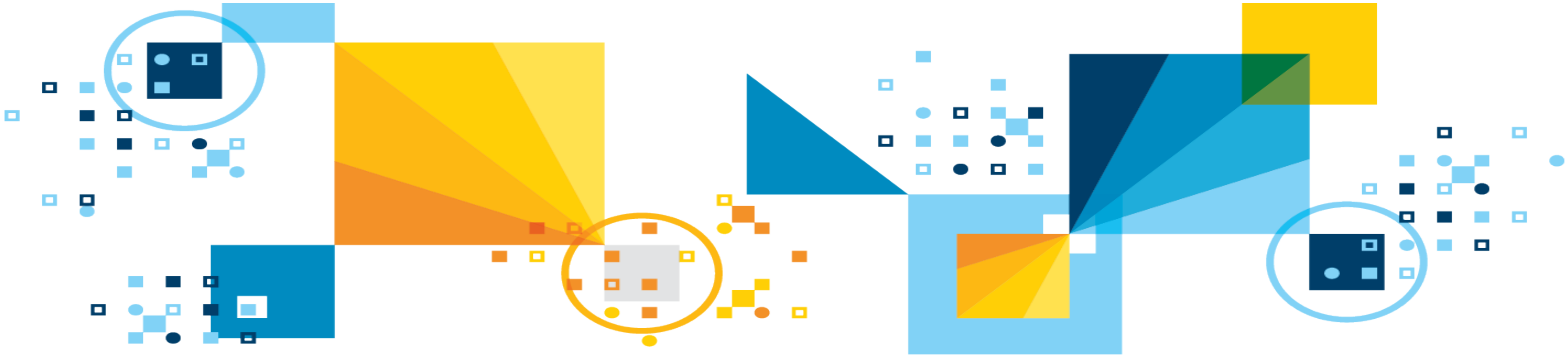
TRIDEX
October 5, 2017

Let Me Make THIS Clear – More Oft-Misunderstood DB2 for z/OS Concepts and Facilities

Robert Catterall
IBM Senior Consulting Db2 for z/OS Specialist



Agenda

- Base and archive tables
- Archive tables and the DB2 Analytics Accelerator
- Data sets: “hard close” and “soft close”
- High-performance DBATs and PKGREL(BNDOPT)
- KEEPDYNAMIC(YES) and DBATs
- Reusing pooled DBATs
- PGFIX and PGSTEAL
- PGFIX(YES) and large page frames
- 2 GB page frames
- Prefetch reads
- DB2 arrays
- Global variables
- Extents

DB2-managed archiving: base and archive tables

- DB2-managed archiving (sometimes referred to as “transparent archiving”) was introduced with DB2 11 for z/OS
  - This feature makes it easy to concentrate “popular” rows (i.e., frequently-accessed rows; often, recently inserted rows) in a base table for enhanced access efficiency, while storing “older and colder” rows in an associated archive table, while presenting to applications a single logical table
- *Some people think that an archive table has to be identical to its associated base table, in every way*
- In fact, what is required is that a base table and its archive table are <u>logically</u> identical
  - Same column names, in the same order, with the same data type
- Can a base table and its archive table be physically different?

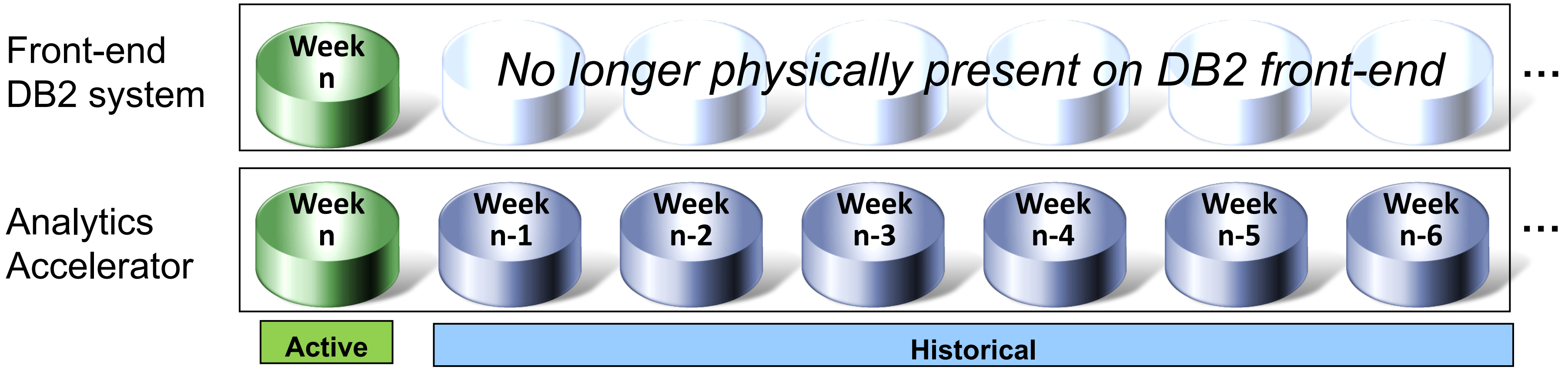
Implications of archive table "physical design freedom"

- Maybe a base table is medium sized, while its archive table is very large
  - If the base table is in, say, a traditional segmented table space, could the associated archive table be in a partitioned table space?
  - **Sure!** And, it could be a partition-by-range or a partition-by-growth table space – doesn't matter!
    - Table spaces have to do with physical database design – all you need is logical equivalence
  - If I have, say, 8 indexes on the base table (for query performance purposes), do I have to have the same indexes on the associated archive table?
  - **No!** Maybe it is understood that queries targeting older rows (stored in an archive table) will not perform as well as queries targeting newer rows (in a base table)
    - If that's the case, reduce the number of indexes defined on an archive table (relative to the number of indexes defined on the associated base table) – save disk space, reduce INSERT/DELETE overhead, and make utility execution more CPU-efficient

Gotta love physical design flexibility!

Archive tables and the IBM DB2 Analytics Accelerator (IDAA)

- *Some people think that DB2 archive tables cannot be used with the DB2 Analytics Accelerator's High-Performance Storage Saver (HPSS) feature*
- Background: if you have a table partitioned on a time-period basis (e.g., 1 week per partition), and if data in partitions other than "current" partition (e.g., other than this week's partition) is read-only, you can leverage High-Performance Storage Saver
  - The "current" partition exists physically on the front-end DB2 system and on the Accelerator
  - <u>All other partitions</u> exist physically ONLY on the Accelerator (they exist <u>logically</u> on front-end DB2)



Why people think that transparent archiving and HPSS don't mix

- One reason: they think that it can't be used for a base table that is anything other than range-partitioned on a time period basis
  - They believe that a base table and its archive table have to be identical in every way

WRONG (see preceding slide)

- Another reason: for combo (transparent archiving + HPSS) to work, archive table has to be partitioned on ROW CHANGE TIMESTAMP column (such a column has to be added to base table, if not already present), and people think you can't do that
  - Actually, you couldn't do that until quite recently
  - APARs PI63830 (DB2 11) and PI66827 (DB2 12) provided the ability to partition a table on a ROW CHANGE TIMESTAMP
- So you can, in fact, have DB2 manage archiving for you, while using the DB2 Analytics Accelerator's High-Performance Storage Saver to enable cost-effective, high-performance querying of archived rows

“Hard” and “soft” closing of DB2 data sets

- *There is some confusion out there as to what these terms mean, and about the effect of CLOSE YES and CLOSE NO for a table space or index*
- “Hard close” is sometimes used to describe physical closing of a DB2 table space or index data set (could be associated with a partition of a table space or index)
  - Aside from shutdown of DB2 subsystem, what would cause DB2 data set to be physically closed?
    - In any DB2 environment, when number of data sets open and allocated to DB2 subsystem reaches limit specified via DSMAX parameter in ZPARM, DB2 will physically close some of the open data sets
    - *In a DB2 data sharing system,* data set physical close actions can also be triggered by “soft close” activity

Next slide

“Soft close” of data sets explained

- “Soft close” is another term for *pseudo-close*
  - Pseudo-close occurs when a DB2 data set that is open for read/write access goes for a pseudo-close interval of time without any update activity
  - The pseudo-close interval is determined by two ZPARM parameters: PCLOSEN (a number of DB2 checkpoints) and PCLOSET (a number of minutes) – the default value for both is 10
    - Whichever occurs first since the last update of a data set – PCLOSEN checkpoints or PCLOSET minutes – will trigger pseudo-close of the data set
  - When data set is pseudo-closed, its <u>state</u> is changed from read/write to read-only, and this is recorded in SYSLGRNX table in DB2 directory (next data-change action affecting a pseudo-closed data set will change its state back to read/write – that will also be recorded in SYSLGRNX)
  - Primary purpose of pseudo-close is to speed up table space and/or index recovery operations
    - When a recovery operation involves log-apply processing, the RECOVER utility can use SYSLGRNX information to skip over portions of the log covering time periods during which the object being recovered was in a read-only state

What about CLOSE YES and CLOSE NO?

- CLOSE YES/NO specification for table space/index affects "hard close" processing, but not "soft close" (though in data sharing group, soft close can lead to hard close)
  - When DSMAX reached, DB2 closes some data sets, *starting with those associated with CLOSE YES objects* (CLOSE YES data sets that have gone the longest without being referenced are closed first)
    - Thus, CLOSE YES/NO lets you influence which data sets will be physically closed when DSMAX is reached
    - That said, my preference is to set DSMAX high enough so that it is rarely, if ever, reached
- DB2 data sharing: when data set in which there is inter-DB2 read/write interest gets pseudo-closed on DB2 member, and then goes another pseudo-close interval with no access at all from that member, it will be physically closed if it a CLOSE YES data set
  - That physical close on member A could result in member B getting an exclusive page set P-lock on the data set – that would reduce data sharing overhead for member B
  - Flip side: if CLOSE YES widely used and there is a lot of pseudo-close activity, data sets could be frequently going into and out of group buffer pool dependency – that would increase overhead
  - CLOSE YES could be good for objects that regularly end up being accessed from only one member

High-performance DBATs and PKGREL(BNDOPT)

- *Some folks have RELEASE(DEALLOCATE) packages that are executed by way of DBATs (DDF threads), and they don't see any high-performance DBAT activity – why?*
- First of all, what do I mean by "don't see any high-performance DBAT activity?"
  - Check a DB2 monitor statistics long report (or online display), in GLOBAL DDF ACTIVITY section:

```
GLOBAL DDF ACTIVITY            QUANTITY
---------------------------    --------
HWM ACTIVE DBATS-BND DEALLC        0.00
```

- Question: Why no high-performance DBATs?
  - Answer: DDF is not enabled for high-performance DBATs

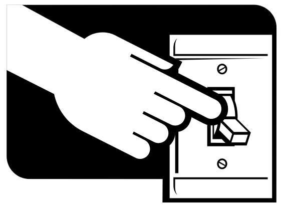
Enabling DDF for high-performance DBATs

- How do you know if DDF is enabled for high-performance DBATs?
  - Issue the DB2 command -DISPLAY DDF DETAIL, and check the output:

```
DSNL106I PKGREL = COMMIT
```

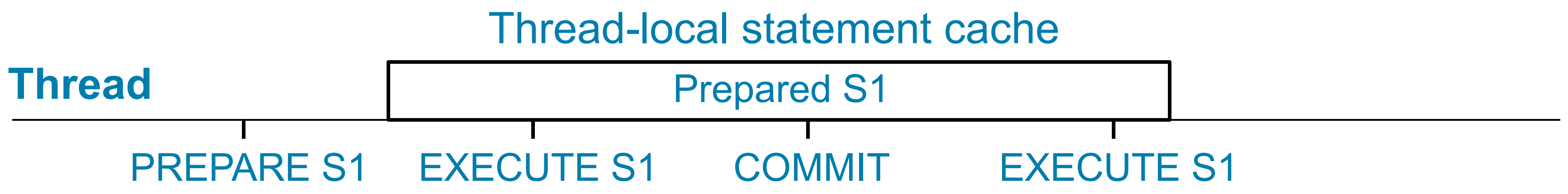
$\xleftarrow{}$ DDF is not enabled for high-performance DBATs

  - To change the situation, issue the DB2 command -MODIFY DDF PKGREL(BNDOPT)
  - Note that it might be necessary sometimes to “turn off” high-performance DBATs to accomplish some database administration tasks:
    - BIND REPLACE or REBIND of RELEASE(DEALLOCATE) package allocated to high-performance DBAT, or - ALTER or pending DDL-materializing online REORG that would invalidate such a package
  - To temporarily turn off high-performance DBATs, issue command -MODIFY DDF PKGREL(COMMIT)
    - Frees up even RELEASE(DEALLOCATE) packages allocated to high-perf DBATs that are not in-DB2
  - Turn high-performance DBATs back on with the command -MODIFY DDF PKGREL(BNDOPT)



KEEPDYNAMIC(YES) and DBATs

- *Some people are unaware of the negative effects of the KEEPDYNAMIC(YES) package bind specification in a DBAT context*
- First, the good: with KEEPDYNAMIC(YES) in effect, two performance-boosting benefits are enabled:
  - Prepared dynamic SQL statements do not have to be re-prepared by a process after commit (to take full advantage of this benefit, *application has to be coded with KEEPDYNAMIC(YES) in mind*)
  - Dynamic SQL statements are kept in their prepared form in a local-to-the-thread cache
    - Getting a "hit" in this local cache (resulting in *PREPARE avoidance*) delivers even more CPU savings than a hit in the global dynamic statement cache (resulting in a *"short PREPARE"*)



KEEPDYNAMIC(YES) and DBATs – the flip side

- A “regular” DBAT (versus a high-performance DBAT) cannot go into a disconnected state (i.e., cannot separate from a connection and go into the DBAT pool) following transaction completion, *if anything is allocated to the DBAT*
- KEEPDYNAMIC(YES): locally-cached prepared statements are allocated to DBAT
  - With the DBAT unable to go into the DBAT pool, it is more likely to hit the idle thread timeout limit than would be the case with KEEPDYNAMIC(NO)
    - Pooled DBATs are not subject to idle thread timeout
  - Also because KEEPDYNAMIC(YES) can keep DBATs from going into a disconnected state, it can reduce the dynamism of Sysplex workload balancing in a DB2 data sharing environment
    - Transaction-level workload balancing depends on DBATs becoming disconnected at transaction completion
- My take: unless application is specifically designed to exploit KEEPDYNAMIC(YES), use KEEPDYNAMIC(NO) together with high-performance DBATs
  - You’ll still get hits in the global dynamic statement cache

Reusing pooled DBATs

- *How can I monitor reuse of pooled DBATs?*
- Look at a DB2 monitor statistics long report (or an online display), in the GLOBAL DDF ACTIVITY section:

```
GLOBAL DDF ACTIVITY               QUANTITY
---------------------------      ---------
DBATS CREATED                       241.98
DISCON (POOL) DBATS REUSED         1399.0K
```

DB2 needed a DBAT 1,399,242 times during the reporting interval, and needed to create a DBAT 242 times – that's a 99.98% rate of thread reuse

- If the rate of pool thread reuse is less than you'd like it to be, consider increasing the value of POOLINAC in ZPARM
  - That's the amount of time a DBAT can sit there in the pool without being reused (default is 120 seconds) – the DBAT will be terminated at that point
  - Perhaps DBATs in your pool are often terminated just before they would have been reused – in that case a modest increase in POOLINAC might significantly boost re-use of pooled DBATs

The PGFIX and PGSTEAL buffer pool configuration settings

- *Some folks think that dependencies exist between PGSTEAL and PGFIX*
- In fact, PGSTEAL and PGFIX are independent of each other – setting one does not affect behavior pertaining to the other
- Key difference: PGSTEAL has to do with DB2 management of its buffer resources (a *virtual storage* thing), while PGFIX has to do with z/OS management of *real storage*
  - DB2's management of buffers in a pool, as indicated by the PGSTEAL setting (LRU, FIFO, or NONE), is NOT affected by a pool's PGFIX attribute (YES or NO)
  - Similarly, z/OS's management of real storage resource used by a buffer pool, affected by pool's PGFIX setting, is NOT affected by the way in which DB2 manages the pool's buffers (PGSTEAL)
- Both attributes having "PG" (short for "page") in the name may be what leads some people to think that PGSTEAL and PGFIX have to do with the same thing
  - In hindsight, things might have been made more clear if PGSTEAL had instead been labeled BSTEAL, indicating that its about buffers that hold pages, versus page frames that hold buffers

More on PGSTEAL

- Regardless of the PGSTEAL setting, DB2 will always steal a buffer in a pool whenever it has to – and it has to if all buffers are occupied by table space and/or index pages, and a new page has to be read into memory
  - PGSTEAL(NONE): “DB2, you should not have to do buffer stealing for this pool, because the pool should have more buffers than there are pages belonging to objects assigned to the pool. Help me maximize performance for those objects, and if you do have to steal a buffer, use FIFO algorithm.”
    - FIFO = first in, first out
  - PGSTEAL(FIFO): “DB2, you will likely have to do a little buffer stealing for this pool. Don’t waste CPU cycles keeping track of which buffer has gone longest time without being accessed. When you have to steal, take buffer holding page that has been in memory the longest time.”
  - PGSTEAL(LRU): “DB2, your are likely to have to do a lot of buffer stealing for this pool. When you steal a buffer, take the one holding the page that’s gone the longest time without being accessed.”

More on PGFIX

- PGFIX(YES): "z/OS, this pool's buffers are fixed in memory – they cannot be paged out to auxiliary storage. When you need to steal a page frame so that something new can be read into memory, look elsewhere."
- PGFIX(YES) makes I/Os more CPU-efficient (no need to fix a buffer in memory before the I/O, and release it after the I/O, because it is <u>already</u> fixed in memory)
  - The higher the level of I/O activity for a pool, the greater the CPU savings (applies to reads and writes to/from group buffer pools in data sharing system, in addition to disk reads and writes)
- Enables use of large page frames for a pool
  - More CPU savings, due to more-efficient translation of virtual storage addresses to real storage addresses
  - The more "active" a pool is (i.e., the higher the level of GETPAGE activity), the greater the CPU benefit of more-efficient address translation

PGFIX(YES) and large page frames

- *My PGFIX(YES) buffer pools are not being backed by large page frames – why?*
- First, how do you know if a PGFIX(YES) buffer pool is backed by large page frames?
  – Issue DB2 command -DISPLAY BUFFERPOOL(ACTIVE) DETAIL, and check output for each pool

```
DSNB402I  *DBP1 BUFFER POOL SIZE = 250000 BUFFERS
DSNB406I  *DBP1 PGFIX ATTRIBUTE -
              CURRENT = YES
              PENDING = YES
DSNB546I  *DBP1 PREFERRED FRAME SIZE 1M
         0 BUFFERS USING 1M FRAME SIZE ALLOCATED
DSNB546I  *DBP1 PREFERRED FRAME SIZE 1M
         250000 BUFFERS USING 4K FRAME SIZE ALLOCATED
```

PGFIX(YES) is in effect for this pool

1 MB is the preferred page frame size for a PGFIX(YES) buffer pool

NONE of this pool's buffers are using the preferred 1 MB page frame size – ALL of the buffers are in 4 KB frames

- Why are 1 MB page frames, though preferred, not used for this pool?

To use large page frames, you must HAVE large page frames

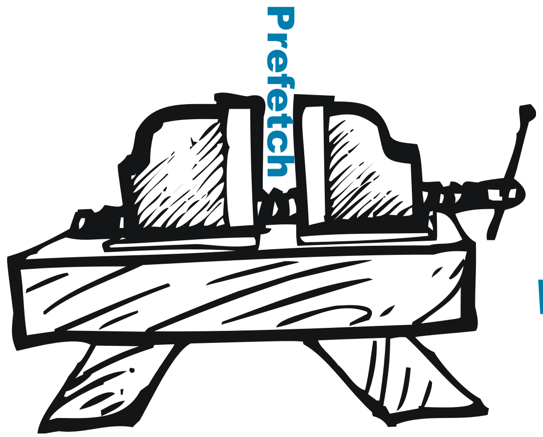
- In my experience, if large page frames not being used for PGFIX(YES) buffer pool, it's either because <u>none</u> of the z/OS LPAR's real storage is being managed using large frames, or there are some large frames but not enough to back the buffer pool
- Large frames are made available in a z/OS LPAR via the LFAREA parameter in the IEASYSnn member of SYS1.PARMLIB
  - LFAREA indicates the amount of the z/OS LPAR's real storage resource that is to be managed using 1 MB page frames, and the amount (if any) to be managed using 2 GB frames
  - Don't go overboard here – specify a 2 GB large frame area that will accommodate the DB2 buffer pools (if any) defined with FRAMESIZE(2G)
  - Specify a 1 MB large frame area sufficient for the buffer pools that will use 1 MB frames, plus a bit of a pad to cover other uses of 1 MB frames – a pad of 5-10% should be fine
    - If you use WebSphere Application Server for z/OS, keep in mind that 1 MB page frames can be used for the Java heap, and plan accordingly

2 GB page frames

- *Some people think that 2 GB page frames should not be used for a buffer pool that is less than 20 GB in size*
- While it is true that 2 GB page frames are unlikely to deliver much of a performance benefit, relative to 1 MB frames, for a pool that is less than 20 GB in size, it is also true that using 2 GB frames will not negatively impact performance for such a pool
- With expectation of roughly equivalent performance for 1 MB and 2 GB frames for a buffer pool sized between 2 GB and 20 GB (2 GB frames not used when a pool's size is less than 2 GB), is there any reason to use 2 GB frames for pools in this size range?
  - Yes – I have seen 2 GB frames used for buffer pools in the 2 GB - 20 GB size range, when a DB2 team has decided that larger pools will be enlarged by multiples of 2 GB (e.g., a 2 GB pool will be enlarged to 4 GB or 6 GB, but not to 2.5 GB)
  - In that case, 2 GB frames and VPSIZE values (for 4K buffer pools) that are multiples of 524,288 (the number of 4 KB buffers that exactly fill one 2 GB frame) helps to formalize that sizing strategy

Prefetch reads

- *Some people believe: prefetch reads not important for DB2 performance*
- Prefetch reads may not be as important as synchronous reads, but they do matter
  - That is why the most important performance metric for a buffer pool, in my opinion, is the total read I/O rate: all synchronous reads plus all prefetch reads, per second
    - Less than 1000 per second for a pool is good, less than 100 per second is great
  - A DB2 person, thinking that prefetch reads don't matter much, may take a pool's VPSEQT setting WAY down – from default of 80 to maybe 20 – in an effort to reduce synchronous reads
    - VPSEQT: percentage of buffers in pool that can receive pages brought into memory via prefetch (note: VPSEQT does not restrict buffers available for synchronous reads – ALL buffers available for those)



What happens when a very low VPSEQT value puts too much of a squeeze on prefetch?

When VPSEQT goes too low

- Might see big increase in prefetch read I/O activity – application performance impact
  - Check average class 3 “wait for other read” time in DB2 monitor accounting long report
- Might see more synchronous reads – look for increase in *sequential synchronous reads* in DB2 monitor statistics long report or -DISPLAY BUFFERPOOL DETAIL output
  - Could be a result of prefetched pages getting “flushed” from memory before they have been accessed by the application process that drove the prefetch read
    - Small VPSEQT value leads to too-short buffer pool residency time for prefetched pages
  - Could be caused by prefetch disabled – check DB2 monitor stats long report or -DIS BPOOL DETAIL
    - **PREFETCH DISABLED - NO BUFFER** – not enough buffer resources to support prefetch activity
    - **PREFETCH DISABLED - NO READ ENGINE** – not enough read engines to handle prefetch activity
- I am not saying, “Don’t ever lower VPSEQT” – I am saying:
  - Have a reason for making change (simulate effect via SPSEQT option of -ALTER BUFFERPOOL)
  - Don’t go overboard – a relatively modest change could yield the benefits you seek

DB2 arrays

- *Some people have gotten DB2 arrays confused with host variable arrays*
- DB2 arrays were introduced with DB2 11 for z/OS
  - A form of a user-defined data type (UDT)
  - Key use case: allow a client program to pass array of values as input to a native SQL procedure
    - Who really wanted this capability: Java developers (arrays are commonly used in Java programs)
- A host variable array is defined in an application program
- A question that came up soon after organizations got to DB2 11 new-function mode:
  - *Can I use a DB2 array as input in a MERGE statement?*
  - Answer: No – an array used as input in a MERGE statement must be a host variable array
  - Why I wouldn't be torn up about that: MERGE is majorly enhanced with DB2 12, so that you won't need to – and likely won't want to – use arrays with the statement
  - Another DB2 12 goody: global variables can be arrays

Global variables

- *Some people are a little confused about "what" and "why" of DB2 global variables*
- Introduced with DB2 11 for z/OS
- What: a global variable (in DB2 11) is a DB2 object into which a single value can be placed
  - You create a global variable, and it can then be populated with a SET statement
  - In a subsequent SQL statement, if the global variable is referenced in a query predicate, the value placed in the predicate will be used in evaluating the predicate
  - "Global" means the variable can be made globally available in a DB2 environment – it does not mean that a value placed in the variable is globally in effect
    - Context of a global variable value is a thread (i.e., a DB2 session) – if process XYZ puts the value 'SMITH' in global variable GV_NAME, that has zero effect on what any other process sees in global variable GV_NAME (each application process has its own instance of a particular global variable when it references the variable)

More on global variables

- The why:
  - They provide a means whereby a data value obtained by one SQL statement can be passed to another SQL statement, with no need for application program code in-between
  - One useful application of the feature: you can use global variable to pass a value to a DB2 trigger
- DB2 12 (as previously mentioned): array-type global variables
  - An array global variable can hold a “stack” of values
  - Possible use case: an array global variable would provide a lighter-weight, easier-to-use means of passing a stack of values from one SQL statement to another versus a one-column declared or created global temporary table (or 2-column temporary table, in the case of an associative array)
    - The SQL specification UNNEST can be used to turn an array into a 1- or 2-column table
- DB2 12 also allows placement of a value in a global variable via FETCH INTO
  - If it’s an array-type global variable, the FETCH INTO has to be in a SQL PL routine

Extents

- *Some people are concerned about DB2 data sets going into extents*
- I’m not – here’s why:
  - In most production DB2 for z/OS systems, vast majority of GETPAGEs satisfied from buffer pool
  - Typically, the large majority of GETPAGEs that are not satisfied from a buffer pool result in a read from disk controller cache, not spinning disk
  - Even when reads from spinning disk occur, the architecture of the disk subsystems typically used with mainframe servers is such that extents are of little consequence, performance-wise
- What about hitting an extent limit?
  - Not likely – the limit is 7257 (z/OS 1.7 or later, data set is SMS-managed, and Extent Constraint Removal option is set to YES in the SMS data class; otherwise, it’s 251)
- Recommendation: let DB2 manage secondary disk space allocation (MGEXTSZ YES in ZPARM, SECQTY not specified in CREATE TABLESPACE, or SECQTY set to -1)
  - Avoid extent limit, likely to avoid hitting max # of volumes across which data set can be spread (59)

Robert Catterall

rfcatter@us.ibm.com