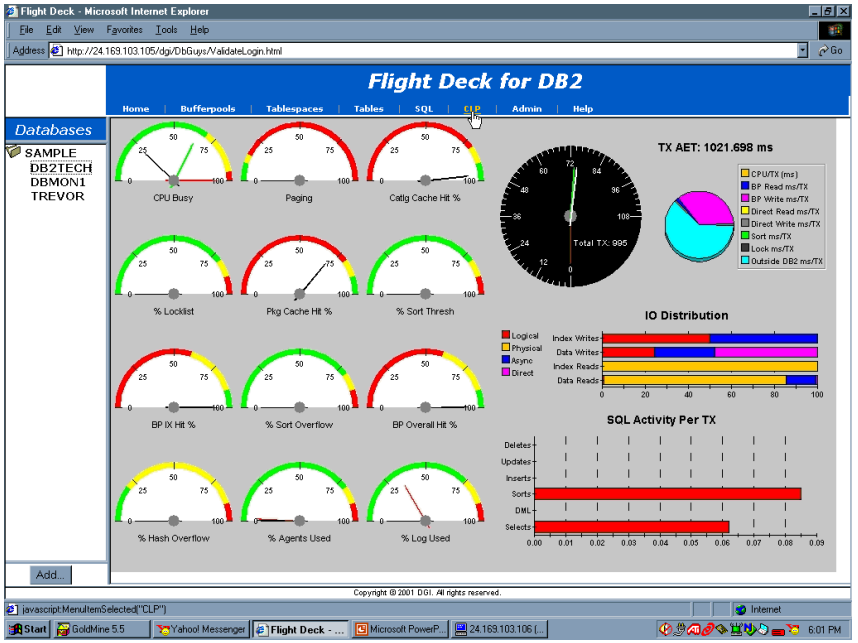
Linux, UNIX, Windows

Cut Query Times in Half / SORT is a FOUR Letter Word

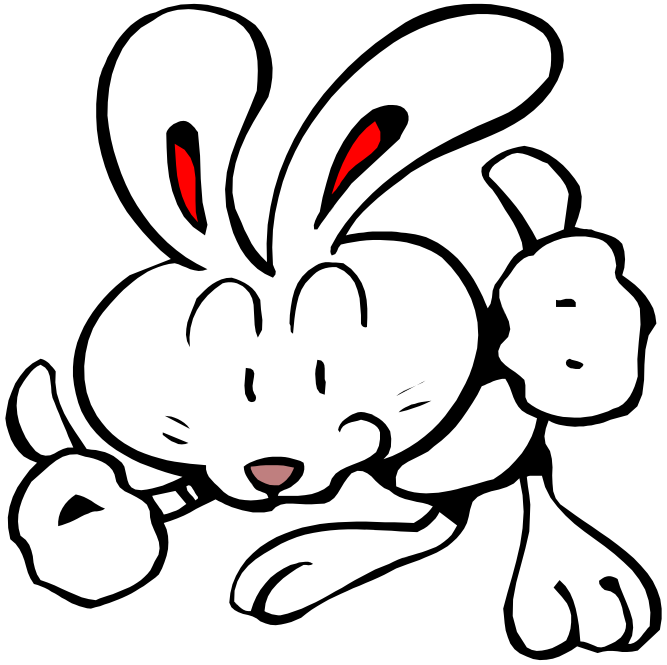
Philip K. Gunning
Vice President/DGI

March 6, 2002



Session Agenda

- How do we measure sort performance?
- How do we improve sort performance?
  - Hash Joins?
  - Index Builds?
- Science Projects
- Case Studies
- Summary
  - Ideas to take home



SORT HAPPENS

- Business Applications
  - Order By
  - MAX
  - Joins
- Small SORTS (Hopefully) < 1 MB
  - Piped, No Overflows
  - Smaller the better
  - Burn CPU
- Decision Support Queries
  - ORDER BY
  - GROUP BY
  - CUBE
  - ROLLUP, RANK
  - JOINS
- Larger Sorts > 1 MB
  - Overflows
  - Optimize I/O

Attendee Notes

- People like order in their lives. Therefore, sorts happen.
- Transactional applications tend to perform small composite sorts, often of a very few rows. Large or frequently executed sorts can bring an application to its knees.
- Decision Support Data Warehouses tend to perform much larger sorts. Larger SORTHEAPS can help, but overflows are likely. I/O subsystem tuning becomes very important for DSS.

SORT This, SORT That, oh SORT!

- SORT VOCABULARY
  - Sort Size
  - Sort Heap
  - Sort Overflows
  - Sort Heap Threshold
  - Sort Capacity
  - Post Threshold Sorts
  - Private Sort
  - Shared Sort
  - Bin Sort
  - The Perfect Sort

Attendee Notes

- Sort Size: the row width/bytes multiplied by the number of rows to be sorted
- Sort Heap: That special memory place where sorts happen without physical I/O
- Sort Overflow: Should the sort size exceed the Sort Heap, the data to be sorted spills over (overflows) into TEMPSPACE tablespace via the bufferpool
- Sort Heap Threshold: DB2's allowance for total sort memory within an instance
- Sort Heap Threshold / Sort Heap Size = Sort Capacity = Number of Eligible sorts prior to post threshold sort
- Post Threshold Sorts - What happens when DB2 exceeds its sort memory "allowance"
- BIN SORT - A fast binary sort technique

Speaking of … DB2_BINSORT

- db2set DB2_BINSORT=YES (default)
- “Enables a new sort algorithm that reduces the CPU time and elapsed time of sorts. This new algorithm extends the extremely efficient integer sorting technique of DB2 UDB to all sort data types such as BIGINT, CHAR, VARCHAR, FLOAT, and DECIMAL, as well as combinations of these data types.” Administration Guide, Appendix D - Registry Variables, Table 69, Performance.
- Are BINSORTS enabled by default in V6? - No.

Attendee Notes

- This page intentionally designed for your doodle.

Configuration Settings that affect your SORT physical memory

- INTRA_PARALLEL =
  - YES, Shared Memory
    - Decision Support Databases
    - Large Data Volume Queries
  - NO, Private Memory
    - Transactional Applications (DEGREE = 1)
- DB2MEMDISCLAIM (AIX) = YES
- DB2MEMMAXFREE = 8192000
- SHEAPTHRES
- BUFFERPOOL SIZES

Attendee Notes

- This page intentionally designed for your doodle.

Sampling SORT Performance

- Sources of SORT Data
  - Database Snapshots
  - Application Snapshots
  - Connection Events
  - SQL Events
- DB2 “GET SNAPSHOT FOR DATABASE ON *DBNAME*”
- DB2 “GET SNAPSHOT FOR DATABASE MANAGER”

- Database Snapshot

```
Total sort heap allocated = 0
Total sorts               = 237
Total sort time (ms)      = 10190
Sort overflows            = 29
Active sorts              = 0

Number of hash joins      = 0
Number of hash loops      = 0
Number of hash join overflows = 0
Number of small hash join overflows = 0
```

- Database Manager Snapshot

```
Sort heap allocated       = 0
Post threshold sorts      = 0
Piped sorts requested     = 181
Piped sorts accepted      = 181
```

SORT Formulas

- % SHEAPTHRES Allocated = Sort Heap Allocated * 100 / SHEAPTHRES
  - If > 90%, larger SHEAPTHRES or smaller SORTHEAP
- If Post Threshold Sorts > 0 (rule above violated), same remedies
- Average Sort Time = Total Sort Time(ms)/Total Sorts
- % SORT Overflows = # Overflows * 100 / Total Sorts
  - If OLTP & If > 3%, cure sorts or increase SORTHEAP
- Sorts / TX, Sorts / SQL, Sort Time / TX, “TCA”
- High Water Sort Heap Used, Average Sort Heap Used
- SORT RQMNT Per Connection = Average Sort Heap Used / Average Number Connections Executing

Attendee Notes

- TCA - Transaction Composition Analysis - The fine art of accurate understanding of where the time goes. What % of time goes to CPU? Sorts? Bufferpool I/O? Locks? Direct I/O? Connection Events have the answer.
- High Water Mark and Average SORTHEAP in use not provided by DB2, DBA must take periodic snapshots and compute/track these.
- Overflows = TEMPSPACE I/O
- High Sort Count & Few Overflows = High CPU burn

Measuring SORT Performance

- Connection Events provide **true cost** of Sort activity for an Application
- create event monitor DGICONN for connections write to file 'e:\tmp\dbaudit\conn\' maxfiles 1 maxfilesize 1024 blocked replace manualstart;
- Set event monitor DGICONN state = 1;
- Set event monitor DGICONN state = 0;

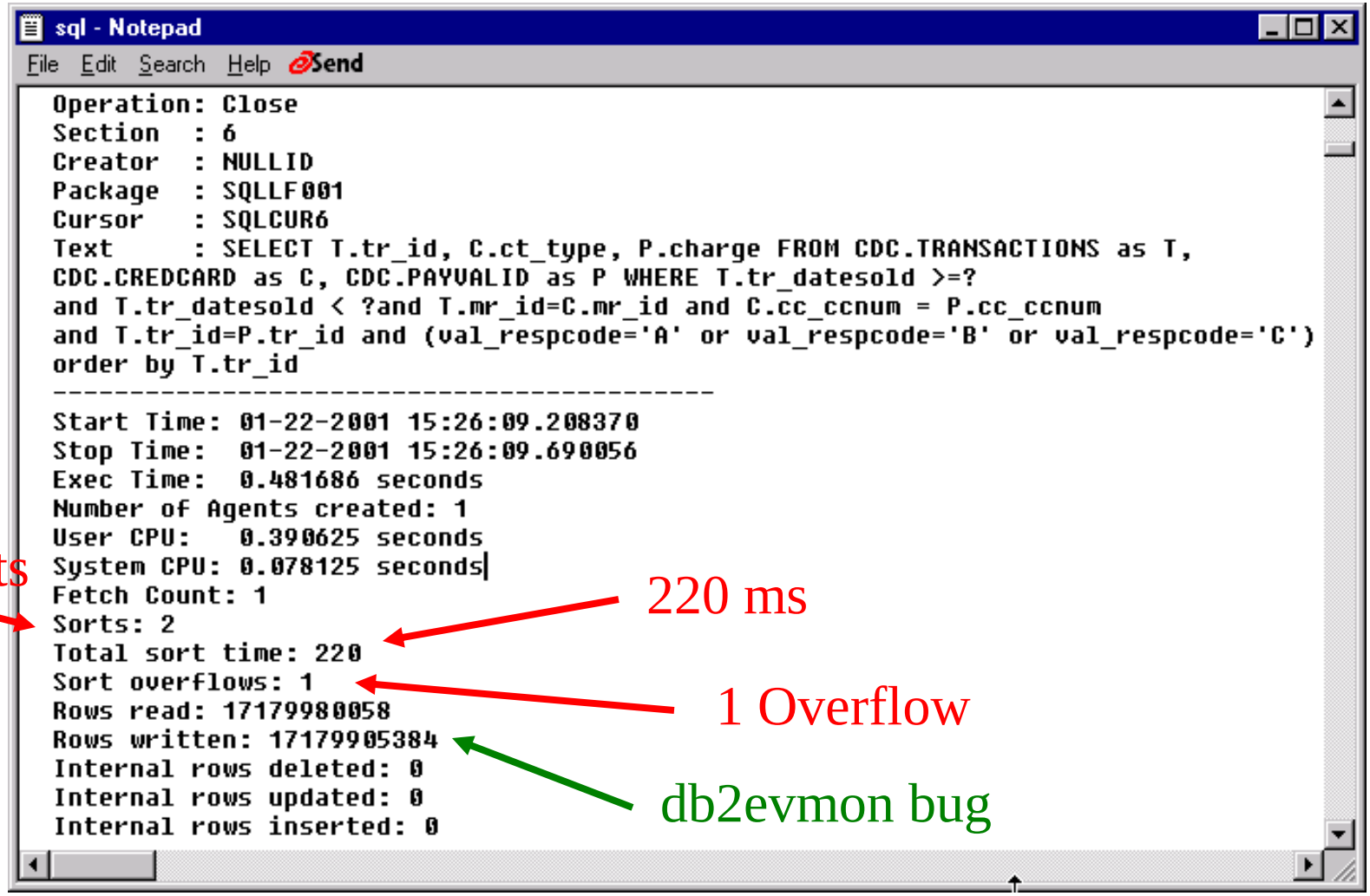
- SQL Events provide **true cost** of Sort activity for an **individual statement**
- create event monitor DGISQL for statements write to file 'e:\tmp\dbaudit\sql\' maxfiles 1 maxfilesize 2048 blocked replace manualstart;
- set event monitor DGISQL state = 1;
- set event monitor DGISQL state = 0;

- db2evmon -path e:\tmp\dbaudit\DIRNAME

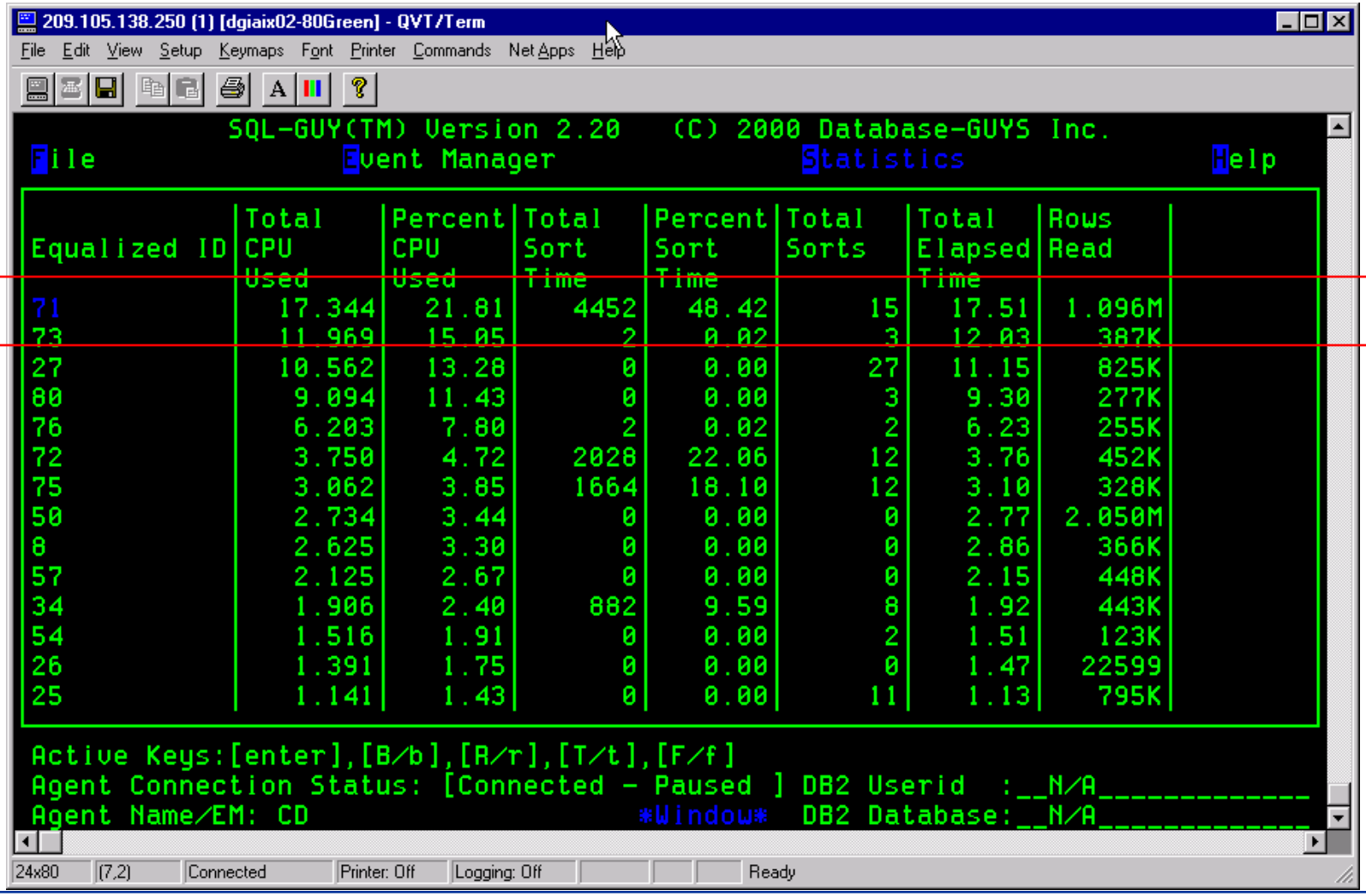
Attendee Notes

- File -
  - Blocked - ensures no data loss at the expense of possible performance delays, should the buffer become full
  - Nonblocked - Fastest capture of event data, but event data could be lost if buffers become full
- Pipes
  - Memory address used to pass event data from buffers to application program reading pipe, extremely fast. “nonblocked” is only option

SORT Facts of Life



SORT Harsh Reality



Attendee Notes

- SQL True Cost = Individual Execution Cost multiplied by Frequency of Execution in the application workload mix.
- The previous 2 slides have demonstrated how a seemingly fast (elapsed time < .5 second) SQL statement consumes over 1/5th (21%) of CPU time in the application, and 48%, almost half, of all sort time in the database.
- It is extremely unlikely that a 1/2 second duration statement would be captured by a snapshot, and, if it were, based on the “low” estimated optimizer cost, such a statement might miss the DBA attention “clip level”

SORT Assessment Summary

- Sort Performance data is available from many sources.
- Snapshots will give you a “taste test” of performance that is presently underway
  - Depending on when snapshots taken, data has varying degrees of value. What if nearly every time you snapped the Sort Heap Allocated was zero?
  - A Snapshot is like looking at your fuel gauge… at the moment… 1/2 full
- Events report total actual execution costs.
  - Equivalent of knowing how much fuel it took to get from point A to point B

How big is your SORT?

- dynexpln -g -d DBNAME -q “Select * from … order by …”

```
|  Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 31
|  |  #Sort Key Columns = 1
|  |  |  Key 1: NAME (Ascending)
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows      = 1981
|  |  |  Row Width = 260
|  |  Piped
```

- (260**+8**) x 1981 = 530,908 bytes / 4096 = 129.6 4K Pages = 530,908 / 1,048,576 = .51 MB …. 50% of default 256 4K SORTHEAP

Attendee Notes

- The SORTHEAP requirement for an SQL statement can be learned from Explain.
- Some SQL statements perform several sorts of varying sizes to arrive at the result set.
- If Rows x Row Width > SORTHEAP size, an overflow will occur to TEMPSPACE via bufferpool
  - Sorts don't overflow "just a little bit", they overflow in their entirety.
  - Hash Joins overflow only to the extent necessary. If > 5% Hash Joins have small overflows, increase SORTHEAP size.

How do I speed up my SORT?

- DON'T DO IT
  - The "Perfect Sort" is the one that never occurs
  - Skip the Order By
  - Check/Change Cluster Sequence
  - Ask the Index Advisor
- IF YOU MUST
  - Make it smaller (fewer rows or smaller width)
  - Increase SORTHEAP to lessen overflows

Attendee Notes

- Even the smallest, seemingly inconsequential, SORTS can bring the biggest machines to their knees.
  - At one site, “Select max(date) from table where primary_key = ‘value’” consumed 33% of an SMP 4-way processor
  - At another site, “Select * from table350rows where primary_key = ‘value’ order by column3” consumed almost 70% of an SMP 8-way processor
- Increasing SORTHEAP to avoid overflows can only mask a performance problem until CPU hits max
- After making any configuration change, re-measure performance to validate the success of your changes.

My SORT is bigger than yours!

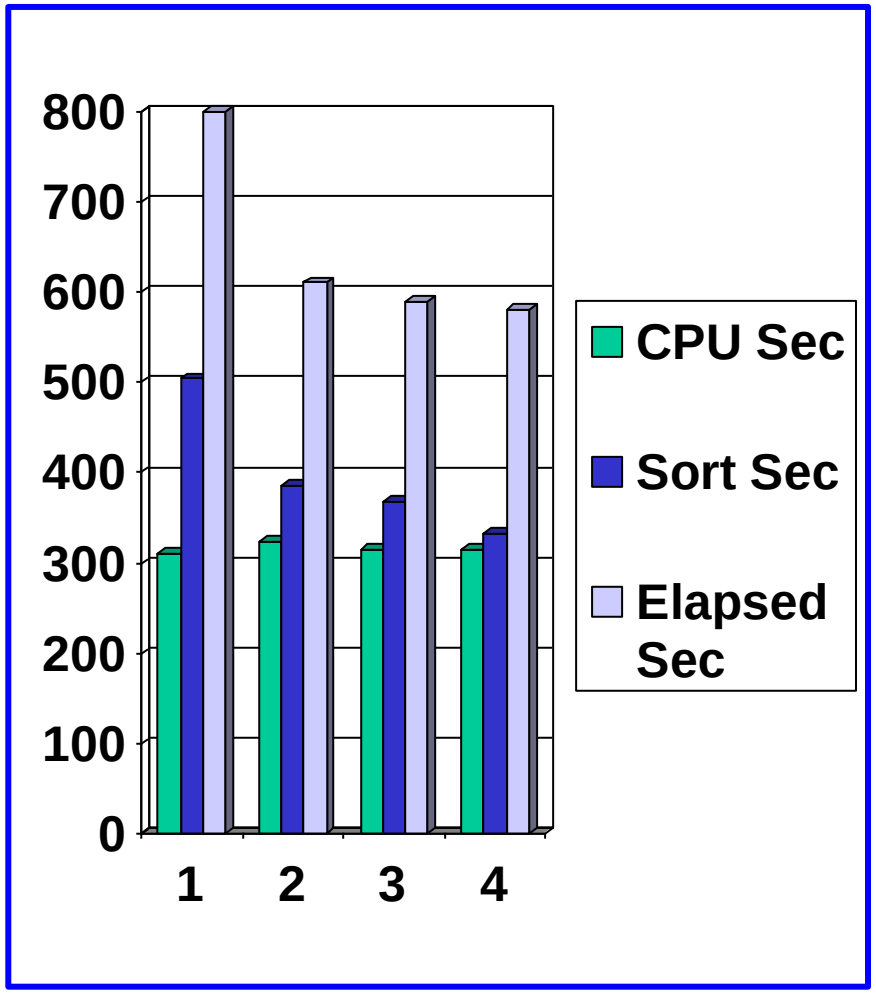
- Big Sorts = Big Overflows
  - Pay Careful Attention to tuning I/O …
  - Container Placement for TEMPSPACE
  - Number of Containers
  - Prefetch Sizes
  - NUM_IOSERVERS
  - 1 big Bufferpool (did I say that?!?!?)
  - DB2_PARALLELIO= for RAID
  - DEGREE of Parallelism
  - DMS DATA tablespaces
  - LONG TABLESPACES

Attendee Notes

- Sorts that cannot be completed within a SORTHEAP will overflow to TEMPSPACE via the bufferpool. Recognizing that some number of Overflow sorts will invariably occur, there are several tuning opportunities to optimize overflow behavior.
  - Dramatic sort performance improvements can be achieved by adhering to essential principles:
    - Container Placement
    - Number of Containers
    - Prefetch Sizes
    - TEMPSPACE Bufferpool

Container Placement Science Project 1

- 556 MB table, 142,514 4K pages, 4.7 million rows
- Test 1 - Sort the full table, TEMPSPACE same disk & file system as WEBHITS_TB.
- Test 2 - Sort the full table, TEMPSPACE different disk & file system, 1 Container
- Test 3 - Sort the full table, TEMPSPACE has 2 containers on different disks
- Test 4 - Same as 3, but 4 containers on different disks.



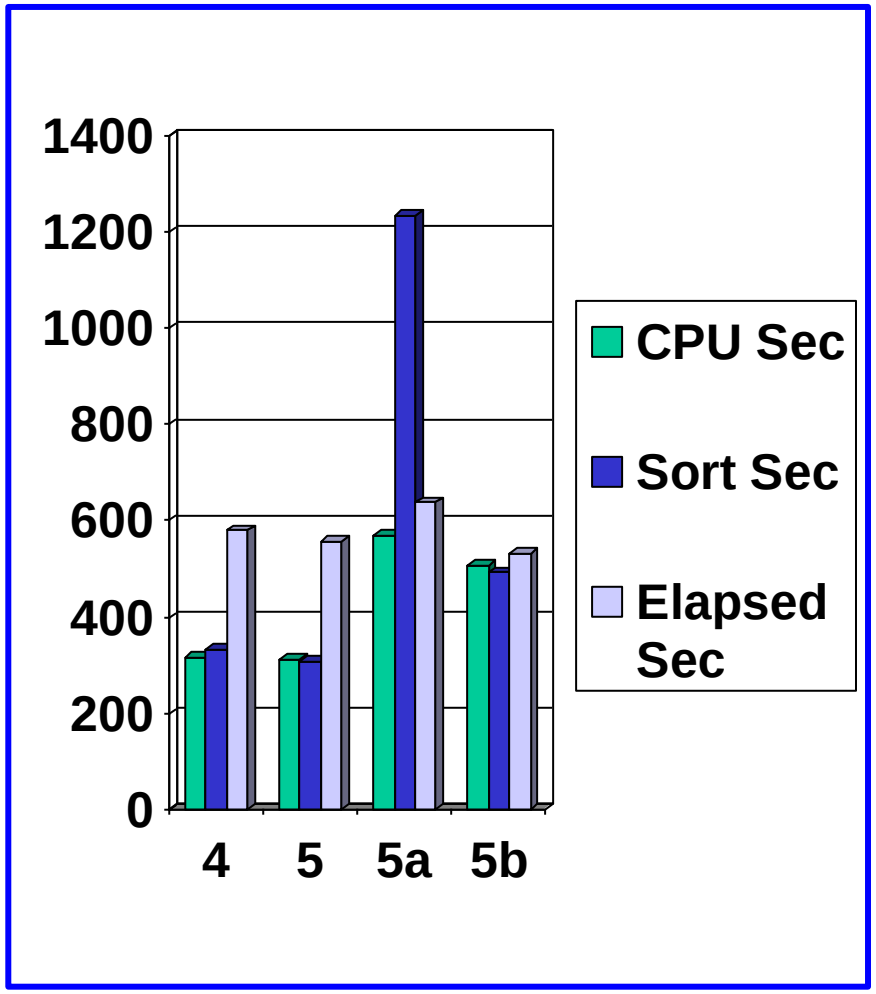
Attendee Notes

- db2batch -d IDUGDB

  -f sort.sql -o f -1 r 20 p 5 o 5 e 0 -r rpt
- Test1- This is intended to be worst case configuration. Our objective now is to tune physical I/O and configuration parameters to see how fast we can make this sort go.
- Test2- Significant improvement just by moving TEMPSPACE 1 container to another disk.
- Test3 & Test4 - Expected better improvements from increasing the number of containers. These results are a bit disappointing. As we'll later see, other configuration parameters have to be changed before multiple containers can provide better performance.

Prefetch Size Science Project 2

- Same 556MB 4.7 million row table. TEMPSPACE has 4 containers on 4 disks.
- Test 4- PrefetchSize = ExtentSize
- Test 5- PrefetchSize = ExtentSize x Number Containers
- Test 5a - INTRA_PARALLEL =YES, DFT_DEGREE = 4
- Test 5b - INTRA_PARALLEL = YES, DFT_DEGREE = -1



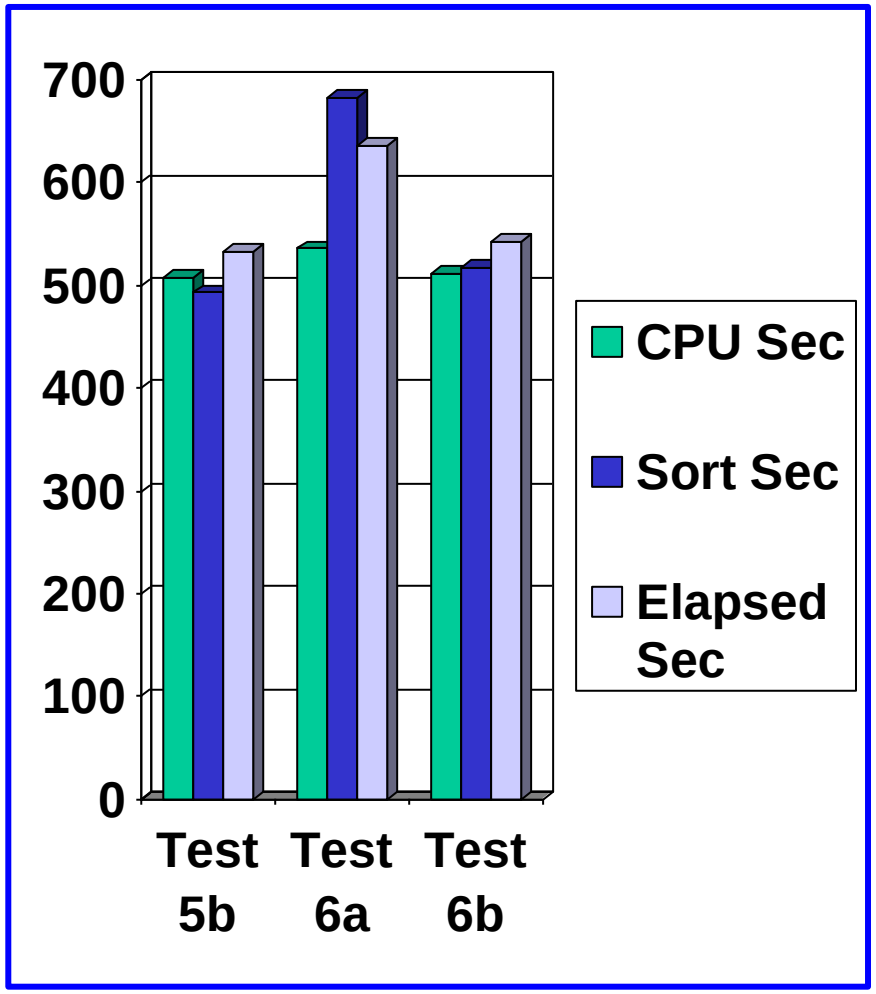
Attendee Notes

- db2batch -d IDUGDB

  -f sort.sql -o f -1 r 20 p 5 o 5 e 0 -r rpt
- Test 5 properly set the prefetch size = number containers X extent size. 557 Seconds, down from 589 seconds… 6% improvement.
- Test 5a turned on intra_parallel w/ dft_degree = 4 (note, 2 cpus on test machine) Ouch! 638 seconds!
- Test 5b changed dft_degree to ANY. The result was best elapsed time so far (533 seconds), with higher CPU expense.

NUM_IOSERVERS Science Project 3

- Sort 556MB 4.7 million row table. TEMPSPACE has 4 containers on 4 disks.
- **Test 5b**- NUM_IOSERVERS = 3, dft_degree = any
- Test 6a- NUM_IOSERVERS = 7, number of disks + 1 (APPR = 1)
- Test 6b- NUM_IOCLEANERS = 2, NUM_IOSERVERS=4



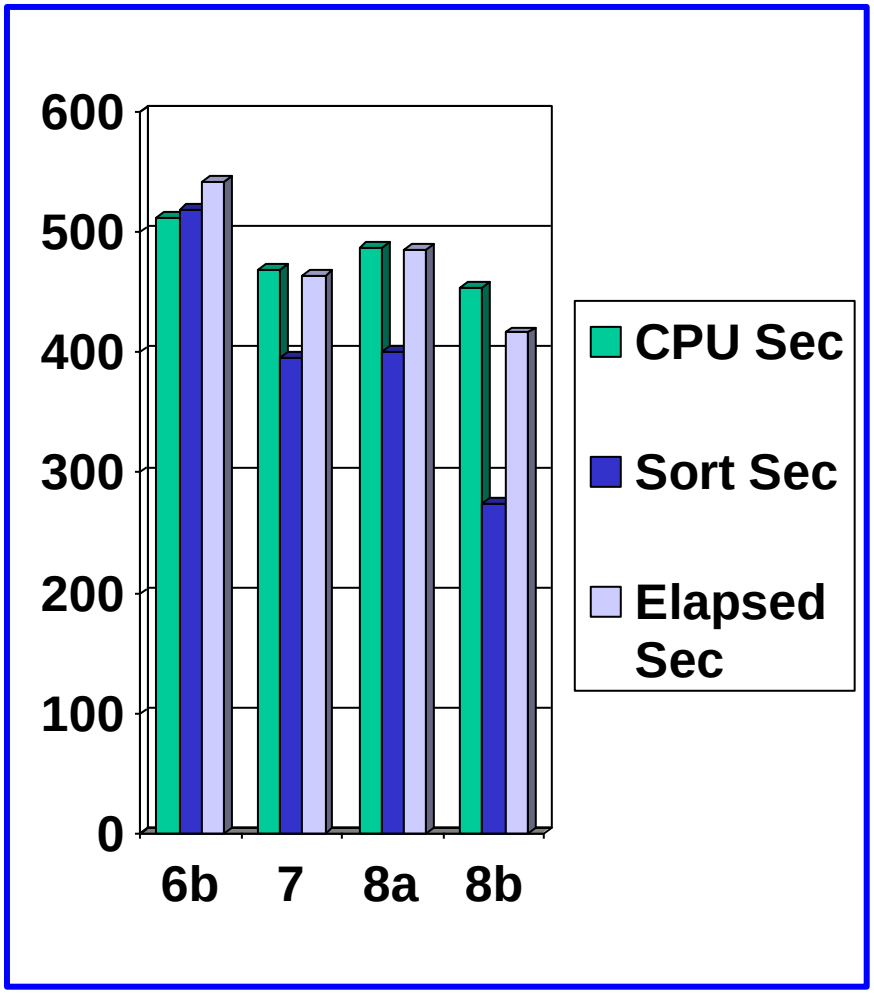
Attendee Notes

- db2batch -d IDUGDB

  -f sort.sql -o f -1 r 20 p 5 o 5 e 0 -r rpt
- Test6a, num_ioservers = 7, bad move.
- Test6b, num_ioservers = 4, num_iocleaners = 2, good move. 9 seconds slower than Test5b (3 ioservers, 1 cleaner).
- Test 7 - Too many I/O servers caused disk contention. Worse yet, bufferpool is over crowded, each prefetch request is yielding only 1 prefetch (async) read, so disks are getting crushed with I/O requests.
- APPR = 122839 / 122070 = ~1 Yikes!
- ARMS = 422799 ms / 265271 = 1.59 ms
- Based on APPR, bufferpool is way too small.

TEMPSPACE BUFFERPOOL Science Project 4

- Sort 556MB 4.7 million row table. TEMPSPACE has 4 containers on 4 disks. 4 IOSERVERS, 2 Cleaners
- Test 6b- IBMDEFAULTBP 4MB
- **Test 7**- IBMDEFAULTBP 64MB (APPR=16!)
- Test 8a- IBMDEFAULTBP 64MB, TEMPSPCBP 64MB
- Test 8b - IBMDEFAULTBP 128 MB!



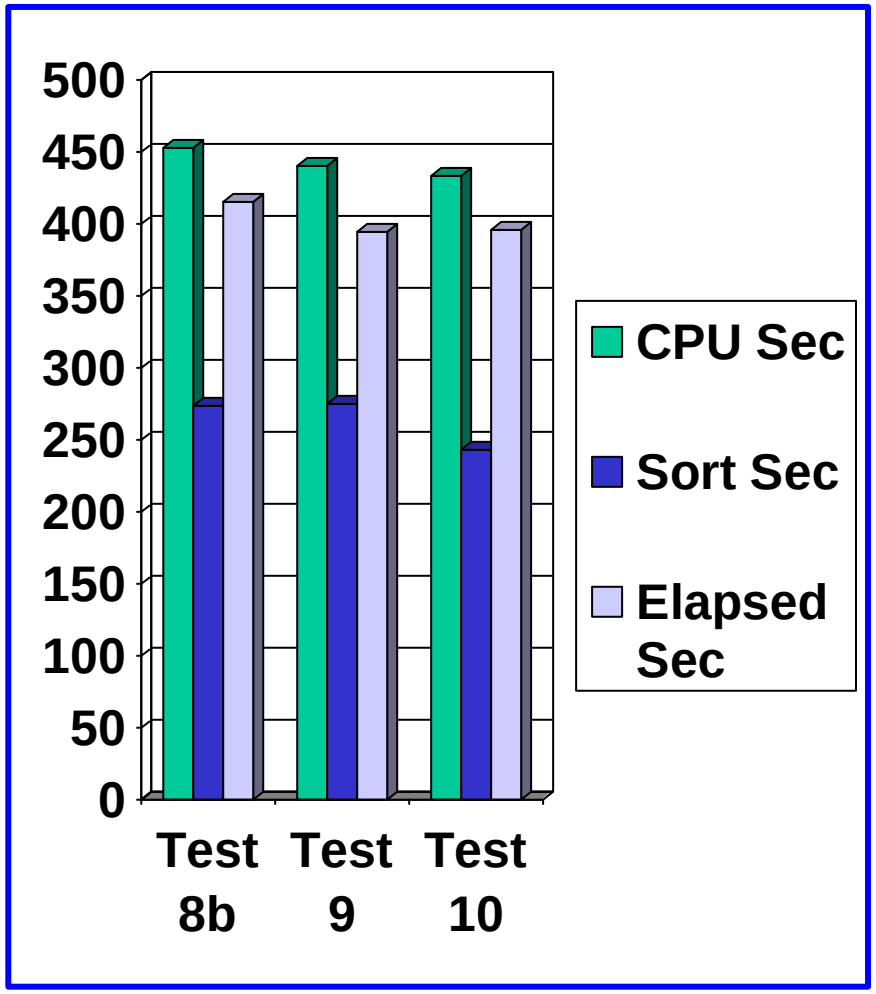
Attendee Notes

- db2batch -d IDUGDB

  -f sort.sql -o f -1 r 20 p 5 o 5 e 0 -r rpt
- Test6 - 4MB bufferpool was overwhelmed with prefetch requests. APPR (Asynchronous pages read per request was = 1!!!)
- Test7- 64MB bufferpool paid off favorably (16 times larger than test 6). APPR jumped to 16!
- Test 8- A separate bufferpool for TEMPSPACE actually hurt performance… cost more CPU
- Test 8b - A single 128MB bufferpool for decision support query w/ large sort is best performing configuration so far…

SMS Data > DMS Data Science Project 5

- Sort 556MB 4.7 million row table is loaded into SMS TS w/ 4 Containers/Dirs, then DMS w/ 4 Containers
- Test 8b- IBMDEFAULTBP 128MB
- Test 9- SMS 1 Dir > SMS 2 Dirs, 5% elapsed improvement
- Test 10- SMS 2 Dir > DMS 2 Containers, 5% elapsed improvement



Attendee Notes

- db2batch -d IDUGDB

  -f sort.sql -o f -1 r 20 p 5 o 5 e 0 -r rpt
- The final frontier - our tablespace containing the 556MB table is SMS, 1 container. What happens when we go to 2 SMS containers? 2 DMS containers? We'd expect some parallelism to feed the input to sort more quickly.
- By putting the data in SMS or DMS with 2 containers instead of one, elapsed time improved by 5%.

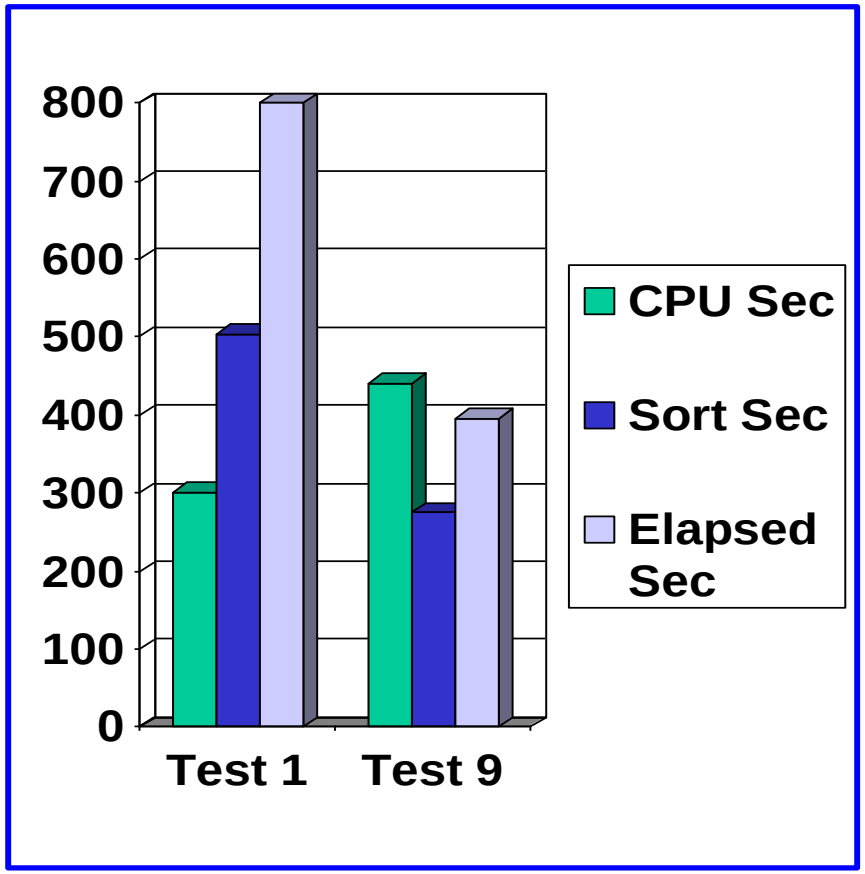
Attendee Notes
iostat

```
tty:      tin          tout    avg-cpu:  % user     % sys      % idle     % iowait
          2.0          25.8                66.4      31.9        0.0        1.7

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           23.4   1898.9     62.0      56818       158
hdisk1           26.1    860.7     32.7        220     25604
hdisk2           27.4    869.3     32.4        312     25772
hdisk3           21.2   1890.2     60.3      56700        16
hdisk4           20.5    863.3     42.4        308     25596
hdisk5           20.4    864.7     41.9        300     25644
cd0               0.0      0.0      0.0          0         0
```

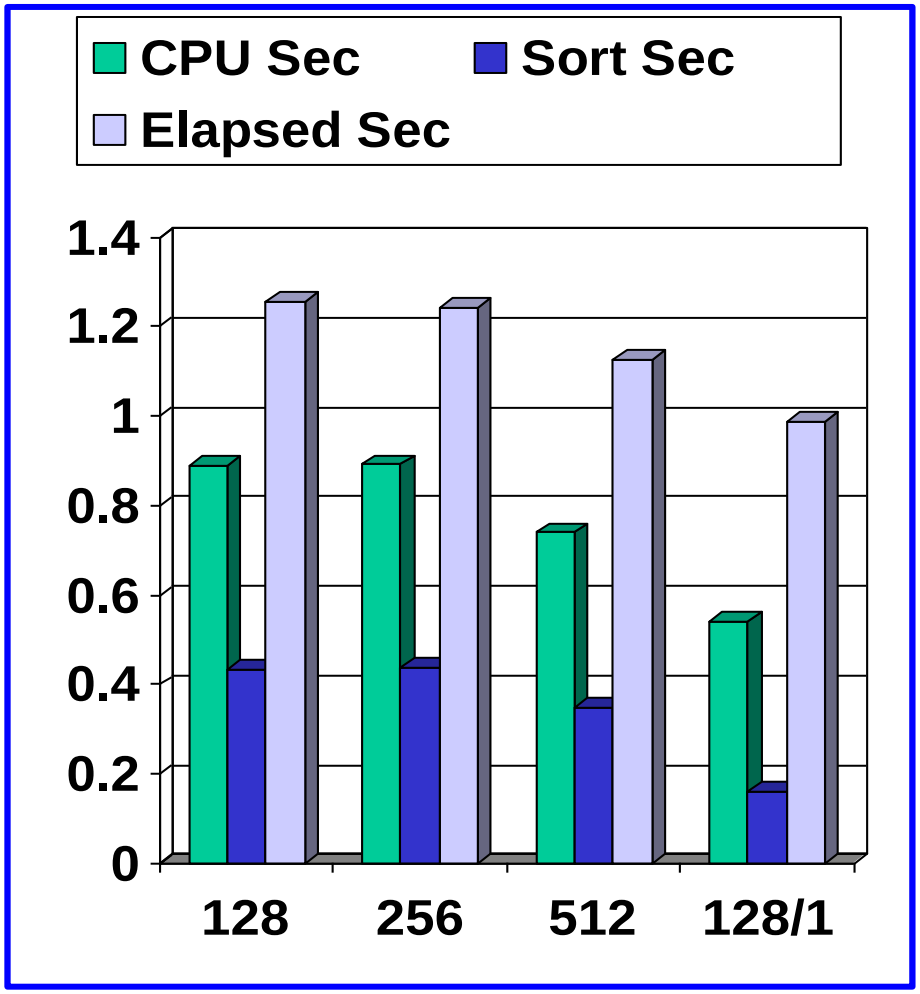
SORT - (You can say this to your kids)
I cut elapsed time of queries in half!

- Test 1 - Worst Case
- Test 9 - Best Case
- Doing physical I/O tuning “right” reduced elapsed time by 51%, Sort time by 45%, at expense of 46% more CPU.
- What is the benefit to your business when you make queries run more than twice as fast?
- Get Results.



Impact of SORTHEAP / Small Sorts Science Project 6

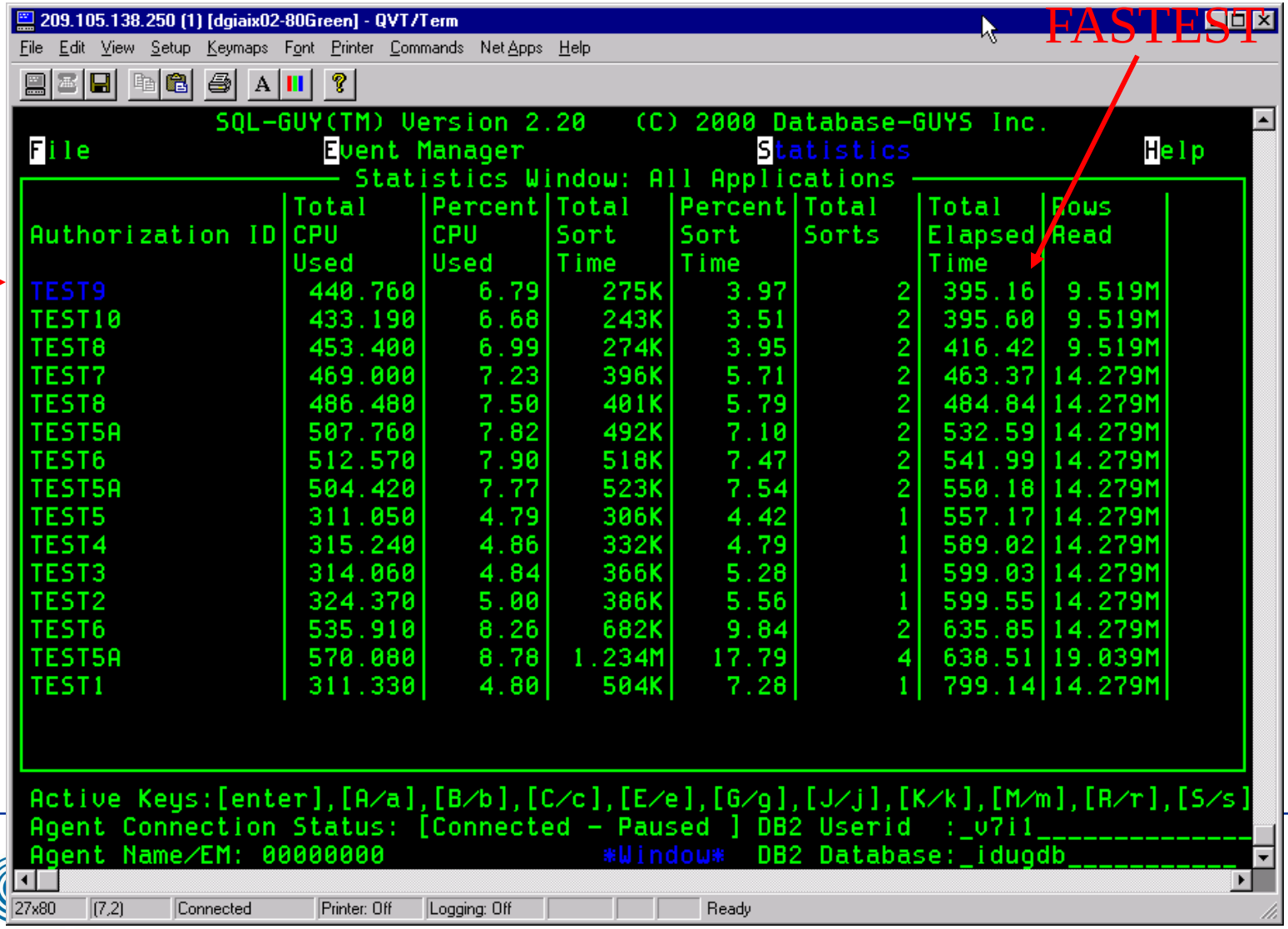
- Test 11 - “Small” Sort Overflows 1MB SORTHEAP, 2 sorts, 1 overflows
- Test 12 - “Small” Sort completes within 2MB SORTHEAP, 2 sorts, 0 overflows
- Avoiding the overflow cut sort time by 20% (1/5th), reduced elapsed time by 10%, and sliced CPU burn by 15%
- 128 - No Order by clause (1 index rid sort)



Attendee Notes

Sortheap	Elap. Sec.	CPU Sec.	Sort Sec.	# Sorts	# Overflows
128	1.257	.890	.433	2	1
256	1.243	.892	.437	2	1
512	1.126	.760	.349	2	0
128, no order by	.989	.540	.160	1	0

Science Projects Performance Summary



Authorization ID	Total CPU Used	Percent CPU Used	Total Sort Time	Percent Sort Time	Total Sorts	Total Elapsed Time	Rows Read
TEST9	440.760	6.79	275K	3.97	2	395.16	9.519M
TEST10	433.190	6.68	243K	3.51	2	395.60	9.519M
TEST8	453.400	6.99	274K	3.95	2	416.42	9.519M
TEST7	469.000	7.23	396K	5.71	2	463.37	14.279M
TEST8	486.480	7.50	401K	5.79	2	484.84	14.279M
TEST5A	507.760	7.82	492K	7.10	2	532.59	14.279M
TEST6	512.570	7.90	518K	7.47	2	541.99	14.279M
TEST5A	504.420	7.77	523K	7.54	2	550.18	14.279M
TEST5	311.050	4.79	306K	4.42	1	557.17	14.279M
TEST4	315.240	4.86	332K	4.79	1	589.02	14.279M
TEST3	314.060	4.84	366K	5.28	1	599.03	14.279M
TEST2	324.370	5.00	386K	5.56	1	599.55	14.279M
TEST6	535.910	8.26	682K	9.84	2	635.85	14.279M
TEST5A	570.080	8.78	1.234M	17.79	4	638.51	19.039M
TEST1	311.330	4.80	504K	7.28	1	799.14	14.279M

Ideas to Take Home

- Transaction Applications - Very small sorts, < 3% Sort Overflows.
  - Eliminate Sorts (an applications worst enemy)
  - Reduce Sort Size, # rows and/or row width
  - Last resort, Increase SORTHEAP
- Decision Support Queries - SORTS HAPPEN
  - Optimize I/O Paths
    - Containers, Placement, Prefetch Size, IO_SERVERS, Single BUFFERPOOL for DATA & TEMPSPACE.

Thank you!

Phil Gunning

DGI

pgunning@breakthroughdb2.com

877-4DB-GUYS X 8