p_tabname=>'||p_tabname||',
p_method_opt=>'||p_method_opt
     ||', p_allparts=>'||display_bool(p_allparts)||', p_force=>'||display_bool(p_force)||',
p_stats_changed=>'||display_bool(p_stats_changed)||')',p_testmode);

  IF p_allparts THEN
    l_allparts := 'Y';
  END IF;

  FOR i IN ( --partitioned table stale/missing stats
        SELECT  p.owner, p.table_name, p.partition_name
        ,       CASE WHEN p.num_rows IS NULL THEN ' missing'
                     WHEN p.stale_stats = 'YES' THEN ' stale'
                     ELSE ''
                END as reason
        from    all_tab_statistics p
        ,       all_part_tables t
        WHERE   p.owner = p_ownname
        AND     p.table_name = p_tabname
        AND     (p.num_rows IS NULL
       or      l_allparts = 'Y'
        or      p.stale_stats = 'YES')
       and      t.owner = p.owner
        AND     t.table_name = p.table_name
        AND     t.subpartitioning_type = 'NONE'
  ) LOOP
    l_message := i.reason||' statistics on '||i.owner||'.'||i.table_name||' physical partition
('||i.partition_name||')';
    IF NOT p_testmode THEN
      IF p_sched_job THEN
```

```
        msg('Gathering'||l_message,p_testmode);
        sys.dbms_scheduler.create_job
        (job_name   => i.partition_name
        ,job_type   => 'PLSQL_BLOCK'
        ,job_action => 'BEGIN sys.dbms_stats.gather_table_stats(ownname=>'''||i.owner
                              ||''', tabname=>'''||i.table_name
                              ||''', partname=>'''||i.partition_name
                              ||''', cascade=>TRUE, method_opt=>'''||p_method_opt
                              ||''', force=>'||display_bool(p_force)
                              ||  ', granularity=>''PARTITION''); END;'
        ,start_date => SYSTIMESTAMP --run job immediately
        ,enabled    => TRUE --job is enabled
        ,auto_drop  => TRUE --request will be dropped when complete
        ,comments   => 'Gather stats on table partition
'||i.owner||'.'||i.table_name||'.'||i.partition_name
        );
      ELSE
        msg('Submitting job to gather'||l_message,p_testmode);
        sys.dbms_stats.gather_table_stats
          (ownname     => i.owner
          ,tabname     => i.table_name
          ,partname    => i.partition_name
          ,cascade     => TRUE
          ,granularity => 'PARTITION'
          ,method_opt  => p_method_opt
          ,force       => p_force
          );
      END IF;
    END IF;
    l_counter := l_counter + 1;
  END LOOP;

  FOR i IN ( --subpartitioned table stale/missing stats
        SELECT   s.owner, s.table_name, s.subpartition_name
        ,        CASE WHEN s.num_rows IS NULL THEN ' missing'
                      WHEN s.stale_stats = 'YES' THEN ' stale'
                      ELSE ''
                 END as reason
        from     all_tab_statistics s
        WHERE    s.owner = p_ownname
        AND      s.table_name = p_tabname
        AND      s.object_type = 'SUBPARTITION'
        AND      (s.num_rows IS NULL
        or      l_allparts = 'Y'
        or       s.stale_stats = 'YES')
  ) LOOP
    l_message := i.reason||' statistics on '||i.owner||'.'||i.table_name||' physical subpartition
('||i.subpartition_name||')';
    IF NOT p_testmode THEN
      IF p_sched_job THEN
        msg('Submitting job to gather'||l_message,p_testmode);
        sys.dbms_scheduler.create_job
        (job_name   => i.subpartition_name
        ,job_type   => 'PLSQL_BLOCK'
        ,job_action => 'BEGIN sys.dbms_stats.gather_table_stats(ownname=>'''||i.owner
```

```
                            ||''', tabname=>'''||i.table_name
                            ||''', partname=>'''||i.subpartition_name
                            ||''', cascade=>TRUE, method_opt=>'''||p_method_opt
                            ||''', force=>'||display_bool(p_force)
                            ||  ', granularity=>''SUBPARTITION''); END;'
        ,start_date => SYSTIMESTAMP --run job immediately
        ,enabled    => TRUE --job is enabled
        ,auto_drop  => TRUE --request will be dropped when complete
        ,comments   => 'Gather stats on table sub_partition
'||i.owner||'.'||i.table_name||'.'||i.subpartition_name
        );
      ELSE
        msg('Gathering'||l_message,p_testmode);
        sys.dbms_stats.gather_table_stats
          (ownname     => i.owner
          ,tabname     => i.table_name
          ,partname    => i.subpartition_name
          ,cascade     => TRUE
          ,granularity => 'SUBPARTITION'
          ,method_opt  => p_method_opt
          ,force       => p_force
          );
      END IF;
    END IF;
    l_counter := l_counter + 1;
  END LOOP;

  IF l_counter > 0 THEN
    wait_for_jobs(p_ownname, p_tabname, l_counter); --wait for database jobs
    msg('Gathered Statistics on '||l_counter||' physical partitons');
    p_stats_changed := TRUE;
    l_counter := 0;
  END IF;

  FOR i IN (  --last subpartition in partition without stats
        SELECT s.* from (
        SELECT   x.*
        ,        row_number() OVER (
                         PARTITION BY table_name, partition_name
                         ORDER BY stale_stats DESC, subpartition_position DESC) as ranking
        from     all_tab_statistics x
        where    object_type = 'SUBPARTITION'
        ) s
        ,        all_tab_statistics p
        WHERE    s.owner = p_ownname
        AND      s.table_name = p_tabname
        and      p.owner = s.owner
        AND      p.table_name = s.table_name
        and      p.partition_name = s.partition_name
        and      p.num_rows IS NULL
        and      p.object_type = 'PARTITION'
        and      s.ranking = 1
  ) LOOP
    msg('Gathering statistics on '||i.owner||'.'||i.table_name
      ||' physical subpartition ('||i.subpartition_name||') to generate aggregated statistics',p_testmode);
```

```
    IF NOT p_testmode THEN
      IF p_sched_job THEN
        msg('Submitting job to gather'||l_message,p_testmode);
        sys.dbms_scheduler.create_job
        (job_name   => i.subpartition_name
        ,job_type   => 'PLSQL_BLOCK'
        ,job_action => 'BEGIN sys.dbms_stats.gather_table_stats(ownname=>'''||i.owner
                              ||''', tabname=>'''||i.table_name
                              ||''', partname=>'''||i.subpartition_name
                              ||''', cascade=>TRUE, method_opt=>'''||p_method_opt
                              ||''', force=>'||display_bool(p_force)
                              ||  ', granularity=>''SUBPARTITION''); END;'
        ,start_date => SYSTIMESTAMP --run job immediately
        ,enabled    => TRUE --job is enabled
        ,auto_drop  => TRUE --request will be dropped when complete
        ,comments   => 'Gather stats on table sub_partition
'||i.owner||'.'||i.table_name||'.'||i.subpartition_name
        );
      ELSE
        msg('Gathering'||l_message,p_testmode);
        sys.dbms_stats.gather_table_stats
          (ownname     => i.owner
          ,tabname     => i.table_name
          ,partname    => i.subpartition_name
          ,cascade     => TRUE
          ,granularity => 'SUBPARTITION'
          ,method_opt  => p_method_opt
          ,force       => p_force
          );
      END IF;
    END IF;
    l_counter := l_counter + 1;
  END LOOP;

  FOR i IN ( --last partition in table without stats
        SELECT s.* from (
        SELECT   x.*
        ,        row_number() OVER (
                         PARTITION BY table_name
                         ORDER BY stale_stats DESC, partition_position DESC) as ranking
        from     all_tab_statistics x
        where    object_type = 'PARTITION'
        ) s
        ,        all_tab_statistics p
        WHERE    s.owner = p_ownname
        AND      s.table_name = p_tabname
        and      p.owner = s.owner
        AND      p.table_name = s.table_name
        and      p.partition_name = s.partition_name
        and      p.num_rows IS NULL
        and      p.object_type = 'TABLE'
        and      s.ranking = 1
  ) LOOP
    msg('Gathering statistics on '||i.owner||'.'||i.table_name
      ||' physical partition ('||i.partition_name||') to generate aggregated statistics',p_testmode);
```

```
    IF NOT p_testmode THEN
      IF p_sched_job THEN
        msg('Gathering'||l_message,p_testmode);
        sys.dbms_scheduler.create_job
        (job_name   => i.partition_name
        ,job_type   => 'PLSQL_BLOCK'
        ,job_action => 'BEGIN sys.dbms_stats.gather_table_stats(ownname=>'''||i.owner
                              ||''', tabname=>'''||i.table_name
                              ||''', partname=>'''||i.partition_name
                              ||''', cascade=>TRUE, method_opt=>'''||p_method_opt
                              ||''', force=>'||display_bool(p_force)
                              ||  ', granularity=>''PARTITION''); END;'
        ,start_date => SYSTIMESTAMP --run job immediately
        ,enabled    => TRUE --job is enabled
        ,auto_drop  => TRUE --request will be dropped when complete
        ,comments   => 'Gather stats on table partition
'||i.owner||'.'||i.table_name||'.'||i.partition_name
        );
      ELSE
        msg('Submitting job to gather'||l_message,p_testmode);
        sys.dbms_stats.gather_table_stats
          (ownname     => i.owner
          ,tabname     => i.table_name
          ,partname    => i.partition_name
          ,cascade     => TRUE
          ,granularity => 'PARTITION'
          ,method_opt  => p_method_opt
          ,force       => p_force
          );
      END IF;
    END IF;
    l_counter := l_counter + 1;
  END LOOP;

  IF l_counter > 0 THEN
    wait_for_jobs(p_ownname, p_tabname, l_counter); --wait for database jobs
    msg('Gathered Statistics on '||l_counter||' physical partitons');
    p_stats_changed := TRUE;
    l_counter:= 0;
  END IF;

  dbms_application_info.set_module(module_name=>l_module, action_name=>l_action);

END gather_table_stats;
--------------------------------------------------------------------------------
--This procedure provides performs all the steps involved in gathering, exporting,
--correcting and reimporting statistics on a partitioned table
--------------------------------------------------------------------------------
PROCEDURE gather_fix_stats
(p_ownname    VARCHAR2 DEFAULT 'SYSADM'
,p_tabname    VARCHAR2
,p_method_opt VARCHAR2 DEFAULT sys.dbms_stats.get_param('METHOD_OPT')
,p_allparts   BOOLEAN DEFAULT FALSE
,p_statid     VARCHAR2
,p_force      BOOLEAN DEFAULT FALSE
```

```
,p_sched_job  BOOLEAN DEFAULT FALSE
,p_testmode   BOOLEAN DEFAULT FALSE
) IS
  l_module VARCHAR2(48);
  l_action VARCHAR2(32);
  l_stats_changed BOOLEAN := FALSE;
BEGIN
  dbms_application_info.read_module(module_name=>l_module, action_name=>l_action);
  dbms_application_info.set_module(module_name=>k_module||'GATHER_FIX_STATS',
action_name=>p_ownname||'.'||p_tabname);
  msg('gather_fix_stats(p_ownname=>'||p_ownname||', p_tabname=>'||p_tabname||',
p_method_opt=>'||p_method_opt
      ||', p_statid=>'||p_statid||', p_allparts=>'||display_bool(p_allparts)||',
p_force=>'||display_bool(p_force)||')',p_testmode);

  agg_stats.delete_global_stats --delete global stats
  (p_ownname=>p_ownname
  ,p_tabname=>p_tabname
  ,p_force=>p_force
  ,p_stats_changed=>l_stats_changed
  ,p_testmode=>p_testmode
  );

  agg_stats.gather_table_stats --gather missing stale stats
  (p_ownname=>p_ownname
  ,p_tabname=>p_tabname
  ,p_method_opt=>p_method_opt
  ,p_allparts=>p_allparts
  ,p_force=>p_force
  ,p_sched_job=>p_sched_job
  ,p_stats_changed=>l_stats_changed
  ,p_testmode=>p_testmode
  );

  exec_sql('DELETE FROM sysadm.gfc_stat_table WHERE statid = '''||p_statid||''' AND c1 = '''
          ||p_tabname||''' AND c5 = '''||p_ownname||'''',p_testmode); --delete old export
  msg(''||SQL%ROWCOUNT||' rows deleted.');

  msg('Exporting statistics for table '||p_ownname||'.'||p_tabname,p_testmode);
  IF NOT p_testmode THEN
    sys.dbms_stats.export_table_stats --export stats
    (ownname=>p_ownname
    ,tabname=>p_tabname
    ,statown=>k_ownname
    ,stattab=>k_stattab
    ,statid=>p_statid
    ,cascade=>TRUE
    );
  END IF;

  agg_stats.fix_stats --fix exported stats
  (p_ownname=>p_ownname
  ,p_tabname=>p_tabname
  ,p_statid=>p_statid
  ,p_stats_changed=>l_stats_changed
```

```
  ,p_testmode=>p_testmode
  );

  IF l_stats_changed THEN

    msg('Importing corrected statistics for table '||p_ownname||'.'||p_tabname,p_testmode);
    IF NOT p_testmode THEN
      sys.dbms_stats.import_table_stats --reimport fixed statistics
      (ownname=>p_ownname
      ,tabname=>p_tabname
      ,statown=>k_ownname
      ,stattab=>k_stattab
      ,statid=>p_statid
      ,cascade=>TRUE
      ,force=>p_force
      );
    END IF;
  ELSE
    msg('No statistics have changed for table '||p_ownname||'.'||p_tabname,p_testmode);
  END IF;

  commit;
  msg('Finished processing statistics for table '||p_ownname||'.'||p_tabname,p_testmode);
  dbms_application_info.set_module(module_name=>l_module, action_name=>l_action);

END gather_fix_stats;
--------------------------------------------------------------------------------
--This procedure calls gather_fix_stats for all partitioned tables whose stats are locked.
--------------------------------------------------------------------------------
PROCEDURE gather_all_locked_part_tables
(p_ownname    VARCHAR2 DEFAULT 'SYSADM'
,p_method_opt VARCHAR2 DEFAULT sys.dbms_stats.get_param('METHOD_OPT')
,p_allparts   BOOLEAN DEFAULT FALSE
,p_statid     VARCHAR2
,p_testmode   BOOLEAN  DEFAULT FALSE
) IS
  l_module VARCHAR2(48);
  l_action VARCHAR2(32);
BEGIN
  dbms_application_info.read_module(module_name=>l_module, action_name=>l_action);
  dbms_application_info.set_module(module_name=>k_module||'GATHER_ALL_LOCKED_PART_STATS',
action_name=>p_ownname);
  msg('gather_all_locked_part_tables(p_ownname=>'||p_ownname||', p_method_opt=>'||p_method_opt||',
p_allparts=>'
     ||display_bool(p_allparts)||', p_statid=>'||p_statid||')',p_testmode);

  FOR i iN(
    SELECT t.owner
    ,      t.table_name
    from   all_tab_statistics s
    ,      all_tables t
    where  t.owner = s.owner
    and    t.table_name = s.table_name
    and    s.partition_name is null
    and    s.stattype_locked != 'NO'
```

```
    and    t.partitioned = 'YES'
    and    (t.owner = p_ownname OR p_ownname IS NULL)
  ) LOOP
    gather_fix_stats
    (p_ownname=>i.owner
    ,p_tabname=>i.table_name
    ,p_method_opt=>p_method_opt
    ,p_allparts=>p_allparts
    ,p_statid=>p_statid
    ,p_force=>TRUE
    ,p_testmode=>p_testmode);
  END LOOP;
  dbms_application_info.set_module(module_name=>l_module, action_name=>l_action);
END gather_all_locked_part_tables;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
END agg_stats;
/
show errors
--------------------------------------------------------------------------------
spool off
```