CONSIDERATIONS AND TECHNIQUES FOR OPTIMAL DATABASE PERFORMANCE

Db2 Night Show

2021-12-10

Jim Bean

Cigna Performance & Forensics

Together, all the way.®

Disclaimer

The information contained in this presentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided “as is” without warranty of any kind, expressed or implied. Cigna shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other documentation provided.

Any statements of performance are based on measurements and projections using nonstandard benchmarks in an uncontrolled environment. The actual throughput or performance improvements that you will experience will vary depending on many factors, including DBMS configuration, OS configuration, I/O configuration, storage configuration, workload processed and many others. Therefore, no assurance can be given that you will achieve similar or comparable results.

IBM, DB2 and Db2 are registered trademarks of International Business Machines Corporation.

Linux is a registered trademark of Linus Torvalds.

Database Performance Analyzer (DPA) software produced by SolarWinds®.

Quest Central is a registered trademark of Quest Software.

Topics

- **HW, SW and Currency**
- DB Configuration
- Maintenance
- Data Archive and Purge
- SQL
- Indexing
- Flash Storage (SSD)
- Implementation Details
- Q & A and More Information

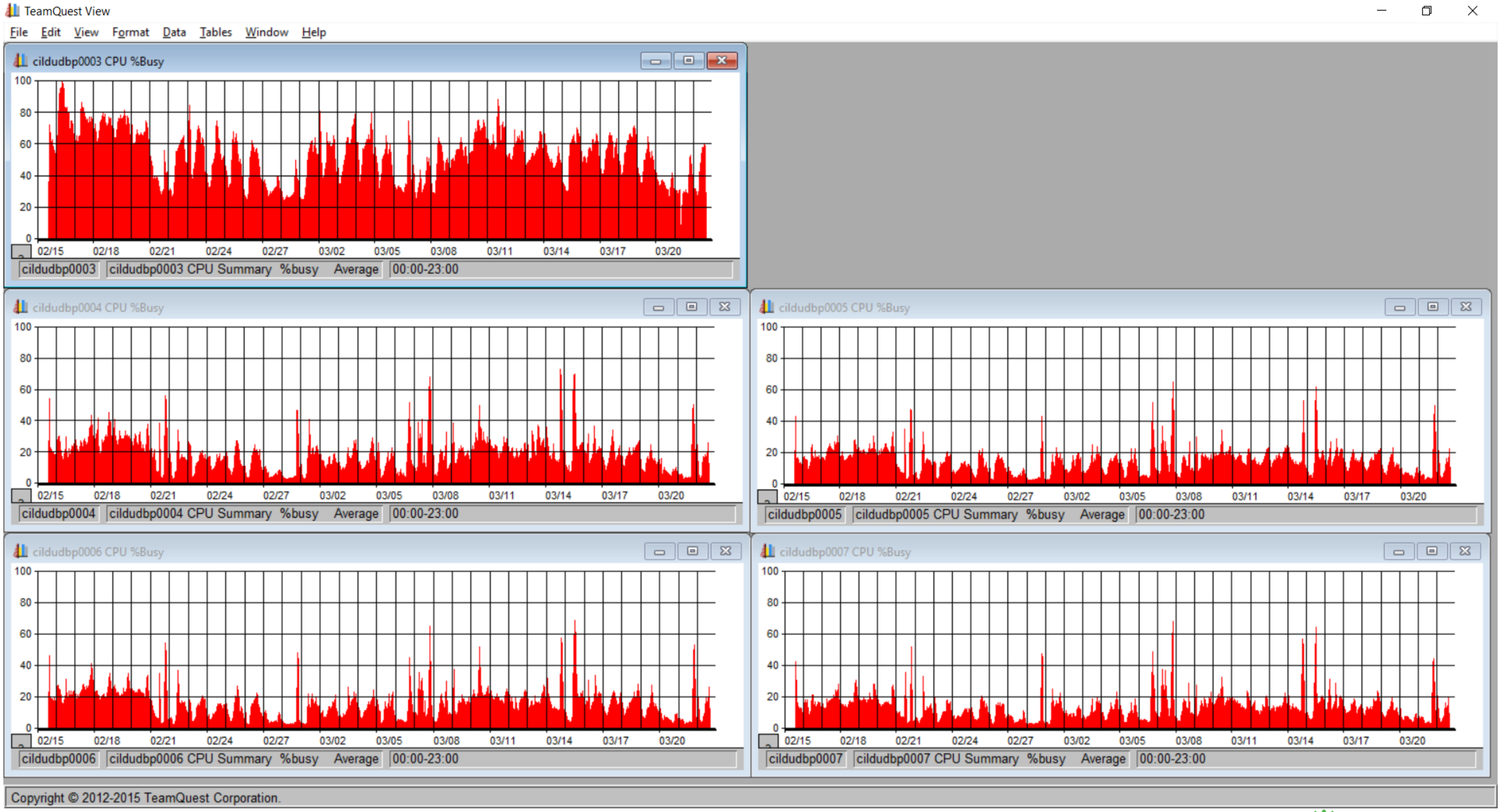
HW, SW and Currency

- DBMS
- OS
- Server
  - More resources (faster CPU; more memory; etc.)
- Storage
- Costs from lack of currency
  - Security vulnerabilities
  - Code defects
  - Support
  - Inability to use / lack of new features and functionality
  - No beneficial performance improvements
  - More CPUs = more licensing costs (generally)

HW, SW and Currency

- Example: 21+TB PRD data warehouse (one of several for this app totaling 85+TB)
- Throughput not fast enough; but “this is not a performance issue”
- Request seeking more and faster CPUs which would increase costs, impact licensing, takes significant man-hours, may require an outage impacting the business, adds risk, etc.
- Issue wasn’t with HW; rather design and implementation of the design resulting in underutilization of the existing HW
- Configuration:
  - One catalog node (coordinator) on one server
  - Eight partitioning nodes (workers) across four servers
  - Each of the five servers has four logical CPUs and 64 GB memory
  - Total of 20 logical CPUs

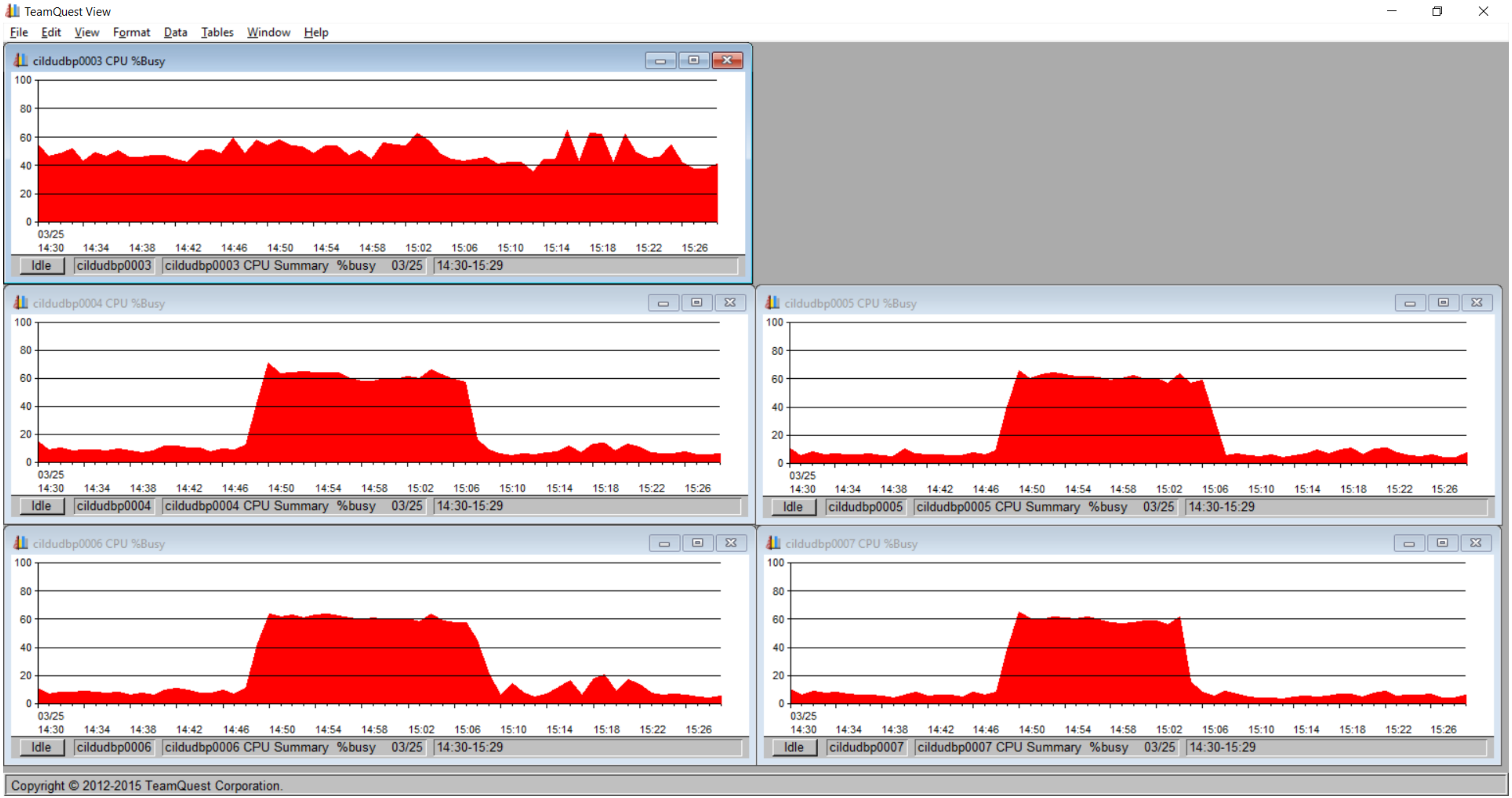
HW, SW and Currency



HW, SW and Currency

- On the prior slide seven logical CPUs in use across the five logical servers
- Very inefficient use of resources from the database perspective
  - Minimal parallelism
  - Partitioning node servers generally under 25% (except backups)
  - Catalog node server very busy, pushing 100% at times and leading to alerts on the logical server
  - Catalog node should not be burdened with “heavy lifting” – it’s supposed to be the coordinator
- Not a shortage of CPU, rather an inappropriate use of CPU
- Based on lack of proper partitioning and placement of database objects
  - Large objects have to be partitioned properly to spread the workload
  - And underlying files spread to obtain maximum parallelism

HW, SW and Currency



HW, SW and Currency

- On the prior slide for 20 minutes 13 logical CPUs in use across the five logical servers
- Essentially using double the CPU, driving significantly more IO and increasing throughput
- Much more efficient use of resources from the database perspective
  - Increased parallelism
  - Partitioning node servers at 60%
  - Catalog node server 50 to 60%
- It’s all about proper partitioning and placement
- Spreads the workload amongst the eight nodes on the four partitioning servers
- Puts more of the HW to use versus having it sit idle
- One example of partitioned workload processing
- Still have much more to go because there is still far too much processing on the catalog node versus the partitioning nodes as the first slide showed

HW, SW and Currency

- Direct ties via configuration parameters
- Parallelism/no parallelism and number of cores vs. speed of cores
- To compress or not to compress
  - impact on processing (CPU)
  - impact on storage and memory footprint
- Offload processing where possible to less expensive processing platforms
  - IDAA (IBM Db2 Analytics Accelerator)
  - Reporting databases
  - Minimized (data: horizontally and vertically)
  - Optimized for specific processing (DBMS configuration; storage; indexing)
- Storage
  - SSD (Solid-State Drives) or Flash
  - Spindle
  - Optical

Topics

- HW, SW and Currency
- **DB Configuration**
- Maintenance
- Data Archive and Purge
- SQL
- Indexing
- Flash Storage (SSD)
- Implementation Details
- Q & A and More Information

Database Configuration

- Some database configuration parameters control enablement of certain features
  - Automatic maintenance activities (reorg, rebuild, stats, etc.)
  - Compression
  - Extent size, prefetching, etc.
  - External tables and locations
  - Parallelism and maximum degree
  - Query optimization levels, statement concentration
  - HA and DR
  - Workload manager
  - Monitoring / tracing activation
  - Self-tuning, health monitoring

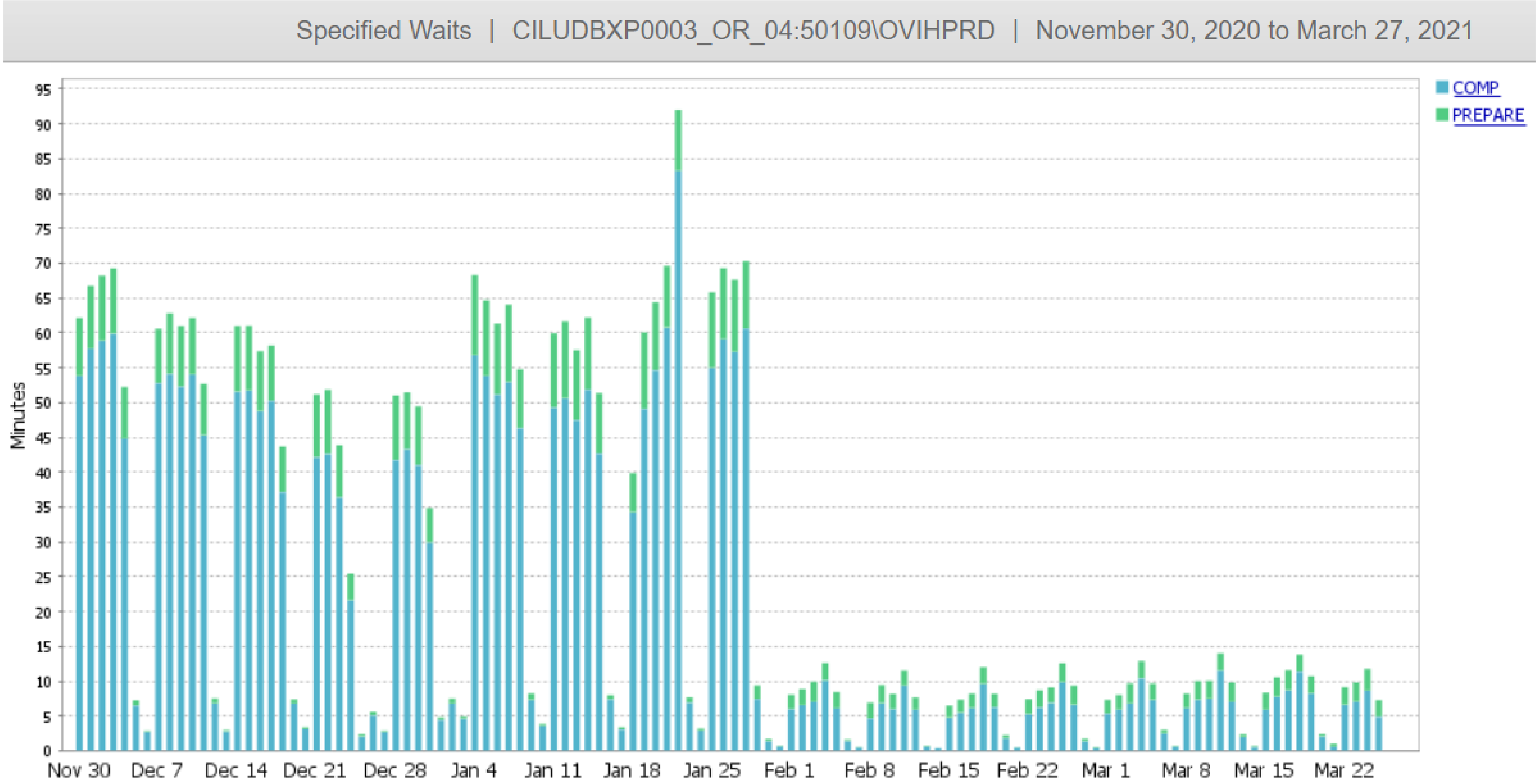
Database Configuration

- Some database configuration parameters control and allocate hardware resources
  - Memory allocations (caches, heaps, etc.) including defaults, min, max, auto
  - Automatic memory management
  - Network buffers
  - HWMs / maximum limits (open files, locks, sessions, init. agents, pooled agents, etc.)
  - Page cleaners, async readers, etc.
  - CPUs, speed (auto detect)
  - Logging (active log, sync points, transaction manager, diaglog, etc.)
  - Recovery (e.g. backups and trackmod, etc.)
  - Utility processing
- Example: Proper use of the Db2 LUW package cache (without statement concentrator)
  - With proper use of parameter markers, saves significant space in the package cache
  - Also reduces CPU significantly
  - Repeatedly-executed statements with varying WHERE criteria in a highly-active OLTP environment run faster and consume less resources than the repeated compilations

Database Configuration

- For a CRM app
  - Recently implemented application changes reduced dynamic SQL compilation and prepare time
  - Compiling a simple dynamic SQL statement may take a ms or two
  - Some can take as much as 500 ms or even more
  - Although generally small, as we'll see later, in very high quantities this can lead to significant time
  - And this is usually all CPU time as all necessary information is cached, assuming other settings in the DBMS are configured appropriately
- The following slide shows the impacts of application changes for six sets of SQL calls
- Reduced total dynamic SQL compilation and prepare time about 60 minutes per day, or about 85% - all CPU time (catalog cache)
- Reduced total database processing time over 12% by eliminating 10s of millions of compilations per day
- Picked the top six sets of SQL calls to get the "biggest bang for the buck"

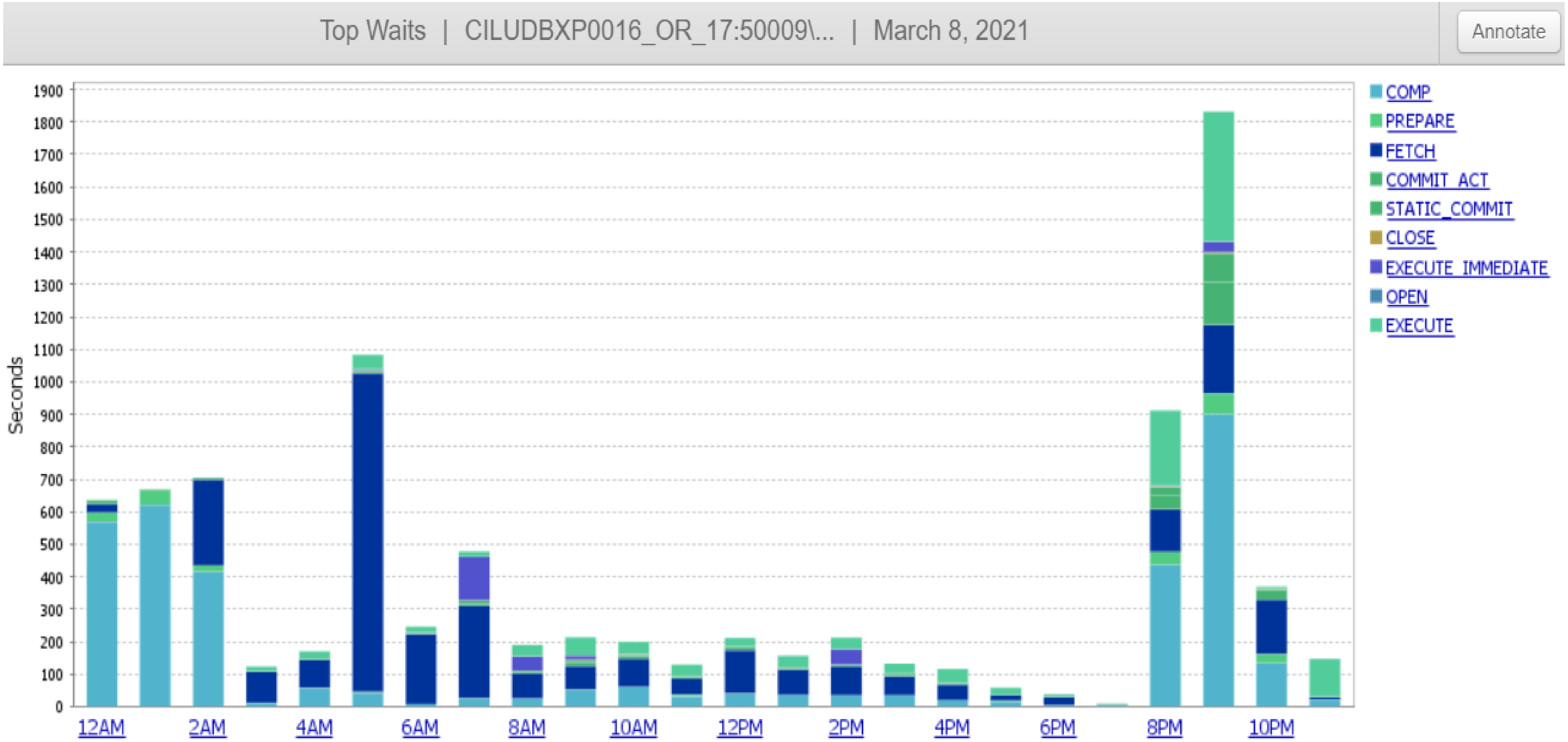
Database Configuration



Database Configuration

- Seen similar issues with batch processing in another app
- Next slide shows high compile and prepare times
- Over the full 24-hour day, averaged >43% of the total database time
- For certain hours, much higher
  - 00:00 93%
  - 01:00 100%
  - 02:00 62%
  - 20:00 52%
  - 21:00 52%
- Significant improvements to batch processing can be realized by using parameter markers versus literals in dynamic SQL calls, to eliminate redundant compiling
- In many of these cases the compile times exceed the execution times

Database Configuration



Database Configuration

- Parameter markers / host variables
  - Use appropriately for frequently-executed dynamic SQL calls that have varying WHERE criteria
  - Eliminates unnecessary compile/prepare/bind time to repetitively determine access paths
  - Compile time can exceed SQL call execution time, and frequently does for optimized SQL calls
  - Can have a drastic impact on CPU consumption reflected as high COMP, PREP or BIND time
  - Can calculate savings from using parameter markers
  - Likewise can identify those candidates ***not*** using parameter markers when they should be
  - A recent example; the next slide shows dynamic SQL ***not*** using parameter markers (how ***not*** to do it)
  - The subsequent slide shows dynamic SQL calls using parameter markers appropriately; examine the execution count (Execs) versus the compilation count (Comps)
  - From the final SQL call using parameter markers appropriately, we see

    ```
    update INTXN.TASK_ATTR set BUS_STEP_ID=? where TASK_ATTR_ID=?
    ```
  - Executed 11,027,200 times since first being inserted into the dynamic SQL statement cache (package cache)
  - Only two compiles at 1 ms each for a total of 2 ms
  - But … 11,027,198 compiles at 1 ms each were saved, for a total of 11,027,198 ms
  - That's 11,027 seconds or over 3 hours of CPU time, for just one SQL statement in the cache

Database Configuration

```
 Execs Comps    Comp_ms SQL
------- -----   ------- ----------------------------------------------------------------------------------------------------------------
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612792)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612794)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612797)
      1     1         2 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612798, 1266612799)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612801)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612802)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612803)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612805)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612806)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612807)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612810)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612811)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612818)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612820)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612821)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612834)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612839)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612842)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612844)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612845)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612846)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612847)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612850)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612853)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612854)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612862)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612863)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612865)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612866)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612867)
      1     1         2 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612868, 1266612869, 1266612870)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612875)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612877)
      1     1         1 update intxn.cmnt c set c.case_id =? where c.cmnt_id in (1266612878)
```

Database Configuration

```
  Execs Comps   Comp_ms SQL
------- -----   ------- -----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
3610593     2         1 update intxn.CMNT set case_id=? where CMNT_ID=?
 131849     1         1 update INTXN.CORR_RCPN set ADDR_LINE1=?, ADDR_LINE2=?, CITY_NAME=?, FIRST_NAME=?, LAST_NAME=?, MIDDLE_INITIAL=?, PRIM_RCPN=?, STATE_CODE=?, SUFFIX_TEXT=?, ZIP=? wh
  74516     1         1 update INTXN.CORR_RCPN set DOC_ROW_ID=null where DOC_ROW_ID=?
 171786     2         1 update INTXN.CORR_RCPN set DOC_ROW_ID=? where RCPN_ROW_ID=?
1803335     1         0 update intxn.DOC set BUS_STEP_ID=null where BUS_STEP_ID=?
  74326     2         2 update intxn.DOC set BUS_STEP_ID=? where DOC_ROW_ID=?
 131767     2         2 update intxn.DOC set CASE_ID=?, ATCH_ID=?, DOC_TYPE=?, DOC_DESC=?, CREATED_BY=?, CREATED_TS=?, LAST_UPD_BY=?, LAST_UPD_TS=?, CREATOR_ROLE=?, UPDATER_ROLE=?, CORR_I
 492801     1         0 update intxn.DOC set case_id=null where case_id=?
1372782     2         1 update INTXN.FIELD_AUDIT set case_id=? where FIELD_AUDIT_ID=?
1289194     1         1 update INTXN.FIELD_AUDIT set FIELD_NAME=?, LAST_UPD_VAL=?, LAST_UPD_FULLNAME=? where FIELD_AUDIT_ID=?
1568128     2         4 update INTXN.INTXN_CONTACT set last_name=?, first_name=?, full_name=?, addr_line1=?, addr_line2=?, city=?, state=?, zip=?, email_id=?, fax_num=?, phone_num=?, PART
3073062     2         4 update INTXN.INTXN_CONTACT set last_name=?, first_name=?, full_name=?, addr_line1=?, addr_line2=?, city=?, state=?, zip=?, email_id=?, fax_num=?, phone_num=?, PART
4737367     2         1 update INTXN.INTXN_EXT set doc_num=?, rcvd_dt=?, acc_method=?, intxn_status=?, rqst_rel=?, followup_intxn_id=?, system_intxn_id=?, last_upd_by=?, CREATOR_ROLE=?, C
 250460     1        11 update intxn.RQST_EXT set PART_PROVIDER=?, REVIEW_TYPE=?, APPEAL_STATUS=?, PROCESSED_STATE=?, RECEIPT_MTHD=?, APPEAL_TYPE=?, CIGNA_RECVD_DATE=?, APPEAL_RECVD_DATE=
3462834     2         2 update intxn.rqst r set r.cmnt_text = ?, r.cmnt_ind = '1', r.updater_fullname = ?, r.last_upd_by = ?, r.last_upd_ts = ? where r.case_id = ?
      1     1         4 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
      2     1         4 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
      4     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
      5     1         4 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
      2     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
      4     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
     18     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    603     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?, ?)
     52     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?, ?)
     56     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?, ?)
    132     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?, ?)
  39390     1         2 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?, ?)
 107023     1         3 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?, ?)
 234909     1         2 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?, ?)
 894731     1         2 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?, ?)
 374267     1         1 update intxn.rqst r set r.rqst_status_code = ?, r.last_upd_by = ?, r.last_upd_ts = ? where case_id in (?)
2698446     2         3 update intxn.RQST set RQST_EXT_ID=?, TYPE=?, sla=?, rqst_reason_code=?, rqst_status_code=?, alliance_name=?, alliance_partner=?, cigna_role=?, RQST_CATG_ID=?, RQST
1382936     1         1 update INTXN.TASK_ATTR set attr_id=?, attr_val=? where TASK_ATTR_ID=?
11027200    2         1 update INTXN.TASK_ATTR set BUS_STEP_ID=? where TASK_ATTR_ID=?
```

Database Configuration

- And of course there are exceptions – “it depends”
- Example: Situation where using parameter markers would degrade performance
  - Non-uniform spread of indexed data (STATUS_CD)
  - With literals provided, and valid runstats, the optimizer knows what to expect
  - With so many rows qualifying (295M rows or 65%), Db2 chose table scan with prefetch as it should
  - With parameter markers, it doesn’t know what to expect, and chose the access path based on certain “assumptions”
  - In this particular case, the optimizer generated an access path using a non-clustered index to subsequently access data pages even though 65% of the data qualified
  - Results in “death by random IO”
  - The SQL will run significantly longer, even though EXPLAIN showed a lower cost (estimated timerons)
  - REOPT VARS option may help
    - REOPT ONCE (first time; may help; “should” choose table scan with list prefetch again)
    - REOPT ALWAYS (back to repeated compiles)

Database Configuration

- Miscellaneous
  - Database status, level, etc.
  - Authentication, security and security groups
  - Data type defaults and conversions
  - Date and other data type formatting
  - Diagnostics (level; location; etc.)
  - Language
  - Codepage
  - Locking and deadlocking, timeout values
  - User exits

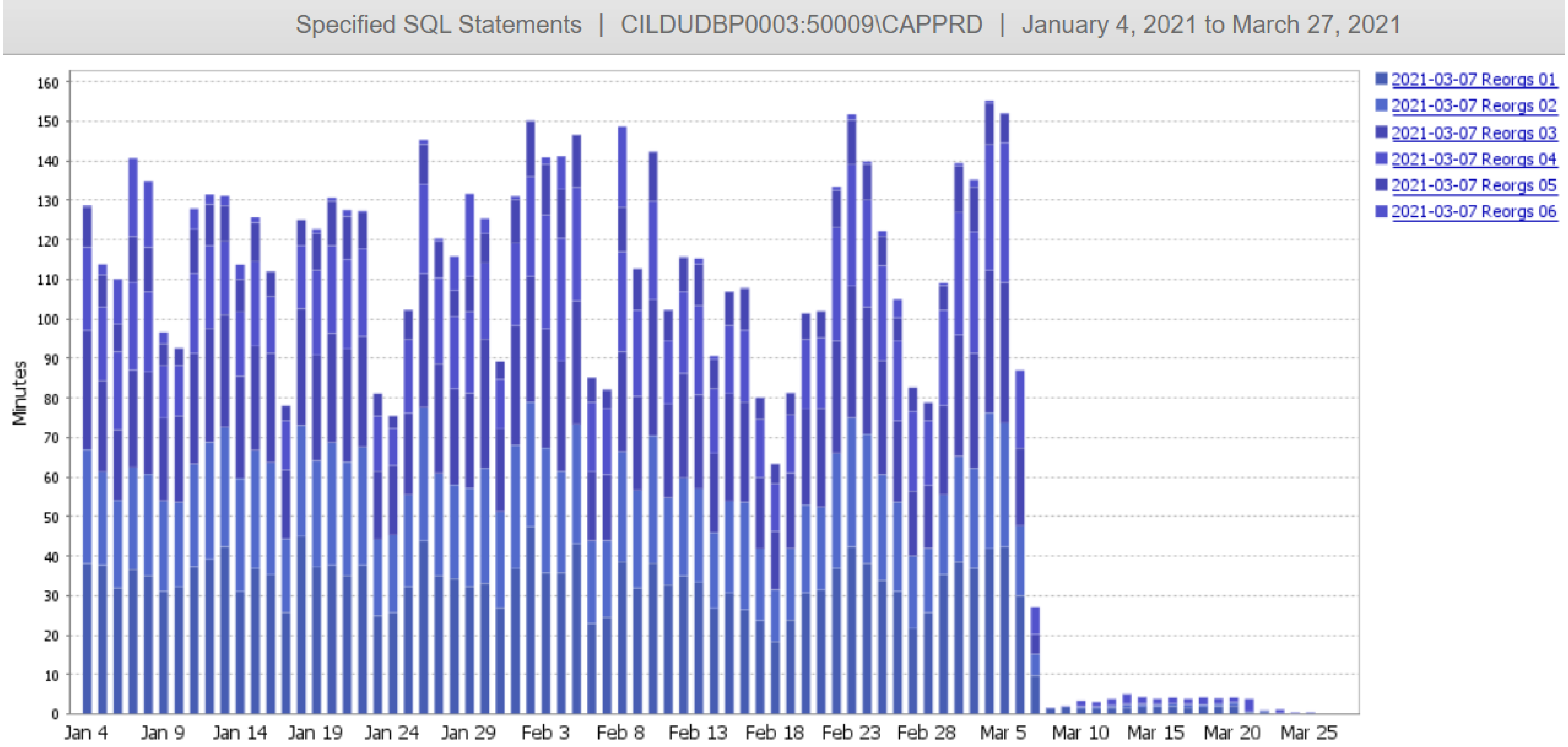
Topics

- HW, SW and Currency
- DB Configuration
- **Maintenance**
- Data Archive and Purge
- SQL
- Indexing
- Flash Storage (SSD)
- Implementation Details
- Q & A and More Information

Maintenance

- Reorgs; when to reorg and why
  - Online vs. offline; be aware of the differences
  - Active tables needing reorg nightly or weekly; offline reorg quarterly
  - Traditional approach as to when and what to reorg
  - Logically deleted rows (e.g. purge; more later)
  - Data re-located to less than optimal page placement (e.g. overflow access)
  - Use a tool (or develop one) to track overflow accesses by table
  - This is an excellent indicator of rows out of place **<u>and</u> that are read**, indicative of needing a reorg
- Database tuning started for the large data warehouse
  - First set of reorgs completed 3/7, and more on 3/21 – see next slide
  - Much more database tuning remains including repartitioning, SQL tuning, index tuning, data purging, etc.

Maintenance



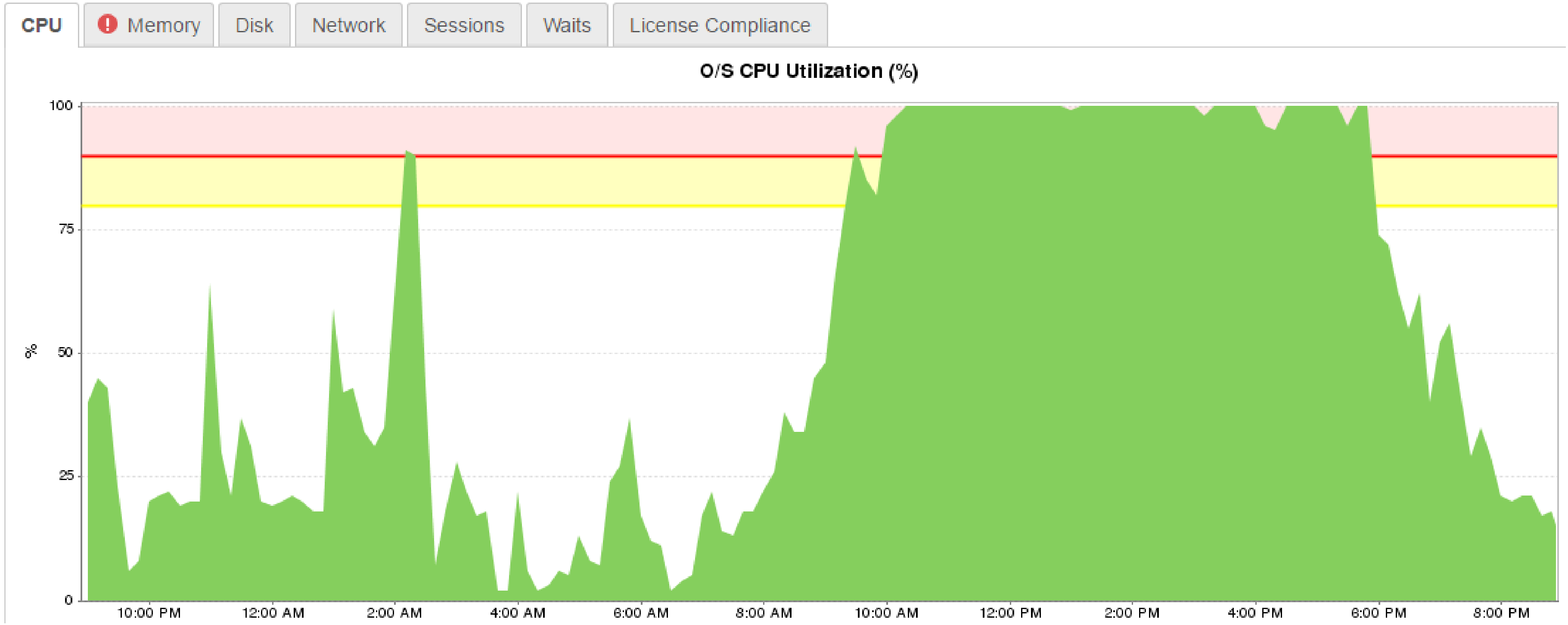
Maintenance

- Runstats can solve major performance issues; two examples below
  - First Incident from a while back
  - Called in to investigate major CPU and locking issues on a Sunday afternoon
  - Maintenance on Db2 zOS side, required bringing down a web app early Sunday morning
  - Issue encountered later that morning after everything was back up

  - Second issue in January
  - This was a reoccurrence
  - Performance would be fine for weeks, then would degrade for a while; then improve again
  - No explanation as to why at the time

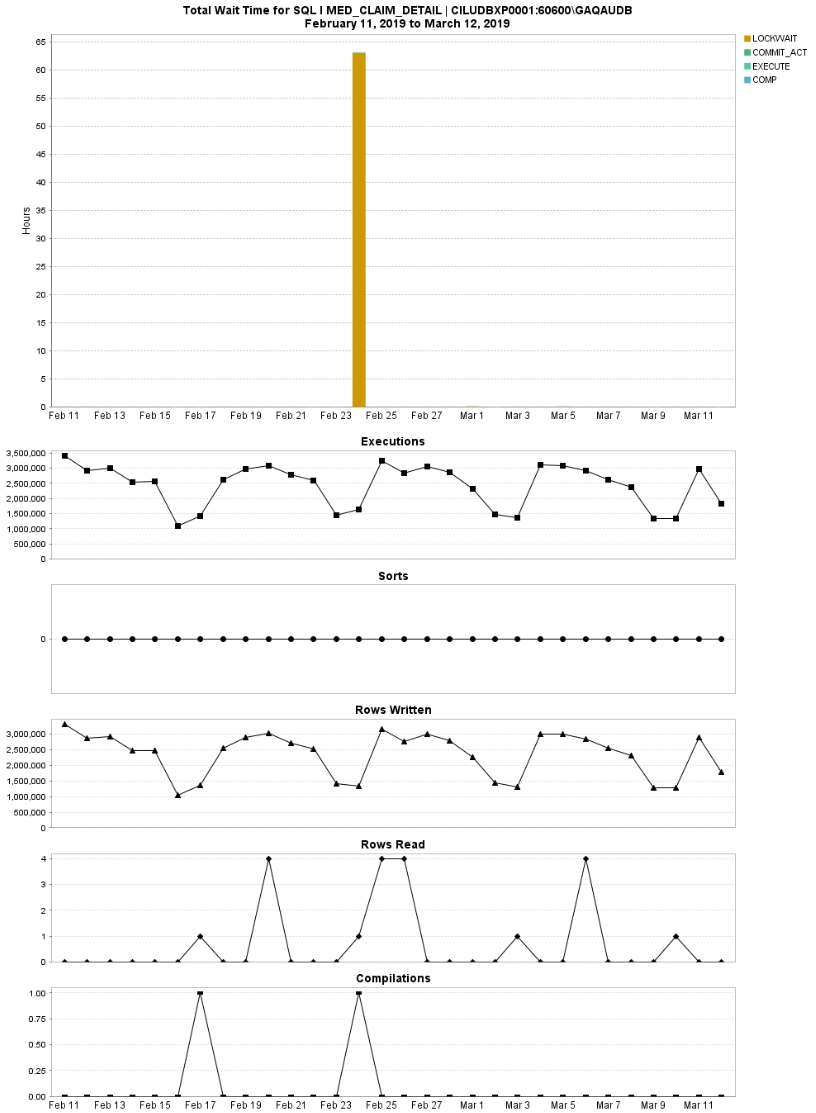
Houston, we have a problem

- An obvious problem, although they're not always as obvious and certainly not always CPU (although there is a tendency to focus on CPU).



Identifying “candidates” for improvement (2|6)

- One thing is for certain – be sure you understand what the problem is and the impact **BEFORE** tuning or taking any actions; else you could be tuning the wrong SQL or taking the wrong actions.
- Example: INSERT statement running long due to locking. Everyone was focused on the INSERTs and locking, but it wasn’t a locking issue; it was a performance issue.
- Many performance issues masquerade or appear as locking issues. The issue wasn’t even with the INSERT; it was a SELECT (two table join) including the table that was INSERTed to, resulting in CPU exhaustion, locking, etc. And it didn’t require an index to resolve either.


Total Wait Time for SQL I MED_CLAIM_DETAIL | CILUDBXP0001:60600\GAQAUDB
February 11, 2019 to March 12, 2019
LOCKWAIT
COMMIT_ACT
EXECUTE
COMP
Hours
Feb 11 Feb 13 Feb 15 Feb 17 Feb 19 Feb 21 Feb 23 Feb 25 Feb 27 Mar 1 Mar 3 Mar 5 Mar 7 Mar 9 Mar 11
Executions
Sorts
Rows Written
Rows Read
Compilations

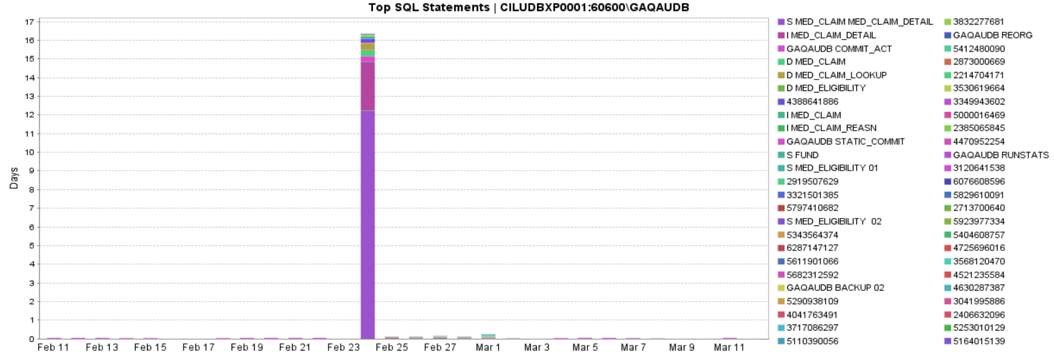
Identifying "candidates" for improvement (4|6)

- With poor response, a crisis call was initiated.
- The focus on this issue was locking experienced by certain INSERTs. Without full understanding, an application recycle was attempted to no avail; same thing happened afterward.
- Upon joining the crisis call, learned they were about to stop the application, recycle Db2 and restart the application (again).
- Halted that activity since the root cause of the problem wasn't fully understood; therefore how could they say this would help?

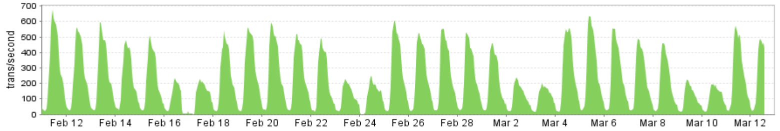
Identifying “candidates” for improvement (5|6)

- Examined other data and stats; see reports on the next slide. Reviewing the issue, shifted focus to a SELECT with a two-table join including the table suffering from locking when performing INSERTs.
- From reports on the next slide, it’s clear the execution count for the SELECT was normal for a Sunday, yet the access path must have changed resulting in significantly more row reads overall and per execution.
- Instead of averaging under 70 row reads and under 0.2ms per execution, now the SELECT was averaging over 370K row reads and over 35 seconds per execution.

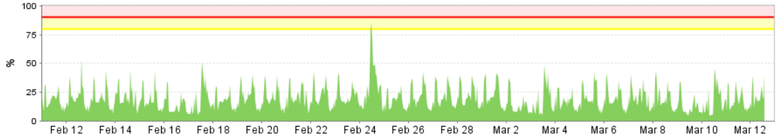
Top SQL Statements | CILUDBXP0001:60600\GAQAUDB



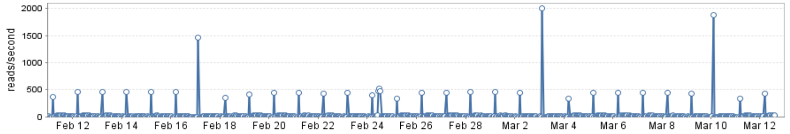
DB Transaction Rate



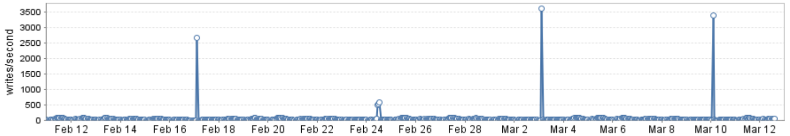
O/S CPU Utilization



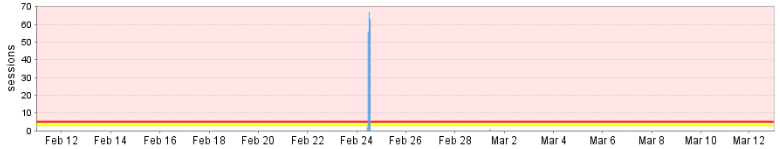
DB Physical Read Rate



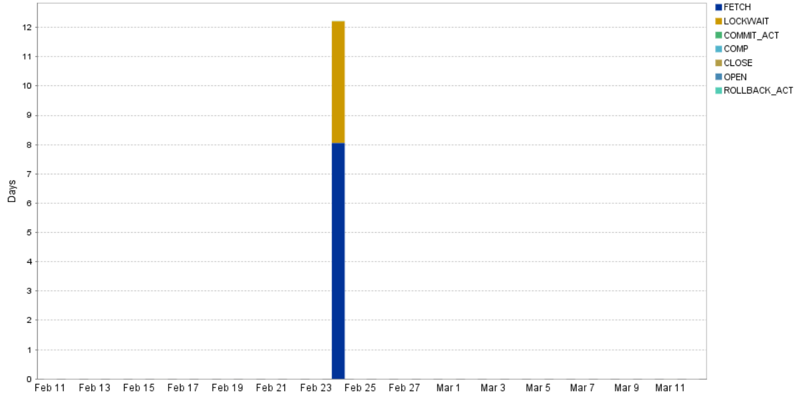
DB Physical Write Rate



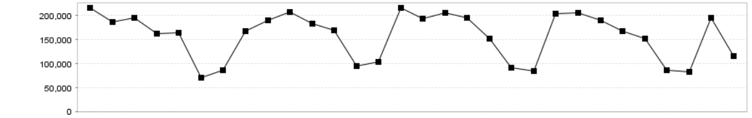
DB Blocked Sessions



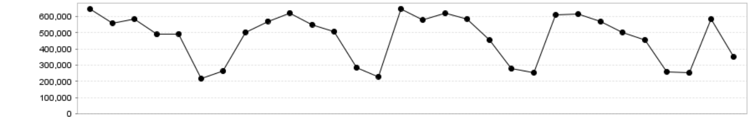
Total Wait Time for SQL S MED_CLAIM MED_CLAIM_DETA... | CILUDBXP0001:60600\GAQAUDB
February 11, 2019 to March 12, 2019



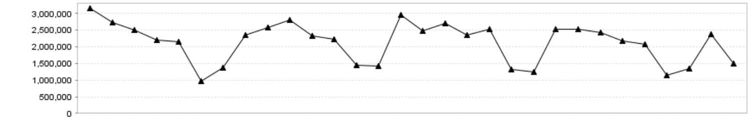
Executions



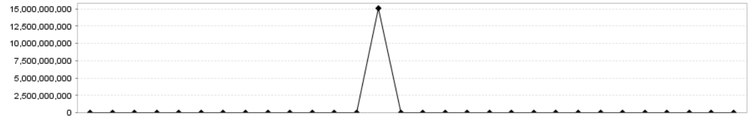
Sorts



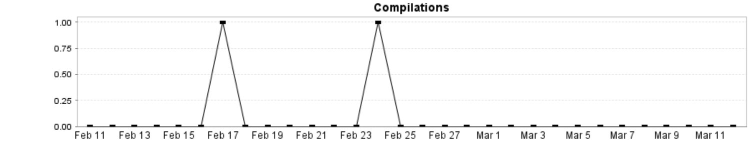
Rows Written



Rows Read



Compilations



When a new index WASN'T needed #1 (1|10)

- Next step is to examine the access path.
- From the next slide showing the "before" and "after" access paths, there's a slight difference resulting in very different performance levels.
- The access path change was a result of running stats when the two tables accessed were empty!
- Solution was to run stats with sufficient data in the tables.
- Difference was incredible; instead of averaging over 370K row reads and over 35 seconds per execution, now the SELECT was averaging under 70 row reads and under 0.2ms per execution again.
- Performance of this two-table SELECT was back to normal, stabilizing CPU and eliminating locking. Performance of the INSERT was back to normal as well.

```
Data operation complete (40.13)
16 Sorted table data scan (40.13)
  15 Data is sorted (40.13)
    14 Sorted table data scan (40.13)
      13 Data is sorted (40.13)
        12 Sorted table data scan (40.13)
          11 Data is sorted (40.13)
            10 Nested loop join (40.13)
              9 Fetch table data - Inner join feed GAQA.MED_CLAIM_DETAIL (19.50)
                8 Index scan GAQA.MED_CLAIM_DETAIL0X (6.78)
              7 Nested loop join - Outer join feed (22.82)
                6 Nested loop join - Inner join feed (22.82)
                  5 Fetch table data - Inner join feed GAQA.MED_CLAIM_DETAIL (13.54)
                    4 Index scan GAQA.MED_CLAIM_DETAIL0X (6.77)
                  3 Fetch table data - Outer join feed GAQA.MED_CLAIM (9.28)
                    2 Index scan GAQA.MED_CLAIM0X (6.77)
                1 Table access full - Outer join feed SYSIBM.GENROW (0.00)
```

Object name	Type	Rows	Pages	Statistics time	File pages	Pct free	RI parents	RI children	RI self	Key unique	Key columns	Col count	Check cour
GAQA.MED_CLAIM	Table	0	0	2019-02-24 05:30:40	1	10	0	3	0	0	1	26	0
GAQA.MED_CLAIM_DETAIL	Table	0	0	2019-02-24 05:30:41	1	-1	1	0	0	0	0	25	0

Index name	Type	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	Internal ID	User defined	System req
GAQA.MED_CLAIM0X	Clustering	Unique	1	1	0	0	100	4	2019-02-24 05:30:40	1	Yes	0
GAQA.MED_CLAIM1X	Regular	Primary Index	1	1	0	0	100	1	2019-02-24 05:30:40	2	Yes	1

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Part key seq	Nulls	Null cour
MBR_NUM	CHARACTER (10)	No	10	Ascending	N/A	0	"	"	0	0	No	0
MBR_SFX_RELCD	CHARACTER (2)	No	2	Ascending	N/A	0	"	"	1	0	No	0
MBR_SORT_ORDER	SMALLINT (2)	No	2	Ascending	N/A	0			2	0	No	0
CLM_NUM	CHARACTER (15)	No	15	Ascending	1	0	"	"	3	0	No	0
CLM_SRC_SYS_CD	CHARACTER (1)	No	1	N/A	N/A	0	"	"	4	0	No	0
CLM_STAT_DESC	CHARACTER (20)	No	20	N/A	N/A	0	"	"	5	0	No	0
CLM_PROCSD_DT	DATE (4)	No	4	N/A	N/A	0	"	"	6	0	No	0

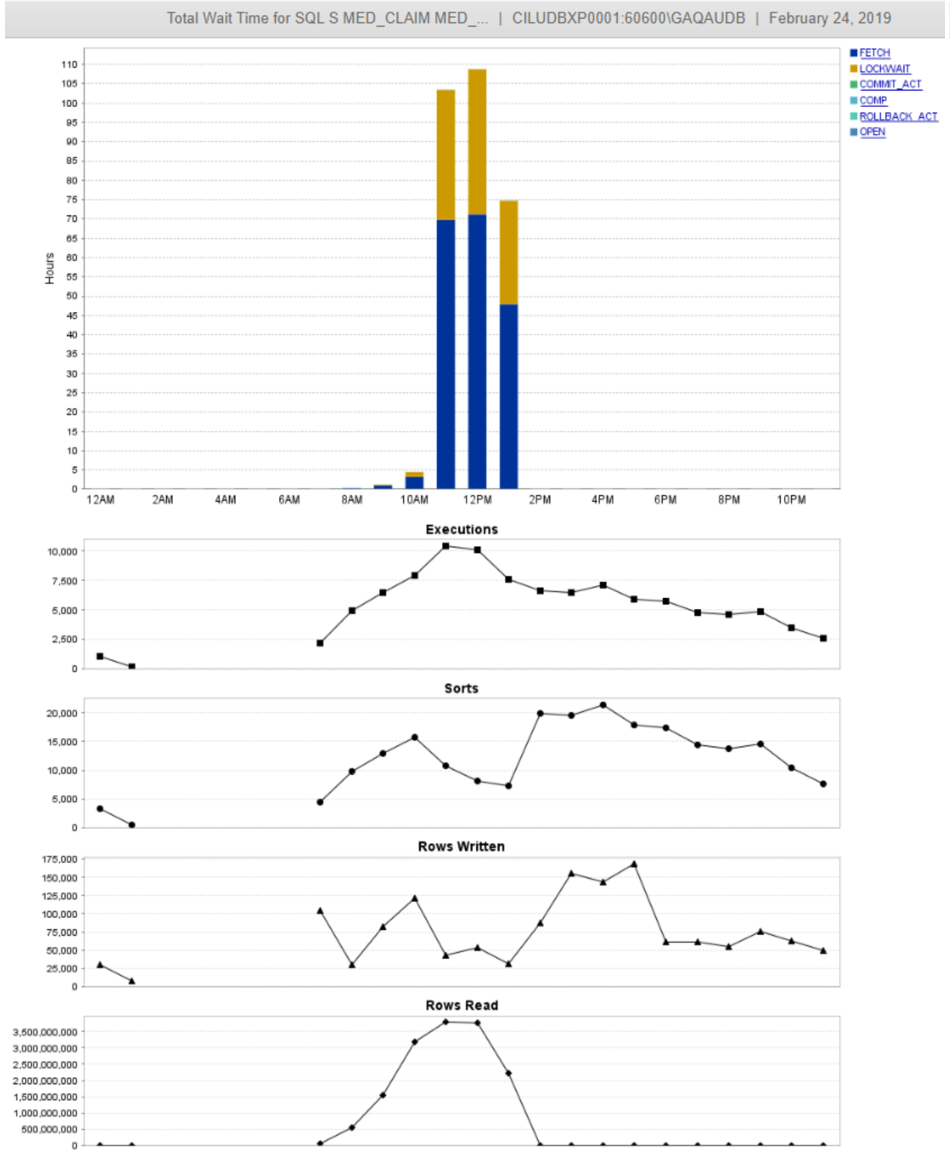
```
Data operation complete (58.35)
16 Sorted table data scan (58.35)
  15 Data is sorted (58.35)
    14 Sorted table data scan (58.35)
      13 Data is sorted (58.35)
        12 Nested loop join (58.35)
          11 Fetch table data - Inner join feed GAQA.MED_CLAIM_DETAIL (20.31)
            10 Index scan GAQA.MED_CLAIM_DETAIL0X (13.54)
          9 Sorted table data scan - Outer join feed (44.81)
            8 Data is sorted (44.81)
              7 Nested loop join (44.81)
                6 Nested loop join - Inner join feed (44.81)
                  5 Fetch table data - Inner join feed GAQA.MED_CLAIM_DETAIL (20.31)
                    4 Index scan GAQA.MED_CLAIM_DETAIL0X (13.54)
                  3 Fetch table data - Outer join feed GAQA.MED_CLAIM (20.41)
                    2 Index scan GAQA.MED_CLAIM0X (13.54)
                1 Table access full - Outer join feed SYSIBM.GENROW (0.00)
```

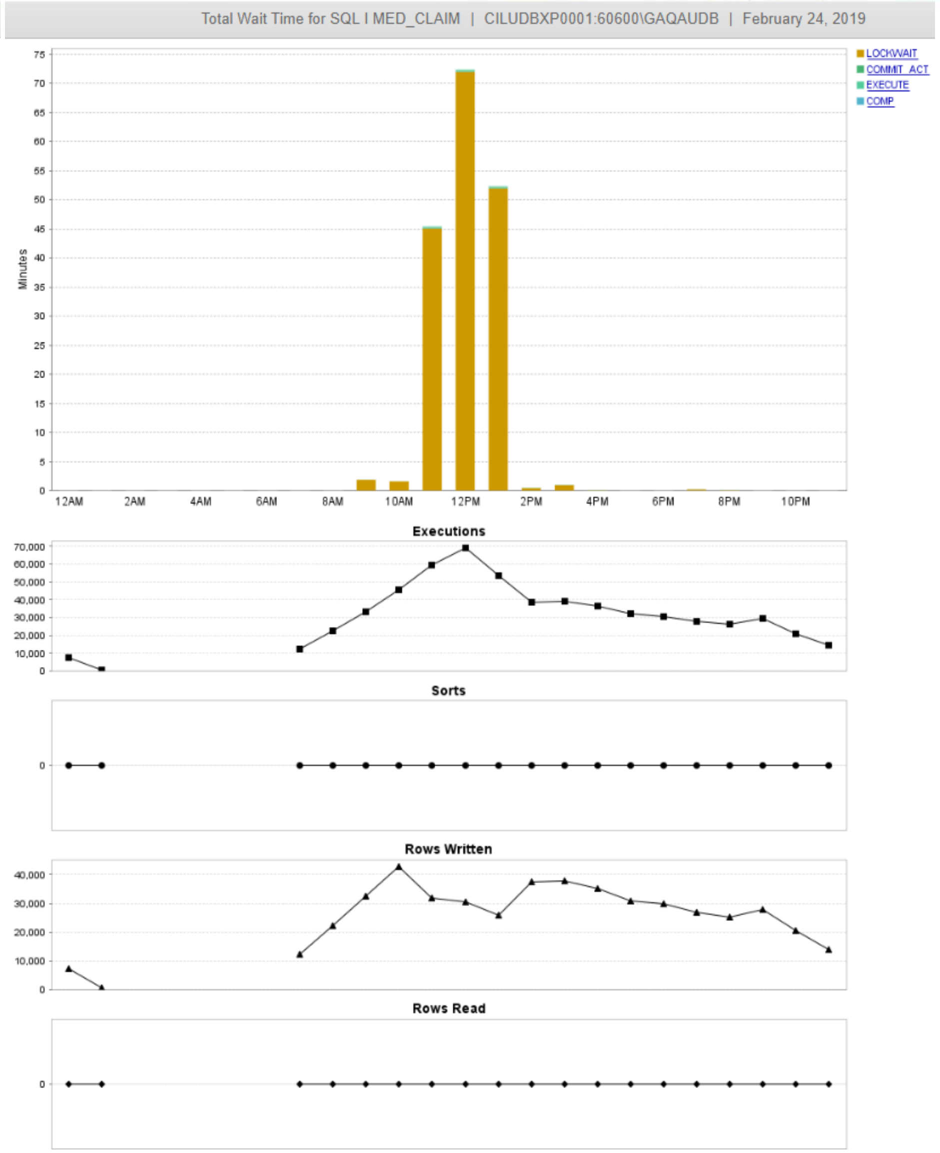
The index is used to access the data

Object name	Type	Rows	Pages	Statistics time	File pages	Pct free	RI parents	RI children	RI self	Key unique	Key columns	Col count	Check cour
GAQA.MED_CLAIM	Table	161582	11549	2019-02-24 13:48:03	11549	10	0	3	0	0	1	26	0
GAQA.MED_CLAIM_DETAIL	Table	438793	29269	2019-02-24 13:48:56	29269	-1	1	0	0	0	0	25	0

Index name	Type	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	Internal ID	User defined	System req	Se
GAQA.MED_CLAIM0X	Clustering	Unique	1796	3	161582	11208	-1	4	2019-02-24 13:48:03	1	Yes	0	17
GAQA.MED_CLAIM1X	Regular	Primary Index	1171	3	161582	161582	-1	1	2019-02-24 13:48:03	2	Yes	1	11

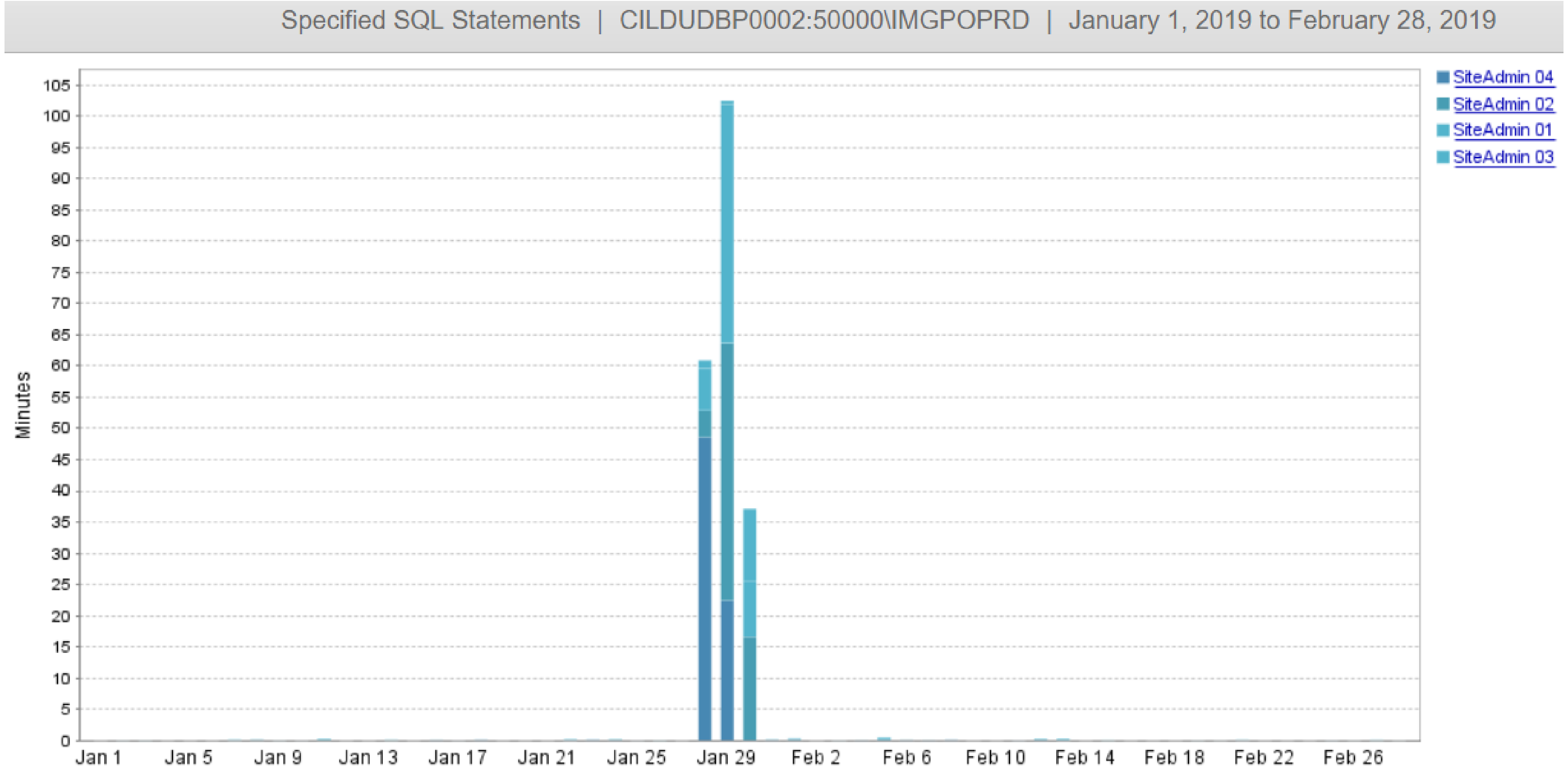
Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Part key s
MBR_NUM	CHARACTER (10)	No	10	Ascending	N/A	11208	'943767392 '	'001442121 '	0	0
MBR_SFX_RELCD	CHARACTER (2)	No	2	Ascending	N/A	17	'EE'	'02'	1	0
MBR_SORT_ORDER	SMALLINT (2)	No	2	Ascending	N/A	12	11	2	2	0
CLM_NUM	CHARACTER (15)	No	15	Ascending	1	161582	'9861905290250 '	'0191813187001 '	3	0
CLM_SRC_SYS_CD	CHARACTER (1)	No	1	N/A	N/A	3	'R'	'J'	4	0
CLM_STAT_DESC	CHARACTER (20)	No	20	N/A	N/A	6	'PENDING '	'DUPLICATE	5	0
CLM_PROCSD_DT	DATE (4)	No	4	N/A	N/A	728	'2019-02-24'	'2017-02-28'	6	0
CLM_CHRG_AMT	CHARACTER (13)	No	13	N/A	N/A	24832	'001237831.22 '	'000000000.01 '	7	0
CLM_PD_AMT	CHARACTER (13)	No	13	N/A	N/A	36352	'000753231.49 '	'000000000.01 '	8	0
OTHER_INS_CONSIDERED	CHARACTER (13)	No	13	N/A	N/A	1	'000000000.00 '	'000000000.00 '	9	0
OTHER_INS_PAID	CHARACTER (13)	No	13	N/A	N/A	3552	'000047292.18 '	'000000000.19 '	10	0
HSA_FSA_IND	CHARACTER (1)	No	1	N/A	N/A	2	'D'	' '	11	0
CCN_IND	CHARACTER (1)	No	1	N/A	N/A	2	'N'	' '	12	0
HSA_PAYMENT_AMT	CHARACTER (13)	No	13	N/A	N/A	1	'000000000.00 '	'000000000.00 '	13	0
MBR_LIABILITY_AMT	CHARACTER (13)	No	13	N/A	N/A	27136	'000147726.45 '	'000000000.01 '	14	0
PAID_FROM_HRA	DECIMAL (11,2)	No	6	N/A	N/A	3616	+000002395.95	+000000000.02	15	0
PAID_FROM_FSA	DECIMAL (11,2)	No	6	N/A	N/A	416	+000001897.03	+000000000.01	16	0
PAID_FROM_HAA	DECIMAL (11,2)	No	6	N/A	N/A	99	+000000300.00	+000000000.30	17	0
REM_MBR_RESP	DECIMAL (11,2)	No	6	N/A	N/A	20992	+000147726.45	+000000000.00	18	0
HRA_TYPE_CD	VARCHAR (30)	No	4	N/A	N/A	5	'GMPVRX'	'GM'	19	0
MERPS_ACCT_NUM	CHARACTER (7)	No	8	N/A	N/A	4	'3323396'	'3208116'	20	0
OTHER_INS_IND	CHARACTER (1)	No	2	N/A	N/A	3	"	'N'	21	0

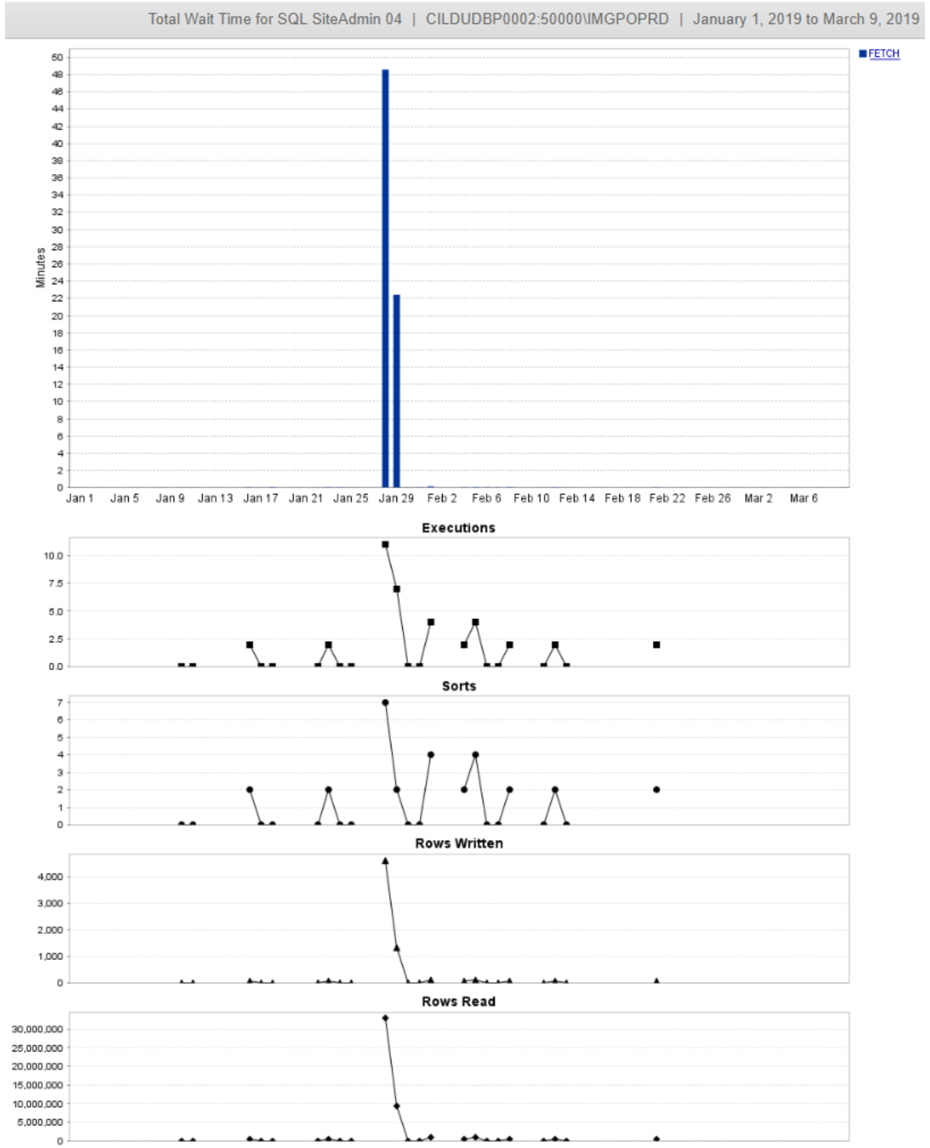

Total Wait Time for SQL S MED_CLAIM MED_... | CILUDBXP0001:60600\GAQAUDB | February 24, 2019
FETCH
LOCKWAIT
COMMIT_ACT
COMP
ROLLBACK_ACT
OPEN
Hours
12AM
2AM
4AM
6AM
8AM
10AM
12PM
2PM
4PM
6PM
8PM
10PM
Executions
Sorts
Rows Written
Rows Read

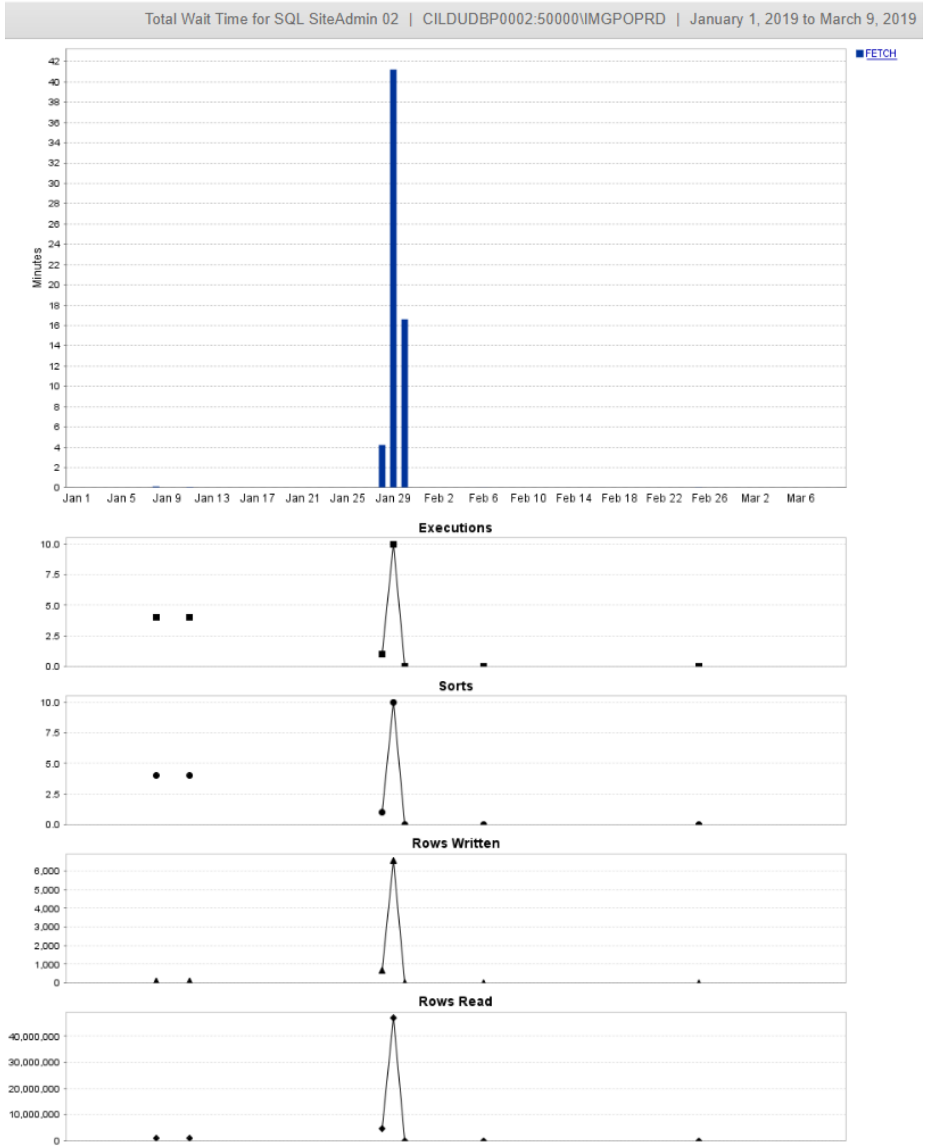

Total Wait Time for SQL I MED_CLAIM | CILUDBXP0001:60600\GAQAUDB | February 24, 2019
LOCKWAIT
COMMIT_ACT
EXECUTE
COMP
Minutes
12AM
2AM
4AM
6AM
8AM
10AM
12PM
2PM
4PM
6PM
8PM
10PM
Executions
Sorts
Rows Written
Rows Read

When a new index WASN'T needed #3 (8|10)

- Here's another issue that started January 28th. Further research and testing showed the access path changed after runstats executed. No other changes were made.
- The SQL calls affected are 7 table joins, unioned together 3 times.
- Although all the tables involved are small (under 100K rows) we found sampling was used.
- Using full runstats resulted in a better access path improving performance as of January 30th PM.
- There have been no access path changes since; the access paths have remained stable since removing "SAMPLED" option.


Specified SQL Statements | CILDUDBP0002:50000\IMGPOPRD | January 1, 2019 to February 28, 2019
Minutes
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
Jan 1
Jan 5
Jan 9
Jan 13
Jan 17
Jan 21
Jan 25
Jan 29
Feb 2
Feb 6
Feb 10
Feb 14
Feb 18
Feb 22
Feb 26
SiteAdmin 04
SiteAdmin 02
SiteAdmin 01
SiteAdmin 03


Total Wait Time for SQL SiteAdmin 04 | CILDUDBP0002:50000\IMGPOPRD | January 1, 2019 to March 9, 2019
FETCH
Minutes
Jan 1
Jan 5
Jan 9
Jan 13
Jan 17
Jan 21
Jan 25
Jan 29
Feb 2
Feb 6
Feb 10
Feb 14
Feb 18
Feb 22
Feb 26
Mar 2
Mar 6
Executions
Sorts
Rows Written
Rows Read


Total Wait Time for SQL SiteAdmin 02 | CILDUDBP0002:50000\IMGPOPRD | January 1, 2019 to March 9, 2019
FETCH
Minutes
Jan 1
Jan 5
Jan 9
Jan 13
Jan 17
Jan 21
Jan 25
Jan 29
Feb 2
Feb 6
Feb 10
Feb 14
Feb 18
Feb 22
Feb 26
Mar 2
Mar 6
Executions
Sorts
Rows Written
Rows Read

Topics

- HW, SW and Currency
- DB Configuration
- Maintenance
- **Data Archive and Purge**
- SQL
- Indexing
- Flash Storage (SSD)
- Implementation Details
- Q & A and More Information

Data Archive and Purge

- Approach is very dependent on the relative amount of data to be archived and purged
- If a large portion of the data is being archived and purged consider:
  - Unloading the data to be archived (if necessary)
  - Unloading the data to be retained
  - Optionally drop unnecessary indexes
  - Sorting the data to be retained in data clustering sequence
  - Loading the data to be retained
  - Optionally recreate indexes
  - Runstat
- This method is extremely efficient if a large percentage of the data is being purged; e.g. no reorgs needed
- If there are HW updates, change of hosting, etc., can minimize data conversion and transport
- Useful if there is no scheduled, automated purge, and periodically (e.g. annually) a manual purge is performed

Data Archive and Purge

- If a small portion of the data is being archived and purged consider:
  - Using “traditional” methods of selecting the data to be archived (if necessary)
  - Optionally drop unnecessary indexes
  - Delete the data to be purged
  - Optionally reorg
  - Optionally recreate indexes
  - Runstat
- This method is preferred if a small percentage of the data is being purged
- Reorgs may be needed
- Useful for a scheduled, automated purge (e.g. weekly) that can be combined with periodic reorgs (e.g. monthly or quarterly)

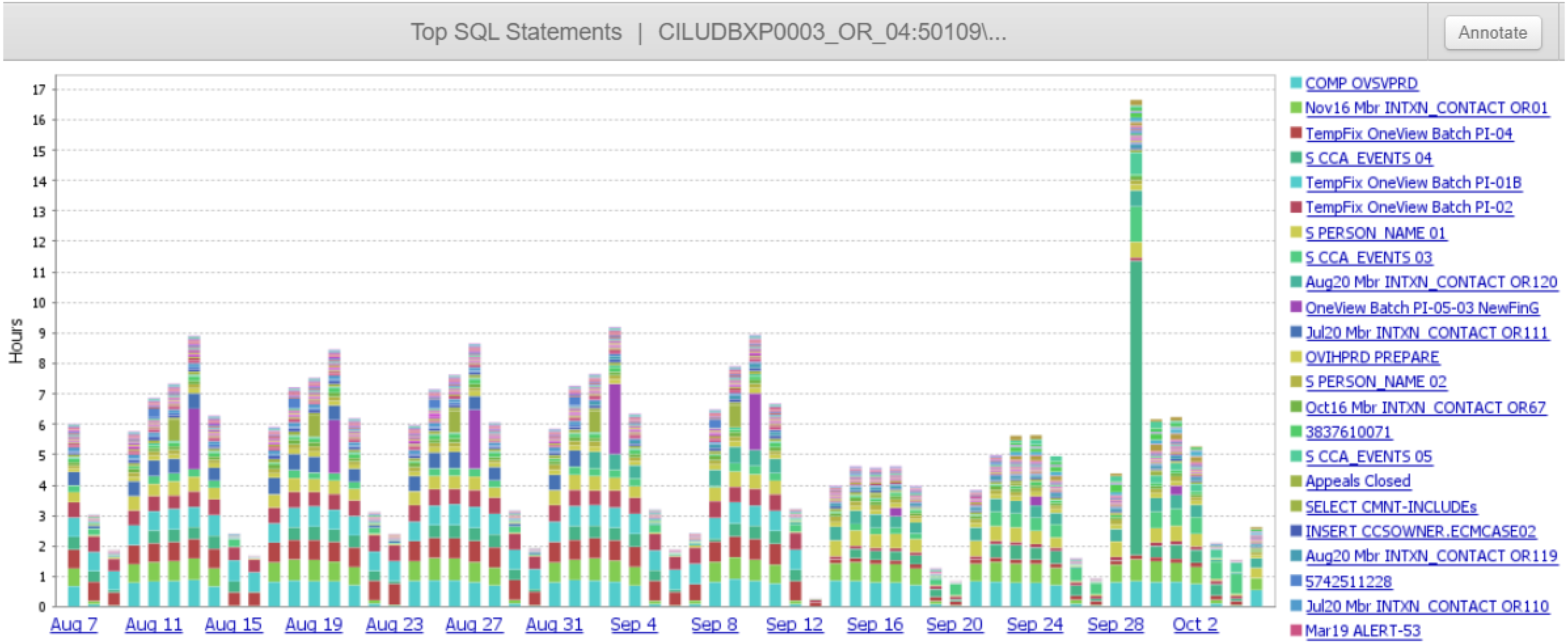
Data Archive and Purge

- In a partitioned environment, partition rotation can be used
- Be aware of any application- or Db2-maintained RI
- RI drives the sequence of tables processed
- Either way, there should be an archive and purge strategy in place to keep size and space in check
- Impacts sequential processing such as some batch, utilities (backup, reorg, runstats)
- Should have minimal impact on OLTP
- Monitor levels on indexes; an increase in levels can indicate an extra physical read
- Keeping objects purged to minimum necessary can keep levels in check, for OLTP
- Keeping objects purged to minimum necessary can keep batch and utilities consistent
- There are tools available, e.g. IBM InfoSphere Optim Archive, to facilitate archive and purge
- And they take into consideration RI
- If you haven’t purged in a “long time” purging can result in huge improvements

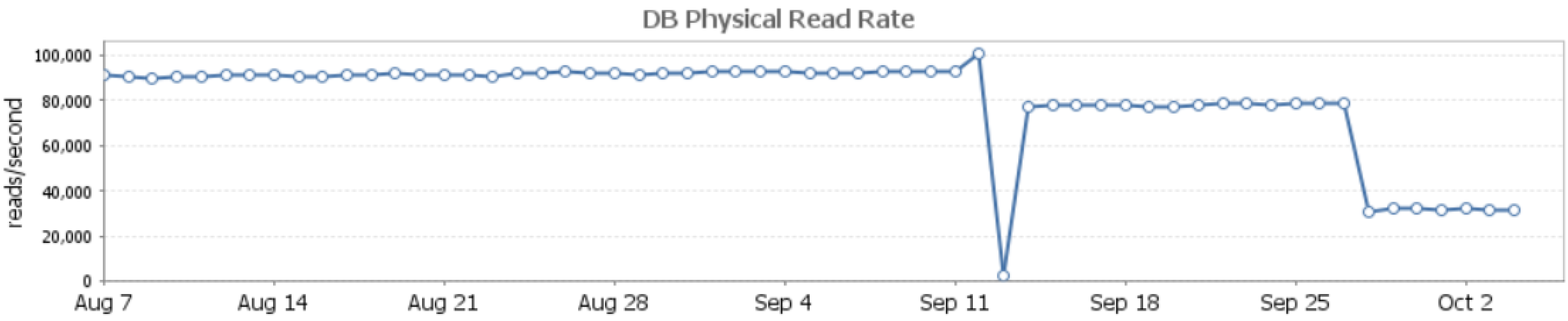
Data Archive and Purge

- Every application, every database must have documented data retention requirements
- From that, archive and purge rules can be defined
- Not all data in an application has to have the same retention; e.g. appeals data is needed longer
- Archive only the data necessary; likely a subset of tables, subset of rows (e.g. appeals data), subset of columns
- The smaller the data needed for retention, the smaller the archive tables or files can be
- Archive and purge on some regular basis:
  - Nightly / weekly during “quiet” times to minimize processing impact and take advantage of available resources
  - Monthly if less-frequent cycle suffices since application volume is low
  - Continuous for some very active applications; watch locking and commit frequency / UOW sizing
- Use your own or use a product; either way just use something to purge unnecessary data
- Be sure to reorg and runstat as required afterward
- Impacts processing that scans; sequential batch processing and maintenance (e.g. backups)
- A recent example

Data Archive and Purge



Data Archive and Purge



- Before purge: 8.052 Billion physical reads/day
- After purge: 6.735 Billion physical reads/day (data copy tables present / image copied)
- After purge: 2.765 Billion physical reads/day (data copy tables dropped)
- Net: 65.7% reduction in physical reads/day

Topics

- HW, SW and Currency
- DB Configuration
- Maintenance
- Data Archive and Purge
- **SQL**
- Indexing
- Flash Storage (SSD)
- Implementation Details
- Q & A and More Information

SQL

- I've presented previously at several CCDUG and IDUG conferences; in those presentations you can find much on SQL tuning
- We'll review some items I've come across multiple times
  - Don't use SELECT * …
    - ✓ Hear this over and over
    - ✓ Really can make a huge difference, especially when physical sorts are involved
    - ✓ And can make all the difference for index-only access considerations
  - Proper placement of parentheses
    - ✓ Include parentheses where needed (required), and for clarification
    - ✓ Be specific and careful about placement
    - ✓ Avoid unnecessary parentheses
    - ✓ Incorrect placement can prevent proper index usage leading to performance issues
    - ✓ … And invalidate the actual query results returned
  - Unnecessary sorting (ORDER BY, GROUP BY, DISTINCT)
  - SQL simplification and function removal; impacting index usage
    - ✓ Know your data
    - ✓ Don't use functions unnecessarily

SQL

- Formatting of SQL calls and proper placement of parentheses are key
- Here’s a portion of a SQL call from one of our FileNet databases:

```
SELECT ...
  FROM PROVIDE2.DocVersion T0
 WHERE (T0.home_id IS NULL
   and T0.recovery_item_id IS NULL
   AND ( ( ( u4c98_producercode = ?
   OR ( u4c98_producercode = ?
   AND u53a3_statementdate <= ?
   AND u53a3_statementdate >= ?
   AND u2922_published = ?
   AND uab72_viewable = ?))
   AND object_class_id=?
   AND is_current = ?)))
 ORDER BY u53a3_statementdate DESC
```

- Received a message of poor performing searches from our DBA; investigated
- I noticed an issue immediately, something with the WHERE criteria different from all other searches

Cigna

SQL

- I added some color and reformatted the SQL snippet to make it clear:

```
SELECT ...
  FROM PROVIDE2.DocVersion T0
 WHERE (T0.home_id IS NULL
   and T0.recovery_item_id IS NULL
   AND (((u4c98_producercode   = ? OR
         (u4c98_producercode   = ? AND u53a3_statementdate <= ? AND
          u53a3_statementdate >= ? AND u2922_published      = ? AND
          uab72_viewable       = ?))
   AND object_class_id=?
   AND is_current = ?)))
 ORDER BY u53a3_statementdate DESC
```

SQL

- What was intended is:

```
SELECT ...
  FROM PROVIDE2.DocVersion T0
 WHERE T0.home_id IS NULL
   AND T0.recovery_item_id IS NULL
   AND (u4c98_producercode = ? OR u4c98_producercode = ?)
   AND u53a3_statementdate <= ?
   AND u53a3_statementdate >= ?
   AND u2922_published = ?
   AND uab72_viewable = ?
   AND object_class_id = ?
   AND is_current = ?
 ORDER BY u53a3_statementdate DESC
```

- Corrected the performance issue, **<u>and more importantly</u> corrected the results returned by the SQL call**
- Using an IN clause is preferred; much more readable and lessens the risk of making a parentheses-related error

```
   AND u4c98_producercode IN (?,?)
```

SQL

- What was intended is:

```
SELECT ...
  FROM PROVIDE2.DocVersion T0
 WHERE T0.home_id IS NULL
   AND T0.recovery_item_id IS NULL
   AND u4c98_producercode IN (?,?)
   AND u53a3_statementdate <= ?
   AND u53a3_statementdate >= ?
   AND u2922_published = ?
   AND uab72_viewable = ?
   AND object_class_id = ?
   AND is_current = ?
 ORDER BY u53a3_statementdate DESC
```

When a new index WASN'T needed #2 (4|10)

- Here's an example of SQL that needs to be tuned first:

```
SELECT ...
  FROM PRV.CHCP_USR_TIN_ACCSS_MGR
 WHERE trim(SSO_ID)=trim(?)
   AND upper(trim(LOB_TY))<>'FIM'
   AND (REGISTERED_TIN is null OR trim(REGISTERED_TIN)='')
```

- which can be rewritten as:

```
SELECT ...
  FROM PRV.CHCP_USR_TIN_ACCSS_MGR
 WHERE SSO_ID=?
   AND LOB_TY<>'FIM'
   AND (REGISTERED_TIN is null OR REGISTERED_TIN='')
```

When a new index WASN'T needed #2 (5|10)

- After verifying data from the table, EXPLAIN estimated a 99.98% reduction with SQL changes alone.
- Db2 chose to use an already-existing index on SSO_ID as one would suspect since it has good cardinality.
- The optimal index for this revised SQL call is an index on CHCP_USR_TIN_ACCSS_MGR (SSO_ID, REGISTERED_TIN, LOB_TY), although we never created it.
- With SQL changes alone these calls were rarely captured afterward and our tooling reflected average run times of 0 to at most 0.76ms.

When a new index WASN'T needed #2 (6|10)

- Before

Data operation complete (171716.30)
6 Fetch table data PRV.CHCP_USR_TIN_ACCSS_MGR (171716.30)
5 Row identifier scan (RID) (10132.60)
4 Data is sorted (5014.16)
3 Index scan (no table access) SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X3 (5013.62)
2 Data is sorted (5118.44)
1 Index scan (no table access) SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X3 (4600.80)

Object name	Type	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Col count	Status	Append mode	Lock size	Volatile
PRV.CHCP_USR_TIN_ACCSS_MGR	Table	2189559	191151	2018-09-09 19:38:29	305413	0	0	0	32	Normal	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	Seq pages	Density	First2 keycard	First3 k
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_PK	Unique	39376	4	2085153	612520	89	4	2018-08-08 19:22:02	28986	42	612537	20025
PRV.CHCP_USR_TIN_ACCSS_MGR_X1	Duplicates	6777	3	329780	329780	13	1	2018-08-08 19:22:02	5111	96	-1	-1
PRV.CHCP_USR_TIN_ACCSS_MGR_X2	Duplicates	21675	4	2002541	612532	72	2	2018-08-08 19:22:02	16661	98	2002541	-1
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X3	Duplicates	4805	3	1	1	98	1	2018-08-08 19:22:02	4804	96	-1	-1
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X4	Duplicates	4805	3	1	1	98	1	2018-08-08 19:22:02	4804	96	-1	-1
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X5	Duplicates	4300	3	1	1	98	2	2018-08-08 19:22:02	4299	99	1	-1

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
REGISTERED_TIN	CHARACTER (1)	No	2	Ascending	N/A	1	"	"	20	Yes	2085153	
CHCP_USR_ID	BIGINT (8)	No	8	N/A	N/A	612520	999592910110717	26679	0	No	0	
SSO_ID	VARCHAR (32)	No	14	N/A	N/A	630315	'zzzsleep'	'0004506n'	1	No	0	"

- After

Data operation complete (28.35)
2 Fetch table data PRV.CHCP_USR_TIN_ACCSS_MGR (28.35)
1 Index scan PRV.CHCP_USR_TIN_ACCSS_MGR_X2 (20.30)

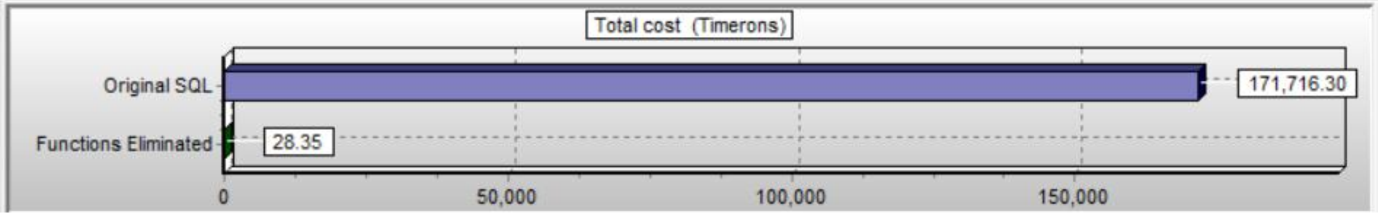
Object name	Type	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Col count	Status	Append mode	Lock size	Volatile
PRV.CHCP_USR_TIN_ACCSS_MGR	Table	2189559	191151	2018-09-09 19:38:29	305413	0	0	0	32	Normal	Rows inserted	Row	

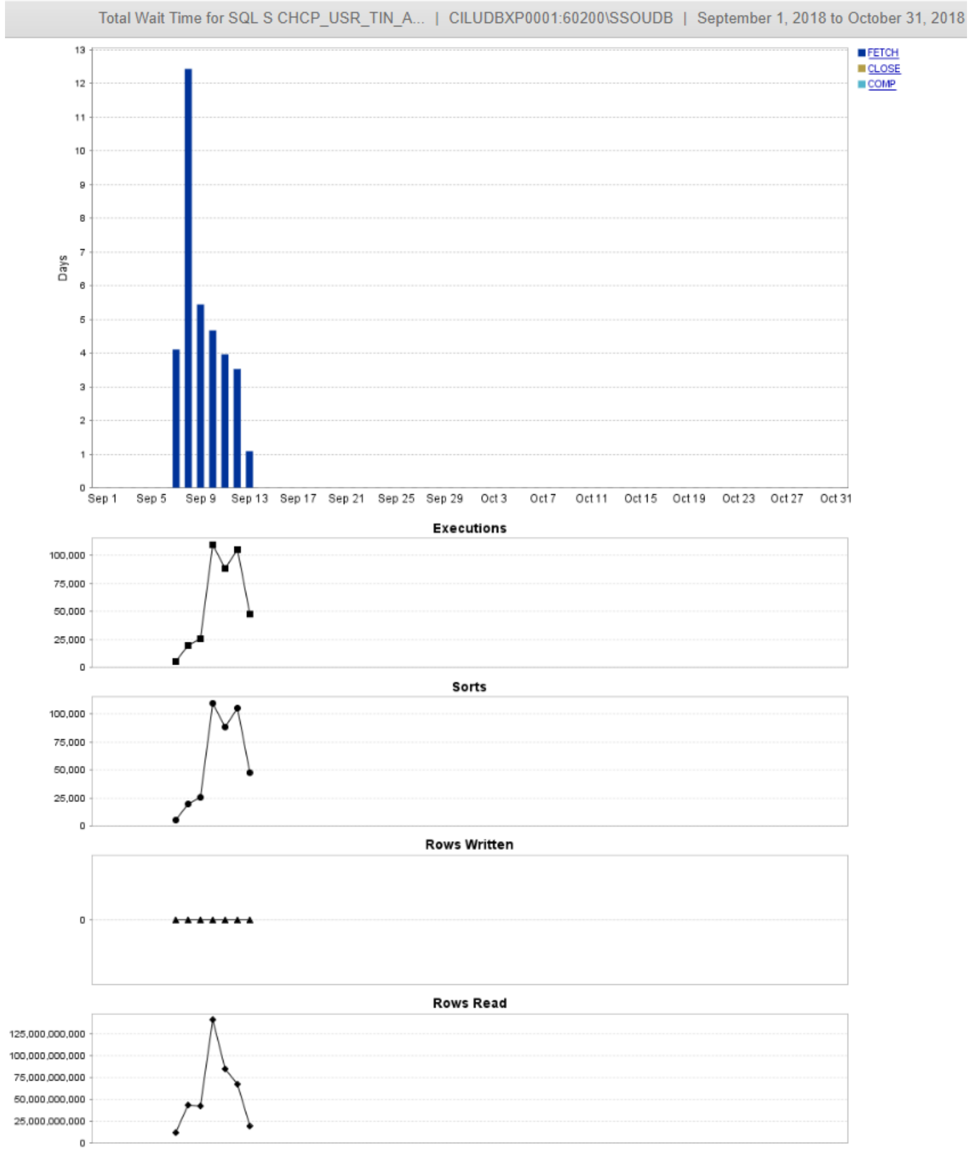
Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	Seq pages	Density	First2 keycard	First3
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_PK	Unique	39376	4	2085153	612520	89	4	2018-08-08 19:22:02	28986	42	612537	2002
PRV.CHCP_USR_TIN_ACCSS_MGR_X1	Duplicates	6777	3	329780	329780	13	1	2018-08-08 19:22:02	5111	96	-1	-1
PRV.CHCP_USR_TIN_ACCSS_MGR_X2	Duplicates	21675	4	2002541	612532	72	2	2018-08-08 19:22:02	16661	98	2002541	-1
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X3	Duplicates	4805	3	1	1	98	1	2018-08-08 19:22:02	4804	96	-1	-1
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X4	Duplicates	4805	3	1	1	98	1	2018-08-08 19:22:02	4804	96	-1	-1
SSOIPRD.CHCP_USR_TIN_ACCSS_MGR_X5	Duplicates	4300	3	1	1	98	2	2018-08-08 19:22:02	4299	99	1	-1

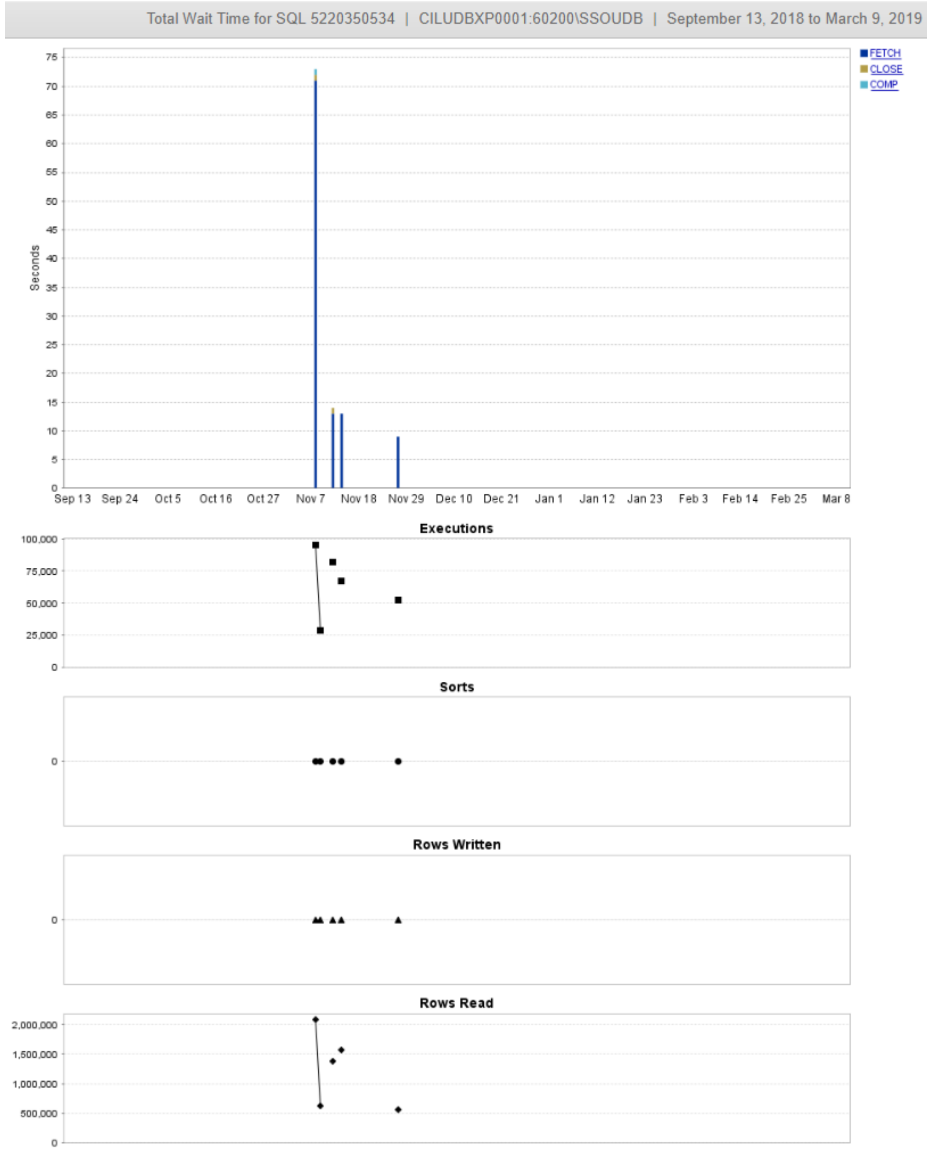
Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
SSO_ID	VARCHAR (32)	No	14	Ascending	N/A	630315	'zzzsleep'	'0004506	1	No	0	"
CHCP_TIN	CHARACTER (9)	No	9	Ascending	N/A	329780	'99999999	'0000000	2	No	0	"
CHCP_USR_ID	BIGINT (8)	No	8	N/A	N/A	612520	99959291	26679	0	No	0	
PROV_ID	CHARACTER (7)	No	8	N/A	N/A	1	"	"	3	Yes	1863610	

- Comparison

Scenario name	Qualifier	Class	Total cost	I/O cost	CPU cost	Elapsed time	Rows	Columns
Original SQL		5	171716.30	192184.88	24841592832.00	N/A	N/A	N/A
Functions Eliminated		5	28.35	4.19	117581.48	N/A	N/A	N/A




Total Wait Time for SQL S CHCP_USR_TIN_A... | CILUDBXP0001:60200\SSOUDB | September 1, 2018 to October 31, 2018
FETCH
CLOSE
COMP
Days
Sep 1 Sep 5 Sep 9 Sep 13 Sep 17 Sep 21 Sep 25 Sep 29 Oct 3 Oct 7 Oct 11 Oct 15 Oct 19 Oct 23 Oct 27 Oct 31
Executions
Sorts
Rows Written
Rows Read


Total Wait Time for SQL 5220350534 | CILUDBXP0001:60200\SSOUDB | September 13, 2018 to March 9, 2019
FETCH
CLOSE
COMP
Seconds
Sep 13 Sep 24 Oct 5 Oct 16 Oct 27 Nov 7 Nov 18 Nov 29 Dec 10 Dec 21 Jan 1 Jan 12 Jan 23 Feb 3 Feb 14 Feb 25 Mar 8
Executions
Sorts
Rows Written
Rows Read

Topics

- HW, SW and Currency
- DB Configuration
- Maintenance
- Data Archive and Purge
- SQL
- **Indexing**
- Flash Storage (SSD)
- Implementation Details
- Q & A and More Information

Indexing

- I’ve presented previously at several CCDUG and IDUG conferences; in those presentations you can find much on Indexing
- We’ll review several items I’ve come across multiple times
  - Unused indexes
  - Single-column indexes, no multi-column indexes
  - Multi-index access versus multi-column index access
  - Physical sorting
  - Referential integrity
  - Index-only access and INCLUDE columns
  - “One fetch” access
  - Expression-based indexes
  - EXCLUDE NULL KEYS

Indexing

- Identify and DROP unused indexes
  - Whenever I'm analyzing SQL and recommending either a new index or an index change, I run a quick check against PRD to see if all the indexes are used, and how recently:

```
SELECT tbcreator, tbname AS TABLE_NAME, creator, name AS INDEX_NAME, lastused AS LAST_USED,
       firstkeycard AS FIRST_KEY_CARD, fullkeycard AS FULL_KEY_CARD, nleaf AS LEAF_PAGES,
       nlevels AS LVLS, numrids AS ROWS, colnames AS COLUMN_NAMES
  FROM sysibm.sysindexes
--HERE tbcreator <> 'SYSIBM' -- Bypass UDB Catalog Tables
 WHERE tbcreator =  'your_table_creator'
   AND tbname   IN ('your_table_name1', 'your_table_name2')
 ORDER BY 1, 2, lastused DESC, firstkeycard DESC
```

  - Why? Why not? Might as well
  - Often I find never-used indexes, not-recently-used indexes, redundant indexes, etc.
  - Eliminates overhead of maintaining indexes
  - For redundant, run query above with `ORDER BY 1, 2, colnames`
  - In earlier versions of Db2, there were issues getting proper lastused values updated in the sysindexes table; that was long ago and is no longer an issue
  - Of course always test thoroughly and be aware of indexes used solely for monthly, quarterly or yearly processing

Indexing

- Single-column indexes, no multi-column indexes
  - I’ve seen various product databases come with many single-column indexes, and no or very few multi-column indexes
  - Forces the Db2 optimizer to choose either a single index or multiple index access, likely impacting performance
  - In some cases, single-column indexes are perfectly fine and all that is needed
  - An example

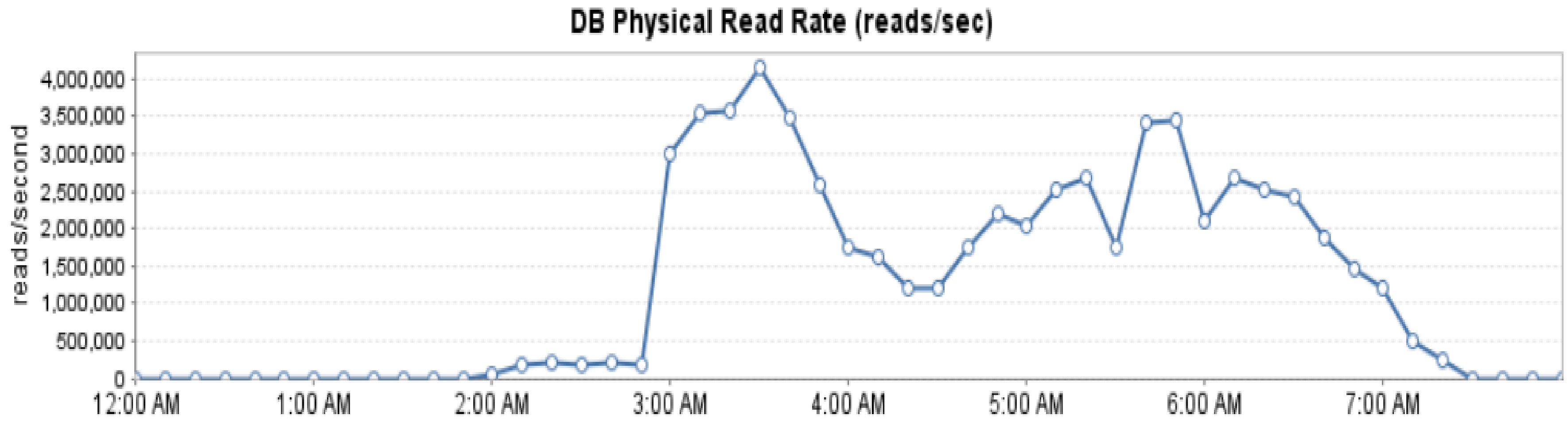
Typical index usage – single column #3 (7|9)

- Application reportedly causing FA utilization alerts; HW team about to add resources.
- Found various SQL calls without parameter markers running and performing excessive asynchronous IO leading to the alerts.

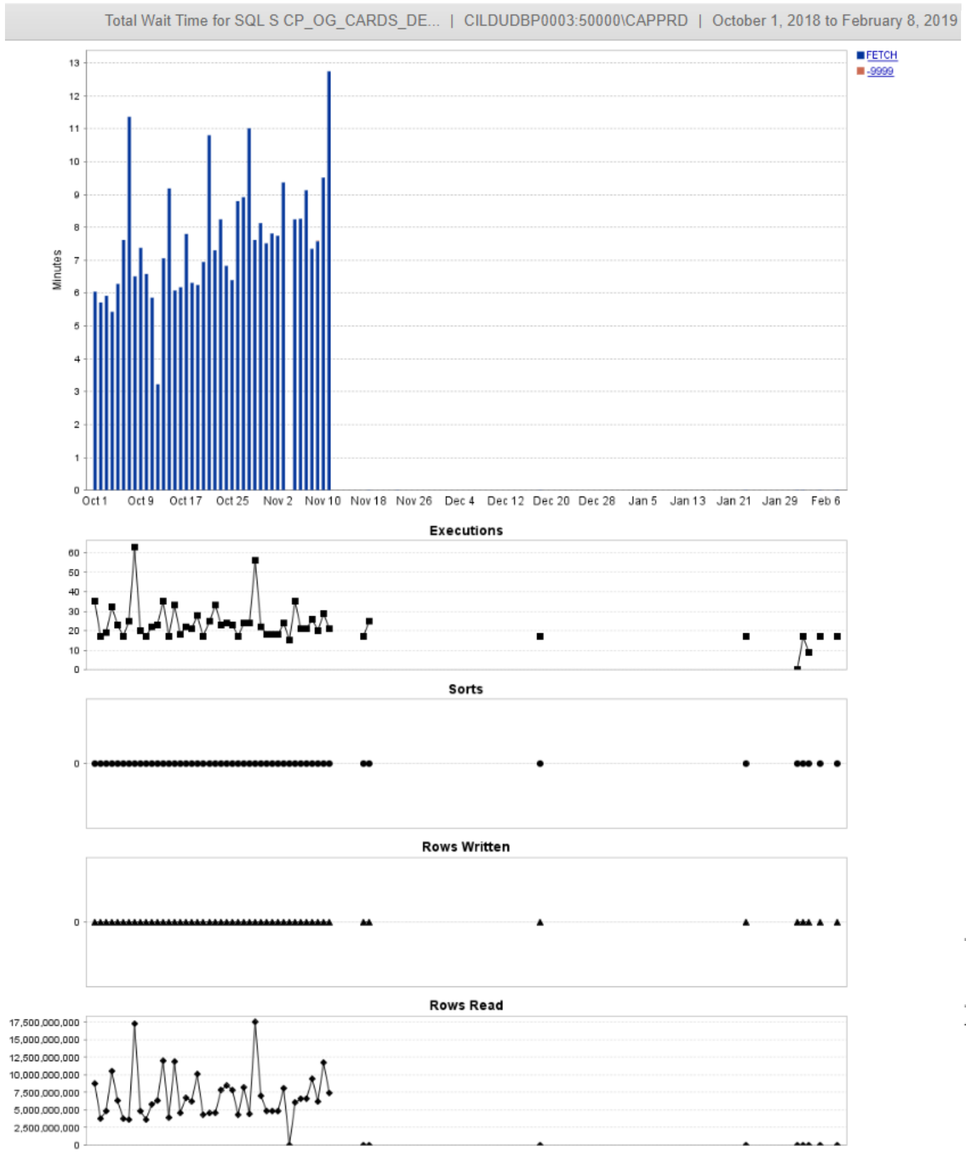
```
DELETE FROM APPDM.CP_OG_CARDS_DETAIL
  WHERE request_id = ######

select count(*) as delete_row_cnt
  from appdm.CP_OG_CARDS_DETAIL
  where request_id=#######
```

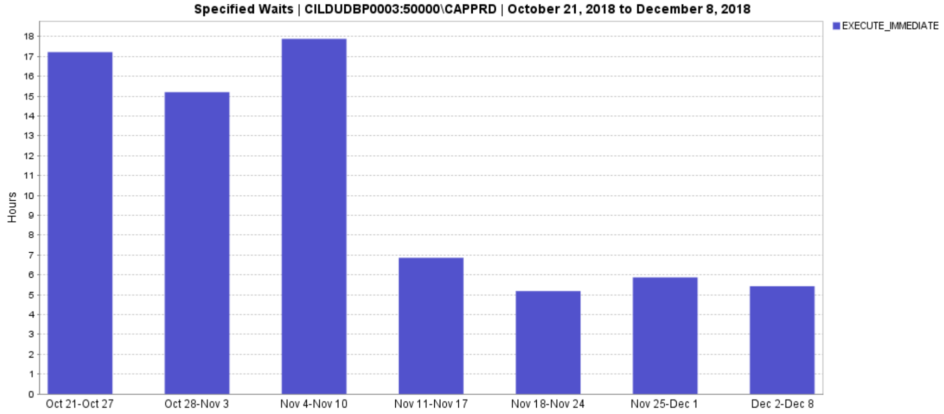
- No indexes on the table; only access path choice was full table scan.
- Needed a new index on CP_OG_CARDS_DETAIL(REQUEST_ID).

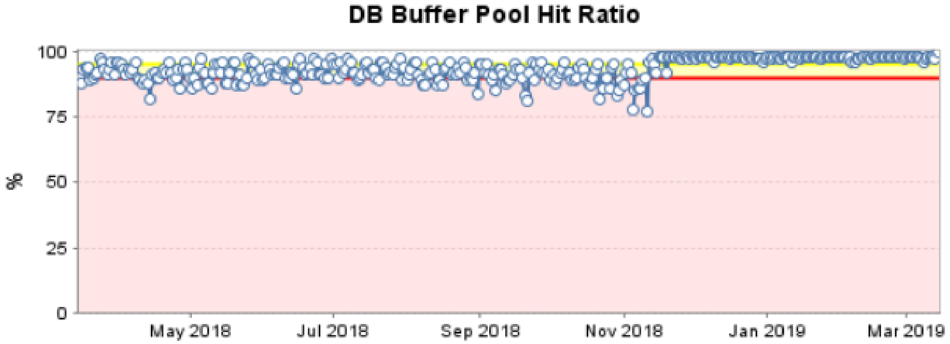
Typical index usage – single column #3 (8|9)

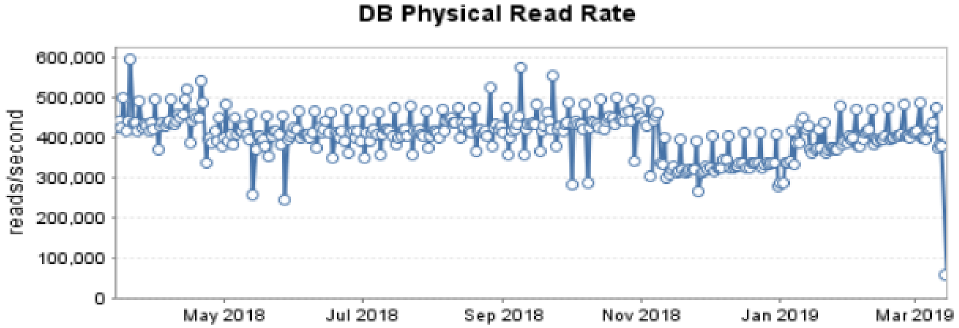


- Savings totaled over 3¼ hours per day and eliminated 198 Billion logical reads per day.
- Experienced a 30% drop in physical reads from 460K to 320K reads/sec averaged throughout the 24-hour day, saving more than 12 Billion physical reads per day.
- Eliminated storage FA utilization impacts affecting other unrelated application processing in the environment.
- 96% of the remaining physical reads were from daily Db2 backups.


Total Wait Time for SQL S CP_OG_CARDS_DE... | CILDUDBP0003:50000\CAPPRD | October 1, 2018 to February 8, 2019
FETCH
-9999
Minutes
Oct 1
Oct 9
Oct 17
Oct 25
Nov 2
Nov 10
Nov 18
Nov 26
Dec 4
Dec 12
Dec 20
Dec 28
Jan 5
Jan 13
Jan 21
Jan 29
Feb 6
Executions
Sorts
Rows Written
Rows Read


Specified Waits | CILDUDBP0003:50000\CAPPRD | October 21, 2018 to December 8, 2018
EXECUTE_IMMEDIATE
Hours
Oct 21-Oct 27
Oct 28-Nov 3
Nov 4-Nov 10
Nov 11-Nov 17
Nov 18-Nov 24
Nov 25-Dec 1
Dec 2-Dec 8


DB Buffer Pool Hit Ratio
%
May 2018
Jul 2018
Sep 2018
Nov 2018
Jan 2019
Mar 2019


DB Physical Read Rate
reads/second
May 2018
Jul 2018
Sep 2018
Nov 2018
Jan 2019
Mar 2019

Typical index usage – multi-column #1 (1|7)

- With more complicated SQL calls, more complicated indexing is required.
  - Think like the optimizer – what's the best index?
  - The smallest index that gives the best filtering
  - When using multiple columns, weigh the length of the column (actual, in bytes) versus the screening or filtering it provides
  - Don't include columns that provide no benefit; e.g. a column with one value (cardinality=1) provides no filtering (although there is at least one exception to this)
  - Don't include long columns providing little benefit; e.g. a low cardinality column provides little filtering
  - What's indexable; = versus <> versus LIKE, ANDs/ORs precedence, INs over NOT INs, etc.

Multi-index usage versus multi-column index (1|4)

- Generally, multi-index access indicates a potential tuning opportunity. The more indexes involved (2, 3 etc.), the greater the opportunity.
- Essentially the optimizer has decided to use two different indexes and (usually) AND the results together and then fetch the qualifying data rows.
- In these cases, a better index choice would be a multi-column index having the necessary columns to optimize access for the query using just one index.
- However there are exceptions to this. If for instance you do need the two indexes for other processing, and it's not worth the overhead of a third index (overlapping columns with the other two indexes), and performance is "sufficient" considering processing volume, then a third index isn't warranted.

Multi-index usage versus multi-column index (2|4)

- In the following example, originally two different indexes were used to resolve the criteria in the WHERE clause, including U7C98_CIGNA_ACCT_NR from the 4X index and U04A6_CIGNA_CLIENT_ID from the 2X index.
- Unfortunately this results in a less-than-optimal multi-index access path with additional processing, including additional physical sorts.
- Since the 2X and 4X indexes were created specifically for other search calls, we didn't want to change them in such a way affecting other SQL.
- We added U04A6_CIGNA_CLIENT_ID to the existing 4X index and on the next two slides are the original and improved access paths.

```
Data operation complete (60.52)
10 Sorted table data scan (60.52)
  9 Data is sorted (60.51)
    8 Nested loop join (60.51)
      7 Fetch table data - Inner join feed CLIENTOS.DOCVERSION (30.27)
        6 Row identifier scan (RID) (30.27)
          5 Data is sorted (30.27)
            4 Index ANDing (30.27)
              3 Index scan (no table access) CLIENTOS.CI_DOCVERSION_02X (15.14)
              2 Index scan (no table access) CLIENTOS.CI_DOCVERSION_04X (15.13)
      1 Table access full - Outer join feed SYSIBM.GENROW (0.00)
```

Object name	Type	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Col count	Status	Append mode	Lock size	Volatile
CLIENTOS.DOCVERSION	Table	2568933	91317	2019-01-23 10:02:30	91317	0	0	0	133	Normal	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Col count	Statistics time	Seq pages	Density	First2 keycard	First3 keycard	First
SYSIBM.SQL150302115638840	Primary Index	2790	3	2568933	2568933	1	2019-01-23 10:02:30	2789	80	-1	-1	-1
CLIENTOS.I_DOCVERSION22	Duplicates	3381	3	2568933	2568933	3	2019-01-23 10:02:30	3380	84	2568933	2568933	-1
CLIENTOS.I_DOCVERSION73	Duplicates	714	2	1	1	1	2019-01-23 10:02:30	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_05X	Unique	5815	3	2568933	1143	5	2019-01-23 10:02:30	5814	88	1145	1147	241
CLIENTOS.UI_UC7B7_OWNERDOCU	Duplicates	714	2	1	1	1	2019-01-23 10:02:30	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_01X	Duplicates	8161	3	2437018	2437018	1	2019-01-23 10:02:30	8160	89	-1	-1	-1
CLIENTOS.CI_DOCVERSION_02X	Duplicates	3566	3	2553343	100940	4	2019-01-23 10:02:30	3565	79	117520	146081	255
CLIENTOS.CI_DOCVERSION_03X	Duplicates	682	2	13533	13498	2	2019-01-23 10:02:30	681	47	13533	-1	-1
CLIENTOS.CI_DOCVERSION_04X	Duplicates	4588	3	2553832	103617	4	2019-01-23 10:02:30	4587	83	156812	254446	255
CLIENTOS.CI_DOCVERSION_06X	Unique	5217	3	2568933	1003	5	2019-01-23 10:02:30	5216	86	1116	1118	181
CLIENTOS.CI_DOCVERSION_07X	Duplicates	4710	3	2412358	1003	6	2019-01-23 10:02:30	4709	85	1810	1906	190
CLIENTOS.CI_DOCVERSION_08X	Duplicates	714	2	168	168	1	2019-01-23 10:02:30	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_09X	Unique	5618	3	2568933	1	5	2019-01-23 10:02:30	5617	99	5	185922	256

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
U7C98_CIGNA_ACCT_NR	VARGRAPHIC (7)	No	18	Ascending	N/A	103617	g'3902341'	g'3320953'	119	Yes	185921	
U66D4_CWR_DOCUMENT_ID	DOUBLE (8)	No	9	Ascending	N/A	31	+2.07000000(	+2.80000000(	109	Yes	185754	
UEF86_FREQUENCY	INTEGER (4)	No	5	Ascending	N/A	4	3	2	116	Yes	185921	
CREATE_DATE	TIMESTAMP (10)	No	11	Ascending	N/A	2146922	'2019-01-18-1·	'2016-03-15-1·	6	Yes	0	
OBJECT_ID	VARCHAR (16)	No	20	N/A	1	2568933	'□□~□H□□□:U□.'	'	0	No	0	
OBJECT_CLASS_ID	VARCHAR (16)	No	20	N/A	N/A	5	'□$□mY□K□? {c	'c_J0	1	No	0	

```
Data operation complete (60.52)
10 Sorted table data scan (60.52)
  9 Data is sorted (60.51)
    8 Nested loop join (60.51)
      7 Fetch table data - Inner join feed CLIENTOS.DOCVERSION (30.27)
        6 Row identifier scan (RID) (30.27)
          5 Data is sorted (30.27)
            4 Index ANDing (30.27)
              3 Index scan (no table access) CLIENTOS.CI_DOCVERSION_02X (15.14)
              2 Index scan (no table access) CLIENTOS.CI_DOCVERSION_04X (15.13)
      1 Table access full - Outer join feed SYSIBM.GENROW (0.00)
```

Object name	Type	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Col count	Status	Append mode	Lock size	Volatile
CLIENTOS.DOCVERSION	Table	2568933	91317	2019-01-23 10:02:30	91317	0	0	0	133	Normal	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Col count	Statistics time	Seq pages	Density	First2 keycard	First3 keycard	First
SYSIBM.SQL150302115638840	Primary Index	2790	3	2568933	2568933	1	2019-01-23 10:02:30	2789	80	-1	-1	-1
CLIENTOS.I_DOCVERSION22	Duplicates	3381	3	2568933	2568933	3	2019-01-23 10:02:30	3380	84	2568933	2568933	-1
CLIENTOS.I_DOCVERSION73	Duplicates	714	2	1	1	1	2019-01-23 10:02:30	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_05X	Unique	5815	3	2568933	1143	5	2019-01-23 10:02:30	5814	88	1145	1147	241
CLIENTOS.UI_UC7B7_OWNERDOCU	Duplicates	714	2	1	1	1	2019-01-23 10:02:30	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_01X	Duplicates	8161	3	2437018	2437018	1	2019-01-23 10:02:30	8160	89	-1	-1	-1
CLIENTOS.CI_DOCVERSION_02X	Duplicates	3566	3	2553343	100940	4	2019-01-23 10:02:30	3565	79	117520	146081	255
CLIENTOS.CI_DOCVERSION_03X	Duplicates	682	2	13533	13498	2	2019-01-23 10:02:30	681	47	13533	-1	-1
CLIENTOS.CI_DOCVERSION_04X	Duplicates	4588	3	2553832	103617	4	2019-01-23 10:02:30	4587	83	156812	254446	255
CLIENTOS.CI_DOCVERSION_06X	Unique	5217	3	2568933	1003	5	2019-01-23 10:02:30	5216	86	1116	1118	181
CLIENTOS.CI_DOCVERSION_07X	Duplicates	4710	3	2412358	1003	6	2019-01-23 10:02:30	4709	85	1810	1906	190
CLIENTOS.CI_DOCVERSION_08X	Duplicates	714	2	168	168	1	2019-01-23 10:02:30	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_09X	Unique	5618	3	2568933	1	5	2019-01-23 10:02:30	5617	99	5	185922	256

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
U04A6_CIGNA_CLIENT_ID	INTEGER (4)	No	5	Ascending	N/A	100940	103857	1002	120	Yes	185921	
U66D4_CWR_DOCUMENT_ID	DOUBLE (8)	No	9	Ascending	N/A	31	+2.07000000(	+2.80000000(	109	Yes	185754	
UEF86_FREQUENCY	INTEGER (4)	No	5	Ascending	N/A	4	3	2	116	Yes	185921	
CREATE_DATE	TIMESTAMP (10)	No	11	Ascending	N/A	2146922	'2019-01-18-1·	'2016-03-15-1·	6	Yes	0	
OBJECT_ID	VARCHAR (16)	No	20	N/A	1	2568933	'□□~□H□□□:U□.'	'	0	No	0	
OBJECT_CLASS_ID	VARCHAR (16)	No	20	N/A	N/A	5	'□$□mY□K□? {c	'c_J0	1	No	0	

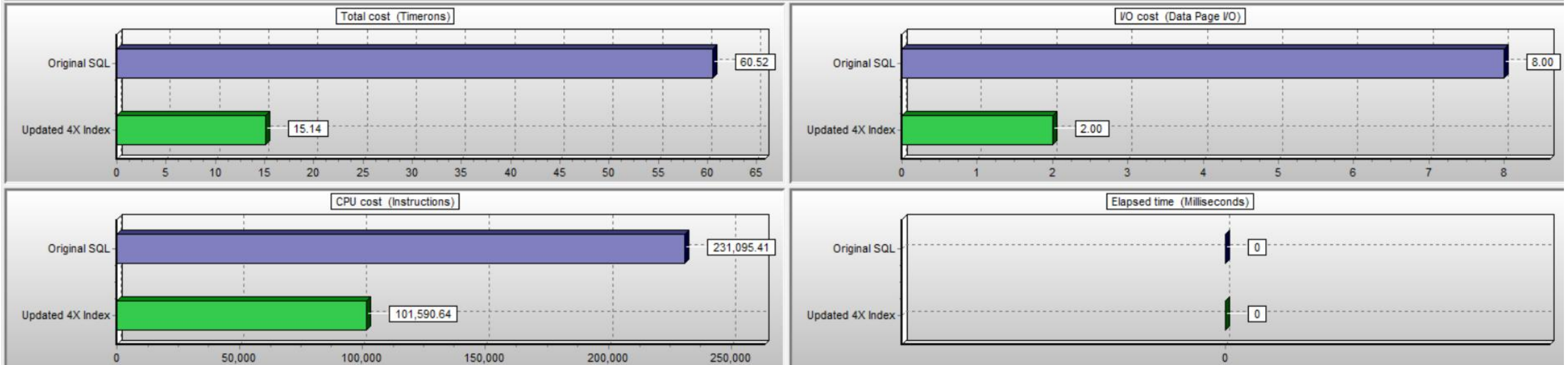
Data operation complete (15.14)
2 Fetch table data CLIENTOS.DOCVERSION (15.14)
1 Index scan CLIENTOS.CI_DOCVERSION_04X (15.13)

Object name	Type	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Col count	Status	Append mode	Lock size	Volatile
CLIENTOS.DOCVERSION	Table	2658933	94421	2019-03-25 13:31:32	94421	0	0	0	133	Normal	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Col count	Statistics time	Seq pages	Density	First2 keycard	First3 keycard	First
SYSIBM.SQL150302115638840	Primary Index	2967	3	2658933	2658933	1	2019-03-25 13:31:32	2789	80	-1	-1	-1
CLIENTOS.I_DOCVERSION22	Duplicates	3594	3	2658933	2658933	3	2019-03-25 13:31:32	3380	84	2658933	2658933	-1
CLIENTOS.I_DOCVERSION73	Duplicates	739	2	1	1	1	2019-03-25 13:31:32	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_05X	Unique	6180	3	2658933	1143	5	2019-03-25 13:31:32	5814	88	1145	1147	249
CLIENTOS.UI_UC7B7_OWNERDOCU	Duplicates	739	2	1	1	1	2019-03-25 13:31:32	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_01X	Duplicates	8161	3	2527018	2527018	1	2019-03-25 13:31:32	8160	89	-1	-1	-1
CLIENTOS.CI_DOCVERSION_02X	Duplicates	3823	3	2643239	100940	4	2019-03-25 13:31:32	3565	79	117523	146107	264
CLIENTOS.CI_DOCVERSION_03X	Duplicates	705	2	13533	13498	2	2019-03-25 13:31:32	681	47	13533	-1	-1
CLIENTOS.CI_DOCVERSION_04X	Duplicates	5194	3	2643789	103637	5	2019-03-25 13:31:32	4861	83	157120	255369	264
CLIENTOS.CI_DOCVERSION_06X	Unique	5541	3	2658933	1003	5	2019-03-25 13:31:32	5216	86	1116	1118	181
CLIENTOS.CI_DOCVERSION_07X	Duplicates	4989	3	2497496	1003	6	2019-03-25 13:31:32	4709	85	1810	1906	190
CLIENTOS.CI_DOCVERSION_08X	Duplicates	739	2	168	168	1	2019-03-25 13:31:32	713	50	-1	-1	-1
CLIENTOS.CI_DOCVERSION_09X	Unique	5957	3	2658933	1	5	2019-03-25 13:31:32	5617	85	5	185922	265

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
U7C98_CIGNA_ACCT_NR	VARGRAPHIC (7)	No	18	Ascending	N/A	103637	g'3902341'	g'3320953'	119	Yes	185921	
U66D4_CWR_DOCUMENT_ID	DOUBLE (8)	No	9	Ascending	N/A	31	+2.07000000	+2.80000000	109	Yes	185754	
UEF86_FREQUENCY	INTEGER (4)	No	5	Ascending	N/A	4	3	2	116	Yes	185921	
CREATE_DATE	TIMESTAMP (10)	No	11	Ascending	N/A	2277776	'2019-03-08-1	'2016-03-15-1	6	Yes	0	
U04A6_CIGNA_CLIENT_ID	INTEGER (4)	No	5	Ascending	N/A	100940	103857	1002	120	Yes	185921	
OBJECT_ID	VARCHAR (16)	No	20	N/A	1	2658933	[illegible]	'	0	No	0	
OBJECT_CLASS_ID	VARCHAR (16)	No	20	N/A	N/A	5	[illegible]	'c_J0	1	No	0	

Scenario name	Qualifier	Class	Connection	Total cost	I/O cost	CPU cost
Original SQL		5	CEOS3PVS.	60.52	8.00	231095.41
Updated 4X Index		5	CEOS3PVS.	15.14	2.00	101590.64



Sorting #1 (1|15)

- One of the most powerful features of indexes is to eliminate reading all qualifying data and then physically sorting it.
- SQL calls with ORDER BY, GROUP BY and DISTINCT clauses frequently require physically sorting the results. And if the results set is large, overflowing to temp space on disk is typical and has even larger impacts on performance.
- This applies to OLTP particularly when paging through multiple rows of data, and to batch processing.
- Proper indexing taking into consideration the WHERE clause predicates AND sort requirements, can lead to huge savings.

Sorting #1 (2|15)

```
SELECT ...
  FROM CGIOS2.DocVersion T0
 WHERE ((T0.object_class_id IN (?))
   AND T0.home_id IS NULL            -- card=1; always NULL
   AND T0.recovery_item_id IS NULL -- card=1; always NULL
   AND ( ( version_status = ?
   AND ( ube06_searchtype = ? OR ube06_searchtype = ?))))
 ORDER BY u1708_documenttitle ASC, object_id ASC
 FETCH FIRST 1000 ROWS ONLY
OPTIMIZE FOR 1000 ROWS
```

- An index on DOCVERSION (VERSION_STATUS, OBJECT_CLASS_ID, U1708_DOCUMENTTITLE, OBJECT_ID, UBE06_SEARCHTYPE) resolves the equal and IN(?) (really an equal) from the WHERE clause, the two order by columns, and the final OR condition on searchtype.

Sorting #1 (3|15)

- The index also resolves paging criteria on subsequent page request SQL calls that look similar, with additional paging WHERE criteria on U1708_DOCUMENTTITLE and OBJECT_ID.
- Allows processing to match and position within the index based on the first two columns, and then retrieve the qualifying data in the desired sort sequence WITHOUT ACTUALLY SORTING because it’s traversing the sorted index.
- While traversing the index, it applies the remaining filtering criteria on searchtype and fetches and returns the data rows in the desired sequence meeting those criteria.
- There are various names for this including “step-through” index processing.

Sorting #1 (4|15)

- Can be very beneficial when processing large amounts of data, for example batch processing.
  - Instead of having to retrieve and fetch all qualifying data and sort it, the data is accessed via the index in sorted fashion.
  - Even better if the table data is “clustered” by this index (coming up shortly) to minimize physical IO to the table data pages.
- Even for smaller processing such as OLTP, instead of retrieving all qualifying data and sorting it, data is identified immediately and in pre-sorted sequence, particularly useful for paging programs.

Sorting #2 (5|15)

- Another example:

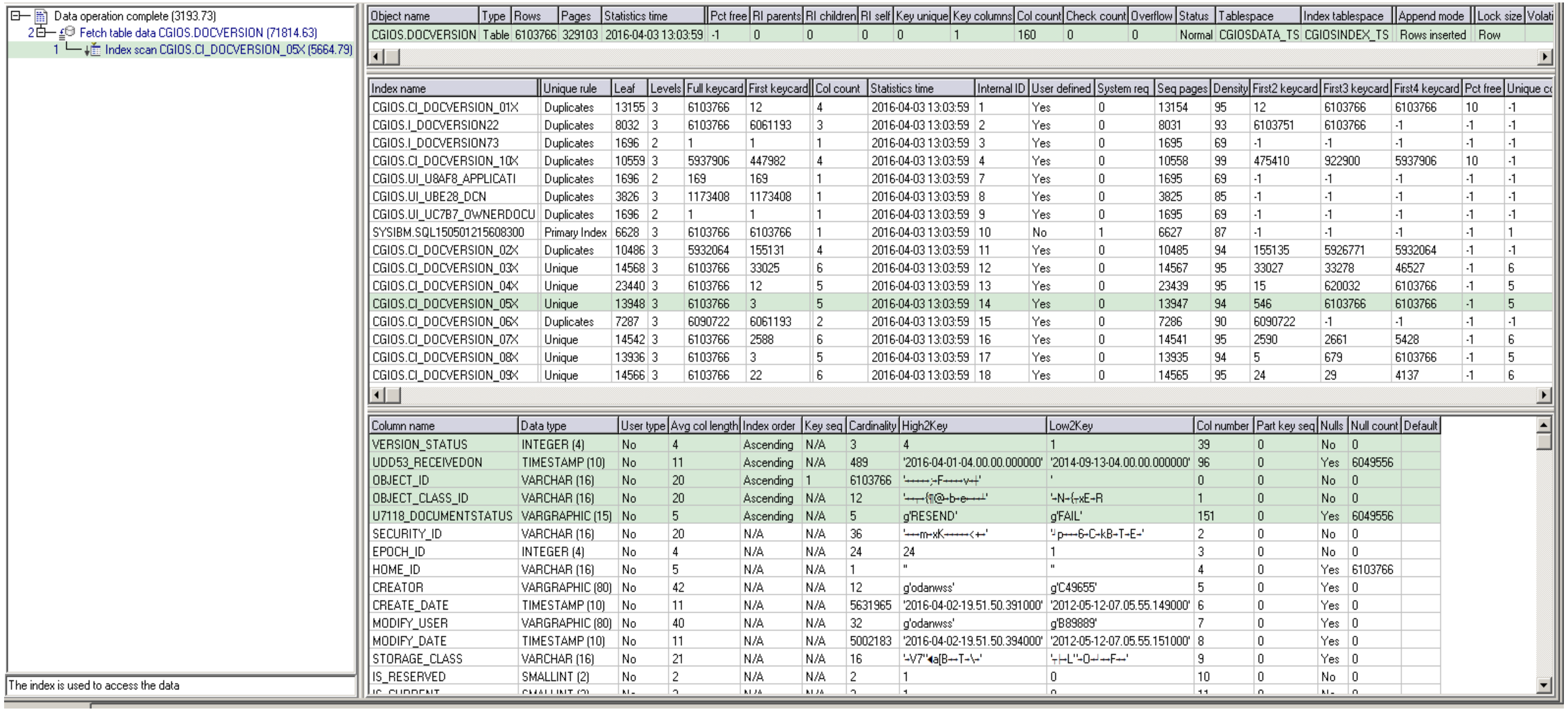
```
SELECT ...
  FROM DocVersion T0
 WHERE (T0.home_id IS NULL              -- card=1; always NULL
   AND ( ( ( u7118_documentstatus = ? -- card=5; 2 selected; poor index
    OR u7118_documentstatus = ?)       -- candidate; but small & needed
   AND object_class_id IN (?, ?, ?, ?, ?) -- card=12; 5 selected; poor
   AND version_status = ?)))               -- card= 3; 1 selected; poor
 ORDER BY udd53_receivedon ASC, object_id ASC
 FETCH FIRST 400 ROWS ONLY
OPTIMIZE FOR 400 ROWS
```

- Created new index (VERSION_STATUS, UDD53_RECEIVEDON, OBJECT_ID, OBJECT_CLASS_ID, U7118_DOCUMENTSTATUS).

Sorting #2 (6|15)

- Even with poor cardinality on the where clauses, this SQL statement ran well by matching on the first column, using the second and third to resolve the physical sort, and the fourth and fifth to provide additional screening on columns where more than one value is specified.
- The DBMS can “step through” the index and retrieve the rows in sequence as they appear on the index, up to 400 as specified in the SQL.
- And since the data is “clustered” (coming up next) on this index, physical IO is minimized.

Sorting #2 (7|15)



Data operation complete (3193.73)
2 Fetch table data CGIOS.DOCVERSION (71814.63)
1 Index scan CGIOS.CI_DOCVERSION_05X (5664.79)

The index is used to access the data

Object name	Type	Rows	Pages	Statistics time	Pct free	RI parents	RI children	RI self	Key unique	Key columns	Col count	Check count	Overflow	Status	Tablespace	Index tablespace	Append mode	Lock size	Volati
CGIOS.DOCVERSION	Table	6103766	329103	2016-04-03 13:03:59	-1	0	0	0	0	1	160	0	0	Normal	CGIOSDATA_TS	CGIOSINDEX_TS	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Col count	Statistics time	Internal ID
CGIOS.CI_DOCVERSION_01X	Duplicates	13155	3	6103766	12	4	2016-04-03 13:03:59	1
CGIOS.I_DOCVERSION22	Duplicates	8032	3	6103766	6061193	3	2016-04-03 13:03:59	2
CGIOS.I_DOCVERSION73	Duplicates	1696	2	1	1	1	2016-04-03 13:03:59	3
CGIOS.CI_DOCVERSION_10X	Duplicates	10559	3	5937906	447982	4	2016-04-03 13:03:59	4
CGIOS.UI_U8AF8_APPLICATI	Duplicates	1696	2	169	169	1	2016-04-03 13:03:59	7
CGIOS.UI_UBE28_DCN	Duplicates	3826	3	1173408	1173408	1	2016-04-03 13:03:59	8
CGIOS.UI_UC7B7_OWNERDOCU	Duplicates	1696	2	1	1	1	2016-04-03 13:03:59	9
SYSIBM.SQL150501215608300	Primary Index	6628	3	6103766	6103766	1	2016-04-03 13:03:59	10
CGIOS.CI_DOCVERSION_02X	Duplicates	10486	3	5932064	155131	4	2016-04-03 13:03:59	11
CGIOS.CI_DOCVERSION_03X	Unique	14568	3	6103766	33025	6	2016-04-03 13:03:59	12
CGIOS.CI_DOCVERSION_04X	Unique	23440	3	6103766	12	5	2016-04-03 13:03:59	13
CGIOS.CI_DOCVERSION_05X	Unique	13948	3	6103766	3	5	2016-04-03 13:03:59	14
CGIOS.CI_DOCVERSION_06X	Duplicates	7287	3	6090722	6061193	2	2016-04-03 13:03:59	15
CGIOS.CI_DOCVERSION_07X	Unique	14542	3	6103766	2588	6	2016-04-03 13:03:59	16
CGIOS.CI_DOCVERSION_08X	Unique	13936	3	6103766	3	5	2016-04-03 13:03:59	17
CGIOS.CI_DOCVERSION_09X	Unique	14566	3	6103766	22	6	2016-04-03 13:03:59	18

Index name	User defined	System req	Seq pages	Density	First2 keycard	First3 keycard	First4 keycard	Pct free	Unique co
CGIOS.CI_DOCVERSION_01X	Yes	0	13154	95	12	6103766	6103766	10	-1
CGIOS.I_DOCVERSION22	Yes	0	8031	93	6103751	6103766	-1	-1	-1
CGIOS.I_DOCVERSION73	Yes	0	1695	69	-1	-1	-1	-1	-1
CGIOS.CI_DOCVERSION_10X	Yes	0	10558	99	475410	922900	5937906	10	-1
CGIOS.UI_U8AF8_APPLICATI	Yes	0	1695	69	-1	-1	-1	-1	-1
CGIOS.UI_UBE28_DCN	Yes	0	3825	85	-1	-1	-1	-1	-1
CGIOS.UI_UC7B7_OWNERDOCU	Yes	0	1695	69	-1	-1	-1	-1	-1
SYSIBM.SQL150501215608300	No	1	6627	87	-1	-1	-1	-1	1
CGIOS.CI_DOCVERSION_02X	Yes	0	10485	94	155135	5926771	5932064	-1	-1
CGIOS.CI_DOCVERSION_03X	Yes	0	14567	95	33027	33278	46527	-1	6
CGIOS.CI_DOCVERSION_04X	Yes	0	23439	95	15	620032	6103766	-1	5
CGIOS.CI_DOCVERSION_05X	Yes	0	13947	94	546	6103766	6103766	-1	5
CGIOS.CI_DOCVERSION_06X	Yes	0	7286	90	6090722	-1	-1	-1	-1
CGIOS.CI_DOCVERSION_07X	Yes	0	14541	95	2590	2661	5428	-1	6
CGIOS.CI_DOCVERSION_08X	Yes	0	13935	94	5	679	6103766	-1	5
CGIOS.CI_DOCVERSION_09X	Yes	0	14565	95	24	29	4137	-1	6

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Part key seq	Nulls	Null count	Default
VERSION_STATUS	INTEGER (4)	No	4	Ascending	N/A	3	4	1	39	0	No	0	
UDD53_RECEIVEDON	TIMESTAMP (10)	No	11	Ascending	N/A	489	'2016-04-01-04.00.00.000000'	'2014-09-13-04.00.00.000000'	96	0	Yes	6049556	
OBJECT_ID	VARCHAR (16)	No	20	Ascending	1	6103766	'------;-F-----v--'	'	0	0	No	0	
OBJECT_CLASS_ID	VARCHAR (16)	No	20	Ascending	N/A	12	'----{¶@-b-e----'	'-N-{-xE-R	1	0	No	0	
U7118_DOCUMENTSTATUS	VARGRAPHIC (15)	No	5	Ascending	N/A	5	g'RESEND'	g'FAIL'	151	0	Yes	6049556	
SECURITY_ID	VARCHAR (16)	No	20	N/A	N/A	36	'----m-xK------<--'	' p----6-C-kB-T-E-'	2	0	No	0	
EPOCH_ID	INTEGER (4)	No	4	N/A	N/A	24	24	1	3	0	No	0	
HOME_ID	VARCHAR (16)	No	5	N/A	N/A	1	"	"	4	0	Yes	6103766	
CREATOR	VARGRAPHIC (80)	No	42	N/A	N/A	12	g'odanwss'	g'C49655'	5	0	Yes	0	
CREATE_DATE	TIMESTAMP (10)	No	11	N/A	N/A	5631965	'2016-04-02-19.51.50.391000'	'2012-05-12-07.05.55.149000'	6	0	Yes	0	
MODIFY_USER	VARGRAPHIC (80)	No	40	N/A	N/A	32	g'odanwss'	g'B89889'	7	0	Yes	0	
MODIFY_DATE	TIMESTAMP (10)	No	11	N/A	N/A	5002183	'2016-04-02-19.51.50.394000'	'2012-05-12-07.05.55.151000'	8	0	Yes	0	
STORAGE_CLASS	VARCHAR (16)	No	21	N/A	N/A	16	'-V7"◂a[B--T-\-'	'-+-L"-0---F--'	9	0	Yes	0	
IS_RESERVED	SMALLINT (2)	No	2	N/A	N/A	2	1	0	10	0	No	0	

Sorting #3 (8|15)

- Another example:

```
SELECT ...
  FROM DocVersion T0
 WHERE (T0.home_id IS NULL           -- card=1; always NULL
   AND T0.recovery_item_id IS NULL -- card=1; always NULL
   AND ( ( ub3e8_clientname = ?     -- card=30,209
   AND object_class_id=?            -- card=12
   AND version_status = ?)))        -- card=3
 ORDER BY ub3e8_clientname DESC, object_id ASC
 FETCH FIRST 400 ROWS ONLY
OPTIMIZE FOR 400 ROWS
```

- Created new index (VERSION_STATUS, OBJECT_CLASS_ID, UB3E8_CLIENTNAME, OBJECT_ID). Even with poor cardinality and one column with fair cardinality on the where clauses, this SQL statement ran well by matching on the first three columns, and using the fourth to resolve the physical sort.

Sorting #3 (9|15)

- The DBMS can "step through" the index and retrieve the rows in sequence as they appear on the index, up to 400 as specified in the SQL.
- And since the data is "clustered" (coming up next) on this index, physical IO is minimized.
- Note the DESC sort sequence doesn't matter in this example since there is an equal clause on UB3E8_CLIENTNAME; in fact the ORDER BY criteria on this particular column isn't even required, only OBJECT_ID.

Sorting #3 (10|15)

- Before

Data operation complete (16616.76)
- 6 Sorted table data scan (16616.76)
  - 5 Data is sorted (16616.76)
    - 4 Fetch table data CGIOS2.DOCVERSION (16616.75)
      - 3 Row identifier scan (RID) (1026.16)
        - 2 Data is sorted (1026.16)
          - 1 Index scan (no table access) CGIOS2.CI_DOCVERSION_05X (969.08)

Object name	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Key unique	Key columns	Col count	Append mode	Lock size	Volatile
CGIOS2.DOCVERSION	1880664	68698	2019-02-10 16:08:00	68698	0	0	0	0	1	113	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	First2 keycard	First3 keycard	First4 keycard	Pct free
SYSIBM.SQL180619113028090	Primary Index	2042	3	1880664	1880664	100	1	2019-02-10 16:08:00	-1	-1	-1	-1
CGIOS2.I_DOCVERSION22	Unique	2475	3	1880664	1880663	98	3	2019-02-10 16:08:00	1880664	1880664	-1	-1
CGIOS2.I_DOCVERSION73	Duplicates	523	2	1	1	100	1	2019-02-10 16:08:00	-1	-1	-1	-1
CGIOS2.CI_DOCVERSION_01X	Unique	4934	3	1880664	59153	80	5	2019-02-10 16:08:00	59154	238162	1880664	10
CGIOS2.CI_DOCVERSION_02X	Unique	5315	3	1880664	153782	90	5	2019-02-10 16:08:00	153783	284313	1880664	10
CGIOS2.CI_DOCVERSION_04X	Duplicates	666	3	102640	102632	94	3	2019-02-10 16:08:00	102633	102640	-1	-1
CGIOS2.CI_DOCVERSION_03X	Duplicates	2873	3	1460203	1459990	86	3	2019-02-10 16:08:00	1459991	1460203	-1	-1
CGIOS2.CI_DOCVERSION_05X	Duplicates	8006	3	1880664	2	86	5	2019-02-10 16:08:00	10	1692670	1880664	-1

Column name	Data type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
VERSION_STATUS	INTEGER (4)	4	Ascending	N/A	2	4	1	40	No	0	
OBJECT_CLASS_ID	VARCHAR (16)	20	Ascending	N/A	9	'}p[illegible]zE[illegible]'	'a	1	No	0	
U1708_DOCUMENTTITLE	VARGRAPHIC (255)	64	Ascending	N/A	1880664	g'sample'	g'30083335 - Vocat'	54	Yes	121556	
OBJECT_ID	VARCHAR (16)	20	Ascending	1	1880664	'h[illegible]P	'd5	0	No	0	
UBE06_SEARCHTYPE	INTEGER (4)	5	Ascending	N/A	2		2	70	Yes	1880656	

- After

Data operation complete (37.04)
- 2 Fetch table data CGIOS2.DOCVERSION (37.04)
  - 1 Index scan CGIOS2.CI_DOCVERSION_06X (15.15)

Object name	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Key unique	Key columns	Col count	Append mode	Lock size	Volatile
CGIOS2.DOCVERSION	1880664	68698	2019-02-12 13:21:17	68698	0	0	0	0	1	113	Rows inserted	Row	

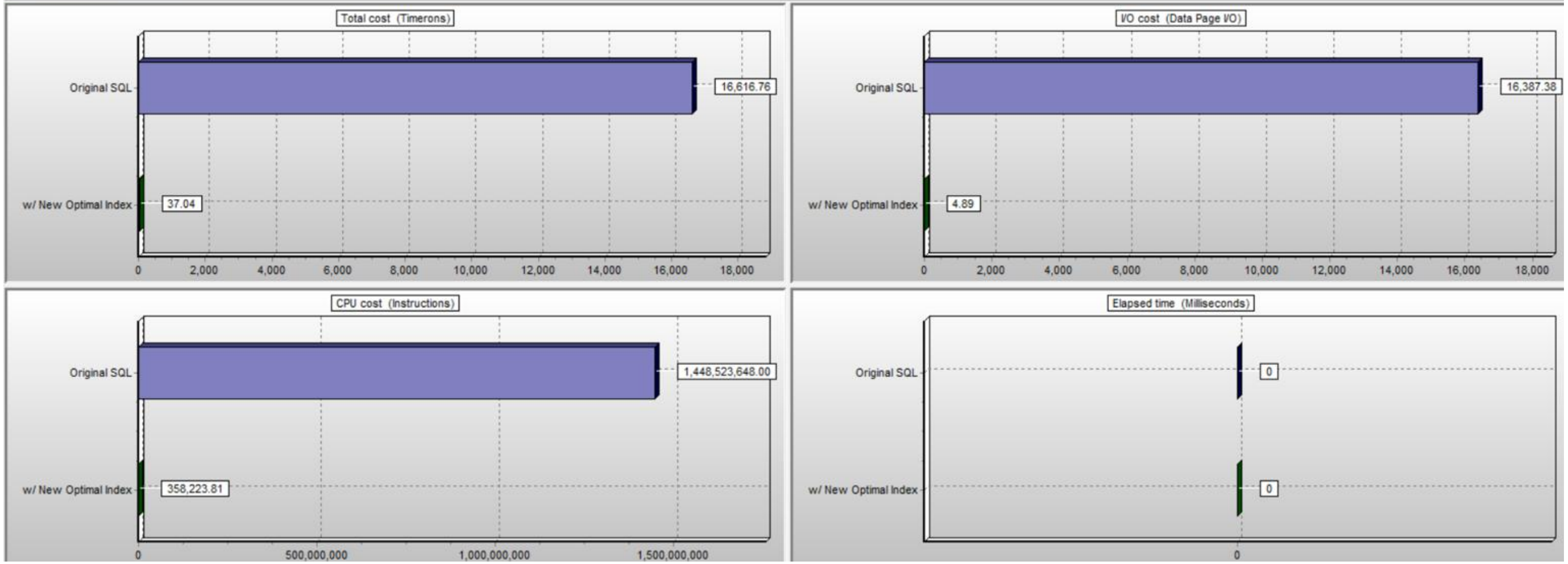
Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	First2 keycard	First3 keycard	First4 keycard	Pct free	Unique col count
SYSIBM.SQL180619113028090	Primary Index	2042	3	1880664	1880664	100	1	2019-02-12 13:21:17	-1	-1	-1	-1	1
CGIOS2.I_DOCVERSION22	Unique	2475	3	1880664	1880663	98	3	2019-02-12 13:21:17	1880664	1880664	-1	-1	3
CGIOS2.I_DOCVERSION73	Duplicates	523	2	1	1	100	1	2019-02-12 13:21:17	-1	-1	-1	-1	-1
CGIOS2.CI_DOCVERSION_01X	Unique	4934	3	1880664	59153	80	5	2019-02-12 13:21:17	59154	238162	1880664	10	5
CGIOS2.CI_DOCVERSION_02X	Unique	5315	3	1880664	153782	90	5	2019-02-12 13:21:17	153783	284313	1880664	10	5
CGIOS2.CI_DOCVERSION_04X	Duplicates	666	3	102640	102632	94	3	2019-02-12 13:21:17	102633	102640	-1	-1	-1
CGIOS2.CI_DOCVERSION_03X	Duplicates	2873	3	1460203	1459990	86	3	2019-02-12 13:21:17	1459991	1460203	-1	-1	-1
CGIOS2.CI_DOCVERSION_05X	Duplicates	8006	3	1880664	2	86	5	2019-02-12 13:21:17	10	1692670	1880664	-1	-1
CGIOS2.CI_DOCVERSION_06X	Duplicates	3691	3	1880664	2	99	4	2019-02-12 13:21:17	10	30011	1880664	10	-1

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Part key seq	Nulls	Null count
VERSION_STATUS	INTEGER (4)	No	4	Ascending	N/A	2	4	1	40	0	No	0
OBJECT_CLASS_ID	VARCHAR (16)	No	20	Ascending	N/A	9	'}p[illegible]zE[illegible]'	'a	1	0	No	0
UB3E8_CLIENTNAME	VARGRAPHIC (100)	No	5	Ascending	N/A	30209	g'VAMSI'	g'1111102'	110	0	Yes	1820660
OBJECT_ID	VARCHAR (16)	No	20	Ascending	1	1880664	'h[illegible]P	'd5	0	0	No	0

Sorting #3 (11|15)

- Comparison

Scenario name	Qualifier	Class	Connection	Total cost	I/O cost	CPU cost
Original SQL		5	CEOSGPVS	16616.76	16387.38	1448523648.00
w/ New Optimal Index		5	CEOSGPVS	37.04	4.89	358223.81



Sorting #4 (12|15)

- Another example:

```
SELECT ...
  FROM DocVersion T0
  WHERE (T0.home_id IS NULL          -- card=1; always NULL
    AND T0.recovery_item_id IS NULL -- card=1; always NULL
    AND ( ( uab28_policynumber = ?  -- card=64,001
    AND object_class_id=?           -- card=12
    AND version_status = ?)))       -- card=3
  ORDER BY ud1e3_policyfromdate DESC, object_id ASC
  FETCH FIRST 400 ROWS ONLY
OPTIMIZE FOR 400 ROWS
```

- Created new index (VERSION_STATUS, OBJECT_CLASS_ID, UAB28_POLICYNUMBER, UD1E3_POLICYFROMDATE **DESC**, OBJECT_ID). Even with poor cardinality and one column with fair cardinality on the where clause, this SQL statement ran well by matching on the first three columns, and using the fourth and fifth to resolve the physical sort.

Sorting #4 (13|15)

- The DBMS can "step through" the index and retrieve the rows in sequence as they appear on the index, up to 400 as specified in the SQL.
- And since the data is "clustered" (coming up next) on this index, physical IO is minimized.
- Note that previously this used the index created in the prior example. Note also the DESC sort sequence does matter in this example. Although with REVERSE SCAN indexing, either works.

Sorting #4 (14|15)

- Before

Data operation complete (8110.23)
- 6 Sorted table data scan (8110.23)
  - 5 Data is sorted (8110.23)
    - 4 Fetch table data CGIOS2.DOCVERSION (8110.23)
      - 3 Row identifier scan (RID) (538.97)
        - 2 Data is sorted (538.97)
          - 1 Index scan (no table access) CGIOS2.CI_DOCVERSION_06X (481.88)

Object name	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Key unique	Key columns	Col count	Append mode	Lock size	Volatile
CGIOS2.DOCVERSION	1880664	68698	2019-02-12 13:21:17	68698	0	0	0	0	1	113	Rows inserted	Row	

Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	First2 keycard	First3 keycard	First4 keycard	Pct free
SYSIBM.SQL180619113028090	Primary Index	2042	3	1880664	1880664	100	1	2019-02-12 13:21:17	-1	-1	-1	-1
CGIOS2.I_DOCVERSION22	Unique	2475	3	1880664	1880663	98	3	2019-02-12 13:21:17	1880664	1880664	-1	-1
CGIOS2.I_DOCVERSION73	Duplicates	523	2	1	1	100	1	2019-02-12 13:21:17	-1	-1	-1	-1
CGIOS2.CI_DOCVERSION_01X	Unique	4934	3	1880664	59153	80	5	2019-02-12 13:21:17	59154	238162	1880664	10
CGIOS2.CI_DOCVERSION_02X	Unique	5315	3	1880664	153782	90	5	2019-02-12 13:21:17	153783	284313	1880664	10
CGIOS2.CI_DOCVERSION_04X	Duplicates	666	3	102640	102632	94	3	2019-02-12 13:21:17	102633	102640	-1	-1
CGIOS2.CI_DOCVERSION_03X	Duplicates	2873	3	1460203	1459990	86	3	2019-02-12 13:21:17	1459991	1460203	-1	-1
CGIOS2.CI_DOCVERSION_05X	Duplicates	8006	3	1880664	2	86	5	2019-02-12 13:21:17	10	1692670	1880664	-1
CGIOS2.CI_DOCVERSION_06X	Duplicates	3691	3	1880664	2	99	4	2019-02-12 13:21:17	10	30011	1880664	10

Column name	Data type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count	Default
VERSION_STATUS	INTEGER (4)	4	Ascending	N/A	2	4	1	40	No	0	
OBJECT_CLASS_ID	VARCHAR (16)	20	Ascending	N/A	9	'}pl'IzE'	'a	1	No	0	
UB3E8_CLIENTNAME	VARGRAPHIC (100)	5	Ascending	N/A	30209	g'VAMSI'	g'1111102'	110	Yes	1820660	
OBJECT_ID	VARCHAR (16)	20	Ascending	1	1880664	'hP	'd5	0	No	0	

- After

Data operation complete (26.05)
- 2 Fetch table data CGIOS2.DOCVERSION (26.05)
  - 1 Index scan CGIOS2.CI_DOCVERSION_07X (15.14)

Object name	Rows	Pages	Statistics time	File pages	RI parents	RI children	RI self	Key unique	Key columns	Col count	Append mode	Lock size	Volatile
CGIOS2.DOCVERSION	1880664	68698	2019-02-13 10:44:00	68698	0	0	0	0	1	113	Rows inserted	Row	

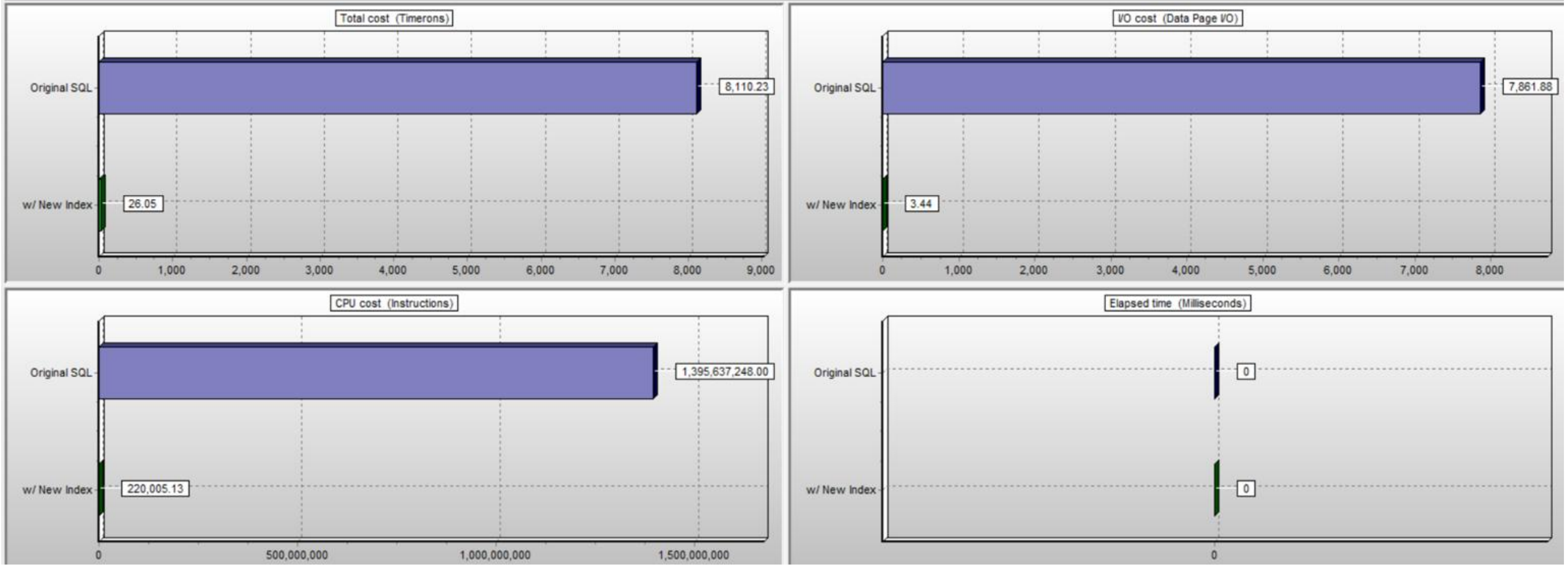
Index name	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	First2 keycard	First3 keycard	First4 keycard	Pct free	Unique…
SYSIBM.SQL180619113028090	Primary Inde	2042	3	1880664	1880664	100	1	2019-02-13 10:44:00	-1	-1	-1	-1	1
CGIOS2.I_DOCVERSION22	Unique	2475	3	1880664	1880663	98	3	2019-02-13 10:44:00	1880664	1880664	-1	-1	3
CGIOS2.I_DOCVERSION73	Duplicates	523	2	1	1	100	1	2019-02-13 10:44:00	-1	-1	-1	-1	-1
CGIOS2.CI_DOCVERSION_01X	Unique	4934	3	1880664	59153	80	5	2019-02-13 10:44:00	59154	238162	1880664	10	5
CGIOS2.CI_DOCVERSION_02X	Unique	5315	3	1880664	153782	90	5	2019-02-13 10:44:00	153783	284313	1880664	10	5
CGIOS2.CI_DOCVERSION_04X	Duplicates	666	3	102640	102632	94	3	2019-02-13 10:44:00	102633	102640	-1	-1	-1
CGIOS2.CI_DOCVERSION_03X	Duplicates	2873	3	1460203	1459990	86	3	2019-02-13 10:44:00	1459991	1460203	-1	-1	-1
CGIOS2.CI_DOCVERSION_05X	Duplicates	8006	3	1880664	2	86	5	2019-02-13 10:44:00	10	1692670	1880664	-1	-1
CGIOS2.CI_DOCVERSION_06X	Duplicates	3691	3	1880664	2	99	4	2019-02-13 10:44:00	10	30011	1880664	10	-1
CGIOS2.CI_DOCVERSION_07X	Duplicates	5349	3	1880664	2	99	5	2019-02-13 10:44:00	10	60229	60229	10	-1

Column name	Data type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Nulls	Null count
VERSION_STATUS	INTEGER (4)	4	Ascending	N/A	2	4	1	40	No	0
OBJECT_CLASS_ID	VARCHAR (16)	20	Ascending	N/A	9	'}pl'IzE'	'a	1	No	0
UAB28_POLICYNUMBER	VARGRAPHIC (11)	20	Ascending	N/A	64001	g'40000226'	g'10000002'	98	Yes	68279
UD1E3_POLICYFROMDATE	TIMESTAMP (10)	11	Descending	N/A	61441	'2174-04-10-04.00.00.000000'	'2010-01-02-05.00.00.000000'	107	Yes	1820660
OBJECT_ID	VARCHAR (16)	20	Ascending	1	1880664	'hP	'd5	0	No	0

Sorting #4 (15|15)

- Comparison

Scenario name	Qualifier	Class	Connection	Total cost	I/O cost	CPU cost
Original SQL		5	CEOSGPVS	8110.23	7861.88	1395637248.00
w/ New Index		5	CEOSGPVS	26.05	3.44	220005.13



Referential Integrity (1|8)

- There are various types of referential integrity or RI.
  - RI can be DBMS-enforced or application-enforced.
  - These days, let the DBMS handle RI unless there are situations the DBMS can't handle.
- The simplest is declaring a column or group of columns unique:
  - A one-column example is the CLAIM table where you can only have one row for a CLAIM_ID; this index would be defined as UNIQUE and the DBMS would prevent inserting a row with a duplicate CLAIM_ID.
  - A two-column example is the CLAIM_LINE table where you can only have one row for the combination of CLAIM_ID and LINE_ITEM. This two-column index would be defined as UNIQUE and the DBMS would prevent inserting a row with a duplicate CLAIM_ID / LINE_ITEM combination.

Referential Integrity (2|8)

- When an index is unique, either one column or multi-column:
  - Define the index as unique to tell the optimizer there will be only one row for any entry (e.g. the chain length is always one).
  - The DBMS can better optimize queries knowing this column or group of columns is unique.
  - You can “INCLUDE” additional columns for index-only access (coming up shortly).
- Another type of RI is the parent-child relationship involving foreign keys:
  - A CLAIM_LINE row can’t be inserted until the corresponding parent CLAIM row has been inserted.
  - The DBMS checks for the existence of the CLAIM_ID on the CLAIM table before allowing any inserts to CLAIM_LINE with that same CLAIM_ID.
  - For this check it’s imperative there's an index on the CLAIM_ID column, which there almost certainly is since it’s likely the primary key of the table.

Referential Integrity (3|8)

- Consider the process to archive and purge old claims after some period. There are three RI options when deleting the parent:
  - To delete all the children too at the same time as part of that delete, use CASCADE.
  - If instead you don't want to delete the parent until all children are deleted first, use RESTRICT to prevent deleting child rows.
  - Can also set the CLAIM_ID on the CLAIM_LINE table to NULL using the SET NULL option.
- Same options on UPDATEs.

Referential Integrity (4|8)

- For example:

```
ALTER TABLE CLAIM_LINE
  ADD CONSTRAINT CLAIM_FK
  FOREIGN KEY(CLAIM_ID)
  REFERENCES CLAIM        -- On CLAIM_LINE insert, ensure CLAIM_ID exists on CLAIM
    ON DELETE CASCADE     -- On CLAIM delete, cascades;  index CLAIM_LINE(CLAIM_ID)
    ON UPDATE RESTRICT    -- On CLAIM update, restricts; index CLAIM_LINE(CLAIM_ID)
```

- Index on CLAIM_LINE (CLAIM_ID, LINE_ITEM) suffices since CLAIM_ID is leading column.

Referential Integrity (5|8)

- Here's an example of a very straightforward DELETE:

```
delete from REASON_NOT_COVERED
 where RNC = 1368
```

Referential Integrity (6|8)

- Here's an example of a very straightforward DELETE:

```
delete from REASON_NOT_COVERED
 where RNC = 1368
```

- And here's the EXPLAIN:

```
Data operation complete (59118904.00)
8 Filter data (59118904.00)
  7 Table access full DBADMIN.PEND_CHASE_RNC_DET (50.27)
  6 Table access full DBADMIN.CLAIM (14461699.00)
  5 Table access full DBADMIN.CLAIM_AU (14857040.00)
  4 Table access full DBADMIN.CLAIM_SERVICE_ITEM (13861336.00)
  3 Table access full DBADMIN.CL_SERV_ITEM_AU (15938724.00)
  2 Delete statement (50.03)
    1 Index scan (no table access) DBADMIN.PK_RNC (25.02)
```

Object name	Type	Rows	Pages	Statistics time	File pages	Pct free	RI parents	RI children	RI self	Key unique	Key columns	Col count
DBADMIN.PEND_CHASE_RNC_DET	Table	273	2	2016-07-10 03:27:09	2	-1	1	0	0	0	2	2
DBADMIN.CLAIM	Table	14450057	3624802	2016-07-10 03:14:26	3691072	-1	7	13	0	0	1	124
DBADMIN.CLAIM_AU	Table	15014211	3774298	2016-07-10 03:12:46	3791936	-1	7	3	0	0	2	126
DBADMIN.CLAIM_SERVICE_ITEM	Table	38444745	3510557	2016-07-10 03:21:49	3532160	-1	5	0	0	0	2	77
DBADMIN.CL_SERV_ITEM_AU	Table	40200693	4043198	2016-07-10 03:29:55	4062445	-1	4	0	0	0	3	78
DBADMIN.REASON_NOT_COVERED	Table	542	47	2016-07-10 03:23:40	47	-1	0	5	0	0	1	8

Index name	Type	Unique rule	Leaf	Levels	Full keycard	First keycard	Cluster ratio	Col count	Statistics time	Internal ID	User defined	System req	Seq pages	De
DBADMIN.PK_RNC	Regular	Primary Index	2	2	542	542	-1	1	2016-07-10 03:23:40	1	No	1	0	0

Column name	Data type	User type	Avg col length	Index order	Key seq	Cardinality	High2Key	Low2Key	Col number	Part key seq	Nulls	Null count	Default
RNC	SMALLINT (2)	No	2	Ascending	1	542	1941	3	0	0	No	0	
RNC_DESCRIPTION	CHARACTER (100)	No	101	N/A	N/A	365	'Zero dollars b	'Allegiance-CH	1	0	Yes	1	
PRINTED_DESC	VARCHAR (1024)	No	194	N/A	N/A	356	'YOUR PLAN	' Based on Ort	2	0	Yes	0	
SAVINGS	CHARACTER (1)	No	2	N/A	N/A	3	'Y'	' '	3	0	Yes	0	
STATUS_IND	SMALLINT (2)	No	3	N/A	N/A	1			4	0	Yes	542	
NETWORK_NAME	CHARACTER (10)	No	11	N/A	N/A	3	''	' '	5	0	Yes	539	
PRINT_DESTINATION	CHARACTER (1)	No	2	N/A	N/A	4	'T'	'P'	6	0	Yes	2	
TERM_DATE	DATE (4)	No	5	N/A	N/A	11	'2016-07-07'	'2007-11-12'	7	0	Yes	413	

- What happened here? Why five tablescans? Four are quite large compared to the simple DELETE using the PK index.

Referential Integrity (7|8)

- Looking at the REASON_NOT_COVERED DDL, we see:

```
ALTER TABLE PEND_CHASE_RNC_DET
  ADD CONSTRAINT FK_PEND_CHASE_D1 FOREIGN KEY (RNC)
    REFERENCES REASON_NOT_COVERED (RNC)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION

ALTER TABLE CLAIM
  ADD CONSTRAINT FK_536 FOREIGN KEY (RNC)
    REFERENCES REASON_NOT_COVERED (RNC)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION

ALTER TABLE CLAIM_AU
  ADD CONSTRAINT FK_20654 FOREIGN KEY (RNC)
    REFERENCES REASON_NOT_COVERED (RNC)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION

ALTER TABLE CLAIM_SERVICE_ITEM
  ADD CONSTRAINT FK_537 FOREIGN KEY (RNC)
    REFERENCES REASON_NOT_COVERED (RNC)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION

ALTER TABLE CL_SERV_ITEM_AU
  ADD CONSTRAINT FK_20657 FOREIGN KEY (RNC)
    REFERENCES REASON_NOT_COVERED (RNC)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
```

Referential Integrity (8|8)

- Each of these tables:

```
PEND_CHASE_RNC_DET
CLAIM
CLAIM_AU
CLAIM_SERVICE_ITEM
CL_SERV_ITEM_AU
```

  requires an index with leading column RNC to eliminate the table scans.

- Table PEND_CHASE_RNC_DET is only two pages so a table scan is acceptable, however an index would be optimal and avoid accessing the data pages at all.

Index-only access (1|2)

- Often we can eliminate access to table data pages entirely by adding one or more columns to an index.
  - Doing this increases the size of the index, and time to create and maintain the index (e.g. INSERTs), but may also benefit other SQL calls.
  - The goal is to eliminate what will likely be physical IO to the table data pages to get additional columns in the select clause.
- Don’t overuse this feature; it may make sense in cases where adding a short column or two helps many SQL calls; that’s where to use this.
- Some DBMS index analyzers make recommendations that flood indexes with additional columns to reduce IO.

Index-only access (2|2)

- Doing this can lead to redundant indexing on primary keys and other unique indexes. DBAs often had to create a unique index to ensure a unique constraint, and for performance reasons create a second index with additional columns for index-only access which is redundant.
- To avoid this, nearly all DBMS have added the INCLUDE feature, to include additional columns on a unique index for index-only access without affecting the unique constraint on the base column or columns. For example:

```
CREATE UNIQUE INDEX CLAIM_PK
  ON CLAIM
 (CLAIM_ID ASC)
  INCLUDE
 (CLAIM_STATUS);
```

"One fetch" access (1|3)

- "One fetch" access is a very special case; a frequently-used example is a subselect to get the maximum (or minimum) of some date column:

```
SELECT MAX(EFF_DATE)
  FROM tablename
 WHERE ACCT_NUM =?
   AND SUBSC_PID=?
```

- This SQL will benefit from an index on (ACCT_NUM, SUBSC_PID) but then require access to data pages to get EFF_DATE, and a sort to derive the maximum value.
- Instead, create optimal index (ACCT_NUM, SUBSC_PID, EFF_DATE **DESC**) to:
  - Make this query index only and
  - "One fetch" because the first entry accessed will be the max since the collating sequence on EFF_DATE is descending.

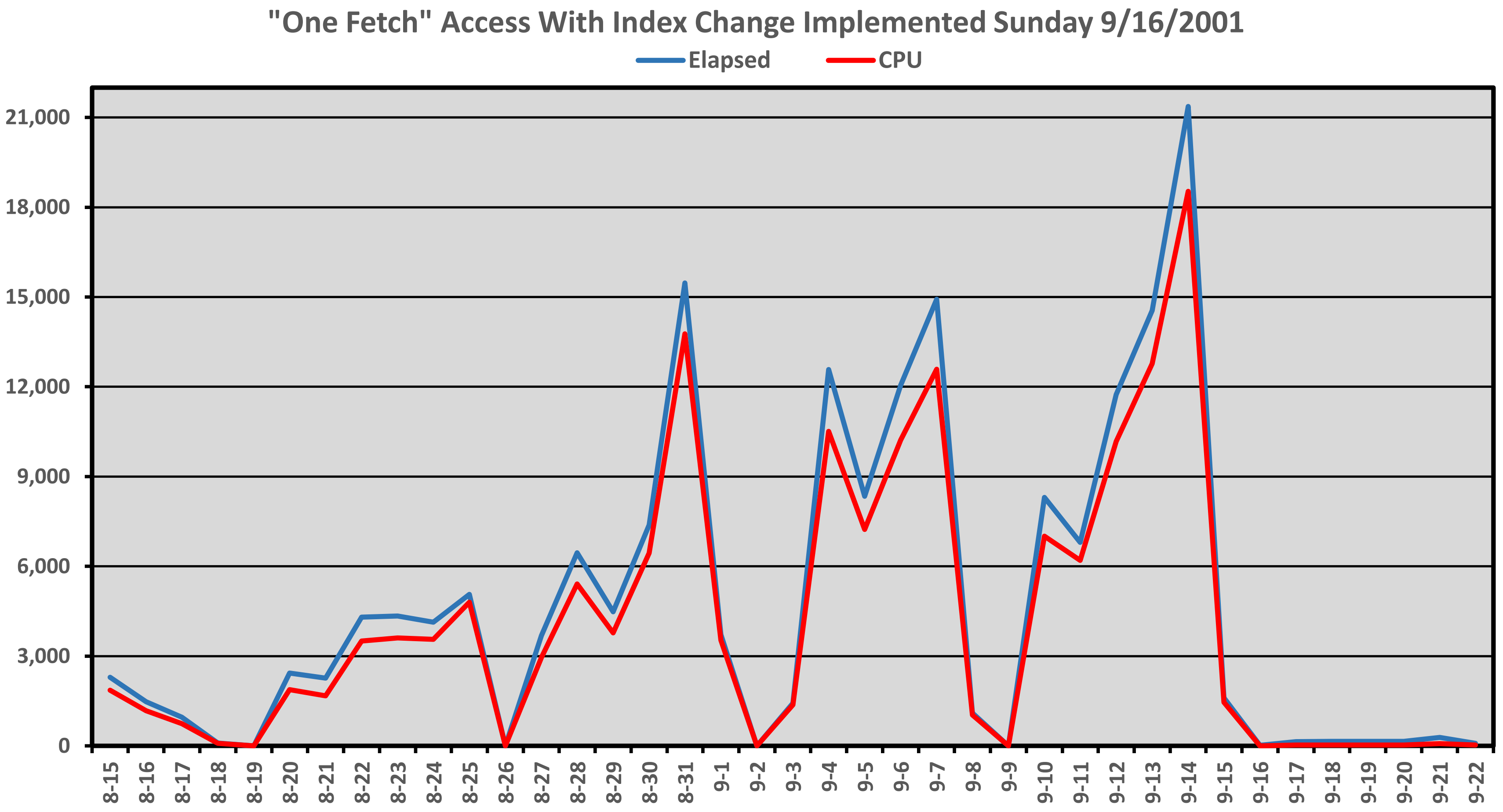
“One fetch” access (2|3)

- This has a lot of applicability especially when retrieving maximum or minimum values in a subselect passed back to complete criteria in the main select.
- An example from 2001 was a recommended index change to improve one SQL call run repeatedly (average of once every 20 seconds):

```
SELECT MAX(A_RANDOM_NUM) INTO :H :H
  FROM GGDD.PORG_PROV_RFL_MAP
```

- Changed index to (A_RANDOM_NUM **DESC**)
- Converted NMI-scan to "one-fetch" access; dramatically reduced lock contention, completely eliminated lock timeout abends (over 15 CICS abends per day), etc.
- **Saved $1,600,000 per year (CPU) plus intangible savings from dramatically reduced customer wait time, abends, frustration and rework.**

“One fetch” access (3|3)

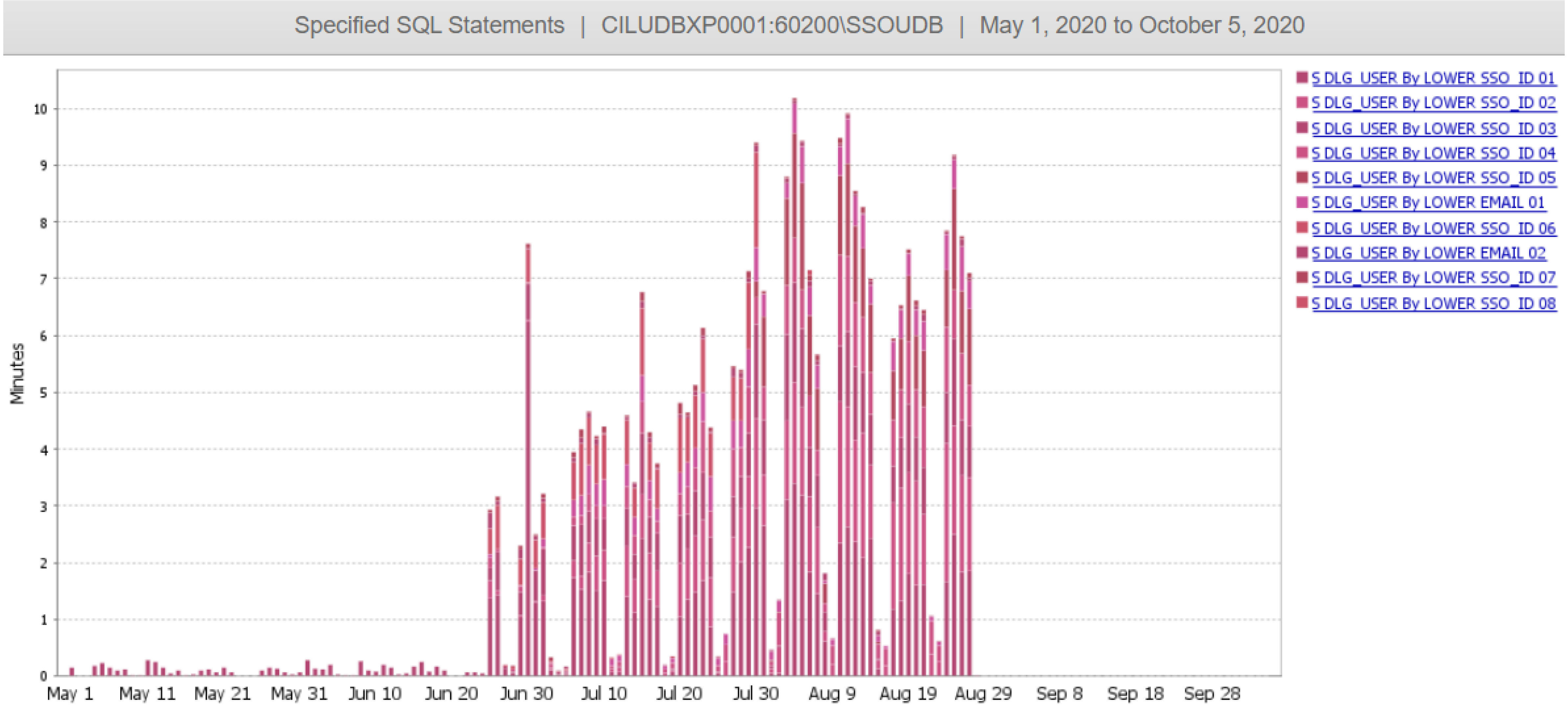


Indexing

- Expression-based indexes
  - Impacts of mixed case data
  - Data entered free form; could be lower, upper or mixed case
  - Data used frequently in searches, but unsure what case to use
  - Concerned about performance of searches? And results returned by searches? You should be
  - Recent example using relatively-new (Db2 10.5) LOWER (SSO_ID) and LOWER (EMAIL)
  - Two options:
    1. If possible, choose case to convert to <u>*before*</u> data row insert, and use for searches; either lower (e.g. email address) or upper (e.g. street address); will save processing time later and avoid potential issues
    2. If mixed case is required (e.g. names):
       - ✓ Insert data in mixed case for proper display
       - ✓ Settle on one case to use when retrieving data from a mixed-case column, either upper or lower (has no impact on how the data is displayed); in this case we'll use upper
       - ✓ Create an index on UPPER (LAST_NAME) for search purposes to avoid table / non-matching index scan
       - ✓ Always use WHERE UPPER (LAST_NAME) = ? (which is all upper case)
  - Recent example using relatively-new (Db2 10.5) LOWER (SSO_ID) and LOWER (EMAIL)

```
CREATE INDEX ERD.DLG_USER6X
  ON ERD.DLG_USER
  (LOWER(EMAIL) ASC)
  ALLOW REVERSE SCANS;
```

Indexing


Specified SQL Statements | CILUDBXP0001:60200\SSOUDB | May 1, 2020 to October 5, 2020
Minutes
10
9
8
7
6
5
4
3
2
1
0
May 1
May 11
May 21
May 31
Jun 10
Jun 20
Jun 30
Jul 10
Jul 20
Jul 30
Aug 9
Aug 19
Aug 29
Sep 8
Sep 18
Sep 28
S DLG_USER By LOWER SSO_ID 01
S DLG_USER By LOWER SSO_ID 02
S DLG_USER By LOWER SSO_ID 03
S DLG_USER By LOWER SSO_ID 04
S DLG_USER By LOWER SSO_ID 05
S DLG_USER By LOWER EMAIL 01
S DLG_USER By LOWER SSO_ID 06
S DLG_USER By LOWER EMAIL 02
S DLG_USER By LOWER SSO_ID 07
S DLG_USER By LOWER SSO_ID 08

Indexing

- EXCLUDE NULL KEYS
  - Situation encountered on FileNet database with column that should be unique
  - However this column is NULL for multiple “template rows”
  - Need to enforce uniqueness via UNIQUE index, but couldn’t until …
  - Relatively-new (Db2 10.5) EXCLUDE NULL KEYS allows multiple rows with NULL while enforcing uniqueness across all other rows

```
CREATE UNIQUE INDEX SVPP8DENTAL.CI_DOCVERSION_13X
  ON SVPP8DENTAL.DOCVERSION
  (U2728_DCN ASC)
  PCTFREE 10
  ALLOW REVERSE SCANS
  EXCLUDE NULL KEYS;
```

  - This was the first time we’ve ever had to use this option
  - Improves compatibility for migrations from Oracle (default in Oracle)

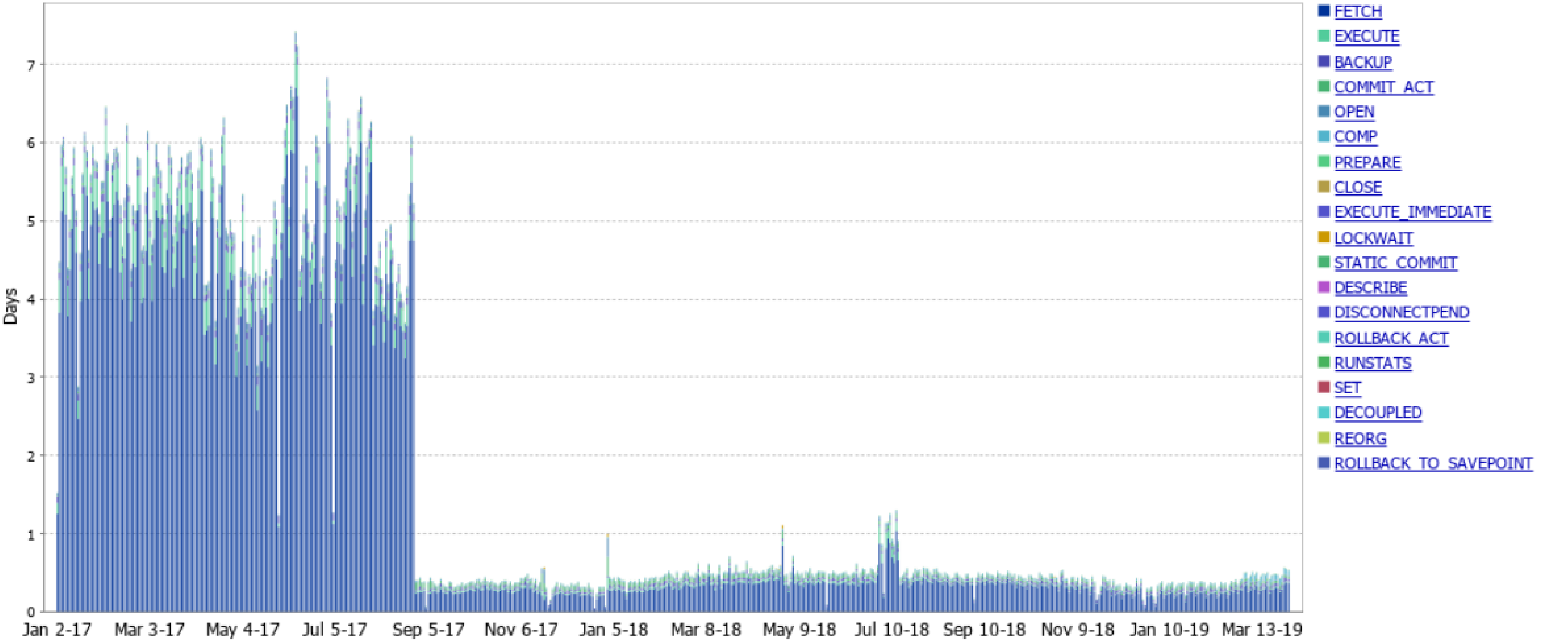
Topics

- HW, SW and Currency
- DB Configuration
- Maintenance
- Data Archive and Purge
- SQL
- Indexing
- **Flash Storage (SSD)**
- Implementation Details
- Q & A and More Information

Flash Storage (SSD)

- Storage types
  - SSD (Solid-State Drives) or Flash
  - Spindle
  - Optical
- Measurable impact
  - Large project that included multiple performance improvements
    1. Major data purging
    2. DB re-configuration, including HADR
    3. Created six new indexes and changed ten existing indexes
    4. HW, OS and DBMS upgrades
    5. CPU cores reduced 25%
    6. SSD or Flash storage for all tables, indexes and transaction logs
- We could easily measure the overall impact, but what impact did SSD alone have?
- We found out on 6/30 when we were “migrated” off SSD in error
- Two weekends later, migrated back onto SSD; see next slide showing the impact

Top Waits | CILUDBXP0003_OR_04:50100\OVIHPRD | January 1, 2017 to March 31, 2019 | Selected Days: Monday,Tuesday,Wednesday,...



Topics

- HW, SW and Currency
- DB Configuration
- Maintenance
- Data Archive and Purge
- SQL
- Indexing
- Flash Storage (SSD)
- **Implementation Details**
- Q & A and More Information

Implementation Details

- HADR
- Replication and bi-directional failback capability
  - Using the prior example
    1. DBMS upgrade required an OS upgrade
    2. OS upgrade required a HW upgrade
    3. Able to provide bi-directional failback capability with the additional HW copy
    4. While still operating on the old platform, we moved data to new platform, purged, reorged, runstat, and we also replicated updates from the operational database on the old platform to the new platform
    5. After two weeks, we “migrated” (re-pointed) the app to the new platform
    6. If we encountered an issue after migration, we could re-point to the old environment where the data was current, again due to replication to keep the old platform data accurate
    7. After several days, we cut the lifeline to reduce overhead and began to free up the old platform resources
    8. We used IIDR\SQL Replication (DProp) for the data propagation in both directions (old to new before migration and new to old after migration)

Topics

- HW, SW and Currency
- DB Configuration
- Maintenance
- Data Archive and Purge
- SQL
- Indexing
- Flash Storage (SSD)
- Implementation Details
- **Q & A and More Information**

Q & A and More Information

- Q & A
- "How To Make Up Time, Or, How To Save Five Days In A Single Day"
  - CRM application upgrade and performance improvement project including HW upgrade, CPU core reduction, OS upgrade, storage conversion to SSD, DBMS upgrade, many configuration updates, major data purges, six index adds and ten index changes, all while maintaining bidirectional failback capability
  - 2018-06-05 Central Canada Db2 User Group (CCDUG) Conference
  - 2019-06-04 International Db2 User Group (IDUG NA) Conference
- "Indexes: They're Not Just For Where Criteria Anymore"
  - Presentation covering many aspects of indexing, many slides included in this presentation
  - 2019-04-30 Central Canada Db2 User Group (CCDUG) Conference
  - 2020-08-13 International Db2 User Group (IDUG NA) Virtual Conference

- Jim Bean, Cigna Performance & Forensics
- Email: jim.bean@cigna.com