

Moving To I I g Statistics

Virginia Oracle Users Group Conference
October 2010

Statistic Changes in Oracle I I g

- dbms_stats package completely rewritten
- Gathering sorting algorithm replaced with hash
 - MUCH faster
 - Uses less TEMP space and memory
- Statistics can be “pending” for concurrent publishing
- Statistic Defaults and Preferences can be set at global, database, schema, and table levels

Statistic Changes in I I g continued

- Table Monitoring is no longer optional
- Statistics History Retention is variable
- Desired (default) sample size is 100% (mostly)
- Automated job calculates actual sample size based on “staleness” and maintenance window
- For partitioned tables, Global Incremental and Global Aggregated Statistics are options
- Extended Statistics for expressions and column combinations

Statistic Changes in I I g continued

- Better Column Histograms – available for “extended” column combinations
- Easier to copy table statistics
- Easier to compare statistic sets
- Statistics not automatically copied during exchange partition operations
- Statistics can be locked at partition level

New dbms_stats routines

- ALTER_STATS_HISTORY_RETENTION
- COPY_TABLE_STATS
- CREATE_EXTENDED_STATS (DROP)
- DELETE_DATABASE_PREFS (SET, EXPORT, IMPORT)
- DELETE_SCHEMA_PREFS (SET, EXPORT, IMPORT)
- DELETE_TABLE_PREFS (SET, EXPORT, IMPORT)
- DIFF_TABLE_STATS_IN_HISTORY (IN_PENDING, IN_STATTAB)

New dbms_stats routines

- GET_PREFS
- LOCK_PARTITION_STATS (UNLOCK)
- PUBLISH_PENDING_STATS
- RESET_GLOBAL_PREFS_DEFAULTS
- RESET_PARAM_DEFAULTS
- SHOW_EXTENDED_STATS_NAME

New 11g Data Dictionary Views

- DBA_COL_PENDING_STATS
- DBA_IND_PENDING_STATS
- DBA_TAB_HISTGRM_PENDING_STATS
- DBA_TAB_PENDING_STATS
- DBA_TAB_STAT_PREFS

New hash sorting algorithm

- Gathering sorting algorithm replaced with hash
 - MUCH faster
 - Uses less TEMP space and memory
- Hash algorithm requires approximately 30% of the time required for 10g's sorting

Statistics can be Pending

- Previous Oracle versions “published” or began using newly collected statistics (and invalidating cached cursors) as soon as a value was collected on any table or index
- 11g allows an entire statistics gathering run to be “pending” until completion
- Statistics can be auto published or examined prior to publication

Statistic Preferences

- CASCADE
- DEGREE
- ESTIMATE_PERCENT
- METHOD_OPT
- NO_INVALIDATE
- GRANULARITY
- PUBLISH
- INCREMENTAL
- STALE_PERCENT

Statistic Preferences

- CASCADE = TRUE
- DEGREE = 2 or 8
- ESTIMATE_PERCENT = DBMS_STATS.AUTO_SAMPLE_SIZE
- METHOD_OPT = FOR ALL COLUMNS SIZE AUTO
- NO_INVALIDATE = DBMS_STATS.AUTO_INVALIDATE
- GRANULARITY = AUTO
- PUBLISH = TRUE
- INCREMENTAL = TRUE
- STALE_PERCENT = 10

Statistics Retention

- ALTER_STATS_HISTORY_RETENTION specifies retention time in days
- 0 = statistics are never saved
- 1 = statistics are never purged
- NULL = default value
- PURGE_STATS procedure provides the ability to purge statistics at will

AUTOSTATS_TARGET Parameter

- AUTOSTATS_TARGET determines which objects are subject to AUTO statistics gathering
 - ALL = all objects
 - ORACLE = Oracle owned objects (including data dictionary and AWR)
 - AUTO = Oracle decides which objects to gather statistics on (default)

Statistics Sampling

- More complete sampling at reduced intervals
- 100% is default target sampling
- 10% is default staleness
- When running AUTO sampling on AUTO objects, the 11g supplied automated gathering job calculates actual sample size based on “staleness” and maintenance window
- Column statistics start with 5570 rows

INCREMENTAL GLOBAL STATISTICS

- Maintains the global statistics of a partitioned table whenever partition statistics are gathered
 - Table's INCREMENTAL = TRUE
 - Table's PUBLISH = TRUE
 - ESTIMATE_PERCENT = AUTO_SAMPLE_SIZE
 - GRANULARITY = AUTO
- On 10g upgraded table, collects statistics on all partitions when “first” partition gathering job is run in 11g or if any partition statistics are missing

Aggregated Global Statistics

- Maintains global statistics on partitioned objects
- Statistics must be available for all partitions (empty stats are allowed, stale stats are not)
- Global statistics must not be present for the partitioned object (gather stats will not be overwritten with aggregated stats)
- GLOBAL_STATS column of the _STATISTICS views = 'YES' then statistics were gathered and will not be overwritten with aggregated values
- Aggregated global statistics are used in 10g whenever partition exchanges are performed
- 11g uses aggregated statistics for all partitioned indexes

Exchange Partition Change

- In 10g, statistics are exchanged automatically with partition exchanges and aggregated statistics are updated where appropriate
 - global gathered statistics may become stale
- In 11g, statistics are not exchanged when partitions are exchanged
- When statistics are gathered or imported for an exchanged partition in 11g, global incremental or aggregated statistics may be updated (if all requirements are met)

Extended Statistics

- Can be gathered on multiple columns or an expression
- DBMS_STATS.CREATE_EXTENDED_STATS function returns system generated name
- See DBA_STAT_EXTENSIONS and DBA_TAB_COL_STATISTICS

Extended Statistics Example

```

DECLARE
sName varchar2(30);
BEGIN
sName := dbms_stats.create_extended_stats(
'fin_base','journal_entry',
'(accounting_period_id,business_unit_id) ');
sName := dbms_stats.create_extended_stats(
'fin_base','journal_entry',
'(upper(journal_name))');
END;
/

```

Column Specified Histograms

```

METHOD_OPT =>
'FOR ALL COLUMNS SIZE AUTO
FOR COLUMNS BUSINESS_UNIT_ID SIZE
255
FOR COLUMNS
(ACCOUNTING_PERIOD_ID,
BUSINESS_UNIT_ID) SIZE SKEWONLY'

```

Oracle Server Environment

- IBM SMP frames with logical (partitioned) servers, 16 to 64 physical processors
- Enterprise shared servers host 75-150 databases each
- Dedicated LPARs for larger, higher usage, or business critical databases

10g Statistics Environment

- Custom PL/SQL scripts gather statistics
- Varying sample sizes
- Hardcoded overrides needed for some tables, indexes, and partitions
- Current partitions gathered daily with 25% sampling
- Sunday gathering of all partitions at less than 100% and global level statistics
- Index statistics are gathered whenever table statistics are gathered

11g Statistics Strategy - Most Databases

- Statistics are retained for 45 days
- Use maintenance window scheduled job to automatically gather statistics
- AUTOSTATS_TARGET = AUTO
- Use Global Incremental Statistics (Partitions)
- Cascade option will gather statistics on needed indexes in a serial manner
- Degree of 2 will use parallelism to speed gathering where parallel options are enabled
- Stagger maintenance windows on enterprise servers or modify resource manager plan to restrict threads

Database Preferences (Most DBs)

- CASCADE = TRUE
- DEGREE = 2
- ESTIMATE_PERCENT = DBMS_STATS.AUTO_SAMPLE_SIZE
- METHOD_OPT = FOR ALL COLUMNS SIZE AUTO
- NO_INVALIDATE = DBMS_STATS.AUTO_INVALIDATE
- GRANULARITY = AUTO
- PUBLISH = TRUE
- INCREMENTAL = TRUE
- STALE_PERCENT = 10

Duke I lg Statistics Strategy - Large Databases

- Statistics are retained for 45 days
- Use customized scripts to run multiple gathering process threads
- AUTOSTATS_TARGET = AUTO
- Use Global Incremental Statistics (Partitions)
- Do not use cascade option to gather index stats
- Gather index stats whenever table stats are gathered
- Use parallelism degree 8
- Daily change based statistic gathering – no special extended weekend jobs

Database Preferences (Large DBs)

- CASCADE = FALSE
- DEGREE = 8
- ESTIMATE_PERCENT = DBMS_STATS.AUTO_SAMPLE_SIZE
- METHOD_OPT = FOR ALL COLUMNS SIZE AUTO
- NO_INVALIDATE = DBMS_STATS.AUTO_INVALIDATE
- GRANULARITY = AUTO
- PUBLISH = TRUE
- INCREMENTAL = TRUE
- STALE_PERCENT = 10

I lg Implementation – No Partitions

- Review SYSAUX tablespace sizing
- Set the statistics retention time
- Set the database defaults
- Set the database preferences
- Review the maintenance window runtime

I lg Implementation with Partitions

- Review SYSAUX tablespace sizing
- Disable GATHER_STATS_JOB
- Create directories on server
 - STATS_DIR
 - /oracle/local/sid/scripts/stats
 - /oracle/local/sid/reports
- Set the statistics retention time
- Set the database defaults
- Set the database preferences

Threaded Scripts - Concept

- Like parallelism, running multiple threads allows gathering stats in a shorter elapsed time (as resources allow)
- Imagine checkout lines at your favorite store:
 - 4 lines open
 - 24 customers to be check out
 - What happens if some customers have a couple of times and others have full carts?
 - What if one checker just started today?

Threaded Scripts - Concepts

- STATS_DIR is used to map logical database directory to physical server directory
- First job generates SQL files of anonymous PL/SQL blocks containing a dbms_stats call
- Second job runs the SQL files in as many threads as desired, spooled run listings are written to a dated log directory
- When runtime error occurs, SQL script and run listing are copied to a dated error directory

Threaded Scripts – Run Count

- SQL*Plus sessions are spawned as background processes of current server process
- Count the number of background processes spawned
- Launch additional SQL*Plus jobs until count equals desired threads
- Sleep a couple of seconds
- Count the number of background processes spawned
- Continue the loop as long as SQL scripts remain to be run

11g Statistics Setup Scripts

- create_stats_dir.sql
- set_stats_retention.sql
- set_global_prefs.sql
- set_db_prefs.sql
- gen_stat_first_part.sql
- gen_stat_part_index.sql
- run_stats.ksh

11g Statistics Threaded Scripts

- gen_stat_obj.ksh calls gen_stat_obj.sql and generates SQL scripts for each stale object
- run_stats.ksh runs SQL scripts with desired number of threads

11g Statistics - Conclusions

- Better, faster, with built in preferences to manage exceptions
- Transition from 10g to 11g statistics requires extended processing and careful setup on larger databases
- Only change based daily statistics gathering runs are needed to maintain optimal statistical levels

```
1  --
2  --  create_stats_dir
3  --
4  declare
5      sDbName varchar2(10);
6      sSql    varchar2(1000);
7  begin
8      select lower(database_name) into sDbName from dual;
9      begin
10         execute immediate 'DROP DIRECTORY STATS_DIR';
11         exception when others then null;
12     end;
13     sSql:= 'CREATE OR REPLACE DIRECTORY STATS_DIR AS ''/oracle/local/' || sDbName ||
14           '/scripts/stats/''';
15     execute immediate sSql;
16 end;
17 /
```

```

1  #!/bin/ksh
2  #-----
3  # file:   gen_stat_first_part.ksh
4  # ver:   1.1 demo
5  #-----
6  # This script will generates the SQL scripts needed to gather initial
7  # partitioned table statistics in an newly upgraded 11g database in
8  # the $2 schema of the $1 database with $3 DOP.
9  #
10 #-----
11 #      DATE      USERID      DESCRIPTION
12 # -----
13 # 07-Jun-2010 mtnorman  initial 11g implementation
14 #-----
15
16 #-----
17 # verify a parameters were passed - SID, schema and number of control threads
18 #-----
19 if [ $# -ne 4 ]
20 then
21 echo "\nYou must provide an Oracle Connect String, schema name, DOP, and PREFIX
22     count"
23 echo "Usage $0 <ORACLE_SID> <SCHEMA> <DOP> <PREFIX>\n "
24     exit 1
25 fi
26
27 #-----
28 # Set up environment variables - replace ? with your values
29 #-----
30 export ORACLE_SID=$1
31 export SCHEMA=$2
32 export DOP=$3
33 export PREFIX=$4
34 export THREADS=$2
35 export ORACLE_BASE=?
36 export ORACLE_HOME=?
37 export PATH=?
38 export SOURCE=$ORACLE_BASE/local/$ORACLE_SID/scripts
39 export REPORTS=$ORACLE_BASE/local/$ORACLE_SID/reports
40 export DATESTAMP=`date +%Y%m%d`\`
41 #-----
42 # Generate gathering SQL scripts for each individual schema object
43 #-----
44 cd $SOURCE
45 sqlplus "/ as sysdba" @gen_stat_first_part $SCHEMA $DOP $PREFIX
46 exit
47

```

```

1  -----
2  ---**
3  ---**  gan_stat_first_part
4  ---**
5  ---**  Generates SQL script files to gathering statistics on the first partition
6  ---**  of each partitioned table
7  ---**
8  ---**  NOTE:  This script uses the database, schema, and table preferences
9  ---**         stored in the database in oracle version 11.1 and later
10 ---**
11 ---**  Positional Parameters
12 ---**  =====
13 ---**  1: Schema owner of objects to gather statistics
14 ---**  2: Degree of parallelism used to gather statistics
15 ---**  3: Count value used to begin file prefix range
16 ---**  4: Force flag used to force gathering of locked statistics
17 ---**
18 ---**  Modification History
19 ---**  =====
20 ---**  07-Jun-2010 mtnorman initial coding
21 ---**
22 -----
23 declare
24 -- parameter values or constants
25   iCnt          integer:= to char(&3);
26   iDegree       integer:= to char(&2);
27   sOracleBase   varchar2(1000):= '/oradm';
28   sSchema       varchar2(1000):= upper('&1');
29 -- variables
30   dNow          date:= sysdate;
31   dToday        date:= trunc(sysdate);
32   sFormat       varchar2(1000):= 'dd-Mon-yyyy hh24:mi';
33   sGran         varchar2(1000):= 'PARTITION';
34 -- derived variables
35   sToday        varchar2(1000):= to char(dToday, 'yyyymmdd');
36   sLogDir       varchar2(1000):= sOracleBase || '/local/' ||
37   lower(sys.database_name) || '/reports/' || sToday || '/';
38 --
39 -- cursors, cursor types and collections
40 --
41 -- table partitions
42 --
43   cursor c_tab part ( p_sOwn in varchar2 ) is
44     select tp.*
45     from (select table owner, table_name, min(partition_name) as partition_name
46           from dba_tab_partitions
47           where table owner = p sOwn
48               and table name not like 'OLD %'
49               and table name not like 'MLOG$_%'
50           group by table owner, table name
51           order by table_owner, table_name
52           ) tp,
53     ( select mview_name, 'YES' as is_mview from dba_mviews
54       where owner = p_sOwn
55       ) m
56   where tp.table name = m.mview_name (+)
57     and m.is_mview is null
58   order by table name;
59   type t tab part is table of c_tab_part%rowtype;
60   tTabPart t tab part;
61 -- application info savepoint variables
62   sAct          varchar2(1000);
63   sMod          varchar2(1000);
64 --
65 -- write_file procedure
66 procedure write_file

```

```

67     ( p iCnt      in out  integer,
68       p sType    in      varchar2,
69       p_sObject  in      varchar2,
70       p_sPart    in      varchar2 default null
71     )
72 is
73     iFile        utl file.file type;
74     sBuffer      varchar2(2000);
75     sLineEnd     varchar2(1000):= chr(13);
76     sFile        varchar2(1000);
77 begin
78     p iCnt:= p iCnt + 1;
79     sFile:= ltrim(to char(p iCnt,'000000')) || '_' ||
80     replace(lower(p_sObject),'$','S');
81     iFile:= utl_file.fopen (
82         location      => 'STATS DIR',
83         filename      => sFile || '.sql',
84         open mode     => 'w',
85         max_linesize  => 2000
86     );
87     sBuffer:= 'set timing on echo on serveroutput on' || sLineEnd;
88     utl_file.put_line(iFile,sBuffer);
89     sBuffer:= 'spool ' || sLogDir || sFile || '_' || sToday || sLineEnd;
90     utl_file.put_line(iFile,sBuffer);
91     sBuffer:= 'declare' || sLineEnd;
92     utl_file.put_line(iFile,sBuffer);
93     sBuffer:= ' stats locked EXCEPTION;';
94     utl_file.put_line(iFile,sBuffer);
95     sBuffer:= ' PRAGMA EXCEPTION INIT(stats_locked, -20005);';
96     utl_file.put_line(iFile,sBuffer);
97     sBuffer:= 'begin' || sLineEnd;
98     utl_file.put_line(iFile,sBuffer);
99     sBuffer:= ' dbms_application info.set_module(''
100         || sSchema || '.' || p_sObject || ','''
101         || nvl(p_sPart,sFile) || ''');'
102         || sLineEnd;
103     utl_file.put_line(iFile,sBuffer);
104     sBuffer:= ' dbms stats.gather table_stats(' ;
105     utl_file.put_line(iFile,sBuffer);
106     sBuffer:= ' ownname          => '' || sSchema      || ',''' ;
107     utl_file.put_line(iFile,sBuffer);
108     sBuffer:= ' tabname          => '' || p_sObject    || ',''' ;
109     utl_file.put_line(iFile,sBuffer);
110     sBuffer:= ' partname         => '' || p_sPart     || ',''' ;
111     utl_file.put_line(iFile,sBuffer);
112     sBuffer:= ' degree           => ' || iDegree      || ',''' ;
113     utl_file.put_line(iFile,sBuffer);
114     sBuffer:= ' cascade          => false' ;
115     utl_file.put_line(iFile,sBuffer);
116     sBuffer:= ' );';
117     utl_file.put_line(iFile,sBuffer);
118     sBuffer:= ' utl_file.fremove ( ''STATS_DIR'', '' || sFile || '.sql'');';
119     utl_file.put_line(iFile,sBuffer);
120     sBuffer:= 'exception ' ;
121     utl_file.put_line(iFile,sBuffer);
122     sBuffer:= ' when stats_locked then utl_file.fremove ( ''STATS_DIR'', '' ||
123         sFile || '.sql'');';
124     utl_file.put_line(iFile,sBuffer);
125     sBuffer:= ' when others then ' ;
126     utl_file.put_line(iFile,sBuffer);
127     sBuffer:= ' dbms output.put_line(sqlcode || '-'' || sqlerrm);';
128     utl_file.put_line(iFile,sBuffer);
129     sBuffer:= ' utl_file.frename(''STATS DIR'', '' || sFile || '.sql' ,
130         ''STATS_DIR'', '' || sFile || '.err' , true);';
131     utl_file.put_line(iFile,sBuffer);
132     sBuffer:= 'end;';
133     utl_file.put_line(iFile,sBuffer);

```

```
131     sBuffer:= '/';
132     utl file.put line(iFile,sBuffer);
133     sBuffer:= 'exit';
134     utl file.put line(iFile,sBuffer,true);
135     utl file.fclose(iFile);
136 end write_file;
137 --
138 -- begin generating one SQL file per table
139 --
140 begin
141     dbms_application_info.read_module(sMod,sAct);
142 --
143 -- populate list of table partitions
144 --
145 dbms application info.set module('Gather ' || sSchema || ' Partition Initial
Stats','generate SQL scripts');
146 open c tab part(sSchema);
147 fetch c tab part bulk collect into tTabPart;
148 close c tab_part;
149 if tTabPart is null then null;
150 else
151     for b in 1..tTabPart.count loop
152         write file(iCnt,'TABLE PART', tTabPart(b).table_name,
tTabPart(b).partition_name);
153     end loop; -- tTabPart
154 end if;
155 --
156 -- reset application info to calling module
157 --
158 dbms application info.set_module(sMod,sAct);
159 exception when others then
160     dbms_application_info.set_module(sMod,sAct);
161 end;
162 /
163 exit
164
```

```

1  #!/bin/ksh
2  #-----
3  # file:    gen_stat_obj.ksh
4  # ver:    1.1 demo
5  #-----
6  # This script will generates the SQL scripts (one per object) needed to
7  # gather statistics in the $2 schema of the $1 database with $3 DOP.
8  #
9  # NOTE: Using "ALL" as the schema gathers statistics on objects in all schemas.
10 #
11 #-----
12 #      DATE      USERID      DESCRIPTION
13 # -----
14 # 18-May-2010 mtnorman  initial coding
15 #-----
16
17 #-----
18 # verify a parameters were passed - SID, schema and number of control threads
19 #-----
20 if [ $# -ne 5 ]
21 then
22     echo "\nYou must provide an Oracle Connect String, schema name, DOP, PREFIX count and
23     FORCE option"
24     echo "Usage $0 <ORACLE_SID> <SCHEMA> <DOP> <PREFIX> <FORCE> \n "
25     exit 1
26 fi
27 #-----
28 # Set up environment variables - replace ? with your values
29 #-----
30 export ORACLE_SID=$1
31 export SCHEMA=$2
32 export DOP=$3
33 export PREFIX=$4
34 export FORCEOPT=$5
35 export ORACLE_BASE=?
36 export ORACLE_HOME=?
37 export PATH=?
38 export SOURCE=$ORACLE_BASE/local/$ORACLE_SID/scripts
39 export REPORTS=$ORACLE_BASE/local/$ORACLE_SID/reports
40 export DATESTAMP=`date +%Y%m%d`\`
41
42 #-----
43 # Generate gathering SQL scripts for each individual schema object
44 #-----
45 cd $SOURCE
46 sqlplus "/ as sysdba" @gen_stat_obj $SCHEMA $DOP $PREFIX $FORCEOPT
47 exit
48

```

```

1  set serveroutput on
2  -----
3  ---**
4  ---** gen_stat_obj
5  ---**
6  ---** Generate SQL scripts that will gather statistics on tables, table
7  ---** partitions, indexes and index partitions where statistics are stale using
8  ---** preference values
9  ---**
10 ---** NOTE: This script uses the database, schema, and table preferences
11 ---**      stored from Oracle version 11.1 and later
12 ---**
13 ---** Positional Parameters
14 ---** =====
15 ---** 1: Schema owner of objects to gather statistics or ALL to gather statistics
16 ---**    on entire database
17 ---** 2: Degree of parallelism used to gather statistics
18 ---** 3: Count value used to begin SQL filename prefix range
19 ---** 4: Force flag used to force gathering of locked statistics
20 ---**
21 ---** Gathering Solution Method
22 ---** =====
23 ---** table is not part; index is not part ==> gather index stats,
24 ---**                                           gather table stats
25 ---** table is not part; index is part    ==> gather index stats on all parts,
26 ---**                                           gather table stats
27 ---** table is part      ; index is not part ==> gather index stats,
28 ---**                                           gather table part
29 ---** table is part      ; index is part    ==> gather index part,
30 ---**                                           gather table part
31 ---**
32 ---** Modification History
33 ---** =====
34 ---** 17-May-2010 mtnorman initial coding
35 ---**
36 -----
37 declare
38 -- parameter values or constants
39   iCnt      integer:= to_char(&3);
40   iDegree   integer:= to_char(&2);
41   sForceOpt varchar2(1000) := upper('&4');
42   sOracleBase varchar2(1000) := '/oracle';
43   sSchema   varchar2(1000) := upper('&1');
44 -- variables
45   bForce    boolean;
46   dNow      date:= sysdate;
47   dToday    date:= trunc(sysdate);
48   iPartCnt  integer;
49   sAct      varchar2(1000);
50   sForce    varchar2(1000);
51   sFormat   varchar2(1000) := 'dd-Mon-yyyy hh24:mi';
52   sGran     varchar2(1000);
53   sMod      varchar2(1000);
54   sPart     varchar2(1000);
55   sTab      varchar2(1000);
56   sTabCnt   varchar2(1000);
57   sTabPart  varchar2(1000);
58   tTab      sys.dbms_stats.objecttab;
59 -- derived variables
60   sToday    varchar2(1000) := to_char(dToday, 'yyyymmdd');
61   sLogDir   varchar2(1000) := sOracleBase || '/local/' ||
62   lower(sys.database name) || '/reports/' || sToday || '/';
63 -----
64 ---** cursors, cursor types and collections
65 ---**
66 -----

```

```

67 --
68 -- indexes (partitioned or non-partitioned) on a given table_name
69 --
70 cursor c index
71   ( p sOwn in varchar2,
72     p sTab in varchar2,
73     p sPart in varchar2
74   ) is
75   select index name
76     from dba indexes
77    where table owner      = p sOwn
78          and table name   = p sTab
79          and partitioned = p sPart
80    order by index_name desc;
81 type t index is table of c_index%rowtype;
82 tIdx t_index;
83 --
84 -- index partitions
85 --
86 cursor c idx part
87   ( p_sOwn in varchar2,
88     p_sIdx in varchar2
89   ) is
90   select partition name
91     from dba ind partitions
92    where index owner = p sOwn
93          and index name = p_sIdx
94          and num rows > 0
95          -- condition added to suppress empty partitions
96    order by partition name;
97 type t idx part is table of c_idx_part%rowtype;
98 tIdxPart t_idx_part;
99 --
100 -- determine index partition name for matching table partition name
101 --
102 cursor c idx part name
103   ( p sOwn in varchar2,
104     p sTab in varchar2,
105     p_sIdx in varchar2,
106     p sPart in varchar2
107   ) is
108   select p.partition name as ind part name
109     from ( select column name, column position
110           from dba part_key columns
111          where owner      = p sOwn
112                and name   = p_sTab
113                and object_type = 'TABLE'
114           ) tk,
115     dba tab partitions t,
116     dba part indexes i,
117     dba ind partitions p,
118     dba part key columns k
119    where t.table_owner      = p_sOwn
120          and t.table name   = p sTab
121          and t.partition_name = p sPart
122          and i.owner        = p sOwn
123          and i.table name   = p sTab
124          and i.index name   = p_sIdx
125          and i.locality     = 'LOCAL'
126          and p.index owner  = i.owner
127          and p.index name   = i.index name
128          and p.partition position = t.partition position
129          and p.high value_length = t.high value_length
130          and p.composite    = t.composite
131          and k.owner        = p.index owner
132          and k.name         = p.index_name
133          and k.object_type  = 'INDEX'

```

```

133         and k.column name           = tk.column name
134         and k.column position       = tk.column_position
135         and i.index_name            = k.name;
136     type t_idx part name is table of c_idx_part_name%rowtype;
137     tIdxPartName t_idx_part_name;
138
139     -----
140     ---**
141     ---** write_file procedure
142     ---**
143     ---** this embedded procedure writes a SQL script file in the STATS_DIR to
144     ---** gather statistics on one database object
145     ---**
146     ---** the SQL script generated includes logging and error handling
147     ---** (1) logs are written to a dated directory (see sLogDir)
148     ---** (2) sql scripts that encounter run time errors are renamed to use the
149     ---**      "err" file extension
150     ---**
151     -----
152     procedure write_file
153     ( p_iCnt      in out  integer,
154       p_sType     in     varchar2,
155       p_sOwn      in     varchar2,
156       p_sObject   in     varchar2,
157       p_sPart     in     varchar2 default null
158     )
159     is
160     iFile      utl_file.file_type;
161     sBuffer    varchar2(2000);
162     sLineEnd   varchar2(1000) := chr(13);
163     sFile      varchar2(1000);
164     begin
165     p_iCnt := p_iCnt + 1;
166     sFile := ltrim(to char(p_iCnt,'000000')) || '_' ||
167     replace(lower(p_sObject), '$', 'S');
168     iFile := utl_file.fopen (
169         location      => 'STATS DIR',
170         filename     => sFile || '.sql',
171         open_mode    => 'w',
172         max_linesize => 2000
173     );
174     sBuffer := 'set timing on echo on serveroutput on' || sLineEnd;
175     utl_file.put_line(iFile,sBuffer);
176     sBuffer := 'spool ' || sLogDir || sFile || '_' || sToday || sLineEnd;
177     utl_file.put_line(iFile,sBuffer);
178     sBuffer := 'declare' || sLineEnd;
179     utl_file.put_line(iFile,sBuffer);
180     sBuffer := ' stats locked EXCEPTION;';
181     utl_file.put_line(iFile,sBuffer);
182     sBuffer := ' PRAGMA EXCEPTION INIT(stats_locked, -20005);';
183     utl_file.put_line(iFile,sBuffer);
184     sBuffer := 'begin' || sLineEnd;
185     utl_file.put_line(iFile,sBuffer);
186     sBuffer := ' dbms application info.set module(''
187     || p_sOwn || '.' || p_sObject || ''','''
188     || nvl(p_sPart,sFile) || ''');'
189     || sLineEnd;
190     utl_file.put_line(iFile,sBuffer);
191     if p_sType = 'INDEX' then
192     sBuffer := ' dbms stats.gather_index_stats(';
193     utl_file.put_line(iFile,sBuffer);
194     sBuffer := ' ownname => '' ' || p_sOwn || ''','' ;
195     utl_file.put_line(iFile,sBuffer);
196     sBuffer := ' indname => '' ' || p_sObject || ''','' ;
197     utl_file.put_line(iFile,sBuffer);
198     sBuffer := ' degree => ' || iDegree || ''','' ;
199     utl_file.put_line(iFile,sBuffer);

```

```

199     sBuffer:= '      granularity      => ''' || sGran      || ', ' ;
200     utl_file.put_line(iFile,sBuffer);
201     sBuffer:= '      force              => '      || sForce;
202     utl_file.put_line(iFile,sBuffer);
203     sBuffer:= '      );' ;
204     utl_file.put_line(iFile,sBuffer);
205     elsif p_sType = 'INDEX PART' then
206     sBuffer:= ' dbms stats.gather index_stats(' ;
207     utl_file.put_line(iFile,sBuffer);
208     sBuffer:= '      ownname           => ''' || p_sOwn      || ', ' ;
209     utl_file.put_line(iFile,sBuffer);
210     sBuffer:= '      indname            => ''' || p_sObject   || ', ' ;
211     utl_file.put_line(iFile,sBuffer);
212     sBuffer:= '      partname           => ''' || p_sPart      || ', ' ;
213     utl_file.put_line(iFile,sBuffer);
214     sBuffer:= '      degree              => '      || iDegree     || ', ' ;
215     utl_file.put_line(iFile,sBuffer);
216     sBuffer:= '      granularity      => 'AUTO', ' ;
-- supports global aggregate update
217     utl_file.put_line(iFile,sBuffer);
218     sBuffer:= '      force              => '      || sForce;
219     utl_file.put_line(iFile,sBuffer);
220     sBuffer:= '      );' ;
221     utl_file.put_line(iFile,sBuffer);
222     elsif p_sType = 'TABLE PART' then
223     sBuffer:= ' dbms stats.gather table_stats(' ;
224     utl_file.put_line(iFile,sBuffer);
225     sBuffer:= '      ownname           => ''' || p_sOwn      || ', ' ;
226     utl_file.put_line(iFile,sBuffer);
227     sBuffer:= '      tabname            => ''' || p_sObject   || ', ' ;
228     utl_file.put_line(iFile,sBuffer);
229     sBuffer:= '      partname           => ''' || p_sPart      || ', ' ;
230     utl_file.put_line(iFile,sBuffer);
231     sBuffer:= '      degree              => '      || iDegree     || ', ' ;
232     utl_file.put_line(iFile,sBuffer);
233     sBuffer:= '      granularity      => 'AUTO', ' ;
-- supports global incremental update
234     utl_file.put_line(iFile,sBuffer);
235     sBuffer:= '      force              => '      || sForce     || ', ' ;
236     utl_file.put_line(iFile,sBuffer);
237     sBuffer:= '      block sample       => false, ' ;
238     utl_file.put_line(iFile,sBuffer);
239     sBuffer:= '      cascade            => false' ;
240     utl_file.put_line(iFile,sBuffer);
241     sBuffer:= '      );' ;
242     utl_file.put_line(iFile,sBuffer);
243     elsif p_sType = 'TABLE' then
244     sBuffer:= ' dbms stats.gather table_stats(' ;
245     utl_file.put_line(iFile,sBuffer);
246     sBuffer:= '      ownname           => ''' || p_sOwn      || ', ' ;
247     utl_file.put_line(iFile,sBuffer);
248     sBuffer:= '      tabname            => ''' || p_sObject   || ', ' ;
249     utl_file.put_line(iFile,sBuffer);
250     sBuffer:= '      degree              => '      || iDegree     || ', ' ;
251     utl_file.put_line(iFile,sBuffer);
252     sBuffer:= '      granularity      => ''' || sGran      || ', ' ;
253     utl_file.put_line(iFile,sBuffer);
254     sBuffer:= '      force              => '      || sForce     || ', ' ;
255     utl_file.put_line(iFile,sBuffer);
256     sBuffer:= '      block sample       => false, ' ;
257     utl_file.put_line(iFile,sBuffer);
258     sBuffer:= '      cascade            => false' ;
259     utl_file.put_line(iFile,sBuffer);
260     sBuffer:= '      );' ;
261     utl_file.put_line(iFile,sBuffer);
262     end if;
263     sBuffer:= ' utl_file.fremove ( 'STATS_DIR', ''' || sFile || '.sql' );';

```

```

264     utl_file.put_line(iFile,sBuffer);
265     sBuffer:= 'exception ';
266     utl_file.put_line(iFile,sBuffer);
267     sBuffer:= '  when stats_locked then utl_file.fremove ( ''STATS_DIR'', '' ||
sFile || '.sql'' ); ';
268     utl_file.put_line(iFile,sBuffer);
269     sBuffer:= '  when others then ';
270     utl_file.put_line(iFile,sBuffer);
271     sBuffer:= '    dbms output.put_line(sqlcode || ''-'' || sqlerrm);';
272     utl_file.put_line(iFile,sBuffer);
273     sBuffer:= '      utl_file.frename(''STATS DIR'', '' || sFile || '.sql'' ,
''STATS DIR'', '' || sFile || '.err'' , true);';
274     utl_file.put_line(iFile,sBuffer);
275     sBuffer:= 'end;';
276     utl_file.put_line(iFile,sBuffer);
277     sBuffer:= '/';
278     utl_file.put_line(iFile,sBuffer);
279     sBuffer:= 'exit';
280     utl_file.put_line(iFile,sBuffer,true);
281     utl_file.fclose(iFile);
282   end write_file;
283   -----
284   --**
285   --**  begin execution of main block
286   --**
287   -----
288   begin
289     dbms_application_info.read_module(sMod,sAct);
290     if   sForceOpt = 'TRUE' then sForce:= 'TRUE';  bForce:= true;
291     elsif sForceOpt = 'T'   then sForce:= 'TRUE';  bForce:= true;
292     elsif sForceOpt = 'YES' then sForce:= 'TRUE';  bForce:= true;
293     elsif sForceOpt = 'Y'   then sForce:= 'TRUE';  bForce:= true;
294     else                                     sForce:= 'FALSE'; bForce:= false;
295     end if;
296     -----
297     --**
298     --**  gather list of objects with stale statistics
299     --**
300     -----
301     if sSchema = 'ALL' then
302       dbms application info.set module('Gather Database Change Statistics','begin');
303       dbms output.put_line('Gather Database Change Statistics begins at ' || to_char(
sysdate,sFormat));
304       dbms stats.gather_database_stats(
305         options => 'LIST AUTO',
306         objlist => tTab
307       );
308     else
309       dbms application info.set_module('Gather ' || sSchema || ' Change
Statistics','begin');
310       dbms output.put_line('Gather ' || sSchema || ' Change Statistics begins at ' ||
to_char(sysdate,sFormat));
311       dbms_stats.gather_schema_stats(
312         ownname => sSchema,
313         options => 'LIST AUTO',
314         objlist => tTab,
315         force   => bForce
316       );
317     end if;
318     sTabCnt:= ' of ' || trim(to_char(tTab.count));
319     -----
320     --**
321     --**  for each table object, collect statistics on every index first, then
322     --**  gather table statistics
323     --**
324     -----
325     for i in 1 .. tTab.count loop

```

```

326     sTab:= ' ' || trim(to char(i)) || sTabCnt;
327     if substr(tTab(i).objname,1,4) in ('BIN$', 'OLD_') then
-- ignore recycle objects
328         null;
329     elsif substr(tTab(i).objname,1,4) = 'SYS_' then
-- ignore system named XML objects
330         null;
331     else
332         dbms_application_info.set_action(tTab(i).objname || ' ' || sTab);
333         dbms_output.put_line('...' || tTab(i).objname || ' ' ||
334                               || tTab(i).partname || ' ' ||
335                               || tTab(i).subpartname || ' ' ||
336                               || to char(sysdate,sFormat));
337         if tTab(i).subpartname is null and tTab(i).partname is null then
338             sGran:= 'GLOBAL';
339             sPart:= null;
340         elsif tTab(i).subpartname is null then
341             sGran:= 'PARTITION';
342             sPart:= tTab(i).partname;
343         else
344             sGran:= 'SUBPARTITION';
345             sPart:= tTab(i).subpartname;
346         end if;
347         if tTab(i).objtype = 'TABLE' then
348             select partitioned into sTabPart
349             from dba tables
350             where owner = tTab(i).ownname
351                   and table_name = tTab(i).objname;
352         elsif tTab(i).objtype = 'INDEX' then
353             select partitioned into sTabPart
354             from dba indexes
355             where owner = tTab(i).ownname
356                   and index_name = tTab(i).objname;
357         else
358             sTabPart:= null;
359         end if;
360         --
361
-- table/index is partitioned and global stats are stale ==> do not gather
global stats on partitioned objects
362         --
363         if sPart is null and sTabPart = 'YES' then
364             null;
365         --
-- table is partitioned and (sub)partition of partitioned table is stale
366         --
367         --
368         elsif tTab(i).objtype = 'TABLE' and sTabPart = 'YES' then
369             open c index(tTab(i).ownname, tTab(i).objname, 'NO');
-- non-partitioned indexes
370             fetch c index bulk collect into tIdx;
371             close c index;
372             for j in 1..tIdx.count loop
373                 if substr(tIdx(j).index_name,1,4) = 'BIN$' then
-- ignore recycle objects
374                     null;
375                 else
376                     write_file(iCnt, 'INDEX', tTab(i).ownname, tIdx(j).index_name);
377                 end if;
378             end loop j;
379             open c index(tTab(i).ownname, tTab(i).objname, 'YES');
-- partitioned indexes
380             fetch c index bulk collect into tIdx;
381             close c index;
382             for j in 1..tIdx.count loop
383                 if substr(tIdx(j).index_name,1,4) = 'BIN$' then
-- ignore recycle objects
384                     null;

```

```

385     else
386         select count(1)
387             into iPartCnt
388             from dba ind partitions
389             where index owner      = tTab(i).ownname
390                and index name      = tIdx(j).index_name
391                and partition name = sPart;
392     if iPartCnt = 1 then
393         -- confirm table and index use same partition name
394         write file(iCnt, 'INDEX PART', tTab(i).ownname, tIdx(j).index_name,
395             sPart);
396     else
397         -- determine index partition name
398         begin
399             open c idx part name(tTab(i).ownname, tTab(i).objname,
400                 tIdx(j).index name, sPart);
401             fetch c idx part name bulk collect into tIdxPartName;
402             close c idx part name;
403             if tIdxPartName.count = 1 then
404                 write file(iCnt, 'INDEX PART', tTab(i).ownname, tIdx(j).index_name,
405                     tIdxPartName(1).ind_part_name);
406             end if;
407             exception when others then null;
408         end;
409     end if;
410 end if;
411 end loop j;
412 write_file(iCnt, 'TABLE PART', tTab(i).ownname, tTab(i).objname, sPart);
413 --
414 -- non-partitioned table
415 --
416 elsif tTab(i).objtype = 'TABLE' then
417     open c index(tTab(i).ownname, tTab(i).objname, 'NO');
418     -- non-partitioned indexes
419     fetch c index bulk collect into tIdx;
420     close c index;
421     for j in 1..tIdx.count loop
422         if substr(tIdx(j).index name,1,4) = 'BIN$' then
423             -- ignore recycle objects
424             null;
425         else
426             write_file(iCnt, 'INDEX', tTab(i).ownname, tIdx(j).index_name);
427         end if;
428     end loop j;
429     open c index(tTab(i).ownname, tTab(i).objname, 'YES');
430     -- partitioned indexes
431     fetch c index bulk collect into tIdx;
432     close c index;
433     for j in 1..tIdx.count loop
434         if substr(tIdx(j).index name,1,4) = 'BIN$' then
435             -- ignore recycle objects
436             null;
437         else
438             open c idx part(tTab(i).ownname, tTab(i).objname);
439             fetch c idx part bulk collect into tIdxPart;
440             close c idx part;
441             for k in 1..tIdxPart.count loop
442                 write file(iCnt, 'INDEX PART', tTab(i).ownname, tIdx(j).index_name,
443                     tIdxPart(k).partition_name);
444             end loop k;
445         end if;
446     end loop j;
447     write file(iCnt, 'TABLE', tTab(i).ownname, tTab(i).objname);
448     *****
449     ***
450     *** gather stats on stale indexes as needed
451     ***

```

```
442  --*****
443      else
444          if sPart is null then
445              write_file(iCnt, 'INDEX', tTab(i).ownname, tTab(i).objname);
446          else
447              write_file(iCnt, 'INDEX PART', tTab(i).ownname, tTab(i).objname, sPart);
448          end if;
449      end if;
450  end if;
451  end loop i;
452  --
453  -- reset application info to calling module
454  --
455  dbms_application_info.set_module(sMod,sAct);
456  exception when others then
457      dbms_application_info.set_module(sMod,sAct);
458  end;
459  /
460  exit
461
```

```

1  #!/bin/ksh
2  #-----
3  # file:    run_stats.ksh
4  # ver:    1.2 demo
5  #-----
6  # This script runs the statistics gathering files found in the "stats"
7  # directory.
8  #
9  #-----
10 #     DATE      USERID      DESCRIPTION
11 # -----
12 # 12-Jul-2009 mtnorman  initial implementation
13 # 18-May-2010 mtnorman  added daily logging directory creation
14 #-----
15
16 #-----
17 # verify a parameters were passed - SID, schema and number of control threads
18 #-----
19 if [ $# -ne 2 ]
20 then
21     echo "\nYou must provide an Oracle Connect String and number of threads"
22     echo "Usage $0 <ORACLE_SID> <THREADS>\n "
23     exit 1
24 fi
25
26 #-----
27 # Set up environment variables - replace ? with your values
28 #-----
29 export ORACLE_SID=$1
30 export THREADS=$2
31 export ORACLE_BASE=?
32 export ORACLE_HOME=?
33 export PATH=?
34 export SOURCE=$ORACLE_BASE/local/$ORACLE_SID/scripts
35 export REPORTS=$ORACLE_BASE/local/$ORACLE_SID/reports
36 export DATESTAMP=`date +%Y%m%d`
37 export MYPID=$$
38
39 #-----
40 # Create daily logging directory as needed
41 #-----
42 export LOGDIR=$REPORTS/$DATESTAMP
43 mkdir -p $LOGDIR
44
45 #-----
46 # Run multiple threads to gather statistics as needed
47 #-----
48 cd $SOURCE/stats
49 for obj in *.sql
50 do
51     i=`ps -ef | grep sqlplus | grep $MYPID | wc -l`
52     while [ i -ge $THREADS ]
53     do
54         sleep 2
55         i=`ps -ef | grep sqlplus | grep $MYPID | wc -l`
56     done
57     sqlplus "/ as sysdba" @$obj > /dev/null 2>&1 &
58 done
59
60 wait
61
62 #-----
63 # Create daily error directory and populate with error source files
64 #-----
65 export ERRDIR=$SOURCE/err $DATESTAMP
66 if ((`ls -al *.err | wc -l` > 0 )) then
67     mkdir -p $ERRDIR

```

```
68  for obj in *.err
69  do
70      mv $obj $ERRDIR/$obj
71      cp -p $LOGDIR/${obj%.*}* $ERRDIR
72  done
73  fi
74
75  exit
76
```

```
1  begin
2  DBMS_STATS.SET DATABASE_PREFS (
3    pname   => 'CASCADE',
4    pvalue  => 'FALSE',
5    add_sys => FALSE);
6  DBMS_STATS.SET DATABASE_PREFS (
7    pname   => 'DEGREE',
8    pvalue  => '8',
9    add_sys => TRUE);
10 DBMS_STATS.SET DATABASE_PREFS (
11  pname   => 'ESTIMATE_PERCENT',
12  pvalue  => 'DBMS_STATS.AUTO_SAMPLE_SIZE',
13  add_sys => TRUE);
14 DBMS_STATS.SET DATABASE_PREFS (
15  pname   => 'METHOD_OPT',
16  pvalue  => 'FOR ALL COLUMNS SIZE AUTO',
17  add_sys => TRUE);
18 DBMS_STATS.SET DATABASE_PREFS (
19  pname   => 'NO_INVALIDATE',
20  pvalue  => 'DBMS_STATS.AUTO_INVALIDATE',
21  add_sys => TRUE);
22 DBMS_STATS.SET DATABASE_PREFS (
23  pname   => 'GRANULARITY',
24  pvalue  => 'AUTO',
25  add_sys => TRUE);
26 DBMS_STATS.SET DATABASE_PREFS (
27  pname   => 'PUBLISH',
28  pvalue  => 'TRUE',
29  add_sys => TRUE);
30 DBMS_STATS.SET DATABASE_PREFS (
31  pname   => 'INCREMENTAL',
32  pvalue  => 'TRUE',
33  add_sys => TRUE);
34 DBMS_STATS.SET DATABASE_PREFS (
35  pname   => 'STALE_PERCENT',
36  pvalue  => '10',
37  add_sys => TRUE);
38 end;
39 /
40
```

```
1  begin
2  DBMS_STATS.SET GLOBAL Prefs (
3    pname => 'CASCADE',
4    pvalue => 'TRUE');
5  DBMS_STATS.SET GLOBAL Prefs (
6    pname => 'DEGREE',
7    pvalue => '8');
8  DBMS_STATS.SET GLOBAL Prefs (
9    pname => 'ESTIMATE PERCENT',
10   pvalue => 'DBMS_STATS.AUTO_SAMPLE_SIZE');
11 DBMS_STATS.SET GLOBAL Prefs (
12   pname => 'METHOD OPT',
13   pvalue => 'FOR ALL COLUMNS SIZE AUTO');
14 DBMS_STATS.SET GLOBAL Prefs (
15   pname => 'NO INVALIDATE',
16   pvalue => 'DBMS_STATS.AUTO_INVALIDATE');
17 DBMS_STATS.SET GLOBAL Prefs (
18   pname => 'GRANULARITY',
19   pvalue => 'AUTO');
20 DBMS_STATS.SET GLOBAL Prefs (
21   pname => 'PUBLISH',
22   pvalue => 'TRUE');
23 DBMS_STATS.SET GLOBAL Prefs (
24   pname => 'INCREMENTAL',
25   pvalue => 'TRUE');
26 DBMS_STATS.SET GLOBAL Prefs (
27   pname => 'STALE_PERCENT',
28   pvalue => '10');
29 end;
30 /
31
```

```
1 begin
2     dbms_stats.alter_stats_history_retention (45);
3 end;
4 /
5
```