DB2 for z/OS LOBs
Experiences & Best Practices

Haakon Roberts
DE, DB2 for z/OS & Tools Development
haakon@us.ibm.com
Disclaimer

- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
Agenda

- LOBs overview
- Inline LOBs
- Import & export of LOBs
- Utilities
- References
Relationship between base and LOB table spaces

LOB table space

LOB table space

LOB table space

LOB table space

LOB COL 1

LOB COL 2

LOB COL 1

LOB COL 2

LOB COL 1

LOB COL 2

aux

aux

aux

aux

LOB

LOB

LOB

Entire base table space

Part 1

Part M

Part N
LOBs Overview

- DB2 9
  - Implicit DDL for LOBs
  - Load/Unload FRV performance
  - SQL support for FRV
  - LOB lock avoidance
  - Faster preformatting
  - LOB APPEND
  - Progressive LOB Streaming
  - FETCH CONTINUE
  - REORG SHRLEVEL REFERENCE
  - Online CHECK LOB
LOBs Overview

**DB2 10**
- Inline LOBs
  - Index on expression
  - Spatial performance
- DEFINE NO
- REORG SHRLEVEL CHANGE
- REORG AUX YES
- LOB materialization avoidance
- Faster LOAD of FRV
- LOAD/UNLOAD support for RECFM=VBS

**DB2 11**
- Crossloader support for FETCH CONTINUE
  - 28% CPU reduction
LOB locking

▪ Pre-V9
  – Insert: X LOB lock, held until commit
  – Delete: S LOB lock, held until commit
  – Select: S LOB lock, held until commit
    – Even for ISO(UR)

▪ V9 onwards
  – Insert: S LOB lock, release after insert
  – Delete & normal select: No LOB lock
  – Select with ISO(UR): X LOB lock, released immediately
LOB locking & data sharing implications

- Since LOB lock released immediately after insert, LOB pages must be written out at end of insert if GBP-dependent

- Use GBPCACHE CHANGED instead of GBPCACHE SYSTEM
  - Default in DB2 9

- If GBPCACHE SYSTEM used then can expect a low GBP hit ratio
  - Always attempt to read data pages from GBP
LOB logging

- Only REDO image of LOB data is logged
- LOG NO = NOT LOGGED for LOBs, but system page information is still logged
  - Header page, HLSM, LLSM, LMAP information
- Aux index logging attribute inherited from base table
- Implications of using LOG NO
  - Does not affect rollback of unit of work
  - Forward log recovery or RECOVER BACKOUT YES will detect a LOB allocation and mark LOB invalid, leave LOB tablespace in AUXW state
    - Attempt to access results in -904
    - LOB can be deleted or updated
  - LPL recovery not allowed for LOG NO LOBs
  - GRECP recovery not allowed for LOG NO LOBs if GBPCACHE CHANGED or ALL
  - Lost LOBs must be restored by external means
- Most customers use LOG YES for recovery purposes
Advantages of Inline LOBs

- **Better performance**
  - For small LOB which is completed inline with base row, only need to access one table space instead of 2
  - For portion of LOB data that is frequently referenced
  - Inline portion of LOB can benefit from compression of the base table
  - Easy

- **Combine with Defer Define option could save DASD**
  - LOB table space is not allocated until the first insert to the LOB table space
  - Possible DBD contention when table has many LOB, AUX columns during the define process in concurrency environment
  - Inline LOBs might never need to allocate the LOB table space
  - Simplifies/Improves backup as empty data sets do not need to be processed
Inline LOB performance consideration

- **When is inlining not good?**
  - If most LOBs will be *split*, unless inlining is needed for indexing or for faster LOB searches
  - If the LOBs are accessed rarely
  - If the LOBs don’t compress well, having to compress or decompress them adds CPU time
  - If fewer than 50% of rows can benefit from inline then reconsider

- **Buffer hit ratio for base table may decrease**
  - But you save I/Os for the aux index and LOB table space
  - Retune the buffer pool configuration
  - Consider adding more real storage

- If LOBs weren’t otherwise logged, in-line data could cause more logging

- Image copies of the base table space will become larger
Inline LOB tuning consideration

- **Use proper page size**
  - Inline LOB can increase size of the record
    - Only increase to next larger page size can reduce the number of split LOB significantly (10% or more)
  - Small page size could cause less record inserted into page and more space map page update is needed

- **Optimal tuning requires knowing the typical row size**
  - Typical compress row size if it is compress table space
  - V10 RTS can help to calculate the average row size
  - You may need to do some Alter/Reorg iterations to find a right size

- **Proper define inline length**
  - Accessing LOB value that is split between base table and LOB table space, might slightly increased CPU due to concatenate LOB data from both table space
Alter and In-Line LOB

- The alter is an immediate change
  - Take effect on the next INSERT or UPDATE
  - The next LOAD or REORG will materialize the old rows with the new inline length.

- Alter inline lob length increase -> table space placed in AREO*

- Alter inline lob length decrease -> table space placed in REORP

- The catalog table – SYSIBM.SYSTABLEPART for LOB table space is updated with ‘I’ for increment inline length and ‘D’ for decrement inline length if the table is not empty
Using Inline LOBs to save DASD space

DASD space used for 1 million LOBs with 4K LOB page size

LOB Size

Gigabytes

Inline
Out-of-line
Random Access to small LOBs

Select 10,000 x 200 byte LOBs

Insert 10,000 x 200 byte LOBs
Faster LOAD REPLACE of small LOBs

**Elapsed Time**

- **200 bytes**
- **3900 bytes**

**CPU Time**

- **200 bytes**
- **3900 bytes**

Bar charts showing the comparison of Elapsed Time and CPU Time for OUTLINE and INLINE methods at 200 bytes and 3900 bytes.
Using UNLOAD and LOAD to Import and Export LOBs

- **DB2 9 (and prior)**
  - SYSREC files used by Load/Unload utilities in DB2 9 (and prior) must be RECFM=VB (not VBS)
    - Since the maximum LRECL for VB is 32760, DB2 9 can’t use SYSREC to load and unload LOBs bigger than 32K
    - May be multi-volume or striped
    - May use tape, which is faster than DASD since DB2 9 can use up to 256K block size on tape
  - Alternative is FRV (File Reference Variables)

- **DB2 10 can use RECFM=VBS (Variable Blocked Spanned)**
  - Supports all LOB sizes
  - Orders of magnitude faster than FRV
  - Potentially faster than VB, but not so in Version 10
LOB File Reference Variables

- Introduced in Version 8 for Load/Unload, and extended to SQL in V9
- Load/Unload limited to PDS/PDSE and Unix File Systems (HFS or zFS), but SQL can use sequential data sets (DASD or tape)
- A FRV column in SYSREC contains the name of the USS file, or PDS/PDSE name and member
- FRV limitations
  - PDS/PDSE limited to one volume
  - PDS limited to 64K tracks
  - PDSE limited to 512K members
  - USS file systems have no such limits and they are faster
  - No utility FRV support for DSORG=PS
Unload using FRV for 20,000 x 200 byte LOBs

- DB2 9 is 4 times faster than V8
- HFS and zFS are faster than PDS/PDSE (especially for Unload, not so much for Load)
- Yet, even with HFS, the throughput is at most 920 LOB/sec, which is very poor for small LOBs
DB2 10: RECFM=VBS data sets

- Enables sequential SYSREC data sets to be used for LOB sizes greater than 32K
  - May be multi-volume or striped
  - May use tape, which is faster than DASD since DB2 9 can use up to 256K block size on tape
  - Orders of magnitude faster than FRV, and avoids the functional limitations of PDS/PDSE

- The system determined block size, used by DB2, is 27K instead of 32K, so that DB2 may store up to 56K per track

- A small LOBVALA value (e.g. 500K) will achieve better LOAD/UNLOAD performance, but a large LOBVALA is recommended for optimal SQL performance in some cases
  - A large LOBVALA value prevents overlap of read/write I/O in DB2 10
VBS versus PDSE

100,000 rows took 2 hours, 55 minutes using PDSE and 21 minutes with PDS

VBS is orders of magnitude faster than PDSE or PDS
VBS versus USS

VBS is 80% faster than HFS (for small LOBs)
Faster LOAD REPLACE of large LOBs
Due to format writes

Load 20K LOBs
DBM1 LOB Materialization Avoidance for DRDA Inserts

- Saves DBM1 storage, CPU time and elapsed time
- Applies to LOBs greater than 2MB
- JDBC 4.0 and later or ODBC/CLI V8 driver FP4 and later. If using JDBC 3.5, the application has to specify the length of the LOB or XML to be -1.
- Restriction: There can be at most one LOB or XML per row for INSERT, UPDATE, or LOAD with file reference variable.
- More information can be found in “DB2 10 for z/OS Performance Topics”
LOB Materialization Avoidance

DRDA BLOB Inserts

Class 1 Elapsed Time

Class 1 CPU Time

- Saves 30% elapsed time and 15% CPU time
REORG of LOB table spaces

- SHRLEVEL NONE
  - Re-chunks the LOBs, but does not reclaim space
  - SHRLEVEL NONE is deprecated in DB2 10 (i.e. it is a no-op)

- DB2 9 introduces SHRLEVEL REFERENCE and DB2 10 introduces SHRLEVEL CHANGE for LOBs. Both reclaim physical space and fully reorganize the LOBs
  - Better availability
  - Temporarily requires additional DASD space for a shadow data set, but once REORG completes the original data set is deleted
  - No mapping table required
  - Recommendation: convert to SHRLEVEL CHANGE in 10 CM
  - Do not use SHRLEVEL NONE in 10 NFM – it’s a no-op!
New AUX keyword on REORG of partitioned base for improved LOB handling

- Permits rows to flow between partitions
- Allows REORG REBALANCE with LOB columns
- Allows ALTER of LIMITKEY with LOB columns
- Permits move of rows between parts on PBG REORG
- Permits deletion of corresponding LOBs on REORG DISCARD
- Default is AUX NO unless LOB objects required to complete REORG
- No XML column support for classic partitioned or PBR
- No mapping table change required
Utility considerations

- Avoid LOB columns in PBG tables prior to DB2 10
  - Rebalancing across partitions not possible prior to V10 without UNLOAD/LOAD

- Convert to REORG SHRLEVEL CHANGE before getting to 10 NFM

- REORG of PBG base tablespace that grows a new PBG partition in LOG phase with corresponding LOB tablespace can leave LOB tablespace in copy-pending

- Do not use RUNSTATS on LOB tablespaces
  - RTS has all information required
CHECK utility considerations

- CHECK DATA on base tablespace
  - SCOPE AUXONLY
  - AUXERROR REPORT
  - AUXERROR INVALIDATE
DB2 10: DEFINE(NO) for LOBs and XML

- Create LOB and XML table spaces and dependent index spaces with an option to defer the physical creation of underlying VSAM data sets until the very first insert
  - DB2 catalog entry will be created

- Performance and usability
  - Shortens the installation process for software packages that use LOB/XML
  - Saves DASD
  - Inline LOBs might never need to allocate the LOB table space
  - Simplifies/Improves backup as empty data sets do not need to be processed
DB2 10: Catalog & directory

- DBD01 converted to use SYSDBDXA LOB tablespace
- SPT01 converted to use SYSSPUXA & SYSSPUXB LOB tablespaces
  - Inlined by default
- APARs
  - PM64226 – Directory LOB space enhancement
  - PM74659 – SPT01 space management
  - PM75921 – LOB space enhancement
  - PM90310 – Permit CHECK LOB AUXERROR INVALIDATE for catalog & directory
References

- SG24-7270-00, “LOBs with DB2 for z/OS: Stronger and Faster”,

- SG24-7892, “DB2 10 for z/OS Technical Overview”,

- SG24-7942, “DB2 10 for z/OS Performance Topics”,

- General DB2 z/OS resources
  - DB2 for z/OS and OS/390
    - http://www.software.ibm.com/data/db2/zos
  - DB2 Family Performance
  - DB2 Solutions Directory Applications + Tool Search
  - DB2 Magazine
    - http://www.db2mag.com