Db2 CPU Optimization: Static SQL In Memory Processing

- Michael W. Moss
- Value-4IT Limited

- 11 November 2020
- Session 1AW




Db2 CPU Optimization-Static SQL In Memory Processing: Agenda

- *Digital Transformation: Big Data Usage Trends; A Digital Universe*
- *IBM Z Server-Db2 for z/OS: Recent Architecture Evolution Observations*
- *I/O Processing Fundamentals: More Memory, Less CPU (MIPS)*
- *Data Access Observations: Hours, Days, Months or Years?*
- *Db2 Buffer Pools: Memory & Performance Optimization Observations*
- *Db2 for z/OS SQL: Static & Dynamic SQL Observations*
- *Db2 for z/OS: Fast Traverse Blocks (FTB) Observations*
- *Db2 for z/OS: General Purpose CPU Offload via zIIP*
- *Db2 for z/OS CPU Usage: Rely on Moore or Do More With Less?*
- *Db2 for z/OS: Other ISV High Performance In Memory Options*
- *Db2 for z/OS: SQL CPU Optimization Software Solution Options*

Digital Transformation: Big Data Usage Trends; A Digital Universe

Real-Time Data Creation: *As of 2020, on average, each living person in the world creates ~2 MB of data every second.*

All-Time Data Creation: *90%+ of all global data was created in the last 2 years.*

Total Data Stored: *~44 ZB (Zettabytes) or 44 Trillion GB (Gigabytes) as of 2020.*

Data Growth Rate: *Doubling in size every 2 years; the data Big Bang explosion happened, a long time ago! By 2025, a daily growth rate of ~450 Exabytes is anticipated…*

Data Analysis: *Only ~5% of data ever created is analysed, classified as target rich; 95% of data remains a mystery...*

Data Reference: *~60% of enterprise data stored on primary storage systems hasn't been accessed for 6+ months.*

Data Origin: *~80% of mission critical corporate data originated from, or is stored in an IBM Z Mainframe environment.*

Data Storage: *~50% of all corporate data uploaded to cloud structures in 2020.*

Unstructured Data: *By 2025, ~80% of global data will be unstructured, difficult to store in database structures & increasingly difficult to process for meaningful access.*

Data Utopia: *There's a good chance that a meaningful Single Source of Truth (SSoT) resides within an IBM Z Mainframe database structure, probably Db2 based…*

Too much data, not enough meaningful business information; Big Data Analytics?

IBM Z Server: Recent Architecture Evolution Observations

Z Server Type	Maximum CEC Memory	Maximum LPAR Memory	Memory Cost/GB	Memory Cost/TB	Maximum # Cores	ROT 20 GB+❷ Memory/Core
zEC12	3 TB	1 TB	$1,500	$1,500,000	120	2.4 TB
z13	10 TB	1 TB	$500❶	$500,000	141	2.82 TB
z14	32 TB	16 TB	$400	$400,000	170	3.4 TB
z15	40 TB	16 TB	$300	$300,000	190	3.8 TB

❶ Assumes customer upgrade of 300%+ memory from zEC12 to z13

❷ Recommended Minimum 20 GB+ of Real Storage per core (GP, speciality)

The cost of IBM Z Mainframe Real Storage Memory declines:

There was a dramatic reduction in cost & increase in capacity of IBM Mainframe real storage during the zEC12 to z13 server transition. If you're back level from an IBM server viewpoint, if for no other reason, upgrade to a z13+ server to realize this cost & performance benefit. Put simply, cost reduced by a factor of 3, with a similar tripling of maximum memory supported for each CEC. From an IBM sales & marketing viewpoint, to benefit from these memory cost reductions, customers had to triple the amount of real storage deployed at their installation, during IBM Z server upgrades. This wasn't too onerous, as the same amount of money delivered 300%+ of the memory resource!

Moore's Law in action, but not everybody has deep pockets & what about Db2?

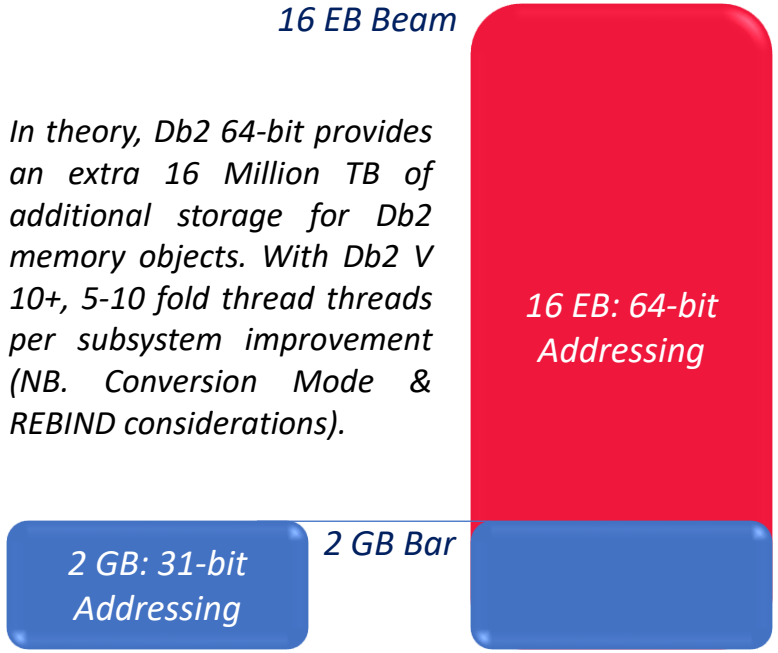
Db2 for z/OS: Recent Architecture Evolution Observations

Db2 Version	Maximum BP (DBM1) Size	z/Architecture 31-bit/64-bit	Support End (EOS)
V8	1 TB	~10% 64-bit	30 Apr 2012
V9	1 TB	~25% 64-bit	27 Jun 2014
V10	16 TB❶	~90% 64-bit	30 Sep 2017
V11	16 TB❶	64-bit❷	31 Mar 2021
V12	16 TB❶	64-bit	TBD

❶ In theory, Db2 V10/V11 can address 16 TB, but practically, only supports 1 TB buffer pools

❷ A 64-bit addressing evolution for Db2 & as of Db2 V11, full (or close) 64-bit data/thread support

In theory, Db2 64-bit provides an extra 16 Million TB of additional storage for Db2 memory objects. With Db2 V 10+, 5-10 fold thread threads per subsystem improvement (NB. Conversion Mode & REBIND considerations).

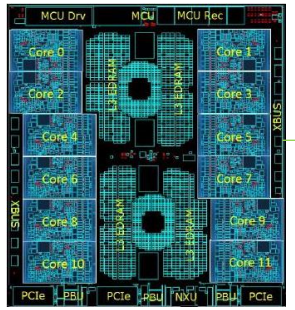


In theory, nearly every IBM Mainframe customer should be running Db2 V11+:

The most basic mechanism for reducing CPU cycles is Data In Memory (DIM), a Mainframe terminology first introduced for the MVS/ESA operating system in 1988, with associated dataspace & hiperspace structures. Accessing data in memory reduces disk I/O & associated CPU cycles, shortening data processing times. Of course, Db2 has always been an early adopter or exploiter of the latest IBM Mainframe In-Memory technologies. As a gradual evolution from Db2 V8-V11, the ability for Db2 to exploit 16 EB (Beam) 64-bit addressing, compared with 2 GB (Bar) 31-bit addressing for data structures was introduced, eradicating Virtual Storage (VSCR) concerns…

All IBM Mainframe customers deploy the latest software & hardware, right!

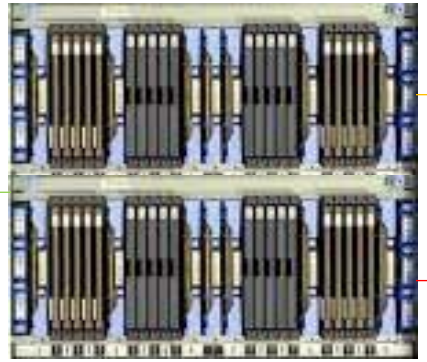
I/O Processing Fundamentals: More Memory, Less CPU (MIPS)



z15 Core: 5.2 GHz
L4 Cache: ~1 GB
Response: ~10 ns
CPC Max: 190 Cores

ICA: Integrated Coupling Adapter

CPC PCIe Fanout



z15 DIMM: DDR4
Capacity: 8 TB
Response: ~400 ns
CPC Max: 160 TB

zHyperLink

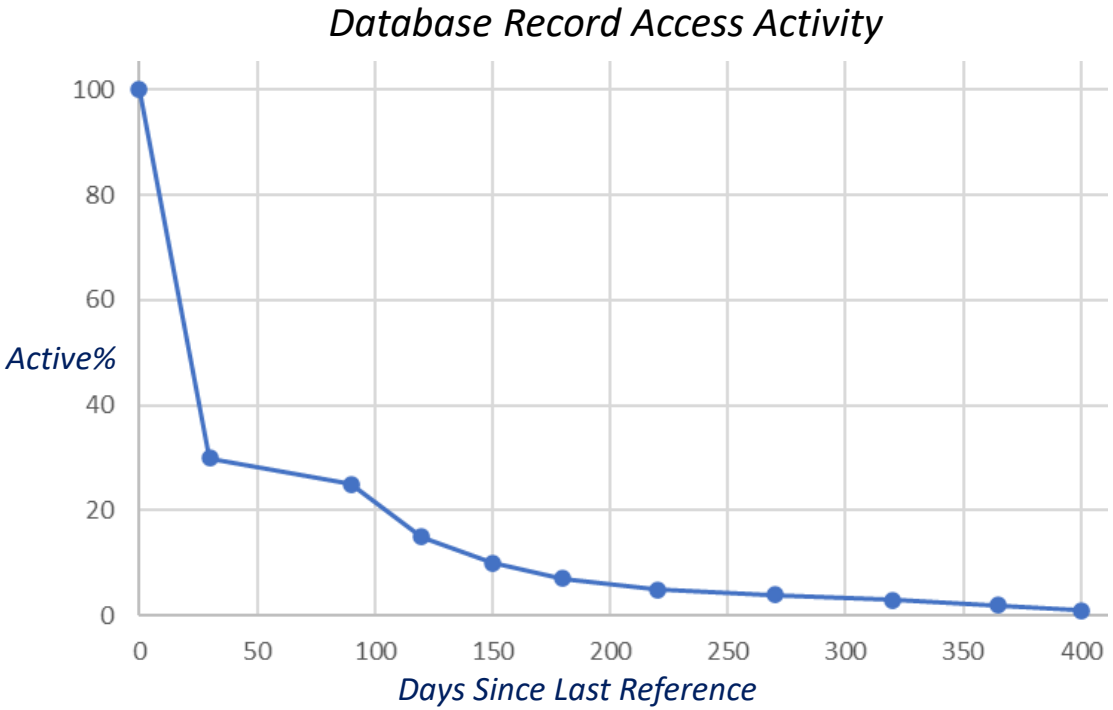
zHPF/FICON

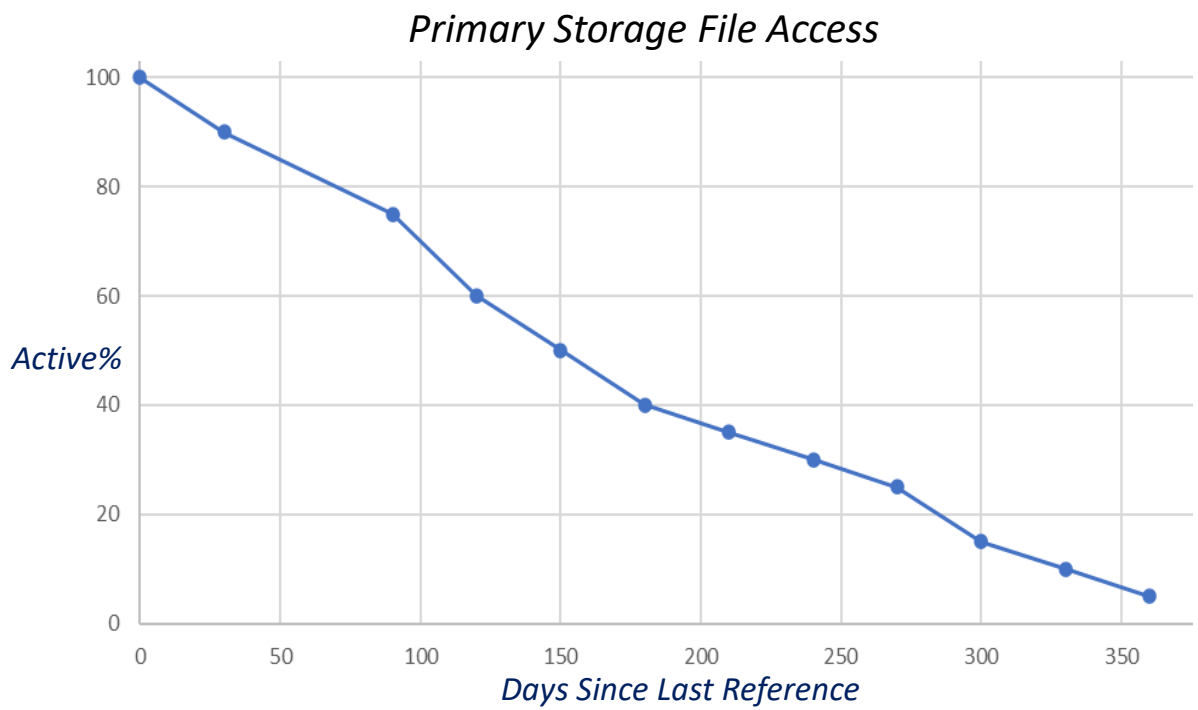
DS8950F: Cache/Flash
Capacity: 3.4 TB/5.9 PB
Response: 20 µs/200 µs
Subsystem Max: 5.9 PB

Urban Myth #1: Many field IBM Z CPCs have large amounts of spare memory & a demand paging rate of zero.
Urban Myth #2: All Flash DASD subsystems are so inexpensive, there's no requirement for physical disk anymore.
Urban Myth #3: The All Flash DS9850F delivers 50%+ reduction in transaction response times for Db2 workloads.
Urban Myth #4: 25-50% of CPC LPAR memory to be allocated for Db2 workloads, without overall system impact.
Urban Myth #5: We don't need to understand our Db2 data, abundant memory & AI will optimize performance…

Deploying the latest & greatest software & hardware helps, but not always…

Data Access Observations: Hours, Days, Months or Years?





When data is first created, it's very active, typically for the first calendar month or so, periodic cycles (E.g. Month, Quarter, Year) then influence subsequent scheduled data access processes. From a Primary DASD or All Flash storage viewpoint, typically 60% of enterprise data hasn't been accessed for 6+ Months. Even when a file or database is accessed, it's doesn't necessarily follow that all records within that data resource are processed. The objective for optimal I/O performance is safeguarding that the right data, is in the right place, at the right time & for the right cost! When data is required, ideally it's accessed from memory storage, minimizing I/O & CPU usage…

Nobody really knows the data access profiles for ever changing Production data!

Db2 Buffer Pools: Multiple Workload Buffer Pool Usage #1

Db2 supports up to 50 buffer pools containing 4 KB pages & up to 10 buffer pools containing 8/16/32 KB pages:

Indexes, data, Large OBjects (LOBs) & work files each have different considerations, partly because pages contain differing amounts of information. Because index keys are small, one index page contains much information & the likelihood of frequent page reference is very high. Rows are larger than keys, each data page contains less information, thus less frequent recurrent page reference. By definition, an LOB is large & no 2 LOBs can occupy the same page & therefore the likelihood of a LOB page being referenced is very small. Pages in a virtual buffer pool can be in-use, updated, or available:

- *In-use: currently being read or updated, data is available for use by other applications.*
- *Updated: changed data not yet written to disk.*
- *Available pages: ready for use; incoming pages of new data can overwrite available pages.*

To avoid disk I/O, updated & available pages containing data can be used. When data in the buffer changes, Db2 does not need to write the data to disk immediately, the data can remain in the buffer pool for other uses. The data remains in the buffer until Db2 needs to use space for another page, applications can read or change the data without a disk I/O operation.

Recommendation: Separate indexes, data & LOBs into separate buffer pools. If LOB pages are unlikely to be referenced, the LOB pages should be assigned to a small buffer pool. Data buffer pools should only be given buffers that are not needed for index buffer pools to achieve a very high index buffer hit ratio. Another reason for separating some page sets into different buffer pools is to assign different buffer pool parameters. Table spaces that are mostly accessed sequentially can be assigned a buffer pool with a 90% or even 99% value for Sequential Steal Threshold (VPSEQT). This avoids significant buffer pool allocation underuse, due to distant time period random getpage requests. It is also useful to assign buffer pools based on update or insert activity. Sequential inserts can devastate the buffer hit ratio for other page sets that share the buffer pool. Therefore, it is useful to isolate page sets used for sequential inserts into a separate buffer pool.

Db2 Buffer Pools: Multiple Workload Buffer Pool Usage #2

Db2 supports up to 50 buffer pools containing 4 KB pages & up to 10 buffer pools containing 8/16/32 KB pages:
Most page sets should use a 4 KB page size, the smallest page size supported by Db2, because a 4 KB page size helps to maximize the random buffer hit ratio.

Recommendation: Separate indexes, data & LOBs into separate buffer pools, considering the associated page size.

Indexes: Using a large page size for indexes minimizes the number of index splits. Index splits cause synchronous log write I/O in a data sharing environment. Compressed indexes can use 8 KB, 16 KB or 32 KB buffer pools. With compressed indexes, more index leaf pages can be stored on disk. The index leaf pages are compressed from either 8 KB, 16 KB or 32 KB to fit into a 4 KB page size. For an 8 KB index page size, the best compression ratio is 2:1; 16 KB index page size, the best compression ratio is 4:1; 32 KB index page size, the best compression ratio is 8:1. 32 KB is probably best…

Table spaces: Using a large page size for table spaces, supports large rows. Since a row cannot span pages, space is often wasted at the end of each page. Typically with variable length row sizes, the amount of wasted space will equate to half a row. For example, with 10 rows per page, expect to waste ~5%. A larger page size that stores more rows reduces the space wastage, but might impact the random buffer hit ratio. Another reason to use a large page size is to maximize the performance of Db2 prefetch & sequential writes. 32 KB is probably best…

Large objects (LOB): Used for columns too large to fit in a row, whose maximum size is 32 KB. When Db2 reads a LOB consisting of a single page, it does a random getpage, which can result in a synchronous I/O. For all subsequent pages in the LOB, getpage operations are classified as sequential & Db2 uses list prefetch. Thus, all but the first LOB page is governed by VPSEQT processing. Because each LOB page is connected with a single row, LOBs tend to be buffer pool unfriendly. A 32 KB LOB size is ideal, maximizing the fitting of an entire LOB in one page, allocated to an efficiently small sized LOB buffer pool, as reference is unlikely. NB. zHPF & modern DASD subsystems support Db2 list prefetch (load pages before application read).

Db2 Buffer Pools: Dedicated Workfile Buffer Pools

Since Db2 V9, increased usage of 32 KB workfile table spaces:

Workfile table spaces, mostly defined to the DSNDB07 database, are used for sort, star join, trigger, created temporary tables, view materialization, nested table expression, merge join, non-correlated subquery, sparse index, temporary tables, et al. For workfile records less than 100 bytes, Db2 prefers 4 KB page table spaces; larger records, Db2 prefers 32 KB page table spaces.

Recommendation: Dedicate Db2 buffer pools to workfile table spaces, 4 KB & 32 KB. Db2 uses prefetch I/O for read data operations in almost all cases, but uses synchronous I/O in certain cases (E.g. Sort). When synchronous read is used, just one page is retrieved at a time. Prefetch is a mechanism for reading a set of pages, usually 32, into the buffer pool with only one asynchronous I/O operation. Sequential Steal Threshold (VPSEQT) is a percentage of the buffer pool that might be occupied by sequentially accessed pages, with a default of 80%. If there is minimal synchronous I/O activity, a higher VPSEQT setting is advised, 90 or 95, because reducing I/O, reduces CPU. Deferred Write Threshold (DWQT) is a percentage of the buffer pool that might be occupied by unavailable pages, including both updated pages & in-use pages, with a default of 30%. Db2 checks this threshold on page update operations. If the percentage of unavailable pages in the buffer pool exceeds the threshold, write operations are scheduled for enough data sets (up to 128 pages per data set) to decrease the number of unavailable buffers to 10% below the threshold. For example, if the threshold is 50%, the number of unavailable buffers is reduced to 40%. Vertical Deferred Write Threshold (VDWQT) is similar to the deferred write threshold, but applies to the number of updated pages for a single page set in the buffer pool, with a default of 5%. If the percentage or number of updated pages for the data set exceeds the threshold, writes are scheduled, up to 128 pages. For workfile buffer pools, setting DWQT of 70-80 & VDWQT of 40-50 is generally OK. Setting these values higher might impact the Data Manager ThresHold (DMTH), triggered when 95% of pool buffers are non-stealable, causing significant disk I/O. Bottom Line: The DMTH hit rate should always be zero (Db2 Monitor Display, Statistics Report, -DIS BPOOL(BPn) DETAIL command).

Db2 Buffer Pools: Real Storage (LPAR) Page Fixing

Since Db2 V11, 2 GB page frames supported, in addition to the pre-existing 1 MB & 4 KB page frames:

Fixing high I/O rate buffer pools in real storage, reduces paging to disk. Buffer pools that have a high number of pages read or written, will benefit from the PGFIX(YES) setting, fixing the Db2 buffer pool in real storage. Otherwise, a buffer pool is fixed in real storage only for the duration of an I/O operation. Conversely, for buffer pools with zero I/O, such as some read-only data or some indexes with a near 100% hit ratio, PGFIX(YES) provides no performance advantage.

Recommendation: Use large-size frames for Db2 buffer pools. In addition to basic 4 KB frames, z/OS supports large-size frames, either 1 MB or 2 GB. Using 1 MB page frames reduces CPU usage by improving the hit ratio of the system Translation Lookaside Buffer (TLB). Typically, 1 MB frames can be non-pageable via PGFIX(YES). Specify large frame support via LFAREA in SYSx.PARMLIB(IEASYSxx) for large 1 MB or 2 GB frame usage. With LFAREA defined, Db2 automatically tries to use 1 MB frames for buffer pools defined with PGFIX(YES). By default, when LFAREA is defined, Db2 tries to use 1 MB frames, overridden by specifying FRAMESIZE(4 K) or FRAMESIZE(2 GB) for the Db2 buffer pool.

2 GB frame sizes require a zEC12+ processor & the buffer pool be non-pageable via PGFIX(YES). Performance advantages of 2 GB frames might be limited for some smaller sites, but it is expected that as memory sizes become much larger, 2 GB frames will be advantageous. For obvious reasons the minimum size of a 2 GB frame buffer pool must be 2 GB & if the buffer pool cannot fully accommodate 2 GB frames, the buffer pool will be utilized by 1 MB &/or 4 KB page frames. Field experience indicates that larger 2 GB frame buffer pools (E.g. 20 GB+) deliver worthwhile CPU usage benefits.

If an installation decides they must use pageable buffer pools via PGFIX(NO) for any reason, use 1 MB frames. However, don't oversize buffer pools. If buffer pools are paging, they're too large, with not enough real LPAR storage to support the buffer pool size. Bottom Line: Try to avoid buffer pool paging at all costs, disk I/O means slower responses & increased CPU!

Db2 Buffer Pools: Real Storage (Db2) Buffer Pool Fixing

In Db2 V12, buffer pool pages are arranged in real storage memory, as if they resided on disk:

With PGSTEAL(NONE), when a Db2 buffer pool object is first accessed, the requesting application process fulfils its data request & Db2 will asynchronously read the remaining objects into the buffer pool via sequential prefetch, arranging the object pages in real storage as per 3390 disk storage. This safeguards efficient page access & reduced CPU overhead, eradicating LRU management. In essence, prefetch will never again be used for that object…

Recommendation: If your installation has a mission critical high use data object, you might want to dictate that the associated page sets remain in memory, eradicating all I/O. Using PGSTEAL(NONE) eradicates the CPU overhead of Least Recently Used (LRU) management & prefetch engine scheduling, which would already find all of the pages in the buffer pool. For the avoidance of doubt, with PGSTEAL(NONE) there will be zero buffer pool steal activity for a pool, because the pool has more buffers than object page objects assigned to the pool.

Db2 automatically creates an overflow area for pages that do not fit in the buffer pool. The overflow area is created when the buffer pool is allocated. The size of the overflow area is based on the VPSIZE value for the buffer pool. The overflow area is typically 10 percent of the VPSIZE value in the range of 50 - 6400 buffers. Page-stealing might occur in the overflow area. Regardless, objects will be allocated in the remaining 90% of the buffer pool, as per 3390 disk storage.

Of course, buffer pool pages can also be managed by traditional LRU & FIFO (First In First Out) mechanisms. LRU is most suited for medium-high levels of buffer pool steal. For example, replacing a buffer resident table or index page, with another page, stealing the buffer with the longest unreferenced time. For low levels, but not zero buffer pool steal activity, FIFO is ideal, avoiding LRU cycles for buffer management. Therefore just stealing the buffer holding the page brought into memory the longest time ago. Bottom Line: For mission critical SLA/KPI related processing, PGSTEAL(NONE) eradicates CPU overhead concerns, optimizing associated response times, but with fixed real storage costs (is all of the data referenced)…

Db2 Virtual Storage Usage: RELEASE(DEALLOCATE)

In Db2 V10 this option was reinstated for distributed packages with generic 64-bit virtual storage usage:

Before Db2 V10, a significant portion of thread related virtual storage was acquired from 31-bit addressing below the 2 GB bar in the Db2 DBM1 address space. From Db2 V10 onwards, using the RELEASE(DEALLOCATE) option reallocated 75-90% of thread related storage to 64-it addressing, above the bar. Additionally threads are no longer allocated to the Environmental Descriptor Manager (EDM) pools, being redirected to 64-bit agent local pools.

In Db2 V10, the RELEASE(DEALLOCATE) option was reinstated for distributed packages with enhancements to use the RELEASE (DEALLOCATE) bind option efficiently, locally or in distributed environments. Performance improvements in Db2 V10 were achieved by avoiding package & section allocations, creation, clean up of table structure control blocks & lock/unlock requests for parent locks.

Using this option allowed DataBase Access Threads (DBAT) to hold package allocation locks while inactive. All of these enhancements deliver CPU reduction for packages bound (BIND, REBIND) with RELEASE(DEALLOCATE) when they contained simple SQL statements with frequent commits. However, packages using many objects did not benefit from these changes or sometimes even degraded compared to packages bound with RELEASE (COMMIT). To provide consistent performance across different types of workloads, the RELEASE(DEALLOCATE) BIND/REBIND option is optimized in Db2 V11.

In Db2 V11, with RELEASE(DEALLOCATE), Db2 monitors the number of locks & table structure blocks held per thread, releasing them at commit once they reach a certain threshold. The default threshold is high enough not to impact existing packages benefited from Db2 V10 enhancements. This ensures consistent performance improvement, irrespective of how many objects are used or how many locks are held by each package.

Recommendation: Use RELEASE(DEALLOCATE) to significantly extend virtual storage addressing from 2 GB (31-bit) to 64 EB (64-bit), with associated CPU reduction benefits (E.g. 3-10%) for an exponentially increased number of active threads.

Db2 for z/OS SQL: Static SQL Observations

The language used to access data in Db2 tables is the Structured Query Language (SQL):

Static SQL statement source is embedded within host language application program (E.g. COBOL). SQL statements are prepared before program execution & the operational form of statements persist beyond program execution. Static SQL source must be processed before program compilation, via the Db2 precompiler or the Db2 coprocessor. The Db2 precompiler or coprocessor checks SQL statement syntax, generating host language statements to invoke Db2 via a DBRM.

When using static SQL, SQL statement form cannot be modified without program changes, but static statements flexibility can be increased by using host variables. For static SQL statements without host variables, Db2 determines the access path during plan or package bind. This combination yields the best performance because the access path is already determined upon program execution. For static SQL without host variables, the time at which Db2 determines the access path depends on the REOPT bind option specified, default of REOPT(NONE) or REOPT(ALWAYS). Db2 ignores REOPT(ONCE) & REOPT(AUTO) for static SQL statements, because Db2 only caches dynamic SQL statements. Db2 determines the access path at bind time, as if there are no input variables via REOPT(NONE). For REOPT(ALWAYS), Db2 determines the access path at bind time & again at run time, using the input variable values from; Host Variables; Parameter Markers; Special Registers.

Db2 spends extra time determining the access path for statements at run time. However if Db2 determines a significantly better access path using variable values, an overall performance improvement may ensue. With REOPT(ALWAYS), Db2 optimizes statements using known literal values, helping Db2 choose a more efficient access path when columns contain skewed data. Db2 also recognizes which partitions qualify if search conditions with host variables on the limit keys of partitioned table spaces exist. Db2 does not start over optimization from the beginning via REOPT(ALWAYS); Db2 does not perform query transformations based on literal values. Static SQL statements using host variables optimized with REOPT(ALWAYS) & similar SQL statements using explicit literal values might result in different access paths.

Db2 for z/OS SQL: Dynamic SQL Observations #1

Before using dynamic SQL, consider the type of dynamic SQL used & any performance implications:

Programs using dynamic SQL can be written in Assembler, C, COBOL, PL/I & REXX. All REXX programs SQL statements are considered dynamic SQL. When a program must use many different SQL statement types & structures of SQL statements & cannot contain a model of each one, the program is suited for dynamic SQL. Dynamic SQL can be executed via the Query Management Facility (QMF), SQL Processor Using File Input (SPUFI) or a UNIX System Services (USS) Db2 Command Line Processor. For dynamic SQL statements, Db2 determines the access path at run time, when the statement is prepared. The repeating cost of dynamic statement preparation can worsen performance when compared with static SQL. If the same SQL statement is processed often, dynamic statement cache decreases the number of times dynamic statements are prepared.

When binding applications containing dynamic SQL statements with input host variables, consider using the REOPT(ALWAYS), REOPT(ONCE) or REOPT(AUTO) bind options, instead of REOPT(NONE). Use REOPT(ALWAYS) when not using the dynamic statement cache. Db2 determines the access path for statements at each EXECUTE or OPEN. This option ensures the best statement access path, but using REOPT(ALWAYS) can increase the cost of frequently used dynamic SQL statements. Consequently, REOPT(ALWAYS) is a bad choice for high-volume sub-second queries. For high-volume fast running queries, the repeating cost of prepare can exceed the execution cost of the statement. Statements that are processed via REOPT(ALWAYS) option are excluded from dynamic statement cache even if when enabled because Db2 cannot reuse access paths.

With REOPT(ONCE), Db2 only determines the access path for statements at the first EXECUTE or OPEN, saving that access path in dynamic statement cache, until the statement is invalidated or removed from the cache. This reuse of the access path reduces the prepare cost of frequently used dynamic SQL statements containing input host variables. REOPT(ONCE) option is ideal for ad-hoc query applications (E.g. SPUFI, DSNTEP2/4, DSNTIAU, QMF). Db2 optimize statements knowing the literal values for special registers (E.g. CURRENT DATE/TIMESTAMP), rather than using default filter factor estimates.

Db2 for z/OS SQL: Dynamic SQL Observations #2

Before using dynamic SQL, consider the type of dynamic SQL used & any performance implications:

When binding applications containing dynamic SQL statements with input host variables, consider using the REOPT(ALWAYS), REOPT(ONCE) or REOPT(AUTO) bind options, instead of REOPT(NONE). With REOPT(AUTO), Db2 determines the access path at run time. For each execution of a statement with parameter markers, Db2 generates a new access path if it is likely to improve performance.

Code PREPARE statements to minimize overhead. With REOPT(AUTO), REOPT(ALWAYS), & REOPT(ONCE), Db2 prepares an SQL statement at the same time as it processes statement OPEN or EXECUTE, processes the statement as if DEFER(PREPARE) was specified. However, Db2 prepares the statement twice in the following situations:

- *Issue the DESCRIBE before the OPEN statement*
- *Issue the PREPARE statement with the INTO parameter*

For the first prepare, Db2 determines the access path without using input variable values. For the second prepare, Db2 uses the input variable values to determine the access path. This extra prepare can decrease performance. With REOPT(ALWAYS), Db2 prepares the statement twice for each run. With REOPT(ONCE), Db2 prepares the statement twice, only when the statement has never been saved in cache. If the statement has been prepared & saved in cache, Db2 uses the saved statement version for DESCRIBE completion. With REOPT(AUTO), Db2 initially prepares the statement without using input variable values. If the statement has been saved in cache, for subsequent OPEN or EXECUTE, Db2 determines if a new access path is needed according to the input variable values.

For SQL statements using a cursor, avoid the double prepare overhead by placing program DESCRIBE statements after OPEN statements.

Db2 for z/OS SQL: Dynamic Statement Cache Observations

Improving dynamic SQL performance by enabling the dynamic statement cache:

The Dynamic Statement Cache (DSC) is a pool where Db2 retains control structures for prepared SQL statements that can be shared among different threads, plans & packages. By sharing these control structures, applications can avoid unnecessary preparation processes & improve performance.

Specifying YES for the CACHEDYN subsystem parameter causes prepared, dynamic SQL statements to be cached in the EDM dynamic statement cache, requiring EDM pool size review accordingly. YES must also be specified for the USE PROTECTION field on panel DSNTIPP. As the Db2 ability to optimize SQL has improved, the overhead of preparing dynamic SQL statements has grown. Applications using dynamic SQL might be forced to pay this cost more than once. When an application performs a commit operation, it must issue another PREPARE statement, if it is to be executed again. For a SELECT statement, the ability to declare a cursor WITH HOLD provides some relief, but requires that the cursor be open at the commit point. Db2 can save prepared dynamic statements in cache; the dynamic statement cache pool that all application processes can use to save & retrieve prepared dynamic statements. After an SQL statement has been prepared & saved in the cache, subsequent prepare requests for the same SQL statement can avoid costly preparation processes, reusing the cached statement, which can be shared among different threads, plans or packages. If a thread attempts to prepare a dynamic SQL statement, Db2 tries to find the cached prepared version of that statement in the Global DSC (GDSC). Db2 employs comparison criteria to determine if there is a match, where 2 statements are considered a match when both of the following criteria are satisfied:

1. They are identical character-for-character (including blanks).
2. They have matching AUTHID, BIND options & special registers.

Db2 provides a concept called Literal Replacement, where white space is removed from any cached dynamic SQL statement being prepared. This Literal Replacement increases the likelihood of finding a cache match, enabled at the application level.

Db2 for z/OS SQL: Static versus Dynamic SQL Observations

Db2 V12 introduces dynamic SQL plan stability, bringing the plan stability feature of static SQL to dynamic SQL:

With dynamic SQL plan stability enabled, Db2 stores statement cache structures for specified dynamic SQL statements in the Db2 catalog. When issued, if a stabilized dynamic SQL statement is not present in the dynamic statement cache, Db2 can load the statement cache structures from the Db2 catalog, avoiding the full prepare operation. The goal is to achieve access path stability comparable to static SQL statements for repeating cached dynamic SQL statements.

Dynamic SQL statements are more susceptible to access path regressions than static SQL statements. Db2 prepares the access path for static SQL statements when packages are bound using the same access path until the next package BIND or REBIND operation. For dynamic SQL statements, Db2 must use the full prepare process for any dynamic SQL statement that is not found in the dynamic statement cache.

Stabilizing dynamic SQL statements requires consideration. Access path changes often improve performance, these potential performance improvements will be compromised for stability. Stabilized dynamic SQL statements also use Db2 catalog space to store the run time structures. The following Db2 subsystem events can invalidate or remove dynamic statement cache entries, once again exposing the dynamic SQL statements to the full prepare process:

- *Db2 subsystem recycle (stop & restart)*
- *Statements exit & re-enter the dynamic statement cache*
- *Statistics are collected by RUNSTATS or other data change utilities*
- *Db2 subsystem parameters change*
- *Db2 subsystem maintenance*
- *Db2 release version upgrade/migration*

Db2 for z/OS: Fast Traverse Blocks (FTB) Observations

Db2 V12 introduces Fast Traverse Blocks (FTB) for random index access performance improvement:

Db2 continuously monitors activity for indexes that are eligible for fast index traversal (AKA FTB), performing fast index traversal on eligible indexes when the indexes exceed internally defined activity levels. Db2 supports fast index traversal for indexes that are defined as UNIQUE with an index key size of 64 bytes or less. To benefit from fast index traversal, data access must use a random pattern. However, SELECT, INSERT, DELETE & UPDATE statements, or any operation requiring index traversal can benefit from FTB. Indexes with frequent index traversals are good candidates; indexes with frequent index leaf-page splits are bad candidates.

Fast index traversal is supported by a separate Fast Traverse Block (FTB) memory area, independent of buffer pools. This memory area uses a concatenated structure containing copies of non-leaf index pages only. Fast index traversal does not use Db2 buffer pools, the non-leaf index pages (excepting the root page) cached in FTB memory area are not fixed in the buffer pool. Index pages in the buffer pool are eligible for stealing & can be removed from the buffer pool, on a Least Recently Used LRU) basis, when the non-leaf pages are stored in the FTB memory area. Performance benefits ensue because the FTB memory area is an L2 cache-aware B-Tree like data structure & each page is equal in size to one cache line (I.E. 256 bytes).

The FTB memory area size is controlled by the INDEX_MEMORY_CONTROL subsystem parameter. An AUTO default setting means the FTB memory area size is allocated at 20% of the total memory allocated for all buffer pools, with a minimum size of 10 MB. FTB memory area size changes as the size of any buffer pool changes, every 10 minutes. Optionally the FTB memory area can be sized manually, up to a maximum size of 200 GB. A zIIP eligible background daemon running for each Db2 subsystem determines which indexes are good candidates for the FTB memory area. To qualify, indexes must be UNIQUE; INCLUDE columns are also supported. The index entry length, including the key & any additional columns, can have a maximum size of 64 bytes. The daemon evaluates & adjusts its priority queue every two minutes.

Db2 for z/OS: CPU Optimization via Memory Usage Reprise

Monitor z/OS Memory Usage: Safeguard a low z/OS demand paging rate (DPAGRT); 0 (ideal, no paging); 1-2 (maybe OK...); 3+ (too high?). In a big memory world, physical disk paging is best avoided. Constantly review Db2 real storage usage.

Monitor Db2 Buffer Pool Sizes: Safeguard read I/O rate (synchronous + prefetch) is balanced for all Db2 buffer pools. If two equally sized buffer pools, have a read rate of 50 for one & 5,000 for the other, allocate increased buffers to the higher I/O rate pool, from memory allocated to the other. Obviously any buffer pool size adjustment needs to be meaningful…

Monitor Db2 Buffer I/O Rates #1: Set a KPI for Db2 buffer pool I/O rates. Anything higher than 1,000 suggests memory constraints; around 100 could be considered OK; many performance critical sites aim for 10 or less. Ideally, avoiding I/O means less CPU, the utopia is 0. Obviously for PGSTEAL(NONE) buffer pools, the target read I/O rate target is 0 (in theory).

Monitor Db2 Buffer I/O Rates #2: For buffer pools with a high read/write I/O page rate, use PGFIX(YES) to fix the buffer pool in real storage, eradicating I/O. For Db2 V11+, safeguard Db2 exploits page frames with a larger 1 MB or 2 GB FRAMESIZE.

Monitor Dynamic Statement Cache (DSC) Usage: Ultimately DSC is a finite memory structure, with two areas; local (DBM1) managed on a FIFO basis, with limited performance improvement; global (64-bit EDM pool), managed on a LRU basis, with good performance improvement possibilities. DSC can be flooded by poor SQL coding (E.g. literal based INSERT), which can flush all other good candidates (I.E. DELETE, MERGE, SELECT, UPDATE) very quickly.

Monitor Fast Traverse Block (FTB) Usage: FTB delivers higher performance, because the allocated memory structure is L2 cache-aware & each page is equal in size to one cache line (I.E. 256 bytes). FTB ineligibility can easily be introduced with well intentioned BAU SQL changes; 1) Index cannot be longer than 64 bytes; 2) Index cannot be versioned (OLDEST_VERSION & CURRENT_VERSION identical); 3) Index contains a TIMESTAMP column with TIMEZONE; 4) Index exceeds maximum supported limit of 2,000,000 leaf pages.

Db2 for z/OS: General Purpose CPU Offload via zIIP

Certain Db2 processing is eligible for IBM Z Integrated Information Processor (zIIP) specialty engine dispatch:

DDF Originated SQL Processing: Up to 60% of the Db2 for z/OS instructions associated with SQL statements initiated from DRDA (Distributed Relational Database Architecture) requesters initiating Enclave Mode SRBs in the DDF (Distributed Data Facility) address space are eligible for zIIP offload. Support was also introduced for native SQL PL (Procedure Language) routines with Dd2 V9. A native SQL procedure executes under the task of the application process that calls it. When the calling process is a DDF connected application, the task is an enclave SRB in the DDF address space & therefore eligible zIIP offload. Converting external Db2 stored procedures called by DRDA requesters to native SQL procedures eligible for zIIP offload is a plausible & pragmatic option for reducing General Purpose (GP) CPU usage.

Parallel Query Child Processes: Including BI application query processing utilizing Db2 star schema parallel access query capabilities & other queries where access has been split into multiple subtasks (CP parallelism) by the optimizer. Up to 80% of child task processing will be zIIP redirected after a specific CPU usage threshold (IBM defined, IBM Z server unique) has been exceeded for each parallel group. Because most BI package queries arrive via TCP/IP using DRDA, they are already eligible to use zIIP. When those queries use parallel star joins, the star join portion is also eligible for zIIP offload.

Db2 Utility Processes: Up to 100% of the portion of LOAD, REORG & REBUILD INDEX utility function used to maintain index structures. Portions of RUNSTATS processing, including column group distributed statistics processing.

XML Processing: Up to 100% of XML schema validation, non-validation parsing & XML obsolete document version deletion.

Db2 System Agents Processing: Up to 100% of processing for Db2 system agents running Enclave Mode SRBs in the MSTR, DBM1 & DIST address spaces, excepting P-Lock (Data Sharing Physical Lock) negotiation processing activities.

NB. zIIP engines require management; safeguard sufficient zIIP weight is defined to Db2 for z/OS LPARs & a minimum of 1 Vertical High logical zIIP engine assigned. zIIP engines perform badly @ 75%+, 50-70% busy is considered good utilization…

Db2 for z/OS CPU Usage: Rely on Moore or Do More With Less?

IBM Z servers continue to evolve, delivering larger & faster memory resources for each new generation, where Moore's Law just about applies, every 2-3 years an increase in speed & capacity for a lower cost. Architectural evolutions deliver functionality that also assists Db2 processing (E.g. IBM Z Integrated Information Processor (zIIP) Engines, Large Page Frames, zEnterprise Data Compression (zEDC), 64-bit Addressing). Some thought, evolution & migration activity is required to leverage from these hardware enhancements.

The Db2 subsystem continues to evolve, with a new version release, every 3 years or so. Latterly major enhancements enable the big data, security & agile development requirements of digital transformation initiatives. Db2 for z/OS V12 included new features such as SQL Pagination Syntax, Increased Partition & Table Sizes, Fast Traverse Blocks (FTB), Insert AlGorithm 2 (IAG2), Dynamic SQL Plan Stability. Some thought, evolution & migration activity is required to leverage from these software enhancements.

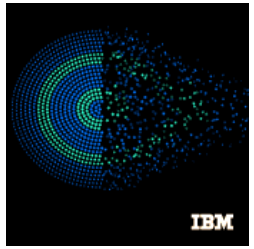
The IBM Z Subject Matter Expert (SME) is classified as such, because they can do more with less. Exploiting the latest hardware & software resources available is always advantageous, but if we look after the bits, the bytes look after themselves! Optimal data processing only occurs when I/O is minimized, if not eradicated, safeguarding the right data, is in the right place, at the right time, for the right cost. Allocation of buffer & cache memory structures various is only the start, manging their size & contents safeguards optimal CPU usage.

Db2 for z/OS: Other ISV High Performance In Memory Options

Even when considering all of the inbuilt memory structures provided by Db2 for in memory processing, maybe there are opportunities for supplementary data in memory functions? In an ideal world, as Db2 data is processed, an intelligent automated daemon (STC/Started Task) would analyse this data, using an advanced algorithm, loading this data into a simplified highly parallel cache memory structure. This cache structure would only maintain active data subsets, not entire table spaces, delivering ultra fast I/O to applications, simplifying the complexities of Db2 data path access. This self managed cache structure would be sized optimally, not requiring large memory resource & eradicate any notion of having to know the data being processed or pre defining Db2 table spaces for eligibility:

Function Description	*QuickSelect for Db2*	*IBM IZTA/DataKinetics tableBASE*
z/OS & Db2 Version Support	All supported z/OS & Db2 versions.	All supported z/OS & Db2 versions.
Cache Management	Only active data records stored, not entire table, threshold processing for smaller (E.g. <10 GB) cache size.	Active data copied from Db2 DBMS into high-performance in-memory tables, accessed using API.
64-bit z/Architecture Support	Yes: 64-bit application, 16 EB addressing.	No: 31-bit application, 2 GB addressing.
OLTP & Batch Support	Yes.	Yes.
In-Built Simulation Analysis	Yes.	No: use standard Db2 reporting or monitors.
Db2 Operating Modes	z/OS Parallel Sysplex, XCF & Db2 Data Sharing.	z/OS Parallel Sysplex, XCF & Db2 Data Sharing.
Performance: Db2 Buffer Pool Comparison	Considerably faster than Db2 buffer pools, even when data is pinned in real storage (PGFIX).	Considerably faster than Db2 buffer pools, even when data is pinned in real storage (PGFIX).
Transparent Implementation	Yes: No application changes required, source code, load module or Db2 package.	No: Application changes (API) required via table services (TS) function for Db2 table space entities.
Data Invalidation	Yes: Invalidates cached result sets upon detection of changes to underlying tables in real-time.	Yes: Data cache is flushed upon detection of changes to underlying tables in near real-time.
Automatic Db2 Table Space Data Load	Yes: caches frequently requested Db2 SQL result sets, only active data, not the entire table space.	Semi: Eligible table spaces need to be pre-defined, only active data, not the entire table space.

Db2 for z/OS: SQL CPU Optimization Software Solution Options



IBM Db2 AI for z/OS leverages from IBM machine learning for z/OS technology, enabling rapid model learning specific to the Db2 data & configuration per LPAR, without requiring data science skills. Db2 AI for z/OS exploits new intelligence to identify optimal access paths for SQL queries, uniquely based on workload characteristics.

CA SQL Performance Suite for Db2 for z/OS helps you quickly identify poor running SQL statements, recommending changes based on expert rules. The access path, SQL performance & Db2 object usage history are stored for future trending & analysis. Application change control procedures automatically detect & prevent inefficient SQL Production promotion.

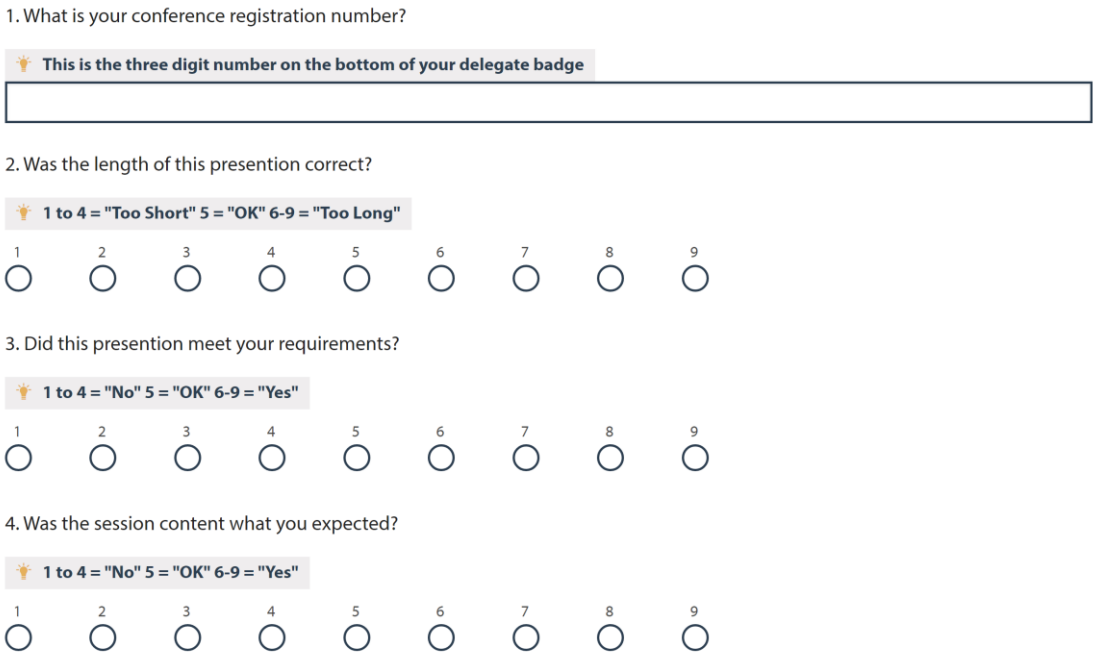
SOFTWARE ENGINEERING SQL WorkloadExpert for Db2 z/OS uses an Intelligent Catcher Facility to create a sturdy workload warehouse environment as the basis for further analysis & reporting. Logging all static & dynamic SQL statements, even statements which have been flushed. Dynamically aggregate & EXPLAIN the data.

BMC AMI SQL Performance for Db2 quickly eliminates wasteful SQL statements, managing performance throughout the application lifecycle & audits privileged access to reduce risk. Rapidly diagnose performance problems, track them to their source, tune SQL to anticipate & resolve slowdowns, eradicating bad SQL & bad access paths.

NB. From previous IBM Z site experience, this is a limited selection of 3rd party ISV products available for Db2 z/OS SQL performance optimization & inclusion within this presentation neither represents endorsement or promotion.

Please submit your session feedback!

- Do it online at http://conferences.gse.org.uk/2020/feedback/1AW
- This session is 1AW



GSE UK Conference 2020 Charity

- The GSE UK Region team hope that you find this presentation & others that follow useful & help to expand your knowledge of z Systems.
- Please consider showing your appreciation by kindly donating a small sum to our charity this year, NHS Charities Together. Follow the link below or scan the QR Code:

http://uk.virginmoneygiving.com/GuideShareEuropeUKRegion