Barclays Capital

Afternoon Tea "Brown Bag"

Performance & Tuning for Developers

Refreshers ~ Opinions ~ Updates ~ Tips

Mark Hudson, Sybase UK Proof of Concept Team
Alan O'Neill, Account Manager

Agenda

QUERY TUNING & TROUBLESHOOTING

- Know your basics ~ tables, indexes, sizes, actuals & estimates
- A few *showplan* gotchas ~ and the future of *showplan*
- The famous (infamous) "302, 310 and Friends" trace flags
- Query tuning ~ approach and decision flow
- Taster demonstration of Database Expert (DB Expert)

I thought showplan was an estate agent's brochure till I discovered Sybase

SYBASE®

KNOW YOUR BASICS

- Rules of thumb for clustered and nonclustered indexes
- Notional and real logical and physical IO rates
- How big is big when it comes to Sybase tables and databases ?
- The terrible twins ~ sp_spaceused and sp_estspace

SYBASE®

Clustered & Nonclustered Indexes
Rules of Thumb

- Clustered indexes
  - Generally slower to create because involve sorting
  - Clustered index order strictly maintained for datapage locking
  - Forwarded rows occur in general for dataonly locking
  - For datapage ~ leaf level contains page pointers
    - Hence leaf level rows in 1:n with data rows ~ relatively small
  - For dataonly ~ leaf level contains row pointers
    - Little difference from nonclustered in this case
  - Traversal of leaf level largely tied to ordered traversal of data level
- Nonclustered indexes
  - Mostly the opposite !
  - Leaf level contains row pointers
  - Hence leaf level rows in 1:1 with data rows ~ relatively large
  - Ordered traversal of leaf level → scattered access to data level
  - Leaf levels themselves always strictly ordered ~ for clustered too

Logical & Physical IOs
Notional versus Real

- Logical IO
  - Every read or write without exception → via a data cache
  - Cache memory access = logical IO often written "lp"
  - Notional value 2 ms (millisec) used by optimizer
  - Real values anywhere down to 0.02 ms (50000/sec)
- Physical IO
  - Every write (except some in tempdb) goes to disk
  - A read may or may not come from disk → always via a cache
  - Often written "pp"
  - Notional value 18ms used in optimization
  - Real values anywhere down to 0.2 ms (5000/sec)
- It's the ratio that matters
  - 9:1 assumed, ? 5:1 – 50:1 in reality ?

e.g. in 302 etc. trace output

tunable ! yes really ...

Sybase Tables & Databases

How Big is Big ? - 1

- Always worth knowing your hardware IO rates
  - If in doubt fairly conservative these days to use 2000 pp, 20000 lp
  - Always try to calculate basic scan time for key tables

EXAMPLE

2,000,000 rows ~ 10 rows/page (don't forget fillfactor)
200,000 pages → 200,000 / 512 = approx. 400 MB (for 2k page size)
basic scan time → 100 seconds physical, 10 seconds logical

- Use this information for the biggest tables
  - Sanity check against more detailed measurements
  - Intuitive understanding of data
  - Never trust blindly what any tool or test is telling you
  - Second guess the broad principles behind any query and be able to visualize / back of an envelope sketch steps
- Basic scan times relevant ~ of course not whole story ~ for:
  - Index re(creates) → sorting can be major overhead of course
  - Table scans for various purposes e.g. brute force batch jobs

Sybase Tables & Databases

How Big is Big ? - 2

- So how big really *is* big ?
  - Depends on the context

```
EXAMPLES ~ all for 10 rows/page
a) 2,000,000 rows → 200,000 2k pages → 100 sec phys, 10 sec logical
b) 20,000,000 → 2,000,000 → 17 minutes, 1.7 minutes
c) 200 million → 20 million → nearly 3 hours, 17 minutes
d) 2 billion → 200 million → over 1 day, nearly 3 hours
```

- These might be big for
  - a) being table scanned in the innermost nested loop of a join
  - b) being table scanned in any time critical interactive user query
  - c) driving any batch job or (re-)creating any index → for testing ?
  - d) recreating any index in production even planned well ahead

The Terrible Twins
sp_estspace

- Purpose
  - Take an *empty* schema → predict table, index sizes
- Parameter usage

```
sp_estspace
    table_name,
    number_of_rows,
    fill_factor,
    cols_to_max,
    textbin_len,
    iosec,
    pagesize
```

tricky ...

default still 30/second but *gotcha* now gone

- Output includes
  - Reasonably plausible index creation times
  - Reliable size estimates for each level of each index

The Terrible Twins
sp_spaceused

- Purpose
  - Take an *actual* table → show data & index size allocated
- Parameter usage

```
sp_spaceused
    [table_name],
    [1]
```

don't forget ! ... very handy

- Output includes
  - Space allocated – slightly higher for various reasons
  - Data and index space used
  - With "1" modifier → index by index breakdown

The Terrible Twins
sp_spaceused ~ alternatives

- Consider using your own more user friendly local system proc or script producing output like this

```
i   NumRows     Isize Dsize TabIdx
--- ---------- ----- ----- ----------------------------------------------------
0   11471119   0.000 393.0 FloatingRatesValues.FloatingRatesValues
0   4036452    0.000 315.3 CurvesRatesNRHist.CurvesRatesNRHist
0   6703476    0.000 242.4 PairsQuotes.PairsQuotes
0   1460892    0.000 219.5 SwapSchedule.SwapSchedule
0   190016     0.000 185.5 FxSwapDeals.FxSwapDeals
0   256815     0.000 167.1 IamDeals.IamDeals
0   321935     0.000 161.0 SpotDeals.SpotDeals
...
2   0          0.193 0.000 StrategiesDeals.StrategiesDealsIdx1
2   0          0.207 0.000 FoldersAllowed.FoldersAllowedIdx1
2   0          0.230 0.000 FraDeals.FraDealsIdx1
```

- Would typically use data_pgs(objid,doampg) function

showplan

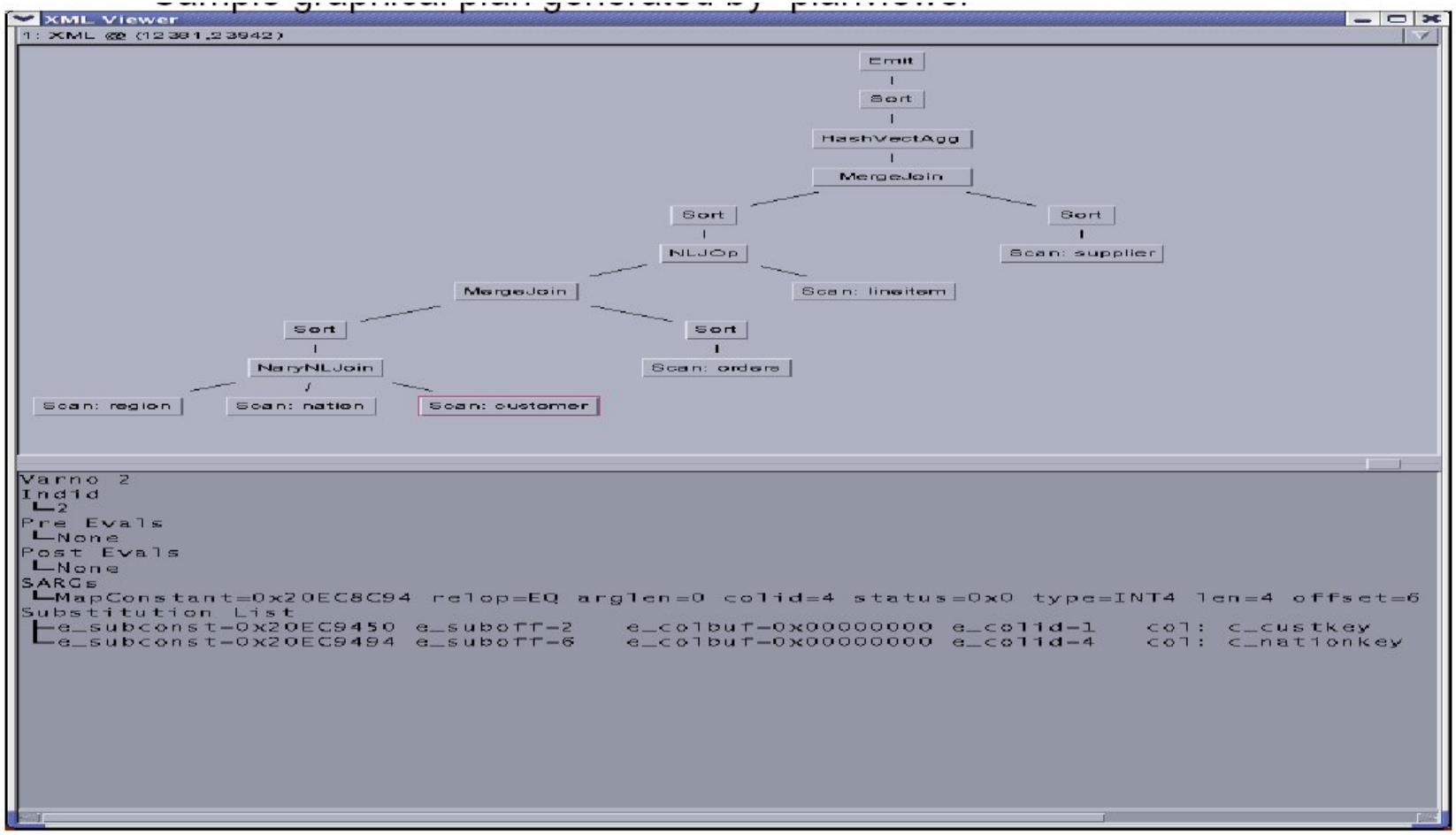
- A few gotchas
- The future of showplan

SYBASE®

showplan
A few gotchas

- Use of an index does not always mean a trouble free query
- Some indexes are not especially selective with certain data
  - Always use common sense
  - Knowledge of data
  - Sanity checks using pp and lp based estimates
  - May well be worth using the 302 etc. trace flags that follow
- Easy to miss that an index is being noncluster leaf level scanned
  - Rather than the usual, and mostly highly efficient, traversal
  - Still useful, but very much second best
  - Remember a difference of a few logical IOs nested well down inside a join loop can build up massively in terms of total IOs
  - Thus can burn a lot of CPU and take a lot of time

showplan
The future

- 

- A text based showplan output in XML will also be available

The "302, 310 and Friends" trace flags

- Outline of the trace flags with a few examples
- Summary of the trace flags
- Future of these trace flags
- Where to go for further information

SYBASE®

"302, 310 and Friends"
Example query

- Example underlying sample outputs shown

```
1> dbcc traceon (3604, 302)
2> go

1> select ta.au_id, t.title_id, t.title, t.pub_id,
   t.pubdate, t.price
2> from titles t, titleauthor ta
3> where type = "cooking"
4> and price = 10
5> and t.title_id = ta.title_id
6> go
```


"302, 310 and Friends"

302 output

WARNING ! rather old style examples ...

```
*******************************
Entering q_score_index() for table 'titles' (objectid 208003772, varno = 0).
The table has 5000 rows and 621 pages.
Scoring the JOIN CLAUSE:
        title_id EQ title_id
Base cost: indid: 0 rows: 5000 pages: 621 prefetch: N
        I/O size: 2 cacheid: 0 replace: LRU
Relop bits are: 5
Estimate: indid 1, selectivity 0.000198, rows 1 pages 3 index height 2
Unique clustered index found--return rows 1 pages 3
Relop bits are: 805
Estimate: indid 2, selectivity 0.000198, rows 1 pages 4 index height 3
Relop bits are: 805
Estimate: indid 4, selectivity 0.000198, rows 1 pages 4 index height 3
Cheapest index is index 1, costing 3 pages and
        generating 1 rows per scan, using no data prefetch (size 2)
        on dcacheid 0 with LRU replacement
Join selectivity is 5000.000000.
*******************************
```

Red = OAM

Blue = Distribution Page

Black = Optimiser

Now → histograms

"302, 310 and Friends"
302 output (cont.)

```
*********************************
Entering q_score_index() for table 'titleauthor' (objectid 176003658, varno = 1).
The table has 6250 rows and 126 pages.
Scoring the JOIN CLAUSE:
        title_id EQ title_id

Base cost: indid: 0 rows: 6250 pages: 126 prefetch: N
        I/O size: 2 cacheid: 0 replace: LRU
Relop bits are: 4
Estimate: indid 1, selectivity 0.000229, rows 1 pages 2 index height 1

Cheapest index is index 1, costing 2 pages and
        generating 1 rows per scan, using no data prefetch (size 2)
        on dcacheid 0 with LRU replacement
Join selectivity is 4369.000001.

*********************************
```

Red = OAM
Blue = Distribution Page
Black = Optimiser

"302, 310 and Friends"
310 – tracing join order decisions

- Information you get from 310:
  - Whether the query is Connected
  - Display the current permutation of tables → "NEW PLAN"
  - For each Join Order (in the order of tables)
    - varno, indexid
    - path, pathtype, method
    - outerrows, rows, joinsel
    - cpages, prefetch, iosize, replace
    - lp, pp, total cost
    - jnvar, refcost, refpages, reftotpages
    - ordercol[0], ordercol[1],
  - Total # Permutations

"302, 310 and Friends"
310 – sample output

QUERY IS CONNECTED

0 - 1 - 2 -
NEW PLAN (total cost = 16548):

varno=0 (stock) indexid=3 (stock_by_name)
path=0x2075c168 pathtype=sclause method=NESTED ITERATION
outerrows=1 rows=500 joinsel=1.000000 cpages=24 prefetch=Y iosize=16
replace=LRU lp=36 pp=24 corder=1

varno=1 (asset) indexid=1 (asset_by_id)
path=0x2075c3d0 pathtype=sclause method=NESTED ITERATION
outerrows=500 rows=75 joinsel=59.000001 cpages=16 prefetch=Y
iosize=16 replace=LRU lp=300 pp=46 corder=2

varno=2 (sec_xn) indexid=2 (sec_xn_date)
path=0x2075c800 pathtype=sclause method=NESTED ITERATION
outerrows=75 rows=35 joinsel=10.000000 cpages=3 prefetch=N iosize=2
replace=LRU lp=4860 pp=272 corder=2

"302, 310 and Friends"
310 – typical features in output

- Query Is Connected
  - Unless there is no join clause, always printed
- Display the current permutation of tables

  0 - 2 - 1 -
  1 - 0 - 2 -
  1 - 2 - 0 -
- Ignoring this Permutation
  - Join order discarded as its cost is already higher than plans already considered
  - If Optimiser ignores what you believe to be a good plan, use trace flag 317 to see calculated costs
- varno, indexid
  - indicated which table and what index
- path, pathtype, method
  - indicates whether index based on join, or, sclause
  - e.g. sclause for outer, join clause for inner

"302, 310 and Friends"
310 – typical features in output (cont.)

- outerrows, rows, joinsel
  - represents how many rows from outertable joining with the inner table
  - how many rows of inner table qualify with indexid
- joinsel
  - join selectivity from (302) output, important field
- cpages, prefetch, iosize, replace
  - shows if/why large block IOs will be used
- refcost, refpages, reftotpages
  - reformat costing information
- ordercol[0], ...
  - column order for joins
- Total Cost
  - Equals 2 * lp + 18 * pp (unless changed)

"302, 310 and Friends"
310 – what to look for in the output

- Are outerrows and rows estimate reasonable?
- Are the lp estimates reasonable?
- Are the pp estimates in line with cache?
- How many permutations are ignored?
- For each table, is the index based on search clause or join clause?
- Whether reformatting is being used?
- Does the join order avoid an extra sort?

"302, 310 and Friends"
The Friends

- dbcc traceon(303)
  - Use when justification of optimization for OR clause is required
- dbcc traceon(311)
  - Additional / alternative info on estimated lp and pp cost factors
- dbcc traceon(317)
  - Same as 310 in terms of information displayed and display format
  - Shows all permutations not just the cheapest
- dbcc traceon(319)
  - Shows the costs of the possible reformatting options when they are considered.
  - WARNING: This also gives a great deal of Query Tree information which is of no value at all

"302, 310 and Friends"
Tracing a Query to a Query Plan

- Parser
- Normalisation
- Query Analyser
- Determine Optimal Access Path
  - Best Index, I/O size etc.
- Explore Possible Join Orders
  - Including Caching Strategy
- Generate Final Query Plan

Possible Trace Flags

} 302 / 303(*)

} 310 / 312 / 317 / 319 (*)

(*) less value

"302, 310 and Friends"
ASE 15.0 – future of the flags

- Trace flags such as 302, 310, 317 will be replaced by extended showplan syntax

```
set option show [brief|normal|long]
set option show_histograms [brief|normal|long]
set option show_engine [brief|normal|long]
```


"302, 310 and Friends"
Where to look for More Information

- The original Ian Smart presentation that introduced the "302, 310 and Friends" title
  - Pre 11.9.2 but still the most helpful tutorial in many ways
- ASE Troubleshooting Guide volume 1
  - Provides some basic explanations
  - Less helpful on why & how to use
- http://www.sybase.com/detail?id=1011767
  - "Technical Issues in ASE 11.9.x and 12.0 Upgrade"
  - Explains the 11.9.2 changes in 302 output
- John Kirkwood ~ "Official Sybase Internals"
  - Some additional remarks e.g. on 311
  - Good treatment of some of calculations for sizes, IOs etc.
  - If you can still get it ! → **ISBN 1-85032-334-8**
- Rob Verschoor ~ "The Complete Sybase ASE Quick Reference Guide"
  - (Very) brief writeups of most flags → **ISBN 90-806117-1-9**
  - www.sypron.nl

SYBASE®

Query Tuning

- Approach → hints & tips
- Decision flow

SYBASE®

Query Tuning
Approach

- First catch your query
  - Maybe already identified and handed to you → fine
  - Otherwise ? identify "hotspot" tables and indexes and related SQL → monitoring tables, DBXray, DB Expert all good for this
- Identify, rank and understand big tables
  - Basic sizing, intuition for how long to scan etc.
  - Primary & surrogate key structure
  - Get systematic sizing list using approx_sizes.sql or similar
- Always worth getting some insight via locks taken
  - Applicable to long running queries
  - Maybe easiest way to find where time being spent
- Consider doing grouped selects
  - Counts of rows grouped by various lists of key values
  - Can be intuitively easier than constantly looking at the intricate selectivities and other statistical info from optdiag, trace flags etc.
  - Use common sense and the basic scan estimates to avoid attempting such selects if they will take unfeasible elapsed times !

Query Tuning
Decision Flow - 1

- Get statistics in good shape first
  - Obvious but vital
  - Drop and recreate all statistics in a copy of data if possible
- Look for potential additional indexes
  - Intuition from joins and subselects etc.
  - Look to improve selectivity, reduce logical IOs on inner tables
  - Test initially by over-specifying indexes (plenty of columns)
  - Then trim them down to minimum necessary later
- Be careful with test cases
  - Easy to convince yourself there's a bug or will'o'the'wisp
  - Oddities like same plan → wildly different logical IOs do exist, but they are rare → more likely a red herring in your test case
- Don't forget the power of nonclustered index covering
  - Discriminatingly chosen additional columns can help
  - Can make index support more queries
  - Don't go mad of course → sanity check index volumes constantly

Query Tuning
Decision Flow - 2

- Start with showplan → of course
  - In many cases you can second guess alternatives
  - Try index variations if any seem promising
  - Showplans (better: AQPs) as last resorts
- If doubts about showplan → consider 302 etc. trace flags
  - Generally 302, 310, 317 best starting set
  - Can be run with `noexec` like `showplan` can
  - Don't so much try to grind through shadowing all the optimizer choices and permutations
  - More a question of looking for inspiration and relating back to the business data, displayed selectivities and any "distribution queries" you have gathered
- Motherhood'n'apple'pie ***BUT***
  - Disciplined approach
  - Test queries in isql with results to files, getdate() timings, clear file naming convention etc. to identify steps taken → results obtained

Barclays Capital

Afternoon Tea "Brown Bag"

Performance & Tuning for Developers

Refreshers ~ Opinions ~ Updates ~ Tips

Mark Hudson, Sybase UK Proof of Concept Team
Alan O'Neill, Account Manager