Large Objects in DB2 for z/OS:

You Better Get Used To Them

Francis Desiron, IBM

DB2 GSE meeting March 2013

Disclaimer

Many thanks to Haakon Roberts and Jeff Berger for allowing me to ‘re-use’ some of their material

Agenda

- Introduction
- The (very) basics
- LOB enhancements at a glance
- Integrity checking
- Locking enhancements
- CHECK LOB/DATA enhancements
- REORG enhancements

- Reference material
  - Application flow optimization
  - File Reference Variables
  - LOAD/UNLOAD/Crossloader enhancements
  - Other

Introduction

- DB2 for z/OS LOB journey started in V6
- Lots of improvements over the years
- In V10 LOBs are now part of the catalog and directory
  - Actually already some catalog tables with LOBs prior to V10
  - V10 also ‘core’ catalog and directory tables have LOB columns
- If you were not using LOBs before, you will be in V10 NFM
  - As LOBs are a bit different from regular objects, it is best to be prepared
    - Practice recovery procedures
    - Start ‘playing’ with LOB columns in regular tables
    - Make sure to update your (disaster) recovery procedures
  - Better to be safe than sorry !

LOBs in C+D

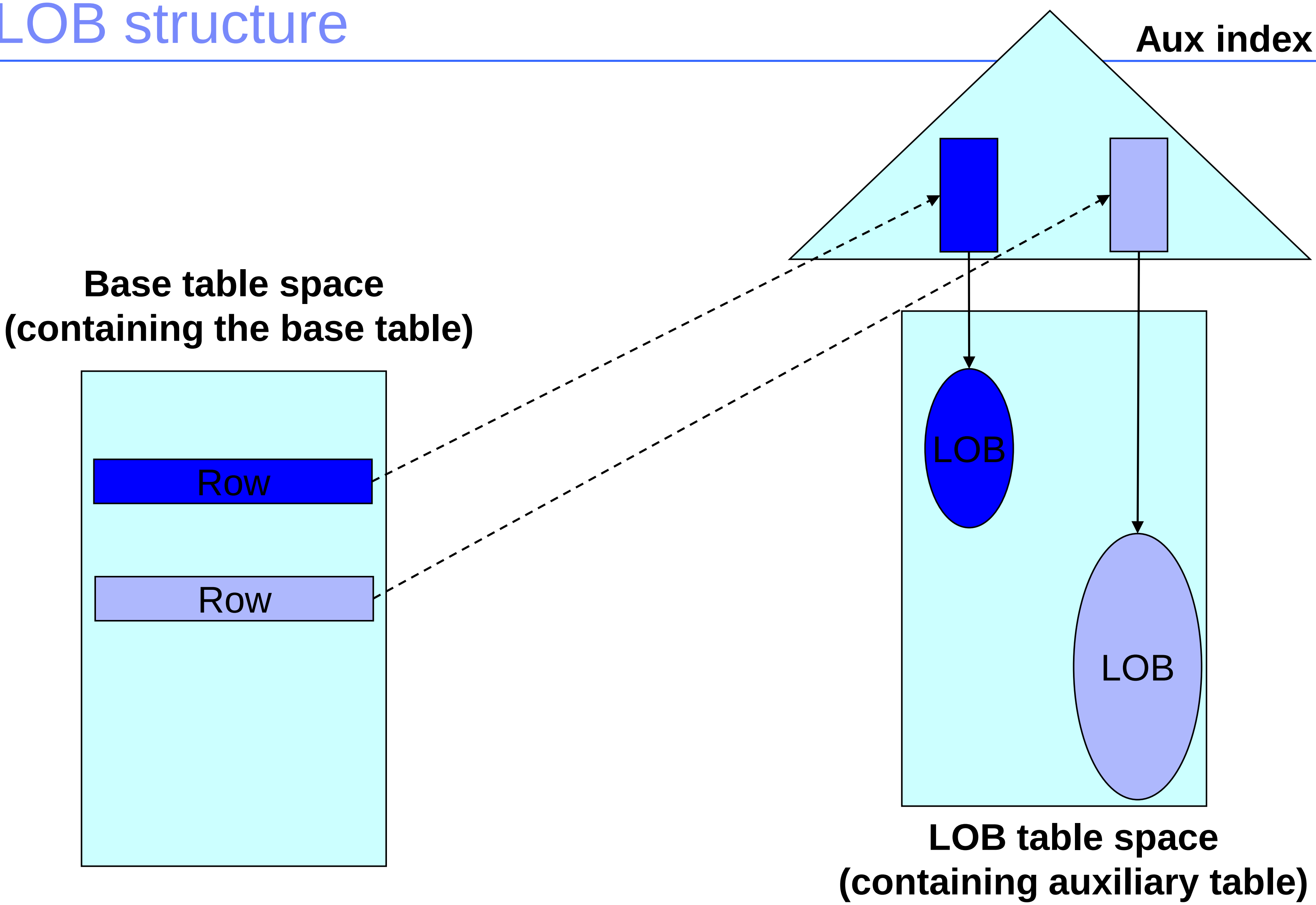
- LOB column actually introduced in the catalog in V7
  - Limited to JAR files for Java SP
- V10
  - Introduced LOBs for core catalog and directory tables
  - Complete V10 list is in the reference material
- V11
  - Add some more

Version	Table Spaces	Tables	Indexes	Columns	LOB columns
V1	11	25	27	269	0
V3	11	43	44	584	0
V5	12	54	62	731	0
V7	20	84	118	1212	2
V8	22	85	132	1265	2
V9	28	104	165	1652	6
V10	95	134	233	2036	36
V11	108	143	250	2202	42

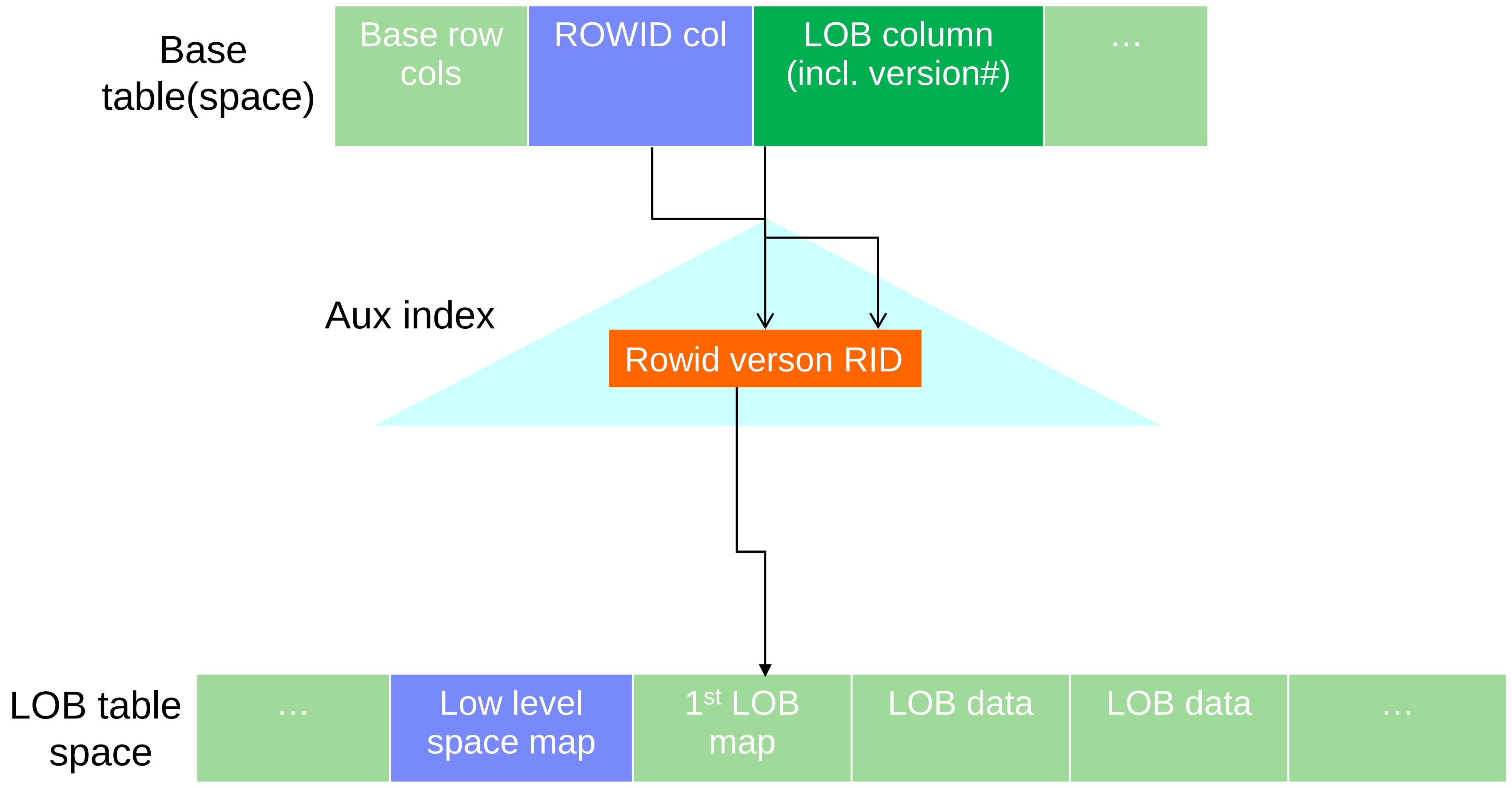
The basics

- One LOB table space per LOB column per partition
  - 2 LOB columns & 4096 parts = 8192 LOB table spaces
  - When a PBG table space grows to a new partition, DB2 creates a new LOB table space using the same STOGROUP as that of the first partition
- One aux table and one aux index per LOB table space
- Access to LOB data is via base row and aux index
- LOB uniquely defined by ROWID and version
  - Update changes version, ROWID never changes
- Internally, no concept of update of LOB
  - Update is de-allocation of old version, allocation of new
- Can be LOGGED / NOT LOGGED
  - NOT LOGGED does not mean zero logging for LOBs
- Separate from base table space
  - REORG, COPY, RECOVER, RUNSTATS
- **Keep LOBs in sync with info in the base table**

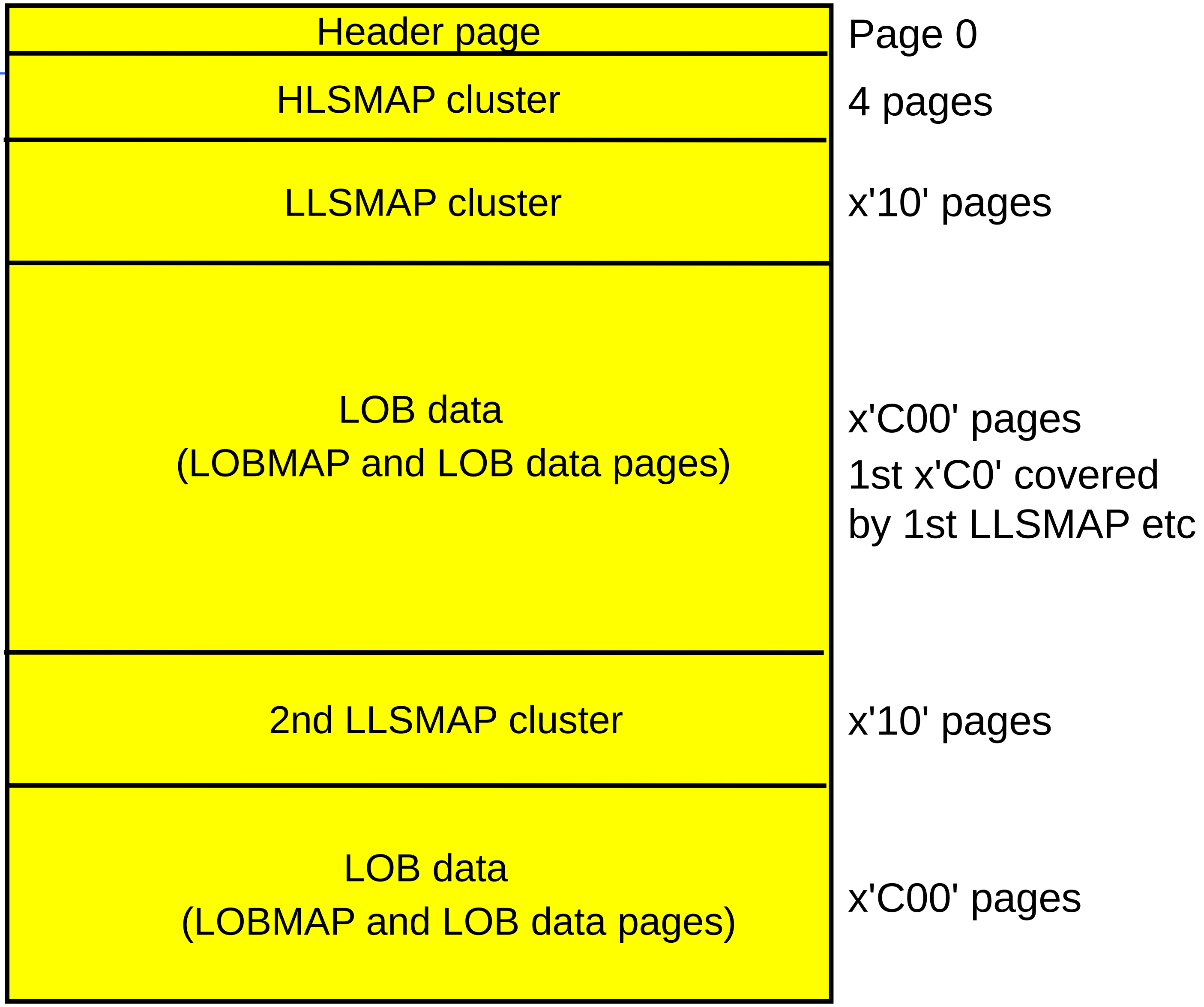
LOB structure



LOB structure



LOB structure



LOB enhancements at a glance

DB2 V8	DB2 9	DB2 10
Load/Unload support for File Reference Variables (FRV)	Implicit DDL for LOBs	Inline LOBs
	Load/Unload FRV performance	- Index on expression
	SQL support for FRV	- Spatial performance
	LOB lock avoidance	DEFINE NO
	Faster preformatting	REORG SHRLEVEL CHANGE
	LOB APPEND YES	REORG AUX YES
	Progressive LOB Streaming	LOB materialization avoidance
	FETCH CONTINUE	Faster LOAD of FRV
	REORG SHRLEVEL REFERENCE	LOAD/UNLOAD support for RECFM=VBS
	Online CHECK LOB	
	Allow LOGGED if > 1G	

LOB integrity

- Relationship between base table (LOB column) and auxiliary table (LOG TS) can be viewed like a ‘referential integrity relationship’
  - A non zero length non null LOB column in the base table has to have a matching LOB in the LOB TS
    - Existing LOB
    - Same ROWID and version-id
- Problems typically occur
  - After PIT recovery of base TS but not the LOB TS
  - After PIT recovery of LOB TS but not the base TS
  - Recovery to a different PIT or non-quiesce point
  - Basically when base and LOB TS are out of sync
  - (DB2 defect)

LOB integrity

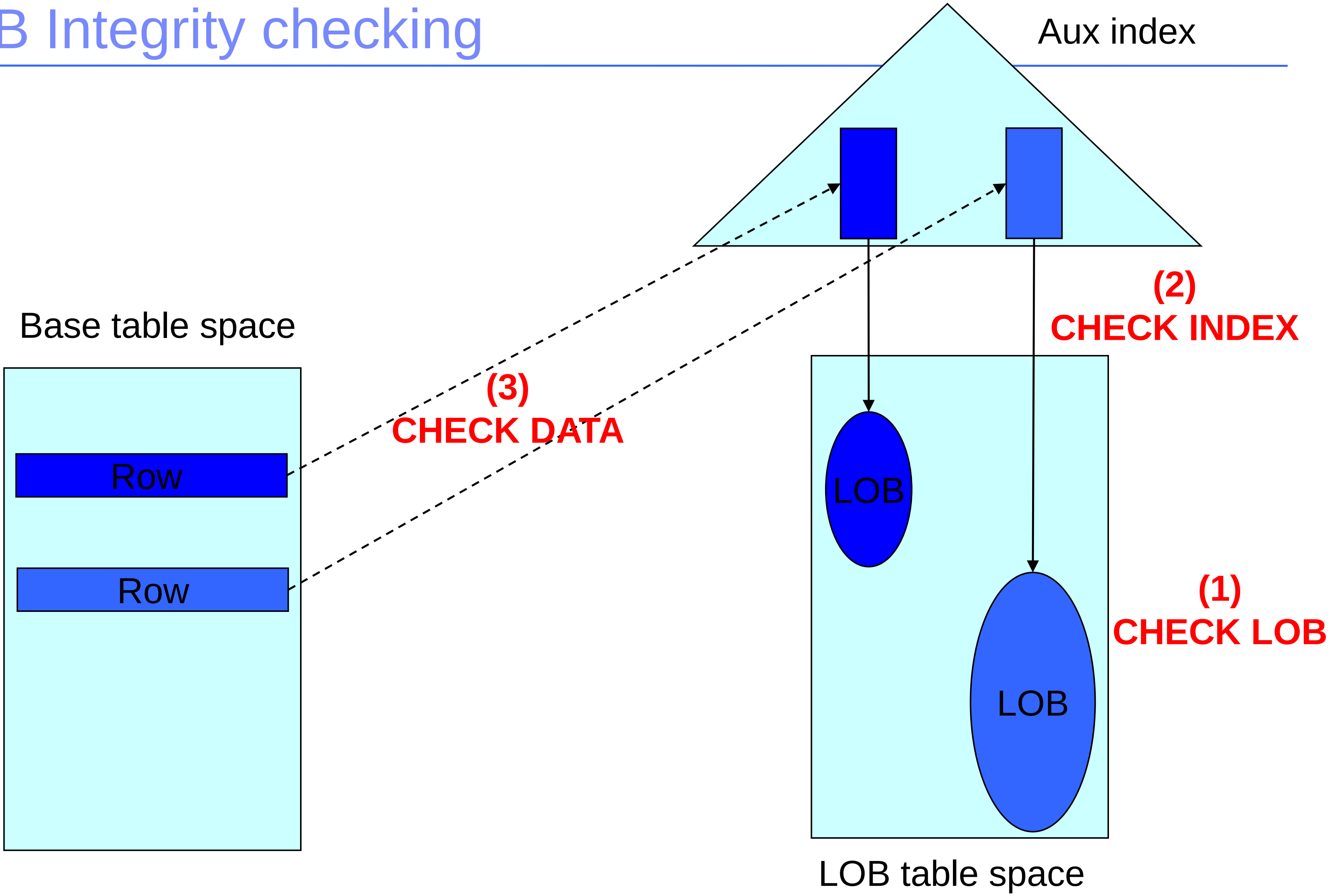
- LOGGED LOBs
  - Recovery options are similar to base tables
  - Make sure to keep base + LOB TS in sync
- NOT LOGGED LOBs does not mean non-recoverable
  - Transaction rollback still possible
    - Enough control information logged to do that
    - Update is not in place (delete/insert)
  - Image copies are allowed
  - RECOVER with roll forward on the log is possible, BUT the LOB itself cannot be recovered of course
    - Marked invalid
- New states introduced with LOBs on both the base and LOB TS

LOB specific DBET states

- AUXW
  - Set on LOB table space or base table space
  - Non-restrictive
  - After forward log recovery of a NOT LOGGED LOB table space if log records encountered
  - Some LOBs marked invalid in LOB table space
  - After CHECK DATA AUXERROR INVALIDATE on base table space
  - Reset by CHECK DATA or REPAIR
- ACHKP – Aux check pending
  - Set on base table space
  - Restrictive – NOT an advisory state
  - Set by CHECK DATA AUXERROR REPORT
  - Reset by CHECK DATA AUXERROR INVALIDATE or REPAIR
- CHKP
  - LOB table space
  - Restrictive
  - Set if error from CHECK LOB
  - Reset by CHECK LOB or REPAIR

New options in V9 and V10 can affect when DBETs states are set/reset (see later)

LOB Integrity checking



LOB integrity checking

- CHECK LOB
  - Verifies integrity of LOB table space
  - PQ91381 (2004/11) made big improvements to CHECK LOB
- COPY CHECKPAGE
  - Since PQ74793 (2003/06) CHECKPAGE also works for LOB TS
- CHECK INDEX on aux index
- CHECK DATA on base table space
  - Can run at part level
  - SCOPE AUXONLY
  - AUXERROR REPORT
  - AUXERROR INVALIDATE

LOB restoration techniques - what to use when

- Can corrupted data be restored by external means?

Easily	Impossible
Recover/salvage/repair	Recover/repair/salvage

- What is the scope or complexity of the corruption?

1 LOB and simple	Extensive or complex
Recover/repair/salvage	Recover/salvage/repair

- Is LOB data corrupted?

No	Yes
Recover/repair/salvage	Recover/repair/salvage

- Is the LOB table space NOT LOGGED

No	Yes
Recover/repair/salvage	Repair/salvage/recover

LOB restoration

- RECOVER
  - Simple - best option if LOB is LOGGED - no loss of data
  - Need good recovery base & no corruption on log
- REPAIR
  - Possibly no loss of data
  - Manual process - OK if scope of corruption limited and simple
  - PQ82063 cuts REPAIR time down from hours to minutes
  - Can be used to remove corrupted LOBs without extracting good data to new table
    - Good if LOB can be restored from image copy or elsewhere
- SALVAGE
  - Guaranteed to work, but guaranteed loss of data
  - OK if LOB can be restored from elsewhere
  - Less necessary with PQ82063

LOB restoration – sample procedures

- Repair & restore
  - Find LOB on earlier image copy
  - Create dummy structure based on existing structure
  - DSN1COPY image copy onto LOB table space
  - Extract base row and LOB to new dummy structure using SQL
  - Recover LOB table space to current
  - Use REPAIR to fix corruption by de-allocating all pages belonging to corrupted LOB
  - Rebuild aux index
  - CHECK DATA SCOPE AUXONLY AUXERROR INVALIDATE
  - Update base row set LOB column=LOB from dummy structure
- Salvage
  - Identify corrupted LOBs, extract all good LOBs to new structure based on old
  - INSERT INTO... SELECT FROM... WHERE rowid-col NOT IN (...)
  - Need PQ80403 for this
  - Drop & recreate original structure then copy back in
  - Restore missing LOB data by external means

LOB Locking

- Locking behavior is different from normal base table page/row locking
  - Locking at the LOB level
  - Use lock on the base table to protect the LOB when possible
- Significant locking changes in V9
  coming up next

Locking enhancements in V9

- Why did we change the locking behavior in V9 ?
  - Availability
    - LOB locks always held until commit
    - LOB locks acquired even for UR readers
      - Lock escalation can occur on LOB table space
  - Performance
    - Significant performance overhead
    - LOB locks acquired for space search purposes

Locking enhancements

- INSERT
  - Pre-V9
    - Get X lock on LOB, hold until commit
  - V9
    - Get X lock on LOB, release after allocation complete
    - In data sharing, need to ensure changed pages are forced out before lock is released, therefore **recommend GBPCACHE(CHANGED) for improved performance**
      (default for LOB TS in V10)
- DELETE
  - Pre-V9
    - Get S lock on LOB, hold until commit
  - V9
    - No LOB lock acquired

Locking enhancements

- SELECT
  - Pre-V9
    - Get S lock on LOB, hold until commit
  - V9
    - For non-UR readers, no LOB lock acquired
    - For UR readers, get S LOB lock & release immediately
- UPDATE
  - Pre-V9
    - Update is delete followed by insert
  - V9
    - Same
      - Benefit from reduced locking for insert, delete & space search

Locking enhancements

- Space search for LOB allocation
  - Pre-V9
    - If a free page belongs to a deallocated LOB then
      - Check if deallocation occurred prior to oldest read claim
        - This test not particularly granular
      - If readlsn checking fails then try to get lock on old LOB
        - If successful, use the page, else try another
    - Can result in many lock requests for a single LOB allocation
  - V9
    - No LOB locks acquired for space search
    - Readlsn granularity improved to page level in LOB table space
      - >=192x more granular

Locking enhancements - summary

- Significant reduction in locking overhead for LOB processing
- Improved availability & performance
- Recommend GBPCACHE(CHANGED) for improved performance
- Requirements
  - V9 NFM
  - Was known as “Locking protocol 3”
    - Automatic in non-data sharing
    - Clean group-wide shutdown in data sharing once NFM enabled
  - PK62027 – remove requirement for clean group-wide shutdown

Other things you should know

- Utility enhancements that affect (re)setting of DBET conditions
- SHRLEVEL CHANGE for CHECK LOB and CHECK DATA
  - Don't (re)set DBET states – runs on shadow objects
  - CHECK DATA SHRLEVEL CHANGE cannot delete rows or mark LOBs invalid
    - Write REPAIR LOCATE VERIFY/REPLACE statements to PUNCHDDN dataset to invalidate LOBs
  - CHECK LOB SHRLEVEL CHANGE cannot delete LOBs
    - Write REPAIR LOCATE DELETE statements to PUNCHDDN dataset
- CHECK_SETCHKP ZPARM in V10
  - Default not to set check pending anymore

REORG - enhancements

- REORG LOB SHRLEVEL NONE
  - Did not reclaim space – only ‘rechunked’ the LOBs
  - No-op in V10
- REORG LOB SHRLEVEL REFERENCE (V9) / CHANGE (V10)
  - Performs ‘Real’ REORG with physical space reclamation
  - Improved availability during REORG
  - Improved LOB retrieval performance after REORG
- REORG LOB SHRLEVEL CHANGE recommended
  - Unload via AUX index (random by design) can be slow (especially the first time, or after massive I/U/D activity)
  - With SHRLEVEL REFERENCE, the LOB TS is RO during the (long) reorg run
- DB2 10 REORG of base table space with AUX YES option enables DB2 to move rows across partitions when a LOB (or XML) column is used
  - Use AUX YES to rebalance PBR partitions or to consolidate PBG partitions

Summary

- LOBs will become part of your life (certainly in V10 NFM)
- LOBs are "different animals" compared to regular table spaces
- Lots of enhancements during the last couple of DB2 versions
  - LOBs are ready for prime time
  - They better be as they are now part of the catalog and directory
- Learn at least how to administer them
  - Locking is different
  - Recovery is different
- Practice recovery/repair/salvation
  - Adjust your DR plan
- Explore the use in applications as well
  - You may find a lot of good use for LOBs

Questions

Questions ?

Reference material starts here

- See also:
  - SG24–7270-00 LOBs with DB2 for z/OS: Stronger and Faster (V9 level)
  - www.ibm.com/redbooks

CHECK utility enhancements

- Why?
  - CHECK LOB & CHECK DATA provide the ability to verify data integrity but…
  - CHECK LOB does not allow update access
  - CHECK DATA does not allow update access either
    - May not allow read access depending on parameters
  - Any inconsistencies encountered may mark entire pageset unavailable
  - In summary, verifying data integrity cannot be done without incurring a (sometimes lengthy) outage

V9 CHECK utility enhancements

- New **SHRLEVEL CHANGE** option for CHECK DATA and CHECK LOB
  - CHECK INDEX already has SHRLEVEL CHANGE option
  - Solution extended to CHECK DATA, CHECK LOB
  - Short term drain of writers to allow flashcopy to shadow
    - Usual drain parameters supported
  - **CHKP/ACHKP/AUXW no longer set if errors detected**
    - Not reset either – use REPAIR
  - CHECK DATA SHRLEVEL CHANGE cannot delete rows or mark LOBs invalid
    - Write REPAIR LOCATE DELETE statements to PUNCHDDN dataset instead of RI discard
    - Write REPAIR LOCATE VERIFY/REPLACE statements to PUNCHDDN dataset to invalidate LOBs
  - CHECK LOB SHRLEVEL CHANGE cannot delete LOBs
    - Write REPAIR LOCATE DELETE statements to PUNCHDDN dataset

V10 CHECK utility enhancements

- New CHECK_SETCHKP ZPARM
  - Whether CHECK DATA or CHECK LOB should turn on Check Pending when it finds a problem
  - Default is no (so changed behavior from V10 unless you change to Yes)
- Setting has no effect for SHRLEVEL CHANGE
  - Operates on a shadow copy of the data and don’t set/reset state (see previous foil)

REORG

- Why?
  - Pre-V9, REORG of LOB table spaces has a number of drawbacks
  - No physical space reclamation
  - SHRLEVEL NONE only
    - No access to LOB data during REORG
  - Re-chunking of LOB data may be sub-optimal
    - Trade-off between space consumption & reorganization
  - LOG YES only
    - May result in excessive logging
  - Complex, susceptible to software defects

REORG

- Introduce SHRLEVEL REFERENCE option for REORG of LOB data
  - LOG NO only permitted option with SHRLEVEL REFERENCE
  - REORG will now load LOBs to shadow LOB pageset
    - Additional DASD space temporarily required
  - Available in both CM and NFM
  - Improved availability
  - Complete reorganization of LOB data
  - Full read access permitted to LOB data except during SWITCH phase
  - Inherit drain options from “standard” REORG
  - Inline imagecopy required to maintain recoverability
  - SHRLEVEL NONE still supported
    - Does nothing in V10 just RC=0
    - Plan to phase out , next step expect RC=8
  - No restart capability
    - Shadow pageset discarded in event of failure

REORG V10

- Finish the job by
- Introducing SHRLEVEL CHANGE for REORG of LOB data
  - Improved availability
  - REORG LOB uses the AUX index to unload the data
  - Good chance that LOBs are accessed in a random fashion
  - First REORG could be be slow
  - With SHRLEVEL CHANGE data is available almost entire time

REORG of partitioned TS with LOBs – AUX YES

- DB2 9 REORG cannot move a row that contains a LOB or XML column from one partition to another
- DB2 10 REORG of base table space with AUX YES enables DB2 to move rows across partitions when a LOB or XML column is used
  - AUX YES used to rebalance PBR partitions or to consolidate PBG partitions
  - AUX NO is the default except when multiple PBG partitions are reorg-ed (MAXPARTS=1 uses AUX NO)
  - REORG with AUX YES may not perform as well as REORG LOB. Use AUX NO if you REORG multiple partitions and don't care about moving rows between partitions

REORG - summary

- REORG of LOB data now much more viable
- Improved availability during REORG
- Improved LOB retrieval performance after REORG
- Physical space reclamation now possible with SHRLEVEL REFERENCE / CHANGE
- Requirements
  - V9 CM SHRLEVEL REFERENCE
  - CMx SHRLEVEL CHANGE
  - Utility job JCL change

Application flow optimization

- Why?
  - Need to reduce flow of unnecessary LOB data for small/medium sized LOB data requests across a network
  - Need to simplify manipulation of large LOBs
  - LOB data transfer currently optimized for large amounts of data
  - Locators may result in poor performance
    - More resource consumption at server
      - Particularly if locator not freed or application doesn’t commit
    - More complex application coding
      - Up to 3 trips to DB2 to materialise LOB
  - For large LOBs, locators often used, but this incurs a separate network flow before data is retrieved
  - For small LOBs, more efficient to retrieve LOB data directly
  - Increased application virtual storage consumption if XML or locator not used
    - Maximum buffer size must be allocated if want to avoid truncation
    - For XML documents, maximum size not known, so educated guesswork
  - Improve handling of large XML objects
    - No locator support for XML

Application flow optimization

- Progressive streaming
  - Provide new LOB/XML data retrieval design
    - Effective for small/medium size objects
    - More efficient use of locators to retrieve large amounts of data
  - Introduce ability for server to dynamically determine most efficient method to return LOB/XML data
  - With dynamic data format enabled, locator is kept for lifespan of cursor, not transaction
  - JDBC, SQLJ, and CLI will let server determine whether to flow LOB values or locators based on size thresholds
    - 2 new JCC T4 datasource properties
      - progressiveStreaming
      - streamBufferSize

Application flow optimization

- FETCH CONTINUE
  - Retrieve LOB or XML data in multiple pieces without use of locators
  - Continue fetch of remaining data when truncation occurs
  - Must specify WITH CONTINUE on initial FETCH
  - Subsequent fetches use FETCH CURRENT CONTINUE
  - Application must manage buffers & reassemble data
    - Not required to fetch entire object before moving to next
  - SQLCA indicates whether data is truncated
  - No multi-row fetch support
  - Universal JDBC driver exploits this to implement progressive streaming
- Locators still recommended for random access to subset of object or if materialisation is to be avoided

Application flow optimization - summary

- Significant reduction in network traffic
- Simplified application design
- Improved performance
  - More efficient retrieval of small/medium LOBs avoiding use of locators
- Improved resource utilization for locators
- Reduced application virtual storage consumption
- Requirements
  - NFM
  - JDBC/CLI/SQLJ dependency also

File reference variables

- Why?
  - Difficult to load/unload large LOBs
  - Poor performance on load/unload
  - Significant application working storage requirement when manipulating large LOBs
  - Cross-platform compatibility

File reference variables

- FRVs defined in a host language
  - Contains file name and allows direct transfer of LOB data between DB2 and the file
- Language support
  - C, C++, Java
  - COBOL, PL/1
  - Assembler
  - REXX
  - No FRV implementation required in JCC type 2 or type 4 drivers
    - Both JCC Type 2 and Type 4 drivers have the capability to read a file to be used in its ‘stream’ (byte/character) methods

File reference variables

- Allow a large LOB or XML value to be inserted from a file or selected into a file rather than a host variable
- Application no longer needs to acquire storage to contain the LOB or XML value
- Bypass host language limitations on the maximum allowed size for LOB values located in working storage
- Support HFS or BSAM
  - HFS used if filename contains a “/”, otherwise BSAM
- FRVs cannot be used as parameters for stored procedures or UDFs
- New SQL host variables
  - BLOB_FILE
  - CLOB_FILE
  - DBCLOB_FILE

File reference variables - summary

- Significantly improve DB2 capability for handling large LOBs
  - Improved performance
  - Improved usability
- Reduce application virtual storage requirements
- Improve family compatibility & application portability
- Requirements
  - V9

LOAD/UNLOAD/Crossloader

- Why?
  - LOAD utility limitation of 32Kb for input row length
- Limitation removal for Crossloader
  - Separate buffer used for LOB column values
  - DSNU1778I only issued if
    - (Sum of lengths of non-LOB columns) + (8 x Number of LOB columns) exceeds 32Kb or
    - Sum of lengths of LOB columns exceeds half of available memory above the line
  - PQ90263 in V7/V8
- LOAD/UNLOAD support for file reference variables
  - PK22910 in V7/V8

LOAD/UNLOAD/Crossloader

- FRV support
  - Load Unload limited to PDS/PDSE and Unix File Systems (HFS or zFS)
    - A FVR 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
- 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
  - Changing the VBS format ??? DCR ???

Other V9 enhancements

- Allow logging of LOB data when max size > 1Gb
  - V9 (DK179)
  - Previously, LOG YES restricted to max size <=1Gb
- New APPEND option for LOB table spaces
  - Prevent exponential growth is space search algorithm
  - 31% CPU reduction measured
  - V9 APAR PK65220

Other V10 enhancements

- INLINE LOBs
  - Including in the DB2 directory
  - See Paul’s presentation
- Eliminate materialization of LOBs in DBM1 during insert
  - DB2 materializes up to 2M before inserting into the database
  - Storage will be reused for subsequent chunks until the whole LOB is processed.
  - Preformatting and update writes are overlapped with network transfer
  - Considerably less CPU time and virtual storage used

Info about LOBs in the DB2 catalog

- SYSIBM.SYSAUXRELS table
  - One row for each auxiliary table created for a LOB column.
    A base table space that is partitioned must have one auxiliary table for each partition of each LOB column
- SYSIBM.SYSCOLUMNS
  - COLTYPE
    - ROWID / BLOB / CLOB / DBCLOB

LOBs in C+D

- The following tables in the C+D contain LOB columns in DB2 10 NFM
  - SYSIBM.DBD01
    - DBD_DATA (BLOB 2G)
  - SYSIBM.SPT01 (BLOB 2G)
    - SPTSEC_DATA (BLOB 2G)
    - SPTSEC_EXPL (BLOB 2G)
  - SYSIBM.SYSAUTOALERTS
    - OUTPUT (CLOB 2M)
  - SYSIBM.SYSAUTORUNS_HIST
    - OUTPUT (CLOB 2M)
  - SYSIBM.SYSAUTORUNS_HISTOU
    - OUTPUT (CLOB 2M)
  - SYSIBM.SYSCONTROLS
    - RULETEXT (CLOB 2M)
    - DESCRIPTOR (BLOB 2M)

LOBs in C+D

- Continued:
  - SYSIBM.SYSCONTROLS_DESC
    - DESCRIPTOR (BLOB 2M)
  - SYSIBM.SYSCONTROLS_RTXT
    - RULETEXT (CLOB 2M)
  - SYSIBM.SYSINDEXES
    - PARSETREE (BLOB 1G)
    - RTSECTION (BLOB 1G)
  - SYSIBM.SYSJARCLASS_SOURCE
    - CLASS_SOURCE (CLOB 10M)
  - SYSIBM.SYSJARCONTENTS
    - CLASS_SOURCE (CLOB 10M)
  - SYSIBM.SYSJAROBJECTS
    - JAR_DATA (BLOB 100M)

LOBs in C+D

- Continued:
  - SYSIBM.SYSPACKSTMT
    - STATEMENT CLOB(2M)
    - [STMTBLOB (CLOB 2M)]
  - SYSIBM.SYSPENDINGDDL
    - STATEMENT_TEXT (CLOB 2M)
  - SYSIBM.SYSQUERY
    - STMTTEXT (CLOB 2M)
  - SYSIBM.SYSROUTINES
    - TEXT (CLOB 2M)
    - PARSETREE (BLOB 1G)
  - SYSIBM.SYSTABLES_PROFILES
    - PROFILE_TEXT (CLOB 1M)
  - SYSIBM.SYSTRIGGERS
    - STATEMENT (CLOB 2M)

LOBs in C+D

- Continued:
  - SYSIBM.SYSVIEWS
    - STATEMENT (CLOB 2M)
    - PARSETREE (BLOB 1G)
  - SYSIBM.SYSXSROBJECTCOMPONENTS
    - COMPONENT (BLOB 30M)
  - SYSIBM.SYSXSROBJECTS
    - GRAMMAR (BLOB 250M)
  - SYSIBM.SYSXSROBJECTS
    - PROPERTIES (BLOB 5M)
  - SYSIBM.SYSXSROBJECTCOMPONENTS
    - COMPONENT (BLOB 5M)