Sybase Masterclass:
Update statistics like we mean business

PRIMA DONNA CONSULTING

Joe Woodhouse

UKSUG
SAP DATABASE & TECHNOLOGY USER GROUP

Contents

- About the author
- Why this presentation?
- Why do database statistics matter so much?
- Why are update stats difficult?
- Update stats… in the right places
- Update stats… at the right times
- Update stats… in the right ways

About the author

- Sybase Australia 1996 – 2003; youngest ever Principal Consultant
- After working with Sybase tech for 20+ years… is no longer young
- Principal Database Engineer at Prima Donna Consulting
- Based in London, UK, and Melbourne, Australia
- International Sybase User Group Board of Directors since 2010
- UK Sybase User Group Board of Directors since 2019
- Not a lawyer – no charge for emails!
- Improves client bottom lines by ~USD$10M/month every month

Why this presentation

- Update stats are one of the most frequently run DBA housekeeping
  - … and yet I have never seen a site with no stats problems ever
- It looks easy and simple
  - It is a dangerously misleading topic
- Consequences for getting it wrong: high-priority incidents
- This presentation is practical and pragmatic
  - The theory and description of the stats are in the manuals

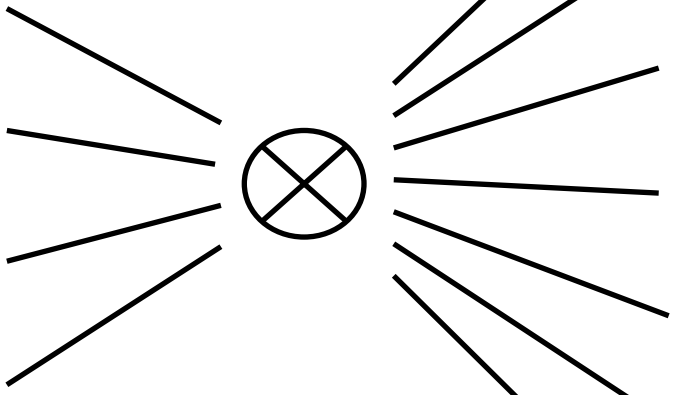
Why do database statistics matter so much?

- It's all about the ASE optimiser
- Unless a statement is one table with no indexes, decisions must be made
- Which access method? Table scan? Index? Which one?
- Which join order?
- Which join method?
- Even a statement with one table with no indexes still needs decisions
  - Parallel query? Lock type? Lock promotions? Fetch-and-discard?

Why do database statistics matter so much?

- These decisions crucially rely on knowing what the data looks like:

- number of pages
- number of rows
- density – how unique?
- distribution – how even?



- per table
- per index
- per column
- per SARG
- per join
- per GROUP BY
- per ORDER BY

Why do database statistics matter so much?

- Number of pages
  - Access methods: how bad is a table scan?
  - Access methods: how good is this index?
- Number of rows
  - Join types: nested loop join starts getting expensive
  - Join order: prefer the smallest table innermost for any loops

Why do database statistics matter so much?

- Density
  - Uniqueness: how many rows will match this value (or range)?
- Distribution
  - How evenly distributed are the values in this column?
  - If we don't know then our predictions get much worse

The cost of missing or incorrect stats

- Missing stats = magic numbers
  - = 10%
  - <, > 25%
  - <=, >= 33%
- Incorrect stats = ASE is misinformed and makes wrong decisions
- Q: Are these scenarios really that bad?
- A: OMG yes
  - One incorrect table scan or join order can bring ASE to its knees

Why this is so difficult to get right

- OK, so missing or wrong stats are bad: run them all the time!
- Update stats consume time & resources, and impact database use
  - There are clear downsides to running them too often
- i.e. this is a Goldilocks problem
  - Run update stats at least as often as required to prevent problems…
  - … and no more often than that

The cost of update stats too much / too often

- Exact cost of update stats depends on which kind of update, but…
  - I/O — table; index(es)
  - memory — procedure cache; data cache
  - tempdb — named tempdbs are bound to login or application
    (how many of you bind sa/DBA to small tempdb?)
  - CPU — small but detectable
  - locks — number of locks; type of locks

Update stats... in the right places

- Three obvious and yet profound truths:
  - Statistics are only necessary when they will change a query plan
  - Statistics don't change if the data doesn't change
  - This means we can't get update stats decisions right without:
    - Knowing all queries against the data, and
    - Knowing what data changes, how, and when

Update stats... in the right places

- Statistics are only necessary when they will change a query plan
  - If there are no indexes on a table, there are no index decisions
  - If there is only one table in a join, there are no join decisions
  - If parallel is disabled (or ruled out), there are no parallel decisions
- Put another way, any query that always table scans a single table...
  - ... doesn't need update stats

Update stats... in the right places

- Statistics don't change if the data doesn't change
  - Read-only tables need stats updated only once
  - Seldom-written tables need stats updated only seldom
  - Append-only tables need update stats only on what is appended
    - Partitions, and only update stats on changed partition(s)?
    - Requires local indexes = problems with global uniqueness

Partitions and local indexes

- Semantic partitions = premium license
- Allows us to define which partition any row lives in
- Allows us to index per partition (local) as well of instead of global
- uniqueness is difficult; needs either:
  - At least one unique global index (defeats the purpose?)
    - But no global index makes quality of global stats worse
  - Partition key must appear in local unique indexes
- If you can make this work, only update stats on changed partitions(s)

Partitions and local indexes

- Fake it with UNION ALL in VIEW?
- Main table + spill table + infrequently batch job to move spill to main
- Most of the time only run update stats on spill table
- Only ever update stats on main table when spill is moved to main
  - Monthly? Quarterly?
  - We don't care if this takes a long time (in terms of locking)
  - We only ever read from main; all writes go to spill table
- Query performance "should" be close; test it first

Update stats... in the right places

- The elephant in the room!
- We don't need to **ever** run update stats in PROD
- Most sites do nightly refresh PROD to UAT / PERF TEST / *somewhere*
  - And if you don't, it's time to start
- Run update stats on one-day-old PROD copy
- optdiag out from UAT
- optdiag in to PROD

Update stats... in the right times

- When to run? As much as is needed, and no more than that
- So when is it needed?
  - When data is changing in ways that change plans
  - When stats are stale
  - When stats are missing
  - When stats weren't updated even though we updated them
- Update stats for free...

When data is changing: datachange()

- Counts inserts and deletes
  - Update = delete + insert = 2 “changes”
  - UPDATE tableA SET colX = colX = 100% “change”
- Expressed as a percentage (can be > 100%)
  - Interpret in context of rowcount
  - 1% x 100 rows = who cares; 1% x 10B rows = we care a lot

When data is changing: datachange()

- Not transactional, rollbacks don't reverse the increment
  - Can be lost from memory
- Refreshed during any polite shutdown
- Refreshed regularly based on "sysstatistics flush interval"...
  - Which defaults to 0 !!!
  - Units are minutes
  - Set to something sensible like 15 or 60
  - You definitely don't need this set to 1

When stats are stale

- Probably the most common cause of stats-related issues
  - Data changes can affect optimiser decisions in non-linear ways
  - i.e. +1% data ⟹ 200x slower because of sudden table scan
  - A clear sign at least some stats aren't updated often enough
- How would we tell?

When stats are stale

- optdiag
- sp_showoptstats
  - Based on sp__optdiag but not as good (thanks Kevin, Raymond)
- select moddate from sysstatistics
- Remember stats are stored per column, not per index
- Check all the columns you care about
- How stale is too stale?
  - Often we can only find out the hard way…

When stats are missing

- **DON'T RUN IN PRODUCTION!!!**
- set option show_missing_stats
- sp_configure "capture missing stats"
  - Does not display them anywhere; sends them to sysstatistics
  - **These are never deleted unless a DBA manually deletes them!**
  - Not human readable, but I've got your back, see next slides
- Only noticed when a query plan is generated
  - **Not captured for plans already in procedure / statement cache!**
- #temp_tables only reliably shown using set option show_missing_stats

When stats are missing (out of the manuals)

```
create   function sp_colidarray
         @colidarray    varbinary(100)
,        @id            int
returns varchar (1500)
as
declare  @s             varchar (1500)
,        @len           int
,        @colid         int
,        @colname       longsysname
set @len = datalength(@colidarray)
while @len> 0
begin
   set @colid = convert(tinyint,
                        substring(@colidarray,@len,1))
   set @colname = col_name(@id, @colid)
   set @s = @colname +
       case @s when NULL then NULL
               else ','
       end + @s
   set @len = @len - 2
end
return @s
```

```
create  proc sp_decode_missing_stats
        @tabname   varchar (100) = '%'
as
select  Dbname = db_name()
,       Tabname = object_name(s.id)
,       NrRows = row_count(db_id(), s.id)
,       ColumnList =
        dbo.sp_colidarray(colidarray, s.id)
,       Captured = moddate
,       Occurs = convert(smallint ,c0)
into    #missing
from    sysstatistics s
,       sysobjects o
where   s.id = o.id
and     object_name(s.id) like @tabname
and     formatid = 110
and     datalength(colidarray) > 0
exec    sp_autoformat
        #missing,
        @orderby="order by 2"
```

When stats are missing (out of the manuals)

```
exec my_db..sp_decode_missing_stats
Dbname   Tabname      NrRows    ColumnList   Captured                Occurs
-------- ---------- -------- ------------ --------------------- ------
mydb     my_table    7800      col_a        May 26 2009 8:26AM   14
mydb     your_table  5         col_b, col_c May 26 2009 8:23AM   1
mydb     other_table 54998666  col_d        May 19 2009 10:03PM 202
```

When stats are missing

- Don't run wild with missing stats
- Need to know how the tables and columns are used
- Need to prove that missing stats are generating bad plans
- We might still want stats even on unindexed columns
  - This can be cheaper than adding/modifying an index
  - More on this later
- You update them, you maintain them
  - Updating special stats only once may be worse than no stats

When stats are missing

- Special case: systabstats
- Table-level stats such as page counts and row counts, among others
- Not rolled back as part of rollback tran
- sp_configure "housekeeper free write percent" **must be at least 1**
  - Otherwise systabstats is only refreshed by
    - update % statistics
    - exec sp_flushstats
  - FYI value of 1 is rarely high enough, consider 2-5

When stats weren't updated by update stats

- Stats are only updated by the same command that generated them
  - If you run `update all statistics` once
  - Thereafter run `update index statistics` weekly
  - The stats on unindexed columns are **not** updated
  - But it's even worse than that…

When stats weren't updated by update stats

- Some attributes of stats have "stickiness"
  - Once set explicitly, they are preserved unless set explicitly again
  - If you don't set them, future commands inherit the values

```
update index statistics tableA using 50 values
update index statistics tableA                    -- also 50 values
```

When stats weren't updated by update stats

- Which stat attributes are sticky (by default) if explicitly set?
  - using NNN values (# of steps)
  - histogram tuning factor
  - sampling
  - hashing / partial_hashing / no_hashing

When stats weren't updated by update stats

- How to change the value of sticky attributes?
  - A new command with explicitly different values
  - Remove the stats completely
  - Remove the stickiness

When stats weren't updated by update stats

- Stats can be deleted explicitly with `delete stats`
  - This is good practice before every update stats command
  - Backup the stats first with optdiag
  - This only deletes column distribution stats from sysstatistics
    - Does not delete table-level stats from systabstats

When stats weren't updated by update stats

- Stats can sometimes be deleted implicitly
  - Drop & recreate index
    - But only if the columns don't appear in any other index
    - Tip: use 0 steps to maintain stats when dropping & recreating
    - Useful when dropping & recreating as second-class reorg
    - … but reorg rebuild online / defrag are both better

When stats weren't updated by update stats

- Remove stickiness…
  - … on this table
    - sp_modifystats … REMOVE_STICKINESS
  - … on every table
    - sp_configure "enable sticky statistics", 0
- This is all or nothing
  - Can't specify that some attributes are sticky and others are not

Update stats for free

- Update stats for free if table already has data
- Create NC index (or DOL CL index) = update statistics … index …
- Create CL index (APL) = update statistics … table
- What's missing?
  - Non-leading columns
    - We cannot do anything about this
  - Type or attributes of the stats
    - We do have some control over these

Update stats for free

```
create index … with statistics using NNN values
,                    statistics hashing
,                               no_hashing
,                               new_hashing
,                    statistics histogram_tuning_factor = N
,                    statistics max_resource_granularity = N
,                    statistics print_progress = N
```

- Only works when the table has data

Update stats... in the right ways

- Which version
- Consumers
- Sampling
- Hashing
- Steps / histogram tuning factor
- Out of range values / dynamic histograms
- Under the hood: sorts and scans
- Avoiding duplicate work / minimum set coverage

Which version: there is only update statistics

- No such thing as update index statistics, or update all statistics
  - Internally transformed into multiple update statistics commands
  - Implications for duplicate effort
  - Implications for minimum set coverage, more on this later

Which version: there is only update statistics

- update index statistics tableA

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

Which version: there is only update statistics

- update index statistics tableA
  - update index statistics tableA index1
  - update index statistics tableA index2
  - update index statistics tableA index3

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

Which version: there is only update statistics

- update index statistics tableA
  - update index statistics tableA index1
    - update statistics tableA index1 -- a
    - update statistics tableA (b) -- b
    - update statistics tableA (c) -- c
  - update index statistics tableA index2
    - update statistics tableA index2 -- a
    - update statistics tableA (d) -- d
    - update statistics tableA (e) -- e
  - update index statistics tableA index3
    - update statistics tableA index3 -- e

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

Which version: there is only update statistics

- update all statistics tableA

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

Which version: there is only update statistics

- update all statistics tableA
  - update index statistics tableA index1
  - update index statistics tableA index2
  - update index statistics tableA index3
  - update statistics tableA (f)
  - update statistics tableA (g)

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

Which version: there is only update statistics

- update all statistics tableA
  - update index statistics tableA index1
    - update statistics tableA index1 -- a
    - update statistics tableA (b) -- b
    - update statistics tableA (c) -- c
  - update index statistics tableA index2
    - update statistics tableA index2 -- a
    - update statistics tableA (d) -- d
    - update statistics tableA (e) -- e
  - update index statistics tableA index3
    - update statistics tableA index3 -- e
  - update statistics tableA (f) -- f
  - update statistics tableA (g) -- g

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

Consumers

- In theory we can divide up the work and do it in parallel
- Many prerequisites for consumers = N
  - Must have **N+1** "number of worker processes" available at runtime
  - Must have "max parallel degree" > **N+1**
  - Must have "max utility parallel degree" > **N+1**
- Incompatible with sampling or hashing

Sampling

- All the pain of update statistics is based on number of pages
- Read fewer pages = less pain… but stats quality suffers…
- `update statistics … with sampling = 10 percent`
- sp_configure “sampling percent”, 10
- Incompatible with consumers or hashing

Sampling

- The devil is in the details
  - Sampling **does not update density** from any non-sampling run
    - `update table statistics tableA`
  - Sampling only updates density if stats are deleted first
  - Sampling only applies to running on individual columns
    - Ignored for running on index or table
  - Sampling only applies to data pages
    - Ignored for all index page scans

Sampling

- Implications for density as it is not adjusted for sampling
- Consider 1000 rows all with same value in this column = density 1.0
- Run update stats with sampling = 10 percent
- ASE will see 10% x 1000 rows = 100 rows all the same, out of 1000 rows
- ASE will mark density = 0.1 !!!
- If you still use sampling, consider sp_modifystats
  - Adjust densities by $\frac{1}{sampling}$
  - Express percentage here as a decimal 0.0 – 1.0

Hashing

- More accurate than sampling, and probably less work
  - Far less tempdb disk space, far less procedure cache
  - A lot more data cache (in whichever case tempdb uses)
- Incompatible with consumers or sampling
- Consumers and sampling are dead, use hashing instead

Hashing

- `with hashing | partial_hashing | no_hashing`
  - hashing = hash regardless of cardinality
  - partial_hashing = hash up to 64K unique values, else sort as usual
    - If existing stats say > 64K unique values, no attempt to hash!
- sp_configure “update statistics hashing”
  - Controls server-wide behaviour during update statistics
- sp_configure “utility statistics hashing”
  - Controls whether “free” stats during create index use hashing

Steps and histogram tuning factor

- Number of steps = **requested** number of histogram steps
- Histogram tuning factor = multiply # steps by this number **if it helps**
- Actual number of steps could be lower or higher than requested
  - 1 or 2 steps (NULL) if all values in column are duplicates
  - Up to $\#\ of\ steps\ *\ histogram\ tuning\ factor$
- Advice from SAP: 1000 total is good, 2000 total is too many
  - With default htf=20, implies request 50 steps

Steps and histogram tuning factor

- Maybe better: 100 steps, htf = 10, set globally via sp_configure
- Histogram tuning factor not guaranteed to increase steps
- Neither is requesting steps… but is a lot more likely to
- The average number of steps is higher for 100..1000 than for 20..1000
- Remember both attributes are sticky!

When the histogram can't keep up with the data

- Two solutions for how to handle column values that only increase
- Default behaviour: out of range SARGs have selectivity = 0 (!!)
  - This can significantly alter query plans

When the histogram can't keep up with the data

- ASE 15.0.2 ESD#2: out of range histograms
- Extrapolate OOR value based on existing histogram behaviour
- Enabled by default server-wide
- Disable server-wide with TF 15355 (documented & supported)
- Disable per column with
  - `update statistics table (column) out_of_range off`
  - Does not actually run update stats; toggles a status
  - May occasionally resolve weird optimiser behaviour

When the histogram can't keep up with the data

- ASE 16.0 SP01: dynamic histograms
  - Documentation bug: not mentioned in some docs > 16.0 SP01
- Update the very last histogram cell transactionally
  - Hold in memory only
  - Flushed at polite shutdown or "sysstatistics flush interval"
    - Remember latter default is zero

When the histogram can't keep up with the data

- ASE 16.0 SP01: dynamic histograms
- Enable per column with
  - `update statistics table (column) using dynamic histogram on`
  - Does not actually run update stats; toggles a status
- Might resolve poor optimiser choices when column only increases
- Definitely makes things worse if column doesn't only increase
  - There is overhead per insert / update / delete

Under the hood... scans and locks

Command	Locking scheme	Index	Scans	Sorts (tempdb)	Locking
update statistics tbl (col)	APL	--	Table scan into worktable	Sorts the worktable	Sh Int Tbl, Sh Page
	DOL	--	Table scan into worktable	Sorts the worktable	Level 0 dirty reads
update statistics tbl index	APL	NC	Leaf level index scan	--	Sh Int Tbl, Sh Page
	APL	CL	Table scan	--	Sh Int Tbl, Sh Page
	DOL	any	Leaf level index scan	--	Level 0 dirty reads
update statistics tbl	APL	any	Table scan + leaf-level index scan per NC index	--	Sh Int Tbl, Sh Page
	DOL	any	Table scan + leaf-level index scan per index	--	Level 0 dirty reads

- Specifying consumers always applies to sorts on worktables
- Specifying consumers only applies to base table if it is partitioned

Under the hood... hashing

Command	Cardinality	Sampling	Partial_hashing	Hashing
update statistics tbl (col)	low	Data only	One hash on column	One hash on column
	high	Data only	One worktable + sort on column	One hash on column
update statistics tbl index	low	Ignored	One hash on index	One hash on index
	high	Ignored	One worktable + sort on index leading column	One hash on index
update statistics tbl	low	Ignored	One hash per index	One hash per index
	high	Ignored	One worktable + sort on index leading column, per index	One hash per index
update index statistics	low	Ignored	One hash per index	One hash per index
	high	Ignored	One worktable + sort per column	One hash per index
update all statistics	low	Ignored	One hash (!)	One hash (!!)
	high	Ignored	One worktable + sort per column	One hash (!!!)

Avoiding duplicate work

- Remember everything is broken down into multiple update statistics
- This means a duplicate scan every time a column appears
- … unless you use (full) hashing!
- `update all statistics … with hashing` uses a single scan!
  - Also little tempdb space and procedure cache
  - But a lot of tempdb data cache
- What to do if this isn't possible or practical?

Avoiding duplicate work

- Minimal set coverage is a famously hard problem
- update statistics = stats on a, e
  = 2 scans
- update index statistics = stats on a, b, c, d, e
  = 7 scans
- update all statistics = stats on a, b, c, d, e, f, g
  = 9 scans
- update statistics (a, b, c, d, e) = stats on a, b, c, d, e
  = 5 scans

Example:

tableA (a, b, c, d, e, f, g)

index1 (a, b, c)

index2 (a, d, e)

index3 (e)

So what is “the right way”?

- DON’T ever run update stats in PROD; optdiag stats from a copy
- DO use datachange() while understanding its limits
- DO use optdiag out to backup stats before any delete stats
- DO use delete stats before any update stats
- DO understand stat stickiness; recommend disabling
- DON’T use consumers
- DON’T use sampling
- DO use sp_modifystats on densities if you still use sampling

So what is “the right way”?

- DO set “sysstatistics flush interval” to 15 or 60
- DO set a sensible global value for number of steps
- DO set a sensible global value for histogram tuning factor
- DO use hashing wherever possible
- DO use minimal set coverage where full hashing is not possible
  - Superior to `update index statistics`!

Q & A, and thank you

Joe Woodhouse
joe.woodhouse@primadonnaconsulting.com

Too busy putting out fires to reduce your toil?
Answering the on-call phone too often?
Drowning in technical debt?

Joe is a freelance consultant available through Prima Donna Consulting and can be engaged flexibly.

Not a lawyer – no charge for emails!