Understanding Optimizer Statistics in Cheetah

Hyun-Ju Vega

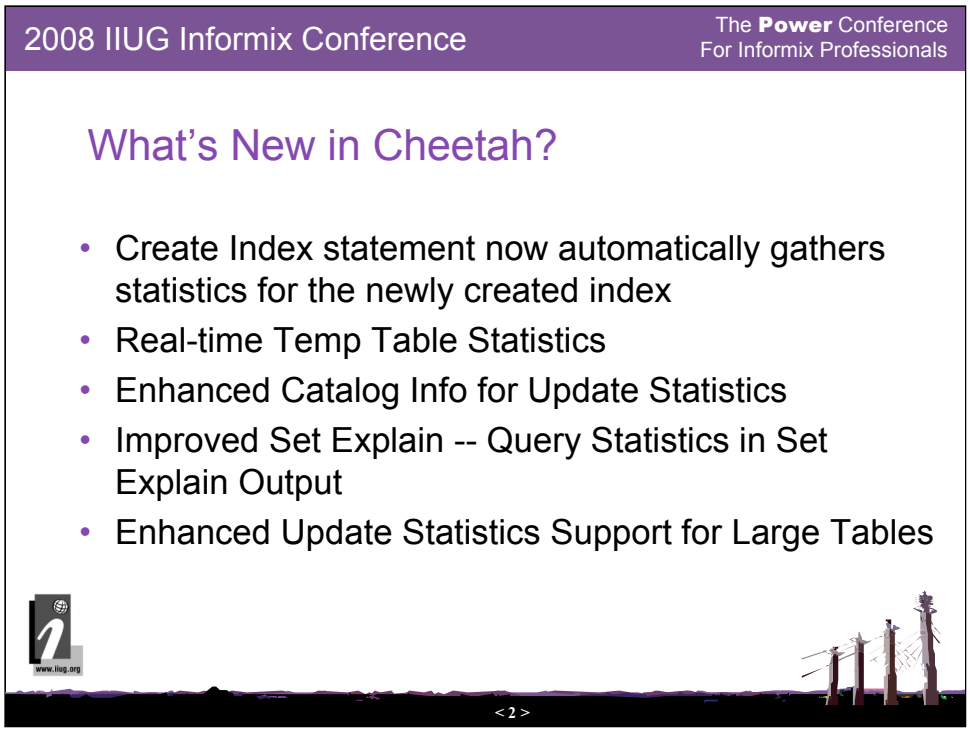
IBM

Session D10

Tuesday, April 29, 2008 • 02:10 p.m. – 03:10 p.m.

2008 IIUG Informix Conference

www.iiug.org



Find out how you can get the most out of the new optimizer statistics features in Cheetah by understanding what's new and different in IDS 11.10. This presentation is a survey of new auto-statistics and update statistics related features in IDS 11. It describes the new functionalities in detail and discusses how the new statistics features in Cheetah may affect current update statistics practices and performance.

Topics discussed include the new auto-statistics feature when creating indexes, real-time temp table statistics, enhanced catalog information for update statistics data, the new "Sampling Size" option in Update Statistics Medium command, and the improved Set Explain.

Quick Review of the Optimization Process -- Why is statistics important?

- Choosing the right QUERY PATH determines how fast you get your results.
- Choosing the Wrong Path can be like going around the world to get to your neighbor's.

- Expensive to go around the world.
- Takes too long.

A query path is the method the optimizer uses to form tuples (the results of a join between two tables). The optimizer uses cost estimates to determine the optimal query path to use. Each path is assigned a cost estimate based on a formula; the cheapest path is considered the optimal path.

A good query path can minimize the amount of data that will be examined. The more you can narrow down the number of possible rows that satisfy the query in the earlier stages, the less time is spent reading rows in other tables that may not match the query criteria. This is why very small tables, or tables with very restrictive filters (example: order_num = 1001), are usually put early in the query path.

A good query path can also prevent extra sorting for ORDER BY or GROUP BY statements. For example, to prevent an ORDER BY in a SELECT statement from requiring a sort, the optimizer may choose to put the table containing the sort columns first (assuming there is an index on the sort columns and the nested loop join is used for subsequent joins). Since the table will be scanned in the order of the column specified in the ORDER BY, no extra sort process is needed.

Query Optimization Process

- Examine all tables (table A, table B, table C)
  - Examine selectivity of every filter (where clauses)
  - Determine if indexes can be used for filters, order by, group by
  - Find the best way to scan a table -- sequentially or by an index
- Identify Join Pairs (AB, AC, BA, BC, CA, CB)
  - Find best join method (nested loop, hash, or sort merge)
  - Decide which indexes are best for the join
  - Calculate the cost of the join
- Repeat for each additional table (ABC, ACB, BAC, ...)

Estimating costs ** -- Need data!

- Find the cheapest/lowest cost path.
  - Cost = I/O cost + Weight * (CPU cost)
  - I/O -- disk access
  - CPU -- Rows processed

- Estimate costs
  - Filters -- Which indexes to be used?
  - Joins -- Nested Loop, Hash, or Sort Merge?
  - Eliminate redundant pairs?

The three basic strategies for joining two tables -- nested loop join, hash joins, sort merge join.

In a nested loop join, the first table is scanned, and the results are matched to the corresponding columns in the second table. The process would work something like this -- fetch a from the first table, use the join column(s) to search for rows in the second table that have matching join column values.

In a sort merge join, both tables are scanned in the order of the join filter. It would work something like this -- sort each table by the join column, then merge the rows from each table into the resulting join tuples.

In a hash join, a sequential scan of the first table is done to build a hash table, then rows from the second table are looked up in the hash table to create tuples. Typically, the hash table is created on the smaller table. Hash joins can be faster than the other join methods, especially where the size of the join tables are very large.

Filter selectivity assignments

- Selectivity is the percentage of rows selected as a result of a filter (number between 0 and 1)

Example:

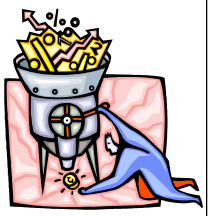
Expression	Filter Selectivity
indexed_col = literal value	F=1/(number of distinct keys in index)
indexed_col > literal value	F = (literal value - 2nd min)/(2nd max-2nd min)
NOT expression	F = 1 - F(expression)
expr1 AND expr2	F = F(expr1) x F(expr2)

Selectivity is a number between 0 and 1 -- closer to 0, more selective.

Look at what kinds of data are needed to figure out how selective a filter is -- number of distinct keys in an index, 2nd min, 2nd max, etc.

How do we Influence Query Optimization?

- OPTCOMPIND
- Optimizer directives (Optimization Goals)
- Statistics (Update Statistics) ***
  - Collect information for the optimizer
  - Statistics -- Update Statistics LOW
  - Distributions -- MEDIUM & HIGH
  - Drop distributions
  - Compile Stored Procedures



OPTCOMPIND -- ONCONFIG parameter (default is 2); environment variable.

2 -- The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

0 -- Use index if available; give preference to nested-loop join (don't do hash joins).

1 -- Behave like 0 if isolation level is Repeatable Read; otherwise, behave like 2.

Optimizer directives:

Example: SELECT {+INDEX(emp idx_dept_no)} ... ;

Optimization goals (ALL_ROWS -- optimize for the total execution time of the query -- or FIRST_ROWS)

Example:

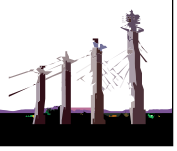
OPT_GOAL ONCONFIG parameter

OPT_GOAL env. variable.

SET OPTIMIZATION FIRST_ROWS;

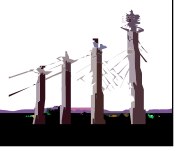
Update Statistics Low Basic Algorithm

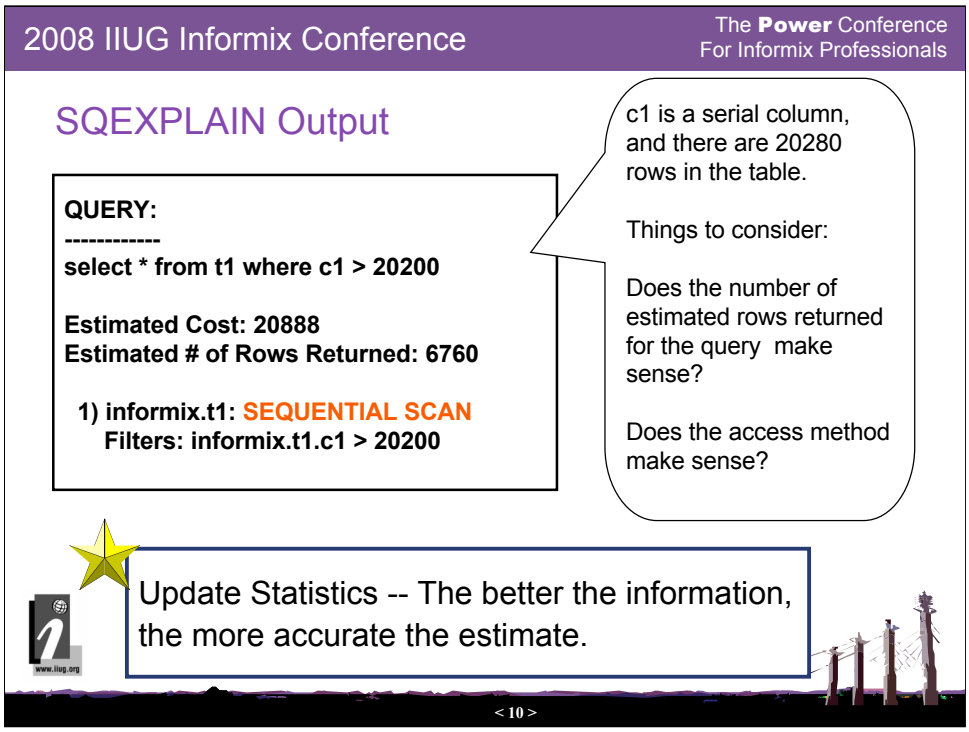
- Walk the leaf pages in each index
- Submit btree cleaner requests when deleted items are found causing indexes to be re-balanced
- Collects the following information
  - Number of unique items
  - Number of leaf pages
  - How clustered the data is
  - Second highest and lowest value



Update Statistics Basic Algorithm for Medium and High

- Develop scan plan based on available resources
- Scan table
  - High ----> All rows
  - Medium ----> Sample of rows
- Sort each column
- Build distributions
- Begin transaction
  - Delete old columns distributions
  - Insert new columns distributions
- Commit transaction





Sequential scans can consume a large quantity of resources so validate that they are appropriate

Sequential scans **are appropriate** if most or all of the table rows are being returned

Example: Update Statistics will do sequential scans

If only a small percentage of the rows are being returned (but a sequential scans are being done), then this indicates that **you may need better indexes** on the table(s).

Find all tables doing sequential scans having a size greater than 100 pages:

```
select dbsname, tabname ,
   pf_seqscans , npdata, npused
   from sysptntab, systabnames, sysptnhdr
   where pf_seqscans > 0
   and sysptnhdr.npdata > 100
   and sysptnhdr.npused > 100
   and sysptnhdr.partnum = systabnames.partnum
   and systabnames.partnum = sysptntab.partnum
   order by pf_seqscans DESC
```

Update Statistics Example

- No Statistics vs Medium Statistics

```
QUERY:
------------
select * from t1 where c1 > 20200

Estimated Cost: 20888
Estimated # of Rows Returned: 6760

  1) informix.t1: SEQUENTIAL SCAN
      Filters: informix.t1.c1 > 20200
```

```
QUERY:
------------
select * from t1 where c1 > 20200

Estimated Cost: 21
Estimated # of Rows Returned: 19

1) informix.t1: INDEX PATH
(1) Index Keys: c1  (Serial,fragments: ALL)
  Lower Index Filter: t1.c1 > 20250
```

- Query plan changed to Index Path for Medium Statistics (vs No Statistics), huge improvement in estimated cost and estimated number of rows (actual number of rows returned is 30)

Update Statistics Example

- Medium vs High Statistics

```
QUERY:
------------
select * from t1 where c1 > 20200

Estimated Cost: 21
Estimated # of Rows Returned: 19

1) informix.t1: INDEX PATH
(1) Index Keys: c1  (Serial,fragments: ALL)
  Lower Index Filter: t1.c1 > 20250
```

```
QUERY:
------------
select * from t1 where c1 > 20200

Estimated Cost: 33
Estimated # of Rows Returned: 30

1) informix.t1: INDEX PATH
(1) Index Keys: c1  (Serial,fragments: ALL)
  Lower Index Filter: t1.c1 > 20250
```

- Query plan did not change, and there is no significant change in performance.

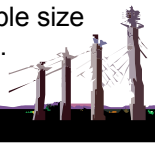
Why Improve Create Index?

- Customers have to run UPDATE STATISTICS LOW at a minimum for newly created indexes to be considered for query access plans.
- Customers are recommended to run UPDATE STATISTICS on table immediately after they create index on the table.

Why can't these be done automatically when new indexes are created?

Create Index Change Summary

- When the CREATE INDEX statement runs successfully, IDS automatically gathers the following statistics for the newly created index:
  - **Index-level statistics**, equivalent to the statistics gathered in the UPDATE STATISTICS operation in LOW mode, for all types of indexes, including B-tree, Virtual Index Interface (VII), and functional indexes.
  - **Column-distribution statistics**, equivalent to the distribution generated in the UPDATE STATISTICS operation in MEDIUM or HIGH mode, for a non-opaque leading indexed column of an ordinary B-tree index. The resolution is 1.0 for a table size that is less than 1 million rows and 0.5 for larger tables.



Implicit or explicit CREATE INDEX automatically calculates distribution statistics for the leading column of the index in HIGH mode, as well as index statistics and table statistics in LOW mode. This feature is always enabled -- cannot turn off.

The automatically gathered distribution statistics are available to the query optimizer when it designs query plans for the table on which the new index was created.

Important!

- Please note that a separate “Update Statistics” command is **not** run to gather any of the statistics and distribution info.
- The optimizer information is gathered during the index building process -- done in an optimized way so that performance is not impacted.
- Internally, we gather the same kind of information **as if** we had run update statistics.

The optimizer information is gathered during the index building process. Internally, we gather the same kind of info **as if** we had run update statistics. Because this is done in an optimized way, the statistics and distribution gathering should not impact performance -- performance should be similar to what it was previous to the new feature(s).

Example:

- CREATE INDEX idx1 on tab1(b_integer)
  - Automatically creates distribution and statistics for the leading column of the index (idx1)
  - Equivalent to running UPDATE STATISTICS HIGH to create distribution for leading index (idx1) column
  - Equivalent to running UPDATE STATISTICS LOW to update index (idx1) and table statistics

SET EXPLAIN Output for CREATE INDEX statement:

```
Index:          idx1 on informix.tab1
STATISTICS CREATED AUTOMATICALLY:
Column Distribution for:        informix.tab1.b_integer
Mode:           HIGH
Number of Bins:     207 Bin size:  4800.0
Sort data:           0.9 MB
Completed building distribution in:     0 minutes 1 seconds
```

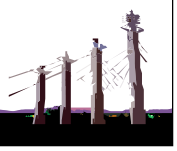
onstat -g ses <sid>

```
IBM Informix Dynamic Server Version 11.50.FCB4    -- On-Line -- Up 2 days 05:20:34 -- 250944
    Kbytes

session           effective                              #RSAM    total      used       dynamic
id       user     user      tty      pid      hostname threads  memory     memory     explain
26       informix -         0        475376   rat-06   5        352256     338128     off

tid       name      rstcb             flags    curstk   status
112       sqlexec   70000001031f8a0   Y-BP---  17936    cond wait  opened_up -
113       mb_colle  7000000103200a8   --B----  2016     sleeping secs: 1     -
117       xchg_1.0  7000000103208b0   Y-B----  1760     cond wait  opened_up -
118       xchg_2.0  7000000103210b8   --B----  1600     running-
119       xchg_3.0  7000000103218c0   --B-R--  1648     ready-

...
```



New thread -- mb_collector -- mini-bin collector for collecting distribution info.

New memory pool in onstat -g mem -- “mb_<sid>”.

Details on distribution creation

- To create distribution information on the leading column of the index, the CREATE INDEX statement automatically gathers statistics by sorting the leading column and building distribution bins from the sorted data.
  - The distribution information can be viewed using the "dbschema -hd" option, which displays the information stored in the database sysdistrib catalog table.
  - Although very similar to UPDATE STATISTICS HIGH, the distribution creation process is not identical to UPDATE STATISTICS HIGH.
    - For tables with less than a million rows, a resolution of 1 is used, so that the number of distribution bins created is roughly 100.
    - For tables with more than a million rows, resolution of 0.5 is used, so that the number of distribution bins created is roughly 200.

Example: Distribution Bin Info

- dbschema -d testdb -hd all

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 11.10.FC1
Copyright IBM Corporation 1996, 2007 All rights reserved
{
Distribution for informix.tlrtab.b_integer
Constructed on 2007-05-29 13:42:45.00000
High Mode, 0.500000 Resolution

--- DISTRIBUTION ---
      (                     NULL)
   1: (    4800,     4800,  -2079917777)
   2: (    4800,     4800,  -2059740802)
   3: (    4800,     4799,  -2039581307)
   4: (    4800,     4798,  -2019086003)
...
 205: (    4800,     4800,   2074411864)
 206: (    4800,     4799,   2094676598)
 207: (    1264,     1264,   2099993300)

--- OVERFLOW ---

   1: (   10123,            NULL)
}
```

-- number of values in each bin, the number of distinct values in each bin, and the high value in the bin

-- frequency and value

How to read Distributions

# of rows represented in this bin

```
   --- DISTRIBUTION ---
   (                        -1
1: (    868317,      70,      75)
2: (    868317,      24,     100)
3: (    868317,      12,     116)
4: (    868317,      30,     147)
5: (    868317,      39,     194)
6: (    868317,      28,     222)
    --- OVERFLOW ---
1: (    779848,                43)
2: (    462364,                45)
```

# of unique values

Highest Value in this bin

To get the range of values look at the highest value in the previous bin.

The value

# of rows for this value

Example - Approximating a Value

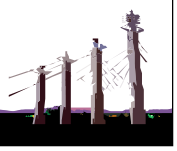
```
   --- DISTRIBUTION ---
   (                        -1
1: (    868317,        70,         75)
2: (    868317,        24,        100)
3: (    868317,        12,        116)
4: (    868317,        30,        147)
5: (    868317,        39,        194)
6: (    868317,        28,        222)
    --- OVERFLOW ---
1: (    779848,                   43)
2: (    462364,                   45)
```

- There are 868317 rows containing a value between -1 and 75
- There are 70 unique values in this range
- The optimizer will deduce 868317 / 70 = 12,404 records for each value between -1 and 75

Example - Dealing with Data Skew

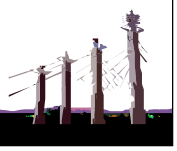
```
   --- DISTRIBUTION ---
   (                        -1
1: (    868317,      70,      75)
2: (    868317,      24,     100)
3: (    868317,      12,     116)
4: (    868317,      30,     147)
5: (    868317,      39,     194)
6: (    868317,      28,     222)
    --- OVERFLOW ---
1: (    779848,              43)
2: (    462364,              45)
```

- Data skew
- For the value 43 how many records will the optimizer estimate will exist?
- Answer 779848 values
- Any value that exceeds 25% of the bin size will be placed in an overflow bin



Update Statistics LOW (what it does)

- UPDATE STATISTICS LOW updates the following info (table and index statistics in the database catalog tables):
  - systables (nrows, npused, ustlowts)
    - nrows -- number of rows in the table
    - npused -- number of pages on disk used for table
    - ustlowts -- the last time update statistics low (implicit or explicit) was run
  - sysindices(levels, leaves, nunique, clust, nrows)
    - leaves -- number of pages on the 0 level of the B+ tree
    - levels -- number of b-tree levels
    - nunique -- number of unique key values (determines selectivity of equality filters)
    - clust -- degree of clustering
    - sysindexes is a view that has similar info as sysindices



systables (nrows, npused, ustlowts)

nrows -- number of rows in the table

npused -- number of pages on disk used for table

ustlowts -- the last time update statistics low (implicit or explicit) was run

sysindices(levels, leaves, nunique, clust, nrows)

leaves -- number of pages on the 0 level of the B+ tree

levels -- number of b-tree levels

nunique -- number of unique key values (determines selectivity of equality filters)

clust -- degree of clustering

sysindexes is a view that has similar info as sysindices

syscolumns(colmin, colmax)

colmin -- second minimum value of column

colmax -- second maximum value of column

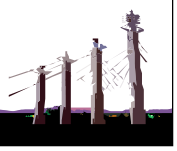
sysfragments

Similar info as systables for fragmented tables

Similar info as sysindices for fragmented indexes

Update Statistics LOW (continued)

- UPDATE STATISTICS LOW updates the following info (table and index statistics in the database catalog tables):
  - syscolumns(colmin, colmax)
    - colmin -- second minimum value of column
    - colmax -- second maximum value of column
  - sysfragments
    - Similar info as systables for fragmented tables
    - Similar info as sysindices for fragmented indexes



systables (nrows, npused, ustlowts)

nrows -- number of rows in the table

npused -- number of pages on disk used for table

ustlowts -- the last time update statistics low (implicit or explicit) was run

sysindices(levels, leaves, nunique, clust, nrows)

leaves -- number of pages on the 0 level of the B+ tree

levels -- number of b-tree levels

nunique -- number of unique key values (determines selectivity of equality filters)

clust -- degree of clustering

sysindexes is a view that has similar info as sysindices

syscolumns(colmin, colmax)

colmin -- second minimum value of column

colmax -- second maximum value of column

sysfragments

Similar info as systables for fragmented tables

Similar info as sysindices for fragmented indexes

Examples

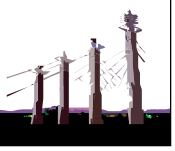
- The following SQL statements will automatically create distributions and statistics
  - CREATE INDEX idx1 ON tab1 (col1);
  - ALTER FRAGMENT FOR TABLE tab2 INIT …. (if table has indexes)
  - ALTER FRAGMENT FOR INDEX idx2 INIT …
  - ALTER TABLE ADD UNIQUE CONSTRAINT …
- CREATE INDEX does not create distributions for VII indexes, functional indexes, or indexes on columns of user-defined data types
  - Manually run the UPDATE STATISTICS command in MEDIUM or HIGH mode to create distribution information on tables that have the above type of indexes.

Details on table and index statistics

- Example

```
create table tab1(col1 int, col2 int);
1) create index idx1 on tab1(col1);
load from tab1.unl insert into tab1;
2) create index idx2 on tab1(col2);
```

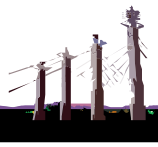
- The first create index is equivalent to running “UPDATE STATISTICS LOW for table tab1”.
- The second create index is not equivalent to running “UPDATE STATISTICS LOW for table tab1” as it will not update index level statistics for existing index idx1.
  - Will not update syscolumns(colmin, colmax) for col1
  - Will not update sysindices, sysindexes for idx1



The only time UPDATE STATISTICS LOW FOR TABLE will still be required after a CREATE INDEX would be if the table had other pre-existing indexes.

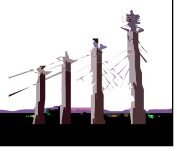
Bottom Line -- What does this mean for me?

- Creation of distributions and statistics during index creation is automatic and cannot be disabled.
- When you create an index, the optimizer will immediately have information on that index and may start using it.
- If you have good Update Statistics Practices, should not need to modify.
- If you do not keep any distribution info and do not want distribution info, then you may need to drop the distributions
  - Update Statistics Drop Distribution ONLY



Limitations to Create Index Distributions

- Index distributions are NOT created when --
  - Undocumented environment variable NOSORTINDEX forces a top down index build which disables this feature.
  - If the lead of the index is a UDT (builtin or non-builtin) as this forces top down index build.
  - Index is a functional index.
  - Index is a VII index.
  - Number of rows in table is < 2.



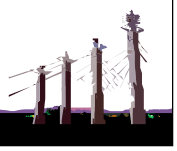
Implementation of Create Index Distribution

- Leverages the sorted data produced by create index
  - Each sort stream creates mini distribution bins
  - Ships the mini-bin via queue to a mini-bin collector thread
  - Mini-bin collector thread sorts mini bins
  - Merges the mini-bins into a final distribution bin.

- Sample size is the data seen during the index build.
  For online index builds, it's data seen during the static phase of the index build, "catch-up" data is ignored.
  - Resolution of 1 (~100 distribution bins) for tables with rows < 1 million
  - Resolution of 0.5 (~ 200 distribution bins) for larger tables

Q's

- Q1) Are distributions for indexes created during dbimport?
- Q2) Are distributions built during CREATE INDEX dropped when the index is dropped?
- Q3) You've created an index, and you do not want distributions, what do you do? How do you check to see if the distributions are gone?



Answers

- Q 1) During the dbimport operation, distribution information is created automatically for non-opaque leading indexed column of an ordinary B-tree index.
- Q 2) Distributions are not dropped when the index is dropped since the distribution information for the column can still be used.
- Q 3) Use the command “UPDATE STATISTICS for table tab1 drop distributions ONLY”. systables tab1 entry should not show a new time stamp for update statistics low (ustlowts). There should not be any entry for table tab1 in sysdistrib.

Real-time Temp Table Statistics

Why Improve Temp Table Statistics?

- **Update Statistics gather information about the table data to help the optimizer make decisions about how to run the query.**
- **Whenever temp tables were used, performance could be greatly affected by whether or not Update Statistics was run on the temp table.**
- **Many people forget to run Update Statistics LOW (as recommended) on a temp table.**
- **Looking at slow running queries with lots of temp tables, it is difficult to figure out which temp tables need Update Statistics...**

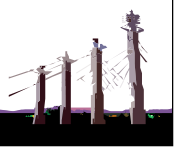
Temp Table Statistics Improvements

- No longer need to run Update Statistics Low on temp tables to get the benefits of running that command.
- AUTOMATIC update of temp table dictionary info.
  - Number of rows and pages is updated every time we access the temp tables data dictionary entry.
  - Adding indexes to temp tables will automatically create distributions for the leading column of the index and statistics for the temp table.

DDL -- data definition language.

Important!

- Please note that a separate “Update Statistics” command is **not** run to gather any of the statistics and distribution info.
- For the automatic update of temp table dictionary info -- the number of rows and pages is updated every time we access the temp tables data dictionary entry.



Details

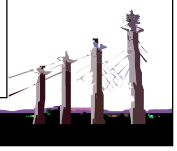
- If you use a lot of temp tables and never ran Update Statistics on temp tables ...
  - You should notice a performance improvement.
- If you use a lot of temp tables and create btree indexes on temp tables, and never ran Update Statistics Medium or High to create distributions ...
  - You may see a slight increase in session memory usage since temp table distributions will be created and kept in memory. However, distribution info really aren't that big so the small bit of extra memory shouldn't be noticeable.

Example (Auto Distribution with Create Index on Temp Table)

- Create temp table tab1(col1 int, col2 int);
- Insert into tab1 select tabid, tabid from systables;
- Set Explain On;
- Create index idx1 on tab1(col1);

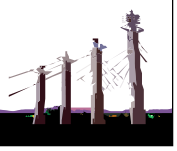
```
CREATE INDEX:
=============

Index:          idx1 on informix.tab1
STATISTICS CREATED AUTOMATICALLY:
Column Distribution for:                informix.tab1.col1
Mode:           HIGH
Number of Bins:      65 Bin size:     1.0
Sort data:           0.0 MB
Completed building distribution in:     0 minutes 0 seconds
```



Temp Tables Questions

- Are temp tables listed in the database systables?
- Can you see temp tables with onstat -g dic ?
- How do you check if an update statistics has been run on a temp table?



Temp Tables Answers

- Temp Tables are private to a session.
  - Temp tables are not listed in systables.
  - Temp table data dictionaries are private to the session. They are not put in the shared dictionary cache.

- How do you check if an update statistics has been run on a temp table?
  - Set Explain will show that distributions are being created when btree indexes are created on the temp table.
  - Also, you may see improved query plans from previous IDS versions (if you've never run Update Statistics on temp tables).

Enhanced Catalog Info for Update Statistics

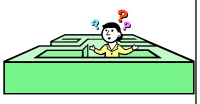
Why Enhance Catalog Data?

- **Missing administrative info on update statistics**

When was the last time UPDATE STATISTICS LOW was run?

What time were the distributions created?

How many rows were sampled for the distributions?



Schema additions to systables table

- **systables** is a database catalog tables
  - systables holds information about tables in the database
- New column in **systables** to track when the last update statistics LOW was run on the table

ustlowts	DATETIME YEAR TO FRATION(5)

This column is updated whenever Update Statistics LOW is run explicitly or implicitly.

Schema additions to sysdistrib table

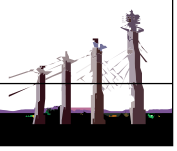
- **sysdistrib** is a database catalog tables
  - sysdistrib holds information about update statistics distributions on table columns
- New columns in **sysdistrib** to track more detailed information about distributions created with update statistics MEDIUM/HIGH

smplsize	INTEGER	(if 0, the smplsize was not specified)
rowssmpld	INTEGER	(number of rows sampled)
constr_time	DATETIME YEAR TO FRACTION(5)	
ustnrows	FLOAT	

Example

- select * from systables where tabname = 'mytab';

```
tabname          mytab
owner            informix
partnum          1051914
tabid            106
rowsize          8
ncols            2
nindexes         0
nrows            39006.00000000 <-- the number of rows in the table
created          06/07/2007
version          6946819
tabtype          T
locklevel        P
npused           233.0000000000
...
pagesize         2048
ustlowts         2007-06-07 18:51:03.00000
secpolicyid      0
protgranularity
```



Example

```
tabid         106
colno         1
seqno         1
constructed   06/07/2007
mode          M
resolution    0.800000012
confidence    confidence   0.949999988
smplsize      0.00   <--- user did not specify, default
                          based on resolution and confidence
                          used by the optimizer

rowssmpld     28931.00000000   <--- number of rows sampled
                                    during distribution build

constr_time   2007-06-07 18:51:03.00000
ustnrows      39006.00000000   <-- number of rows in table
                                   during distribution build
```

- select * from sysdistrib where tabid = 106 and colno = 1

For update statistics medium, unless sampling size is specified, the smplsize is determined by the resolution and the confidence. For update statistics high, confidence is not used, and the smplsize is the number of rows in the table at the time of the distribution build.

Example

If sysdistrib.smplsize is not 0, the 'sampling size' would also be shown.

```
DBSCHEMA Schema Utility
...
{
Distribution for informix.mytab.col1

Constructed on 2007-06-07 18:51:03.00000

Medium Mode, 0.800000 Resolution, 0.950000 Confidence

--- DISTRIBUTION ---

      (                     1048577)
   1: (   312,      52,      1048628)    <--- 312 values in bin 1,
   2: (   312,      52,      1048680)         52 distinct values in bin 1, and
   3: (   312,      55,      1048735)         1048628 is the largest value
   4: (   312,      52,      1048788)         in bin 1.
   5: (   312,      53,      1048840)
...
```


dbschema -d testdb -hd mytab

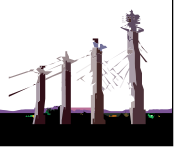
Improved Set Explain

What's New with Set Explain?

- **Set Explain outputs have limited info.**
- **Can't turn on Set Explain dynamically.**
- **Why can't I specify which file to put the explain output to?**

New Set Explain Features

- **Dynamically turn on/off Set Explain**
  - onmode -Y <session_id> [0|1|2]
  - 0 -- turn off
  - 1 -- turn on
  - 2 -- turn on but only display the query plan (skip query statistics)
- **SET EXPLAIN FILE TO command allows users to control where the explain file is generated.**
  - SET EXPLAIN FILE TO '/work/sq1exp.out';



SET EXPLAIN FILE TO command now allows users to control where the explain file is generated

SET EXPLAIN FILE TO '/work/sq1exp.out' ;

Turns on SET EXPLAIN and sets the output file to '/work/sq1exp.out' .

<u>If only the filename is specified (on UNIX):</u>

If the client application is on the same machine as the database server, the explain output file is saved in the current working directory (of where the client application was started). If the client application is on a remote machine, the explain output is saved to the home directory of the application user on the machine where the database server is running.

If the file exists, data is appended to the file.

onmode -Y details

- onmode -Y <sessionid> 2
  - Turns dynamic explain on for <sessionid> and displays the query plan only (does not display query statistics)
- onmode -Y <sessionid> 1
  - Turns dynamic explain on for <sessionid> (displays the query plan and query statistics)
- onmode -Y <sessionid> 0
  - Turns dynamic explain off for <sessionid>
- Explain output goes to file '*sqexplain.out.<sessionid>'* in current working directory (on UNIX).

On Unix, if the client application is on the same machine as the database server, the explain output file is saved in the current working directory (of where the client application was started). If the client application is on a remote machine, the explain output is saved to the home directory of the application user on the machine where the database server is running.

On Windows, the explain output, by default, goes to %INFORMIXDIR%\sqexpln directory.

onstat -g ses

```
session                                    #RSAM    total      used      dynamic
id    user     tty   pid   hostname  threads  memory     memory    explain
29    informix -     0     -         0        12288      11736     off
28    informix 3     595   mach.ibm  1        57344      51368     on
21    informix -     0     -         1        311296     260976    off
```

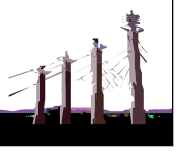
```
Sess  SQL         Current     Iso Lock        SQL   ISAM F.E.
Id    Stmt type   Database    Lvl Mode        ERR   ERR  Vers  Explain
28    -           expl_db     NL  Not Wait    0     0    9.35  Dyn. stat
```

```
Sess  SQL         Current     Iso Lock        SQL   ISAM F.E.
Id    Stmt type   Database    Lvl Mode        ERR   ERR  Vers  Explain
28    -           expl_db     NL  Not Wait    0     0    9.35  Dyn. plan
```


The SQEXPLAIN setting will allow you to capture insert/update/select SQL statements, but it will not capture statements like "Execute function", "Execute procedure“, "CREATE TABLE” and “DROP TABLE”.

Set Explain Feature -- Query Statistics

- **New ONCONFIG parameter EXPLAIN_STAT**
  - 1 is ON; 0 is OFF (default is ON)
  - When enabled, includes query statistics information in your explain output file.
  - EXPLAIN_STAT can be set using onmode -wm and onmode -wf
    - onmode -wf EXPLAIN_STAT=1
    - onmode -wm EXPLAIN_STAT=1



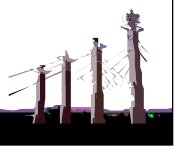
The default value for EXPLAIN_STAT is 1 (ON or Enabled) -- this means that if EXPLAIN_STAT is not in the ONCONFIG, it is ON.

In IDS 11.10, EXPLAIN_STAT is 1 in onconfig.std.

In IDS 11.50, EXPLAIN_STAT is 0 in onconfig.std.

More on Query Statistics

- SET EXPLAIN STATISTICS is available for backward compatibility, but it is superseded by the new SET EXPLAIN ON



Query Statistics section

- The Query Statistics section of the SET EXPLAIN output shows the estimated number of rows that the query plan expects to return, the actual number of returned rows, and other information about the query.
- The Query Statistics section, which also gives you an indication of the overall flow of the query plan and how many rows flow through each stage of the query, can be useful for debugging performance problems.
- If the "estimate" and "actual" number of rows scanned and/or joined are way off, it can be an indication that the Update Statistics info on those tables are out of date and need to be updated.

Query Statistics Example

```
QUERY:
------
select {+USE_NL(tab1)} tab1.c1 , tab1.c2 from tab1, tab2
  where tab1.c1 = tab2.c1 and tab1.c4 < 200

Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                tab2
  t2                tab1

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     10000      10000     10000      00:00:00   566

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     26         90        1          00:00:00   0

  type     rows_prod  est_rows  time       est_cost
  ----------------------------------------------------
  nljoin   26         88        00:00:00         5591
```


The query plan section has been deleted.

SQEXPLAIN Output with PDQ

```
QUERY: (OPTIMIZATION TIMESTAMP: 03-28-2008 09:22:05)
------
select {+ORDERED} * from tab5, tab4, tab3, tab2, tab1
where tab1.c1 = tab2.c1
AND   tab2.c1 =  tab3.c1 AND   tab3.c1 = tab4.c1
AND   tab4.c1 = tab5.c1  AND   tab1.c1 < 800

DIRECTIVES FOLLOWED:
ORDERED
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 955
Estimated # of Rows Returned: 1
Maximum Threads: 8
```

This is just the very first part of the sqexplain output. The query plan and the query statistics parts are not shown.

SQEXPLAIN Output with Remote Queries

```
select c.customer_num, o.order_num, i.item_num
from stores2@ids_1111:informix.customer c,
        stores2@ids_1111:informix.orders o,
        stores1@ids_1110uc2:informix.items i  <------- remote database server
where c.customer_num = o.customer_num AND
          o.order_num = i.order_num AND
          c.customer_num = 104

  ...
 1) informix.o: INDEX PATH
  (1) Index Keys: customer_num   (Serial, fragments: ALL)
     Lower Index Filter: informix.o.customer_num = 104

 2) informix.i: REMOTE PATH
  REMOTE SESSION ID FOR 'ids_1110uc2' is 35   <------------ remote session #

Remote SQL Request:
select x0.item_num ,x0.order_num from stores1:"informix".items x0
where (? = x0.order_num )  <---------------- using a host variable
...
```



```
QUERY: (OPTIMIZATION TIMESTAMP: 03-31-2008 14:53:04)
------
select c.customer_num, o.order_num, i.item_num
from stores2@ids_1111:informix.customer c, stores2@ids_1111:informix.orders o,
stores1@ids_1110uc2:informix.items i
where c.customer_num = o.customer_num AND
o.order_num = i.order_num AND c.customer_num = 104

Estimated Cost: 5
Estimated # of Rows Returned: 3

  1) informix.o: INDEX PATH
    (1) Index Keys: customer_num   (Serial, fragments: ALL)
       Lower Index Filter: informix.o.customer_num = 104
  2) informix.i: REMOTE PATH
    REMOTE SESSION ID FOR 'ids_1110uc2' is 35
    Remote SQL Request:
    select x0.item_num ,x0.order_num from stores1:"informix".items x
    0 where (? = x0.order_num )
NESTED LOOP JOIN
  3) informix.c: INDEX PATH
    (1) Index Keys: customer_num   (Key-Only)  (Serial, fragments: ALL)
       Lower Index Filter: informix.c.customer_num = informix.o.customer_num
NESTED LOOP JOIN
```

Query Statistics on Local

```
type     table  rows_prod  est_rows  rows_scan  time       est_cost
-------------------------------------------------------------------
scan     t1     4          1         4          00:00.00   1

type     rows_prod  est_rows  time       est_cost
-----------------------------------------------
remote   9          67        00:00.06   0

type     rows_prod  est_rows  time       est_cost
-----------------------------------------------
nljoin   9          3         00:00.07   4

type     table  rows_prod  est_rows  rows_scan  time       est_cost
-------------------------------------------------------------------
scan     t2     9          1         9          00:00.00   0

type     rows_prod  est_rows  time       est_cost
-----------------------------------------------
nljoin   9          4         00:00.07   5
```

t1 = orders
t2 = customer

Query statistics:

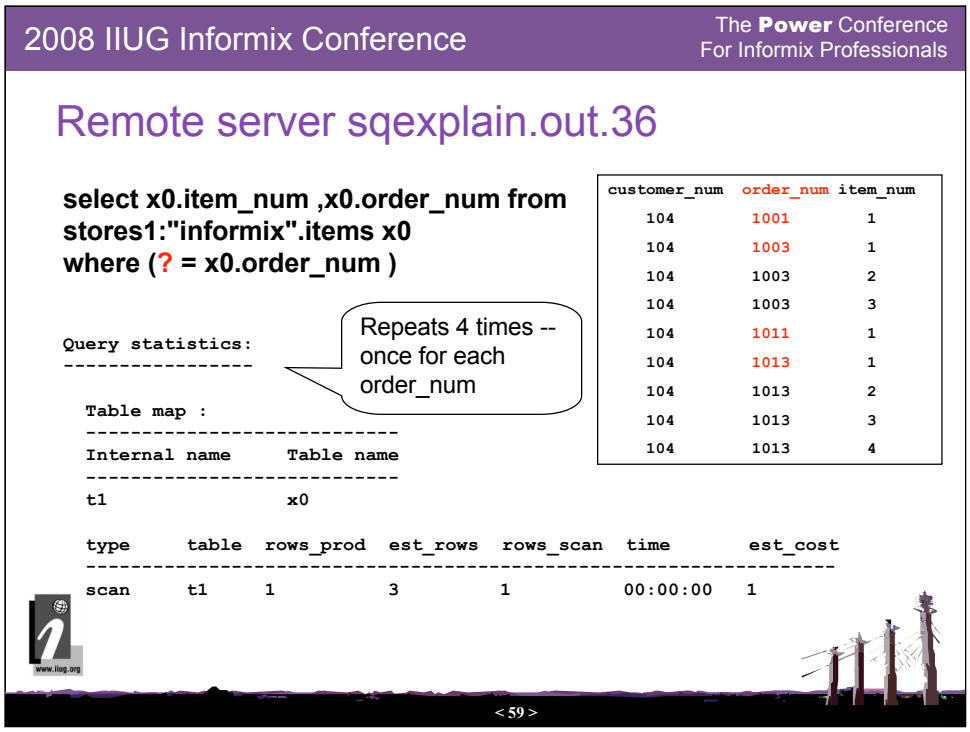
```
----------------
 Table map :
 ----------------------------
 Internal name    Table name
 ----------------------------
 t1               o
 t2               c
 type    table  rows_prod  est_rows  rows_scan  time      est_cost
 ------------------------------------------------------------------
 scan    t1     4          1         4          00:00.00  1

 type    rows_prod  est_rows  time      est_cost
 ----------------------------------------------
 remote  9          67        00:00.06  0

 type    rows_prod  est_rows  time      est_cost
 ----------------------------------------------
 nljoin  9          3         00:00.07  4

 type    table  rows_prod  est_rows  rows_scan  time      est_cost
 ------------------------------------------------------------------
 scan    t2     9          1         9          00:00.00  0

 type    rows_prod  est_rows  time      est_cost
 ----------------------------------------------
 nljoin  9          4         00:00.07  5
```



QUERY: (OPTIMIZATION TIMESTAMP: 03-31-2008 15:05:47)
---
select x0.item_num ,x0.order_num from stores1:"informix".items x0 where (? = x0.
order_num)

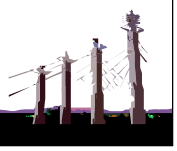
Estimated Cost: 1
Estimated # of Rows Returned: 3
1) informix.x0: INDEX PATH
(1) Index Keys: order_num (Serial, fragments: ALL)
Lower Index Filter: informix.x0.order_num = 1001

Query statistics: --> This section repeats 4 times, once for each order_num
---
Table map :

Internal name Table name
---
t1 x0
type table rows_prod est_rows rows_scan time est_cost
---
scan t1 1 3 1 00:00:00 1
scan t1 4 3 4 00:00:00 1 -- from 2nd Query statistics
scan t1 5 3 5 00:00:00 1 -- from 3rd Query statistics
scan t1 9 3 9 00:00:00 1 -- from 4th Query statistics

Q's

- In the query statistics output, how can you tell which is the most expensive part of each query run?
- In the query plan section, how can you tell if PDQ is set?
- By default, where is the sqexplain output file created on Windows?

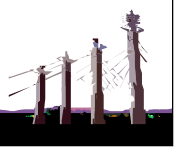


Answer 1) You can look at the "est_cost" for each iterator/step.

Answer 2) sqexplain output displays "Maximum Threads" info.

Answer 3) %INFORMIXDIR%\sqexpln directory, file named <userid>.out .

Enhanced Support for Large Tables



Issue 1 -- MAXINT Overflow

- **Several database catalog table fields used an integer field to store information about total number of rows in a table (and related entries such as number of unique values).**
- **For a large fragmented table, the number of rows can exceed MAXINT.**
- **During optimizer calculations, this can result in overflow of the integer fields in the database catalog tables and cause bad cost calculations, resulting in bad query plans.**

Fixed MAXINT Overflow

- The following catalog table entries are changed from integer to double:
  - **systables**: nrows, npused
  - **sysindices**: leaves, nunique, clust, (nrows was already double)
  - **sysfragments**: nrows, npused, clust

```
dbaccess testdb -

> info columns for systables;

Column name          Type                    Nulls
tabname              varchar(128,0)          yes
...
nrows                float                   yes
...
npused               float                   yes
...
```

Issue 2 -- Sample Size Confusion

- One misconception held by many DBAs is that the sample size for UPDATE STATISTICS Medium is based on the table size.
- **The sample size is only based on the confidence and resolution.**

Resolution	Confidence	Samples
2.5	.95	2,963
2.5	.99	4,273
1.0	.95	18,516
1.0	.99	26,569
0.5	.95	74,064
0.5	.99	106,276

Resolution	Confidence	Samples
.25	.95	296,255
.25	.99	425,104
.1	.95	1,851,593
.1	.99	2,656,900
0.05	.95	7,406,375
0.05	.99	10,627,600

The resolution and confidence are used to determine the sample size for Update Statistics Medium, but it is not easy to see what the actual sample size will be. A range of resolutions, confidences and the associated sample size is shown above. The default resolution of 2.5 and confidence of .95 yields a sample size of 2963 rows, regardless of the table size.

Allow 'SAMPLING SIZE' Option to Update Statistics

in IDS 11

- UPDATE STATISTICS MEDIUM syntax has a new option 'sampling size'.
  - UPDATE STATISTICS MEDIUM SAMPLING SIZE <number>
  - number less than or equal to 1.0 is interpreted as % of number of rows in table to be sampled
  - number greater than 1.0 is interpreted as number of rows to sample
  - User specified SAMPLING SIZE is stored in ***sysdistrib.smplsize***

Allow 'SAMPLING SIZE' Option to Update Statistics (continued)

- User specified sampling size cannot be smaller than the preset sampling size calculated using resolution of 2.5 and confidence of .80 = 1832 rows. (Specified sampling size will not be used if it's too small.)
- Actual number of rows sampled for UPDATE STATISTICS MEDIUM gets recorded in ***sysdistrib.rowssmpld***
- SAMPLING is a new keyword and a table name SAMPLING cannot be used in an UPDATE STATISTICS statement.

Sampling Size Examples

- create table foo (col1 integer)
- Insert … (say 100 rows)

Sample 50% rows
sysdistrib.smplsize = 0.5

- UPDATE STATISTICS MEDIUM FOR TABLE foo (col1) **SAMPLING SIZE 0.5**;
- UPDATE STATISTICS MEDIUM FOR TABLE foo (col1) **SAMPLING SIZE 3000.0**;

Sample 3000 rows
sysdistrib.smplsize = 3000

Update Statistics Improved Tracking

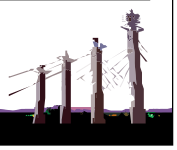
- DBExport and DBSchema have been enhanced to dump out sampling size syntax and value when displaying distributions.

```
dbschema -d testdb -hd mytab

DBSCHEMA Schema Utility       INFORMIX-SQL Version 11.10.UC1

Distribution for informix.mytab.col

Constructed on 2007-06-07 02:42:57.00000
Medium Mode, 4000.000000 Sampling Size, 1.000000 Resolution,
0.900000 Confidence
```



For resolution 1 and confidence of 0.9, the sampling size is 14890.

Update statistics medium for table mytab *sampling size 4000 resolution 1 .9* ; -- Is the specified sampling size used? No, since the resolution and confidence indicate a greater sample size than the one specified by the sampling size option.

dbexport example:

database testdb;

Create table tab1(col1 int, col2 int);

Insert into tab1 select partnum, partnum from sysmaster:sysptnhdr;

Update statistics medium for table tab1(col1) sampling size 0.75;

Update statistics medium for table tab1(col2) sampling size 0.5;

$ dbexport -c -q -o /tmp/testdb -ss testdb

----------dbexport.out----------

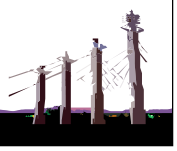
...

update statistics medium for table tab1 (
    col2) sampling size   .50
    resolution   2.50000   0.80000 ;
update statistics medium for table tab1 (
    col1) sampling size   .75
    resolution   2.50000   0.80000 ;

Update Statistics Slow when Dropping Distributions

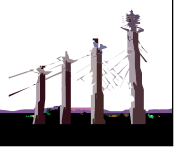
- Currently when you drop distributions it takes a long time because the actions taken include both
  - Dropping the distributions.
  - Updating the statistics (i.e. low mode)
- Solutions is to allow users to ONLY drop distributions
  - The addition of new syntax to allow for the dropping of distributions ONLY

```
UPDATE STATISTICS FOR TABLE tab1
DROP DISTRIBUTIONS ONLY
```



Q's

- Q1: For update statistics medium, unless sampling size is specified, the resolution and confidence is used to calculate the number of rows to be sampled. For update statistics high, what is the number of rows sampled?
- Q2: Use a single update statistics command to remove the distributions, but not run any other update statistics on tab1. How can you show that no statistics were changed on tab1?
- Q3: How many rows will be sampled to gather the distribution info for a column if I run update statistics medium with resolution 2.5 and confidence 0.8 ?



Answer

- Ans 1: For Update Statistics High, the number of rows sampled is the number of rows in the table.
- Ans 2:
  - Update statistics for table tab1 drop distributions only;
  - select ustlowts from systables where tabname = 'tab1' should show no change.
  - select * from sysdistrib where tabid = (select tabid from systables where tabname = 'tab1') should return 0 rows.
- Ans 3:
  - Update statistics medium for table tab1 resolution 2.5 0.8 ;
  - select rowssmpld from sysdistrib where tabid = (select tabid from systables where tabname = 'tab1');
  - Update statistics medium for table tab1 sampling size 1832;

For each column in tab1, get the update statistics mode, resolution, sampling size, number of rows sampled, number of rows in the table when the update statistics was run, and the time when the last update statistics medium or high was run on tab1.

select tabid, colno, mode, resolution, smplsize, rowssmpld, constr_time, ustnrows from sysdistrib where tabid = (select tabid from systables where tabname = 'tab1');

Session D10

Understanding Optimizer Statistics in Cheetah

Hyun-Ju Vega
IBM
vegah@us.ibm.com