IBM Software Group

# Hints and Tips to Get Most out of DB2 for z/OS

# Disclaimer



***THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.***

# DB2 9 – A Rich, Features Filled Release

- SHRLEVEL(REFERENCE) for REORG of LOB table spaces
- Online RENAME COLUMN
- Online RENAME INDEX
- Online CHECK DATA and CHECK LOB
- Online REBUILD INDEX
- Online ALTER COLUMN DEFAULT
- More online REORG by eliminating BUILD2 phase
- Faster REORG by intra-REORG parallelism
- 256K tape block size
- Renaming SCHEMA, VCAT, OWNER, CREATOR
- LOB Locks reduction
- Skipping locked rows option
- Tape support for BACKUP and RESTORE SYSTEM utilities
- Recovery of individual table spaces and indexes from volume-level backups
- Enhanced STOGROUP definition
- Conditional restart enhancements
- Histogram Statistics collection and exploitation
- WS II OmniFind based text search
- DB2 Trace enhancements
- WLM-assisted Buffer Pools management
- . . .

- Global query optimization
- Generalizing sparse index and in-memory data caching method
- Optimization Service Center
- Autonomic reoptimization
- Logging enhancements
- LOBs network flow optimization
- Faster operations for variable-length rows
- NOT LOGGED table spaces
- Index on expressions
- Universal table spaces
- Partition-by-growth table spaces
- APPEND option at insert
- Autonomic index page split
- Different index page sizes
- Support for optimistic locking
- Faster and more automatic DB2 restart
- RLF improvements for remote application servers such as SAP
- Preserving consistency when recovering individual objects to a prior point in time
- CLONE Table: fast replacement of one table with another
- Index compression
- Index key randomization
- . . .

- DECIMAL FLOAT
- BIGINT
- VARBINARY, BINARY
- TRUNCATE TABLE statement
- MERGE statement
- FETCH CONTINUE
- ORDER BY and FETCH FIRST n ROWS in sub-select and full-select
- ORDER OF extension to ORDER BY
- INTERSECT and EXCEPT Set Operations
- Instead of triggers
- Various scalar and built-in functions
- Cultural sort
- LOB File Reference support
- XML support in DB2 engine
- Enhancements to SQL Stored Procedures
- SELECT FROM UPDATE/DELETE/MERGE
- Enhanced CURRENT SCHEMA
- IP V6 support
- Unified Debugger
- Trusted Context
- Database ROLEs
- Automatic creation of database objects
- Temporary space consolidation
- 'What-If' Indexes
- . . .

**TCO Reduction** **Continuous Operations** **Performance** **Scalability** **SQL** **Portability**

# Agenda

- Administration
  - Partition By Growth
  - Reordered Row Format
  - Merged workfile/temp space
  - Utilities
- LOBs and XML
  - What is pureXML?
  - Progressive LOB streaming
  - Fetch Continue
  - File Reference Variables
  - LOB Append
- Index Management
  - Large page sizes
  - Compression
  - What if?
- SQL
  - Native SQL procedures
  - Select from Update/Delete/Merge
  - Order By and Fetch First
  - New data types
  - Skipping locked rows
  - Append
  - Truncate table
  - Merge

# Administration

# Universal Table Spaces – Partition By Growth

- Why use PBG?
  - PBG is a better answer for partitioning than key range partitioning when the size is unknown or no natural partitioning key exists
    - DB2 automatically adds partitions on demand
  - Support of fast mass deletes
- To use PBG specify MAXPARTITIONS
  - If SEGSIZE is not specified, DB2 will use the default SEGSIZE
  - Single table only
- DROP / CREATE to migrate existing page sets
- Simple table spaces can not be created
  - Default table space is now Segmented (CM) or PBG (NFM)
- Incompatible with MEMBER CLUSTER, ADD PARTITION, ROTATE PARTITION

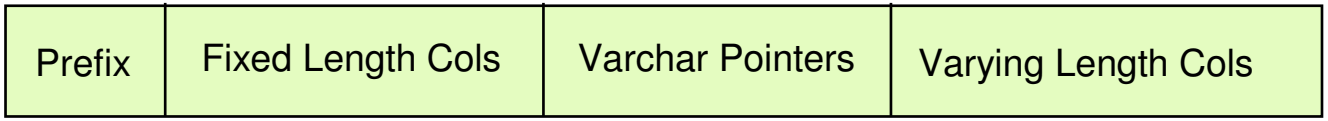
# Partition By Growth – Reorg Considerations

- MAXPARTITIONS can be ALTERed but observe the limits that are dependent from page size and DSSIZE

| DSSIZE \ Page Size | 4K | 8K | 16K | 32K |
|---|---|---|---|---|
| 1 – 4 GB | 4096 | 4096 | 4096 | 4096 |
| 8 GB | 2048 | 4096 | 4096 | 4096 |
| 16 GB | 1024 | 2048 | 4096 | 4096 |
| 32 GB | 512 | 1024 | 2048 | 4096 |
| 64 GB | 256 | 512 | 1024 | 2048 |

- REORG can add new partitions (except if there are LOB or XML columns).
- REORG will not remove empty partitions, but it can shrink them to contain a header and space map page, again, subject to absence of LOB and XML columns
- If a table space contains a LOB/XML column, REORG cannot move a row from one partition to another

# Reordered Row Format (RRF) …




- Potential for significant reduction in CPU resource consumption when accessing many rows with many varying length columns
  - Implements CPU tuning recommendation to place fixed length columns ahead of varying length columns
  - Provides for direct access to each varying length column
  - In some few cases may lead to increased logging volume
- On by default in DB2 9 (NFM) and applies to all table space types
- Application transparent
- REORG and LOAD REPLACE utilities override KEEPDICTIONARY during first time migration when using data compression of variable-length rows
- RRF may not compress as well as BRF (Basic Row Format) if the row has many small VARCHARs (e.g. VARCHAR(1))

# Reordered Row Format (RRF) …

- Recommend applying APARs PK78958, PK78959, PK79127, PK87348, PK85881
- SPRMRRF opaque online changeable zparm
  - Default value is SPRMRRF=ENABLE
- CREATE honors SPRMRRF, except XML table spaces are always RRF
- Adding a partition honors SPRMRRF except:
  - For existing table space with Editproc, always use the format of existing partitions

# Reordered Row Format (RRF) …

- REORG and LOAD REPLACE
  - If SPRMRRF=ENABLE, converts from BRF to RRF
  - If SPRMRRF=DISABLE, does not convert
  - Compression attribute does not affect the choice of format
  - A new utility keyword ROWFORMAT (RRF or BRF) may be specified to choose the format
    - Requires NFM, no affect on catalog, directory, LOB, XML, UTS participating in a CLONE relationship

# More Efficient Workfiles Usage

## V8

- 4K-page workfiles are used whenever row is smaller than 4KB
- Since work file access is often sequential, using larger page size can be more efficient
- E.g. for 2050-byte records:
  15 records on one 32K page
  vs.
  8 records on eight 4K pages

## V9

- 32K-page workfiles are used much more aggressively
- 4K-page workfiles are now used only for small records
  - where the limit of 255 rows per page results in waste of space
  - e.g. over 90% wasted space on 32K page for 10-byte records
- Recommendations
  - ✓ Assign a larger 32K workfile buffer pool
  - ✓ Allocate more 32K workfile space
  - ✓ If 4K workfile buffer pool activity is significantly lower, then the corresponding buffer pool size and work file datasets can be reduced.
  - ✓ Monitor new statistics on how often more optimal 32K workfiles ran out and 4K workfiles had to be used instead, or vice versa

# Large tape block sizes

- Specify TAPEBLKSZLIM=262144 in DEVSUPxx parmlib member
- Use system determined block size
  - Don’t specify BLKSIZE parameter
- Use for image copies and all input/output files used by LOAD and UNLOAD
- Up to 170 MB/sec using IBM 3592 tape drives, faster than DASD

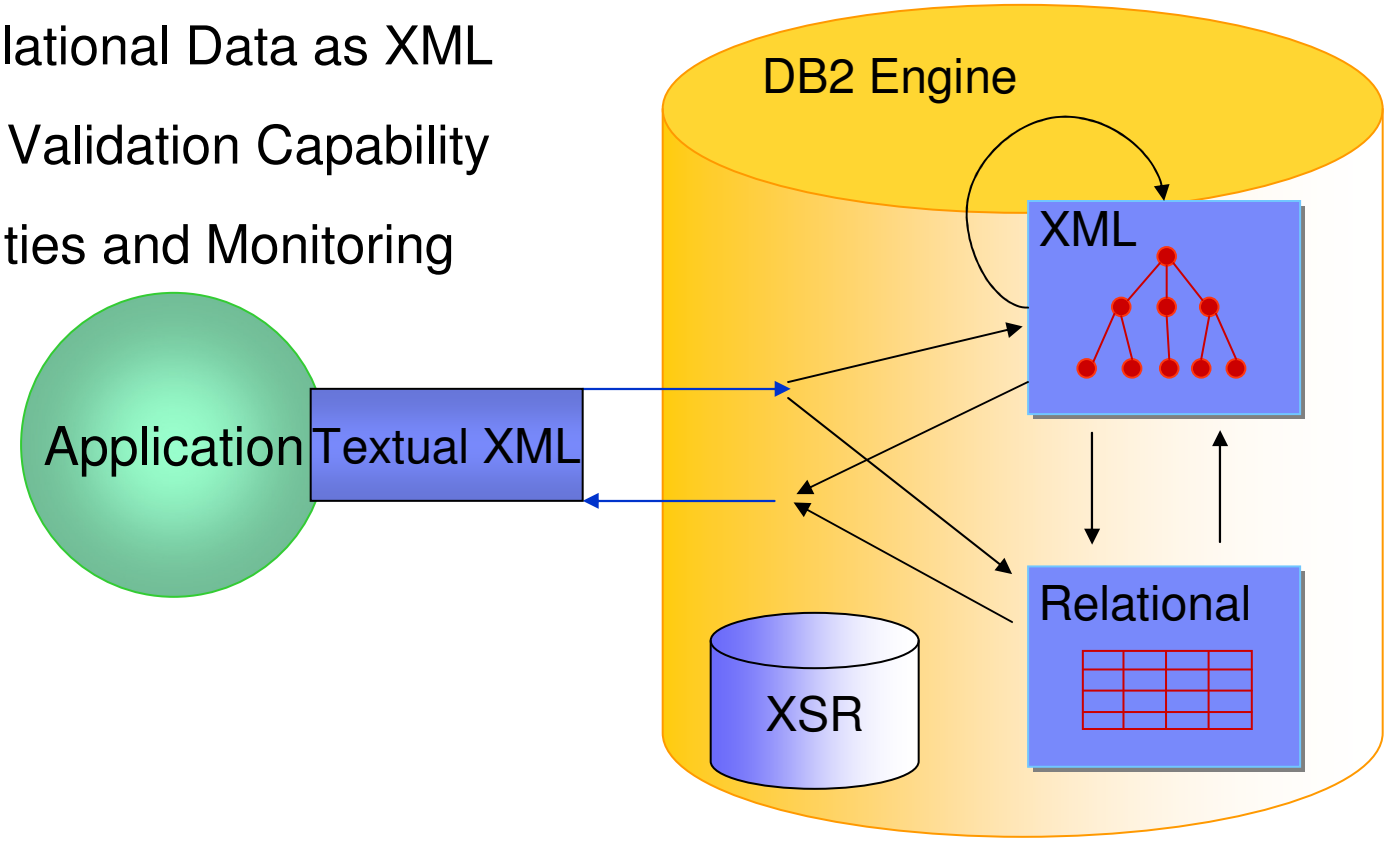
# BACKUP, RESTORE and RECOVER

- BACKUP DUMP
  - DUMP keyword causes HSM to dump the current backup or previous backup to tape
- BACKUP FINCREMENTAL
  - Creates a persistent backup, and each subsequent use of that backup incrementally updates it
  - Only one persistent backup allowed. Best used with VERSIONS=1 for the SMS Copy Pool
  - Since the backup is updated, the backup cannot be use to recover to a point in time prior to the last increment
  - To recover to an older point in time, you can restore from a different backup
- RECOVER
  - Can recover an individual object from a system level backup

# LOBs and XML

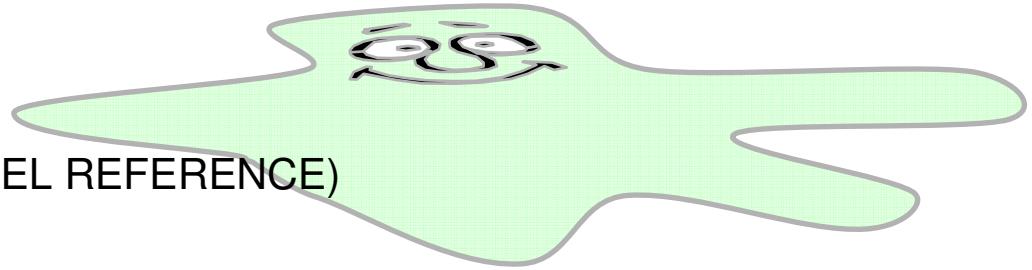
# What is pureXML?

- XML native storage
- Query Capability
- Indexing Inside of XML Documents
- Publishing Relational Data as XML
- XML Schema Validation Capability
- Database Utilities and Monitoring



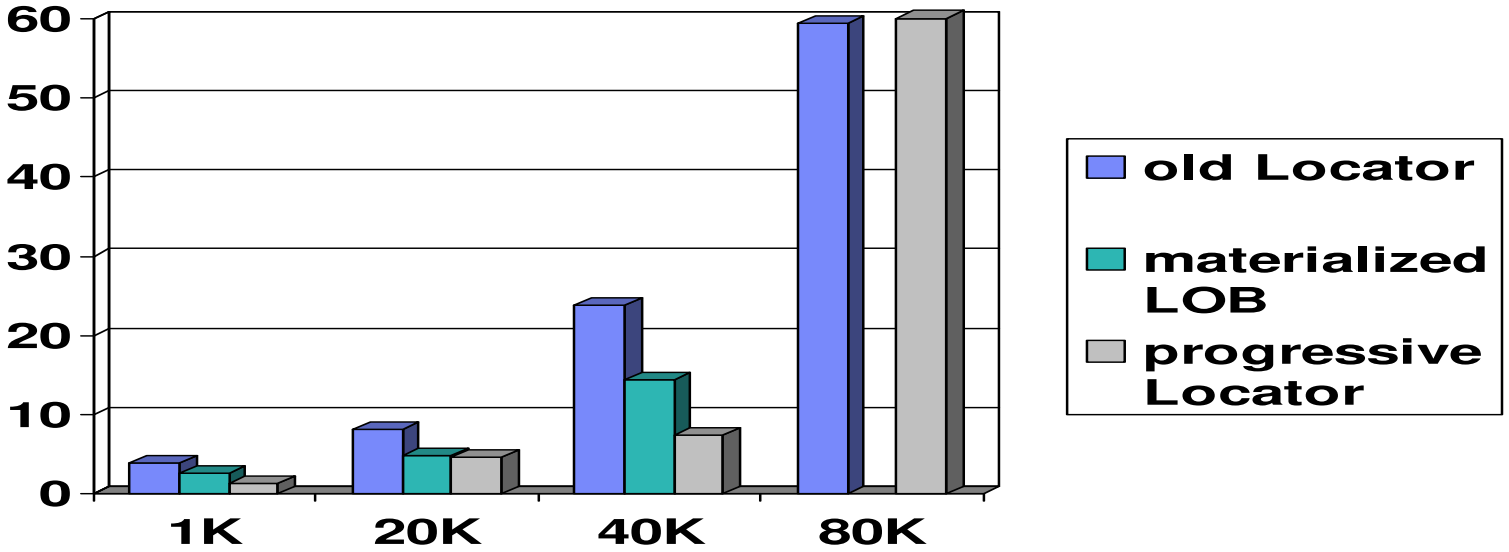
# LOB Improvements: Faster & Easier

- Progressive Streaming for LOB Locator Values
  - DB2 uses LOB size to determine whether to send LOB data to Java or DB2 CLI clients in one (<32KB), in chunks (<1MB) or as LOB locator (>=1MB) [Transparent to application using LOB locators]
- Reduced use of LOB locks
- Utility Changes
  - REORG LOB reclaim space (SHRLEVEL REFERENCE)
  - Logging for > 1GB LOBs
  - Online CHECK LOB and DATA using Flashcopy
  - Better performance of LOB file reference variables
- Implicit object creation (SET CURRENT RULES = 'STD')
- SQL support for LOB file reference variables
  - SQL supports sequential data sets on DASD or tape
- FETCH CONTINUE



# LOB Streaming Performance Results

- Elapsed time to retrieve 1000 CLOB values of varying size streamBufferSize=70000
- Progressive Locator processing
  - 1K LOB send inlined in query result
  - 20K and 40K LOB send chained to query result
  - 80K LOB send via locator



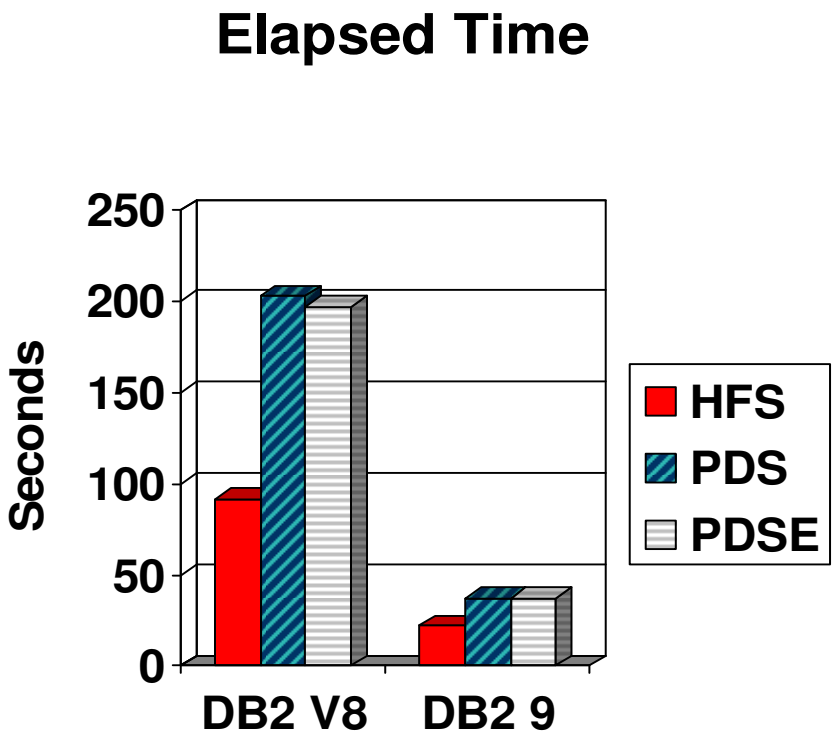
# New Technique to Retrieve LOBs: FETCH CONTINUE

- Why use it?
  - A method of reading a large LOB sequentially, without needing to use locators or SUBSTR function
    - No need to free a locator variable
    - DB2 returns the LOB/XML length and indicates if there was truncation
  - A fast method of scanning rows when
    - most LOBs can be skipped, or
    - most LOBs are small enough to fit in a host variable, or
    - only the first piece of most LOBs need to be examined.
- Cannot use with multi-row fetch with FETCH CONTINUE
- Use locator variables to randomly position within a LOB
- How to use
  - FETCH WITH CONTINUE to fetch base row information
  - FETCH CURRENT CONT[illegible] if LOB was truncated

# File Reference Variables

- Unload and Load utilities limited to PDS/PDSE and USS file systems (HFS or zFS)
- Recommend USS over PDS or PDSE
  - USS file systems can be multi-volume
  - USS file systems have no limitation on directory size
    - PDSE limited to 512K members
    - PDS limited to 64K tracks
  - USS file systems are faster for small LOBs than PDS or PDSE

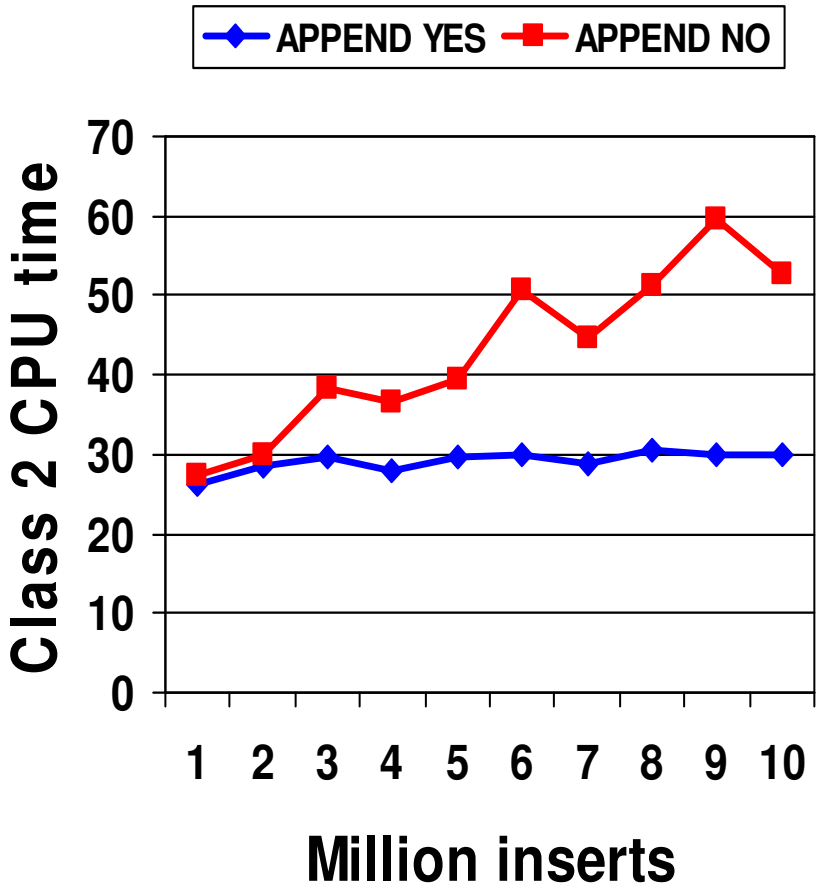
# Unload using File Reference Variables for 20,000 x 200 byte LOBs

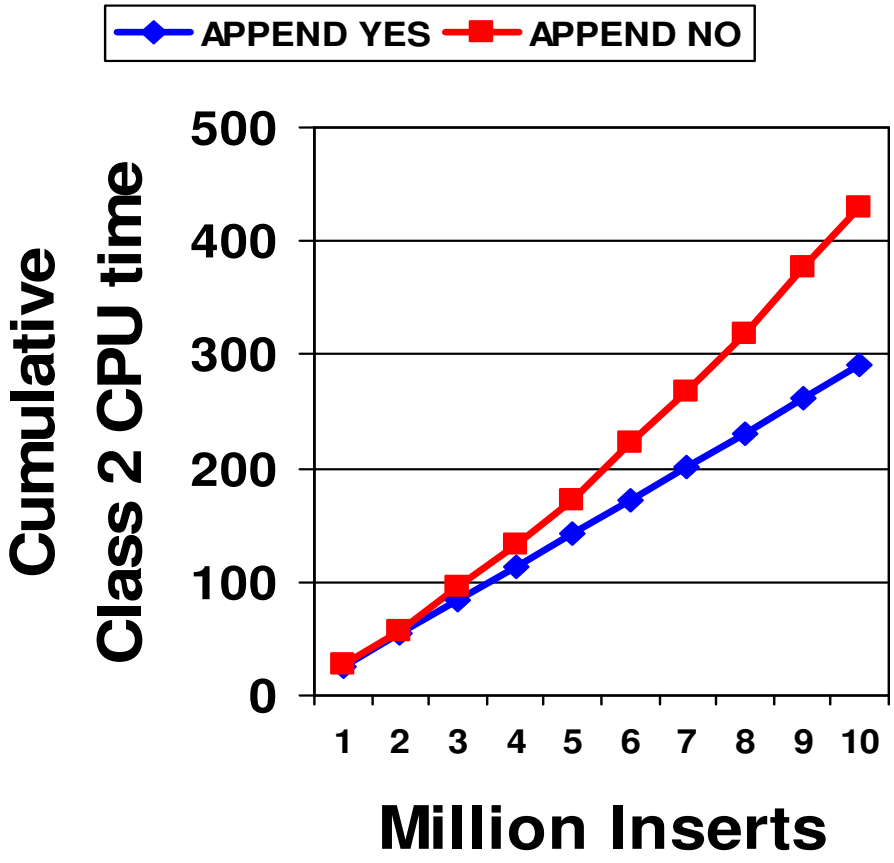



- **DB2 9 is 4 times faster than V8**
- **HFS and zFS are faster than PDS/PDSE (especially for Unload, not so much for Load)**

- With PK83992 (V8 and 9), SYSREC is much faster than FRV, but to use SYSREC the row size including LOBs cannot exceed the block size.

# LOB APPEND using small LOBs

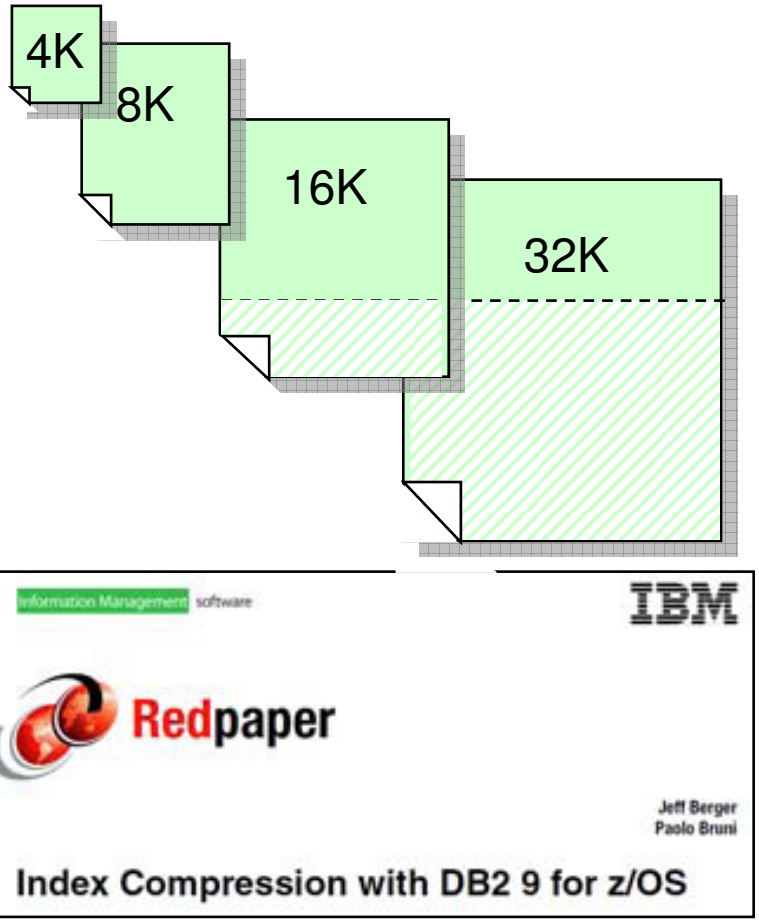


# Index Management

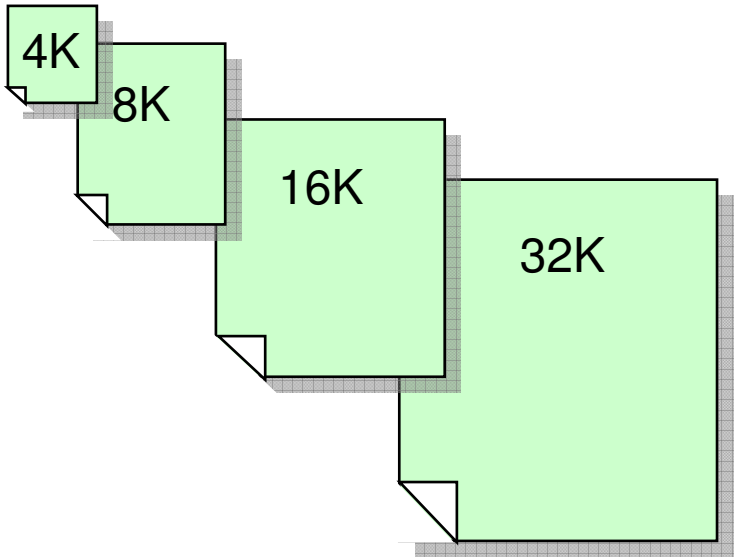
# Indexing enhancements

- Larger index pages
- Index compression provides page-level compression
  - Data is compressed to 4K pages on disk
  - 32K/16K/8K pages results in up to 8x/4x/2x disk savings
  - No compression dictionaries
    - Compression on the fly
    - No LOAD or REORG required
- Rebuild Index SHRLEVEL CHANGE



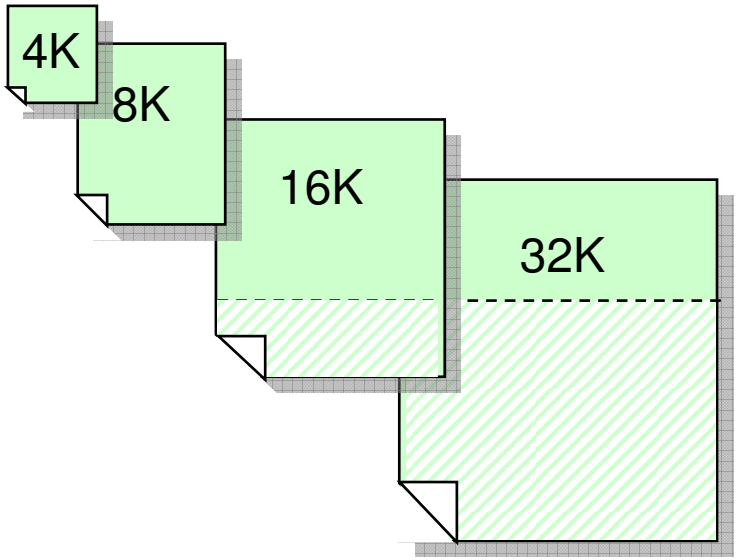
# Large index page size

- Benefits
  - Fewer page splits
  - Potentially fewer index levels
  - Faster sequential I/O
- Large page size may cause
  - Worse buffer hit ratio
  - More deferred writes, which could be especially bad with remote writes



# Index Compression

- VSAM CI size is always 4K
- 8K page size is usually best.
- Larger page size may cause
  - Worse buffer hit ratio
  - More deferred writes, which could be especially bad with remote writes
- Run DSN1COMP before using index compression to see how different page sizes affect the compression and storage usage

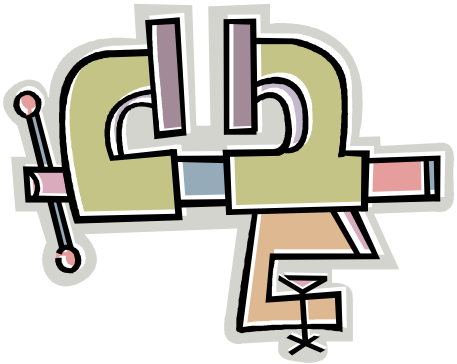


# Comparing table space vs. index compression

| | Table space | Index | Remarks |
|---|---|---|---|
| Unit of compression | Row | Page | Index non-leaf pages are not compressed, but that's typically less than 5% of the entire index |
| Where is data compressed | on disk, in buffer pools, in logs | on disk only | |
| compression timing | at insert and fetch | at I/O | Hence, CPU overhead affected by BP hit ratio. Larger buffer pools strongly recommended for compressed indexes. Do not use page-fix for these buffer pools |
| Hardware assist | Yes | No | Index compression uses sophisticated software compression algorithms |
| Page size restrictions | No | Only for 8K, 16K and 32K-page indexes | Page size on DASD:<br>▪ for table spaces, equivalent to page size in the associated buffer pool<br>▪ for indexes, always 4K |
| Compression dictionary used | Yes | No | |
| When does compression start | After the first Reorg or Load | At the first insert or update | |
| Compression CPU cost reported in | Accounting | Accounting & DBM1 SRB Statistics | No changes in accounting CPU time if index pages brought in by prefetch |
| Typical ratio for compression | 10 – 90% | 25 – 75% | Maximum CR limited by index page size: 50% for 8K, 75% for 16K and 87.5% for 32K<br>Use DSN1COMP to predict compression ratio |

# Compression Rules of Thumb

- Calculate disk space used by indexes
- Avoid high IO situations or add memory to compensate
- Choose large indexes, over 10 MB
  - Especially if long index keys (over 400 bytes), or padded indexes
- Use DSN1COMP to estimate compression
- Where compression is
  - Over 45% and under 70%, use 8K pages
  - Over 70% and under 85%, use 16K pages
  - Over 85% use 32K pages
- Test and stage in implementation



# Identifying Unused Indexes

## V8

- In order to improve the performance of different types of queries, some customers create many indexes just in case one of them can be useful.
- However, there is a significant cost in maintaining such indexes during Insert, Delete, Load, Reorg, ...
- Some of the indexes are no longer needed and they could be dropped, but how can we be sure that they have not been used for a very long time?

## V9

- For each index, a new real-time statistics column (LASTUSED) is maintained in table SYSIBM.SYSINDEXSPACESTATS
- It is updated whenever the index is used in
  - SELECT/FETCH
  - searched UPDATE/DELETE
  - Referential Integrity checking
- The column is not updated for INSERT, LOAD, etc.
- If an index has not been used for a period that covers the entire applications life-cycle (e.g. do not forget periodic processes such as period closing), it can be safely dropped
- Be careful with indexes that enforce uniqueness

# Virtual Indexes a.k.a. 'What If' Indexes

## V8

How to determine that a new index would benefit a given **dynamic SQL** query?

How to determine that dropping an index will not negatively affect a given query?

- In many cases predicting based on modeling is not reliable due to query complexity
- Indiscriminate adding of indexes creates permanent overhead for most operations (SQL and utilities)
- Creating a new index is obtrusive for concurrent operations
- Using a test system for experimenting lacks potentially crucial environmental factors that affect access path selection

## V9

- Virtual i.e. hypothetical indexes can be specified and made visible to statement EXPLAIN STATEMENT FOR
- Table DSN_VIRTUAL_INDEXES is used to specify virtual indexes
  - Table columns include selected columns from SYSINDEXES and SYSKEYS
  - Users need to create the table manually, unless tooling such as Index Advisor does it automatically. Appropriate script is provided.
  - To create/drop an index, the table needs to be populated with a row that provides an appropriate description of index
- At EXPLAIN time, during query optimization, the virtual indexes compete with regular indexes on the tables in a cost-based fashion and the dropped indexes are not considered

DSN_VIRTUAL_INDEXES Table

| | |
|---|---|
| TBCREATOR | Authorization ID of owner (or schema in V9) of table on which the index is being created/dropped |
| TBNAME | Name of the table on which the index is being created or dropped |
| IXCREATOR | Authorization ID (or schema in V9) of the owner of the index |
| IXNAME | Name of the index |
| ENABLE | Whether this index specification will be processed ('Y') or not ('N'). |
| MODE | Whether the index is being created ('C') or dropped ('D') |
| UNIQUERULE | Whether the index is unique: D for No (duplicates are allowed); U for Yes |
| COLCOUNT | The number of columns in the key |
| CLUSTERING | Whether the index is clustered ('Y' or 'N') |
| NLEAF | Number of active leaf pages in the index. If unknown, the value must be -1. |
| NLEVELS | Number of levels in the index tree. If unknown, the value must be -1. |
| INDEXTYPE | The index type: '2' - NPSI; 'D' - DPSI |
| PGSIZE | Size, in bytes, of the leaf pages in the index: 4K, 8K, 16K, 32K |
| FIRSTKEYCARDF | Number of distinct values of the first key column. If unknown, the value must be -1. |
| FULLKEYCARDF | Number of distinct values of the key. If unknown, the value must be -1. |
| CLUSTERRATIOF | Clustering ratio. . If unknown, the value must be -1. |
| PADDED | Indicates whether keys within the index are padded for varying-length column data ('Y' or 'N') |
| COLNO1 | Column # of the first column in the index key |
| ORDERING1 | Ordering ('A' or 'D') of the first column in the index key |
| ... | ... |
| COLNO64 | Column # of the last column in the index key. Needs to be populated only when # index keys = 64 |
| ORDERING64 | Ordering ('A' or 'D') of the last column in the index key. |

# SQL

# Native SQL Procedures

- Potential for significant reduction in CPU resource consumption by avoiding
  - Overhead of stored procedure invocation overhead
  - Overhead of roundtrip between WLM and DBM1 address spaces for each SQL call
- Short running SQL procedure could achieve up to an 40% ITR improvement
- But little or no improvement for long-running SQL procedure
- zIIP-eligible if DRDA as it runs in DBM1, not WLM, address space under DDF enclave SRB
- Easy to code, develop and manage
- Conversion from external SQL procedures to native SQL procedures is easy

# SELECT FROM UPDATE/DELETE/MERGE

## V8

**SELECT FROM INSERT**

Retrieves columns values created by INSERT in a single SELECT statement including:

- Identity columns
- Sequence values
- User-defined defaults
- Expressions
- Columns modified by BEFORE INSERT trigger
- ROWIDs

Avoids possible expensive access path that separate SELECT might be using

## V9

**SELECT FROM INSERT**
**UPDATE**
**DELETE**
**MERGE**

One SQL call to DB2 modifies the table contents and returns the resultant changes to the application program.

E.g. we can now code destructive read from a table when a SELECT FROM DELETE statement is included. This feature is particularly useful when a table is used as a data queue.

# ORDER BY and FETCH FIRST in Subselect

## V8

ORDER BY and FETCH FIRST can be specified only as part of select-statement, i.e. one can write:

```
SELECT * FROM T1
  ORDER BY C1
  FETCH FIRST 1 ROW ONLY
```

but not the following:

```
INSERT INTO T2
 (SELECT * FROM T1
   ORDER BY C1
   FETCH FIRST 1 ROW ONLY)
```

This will give a syntax error

## V9

The restriction has been removed – the clauses can be specified in either a subselect or fullselect.

Interesting example: a loop over the statement followed by a COMMIT deletes rows without acquiring too many locks

```
DELETE FROM T1 X
  WHERE EXISTS
   (SELECT * FROM T1 Y
      WHERE X.KEY1=Y.KEY1
        AND X.KEY2=Y.KEY2
        AND delete_predicate
     FETCH FIRST 10000
                    ROWS ONLY
```

# New Data Types

- BIGINT, 8 byte integer
- BINARY, 1 to 255 bytes
- VARBINARY, 1 to 32704 bytes
  - Unlike FOR BIT DATA the new BINARY and VARBINARY use x'00' as a padding character
  - Comparison rule:
    - If two strings are equal up to the length of the shorter string , the shorter string is considered less than the longer string.
- DECFLOAT
  - IEEE 754r number with a decimal point. The position of the decimal point is stored in each decimal floating-point value. The maximum precision is 34 digits.

# Skipping Locked Rows

## V8

Application does not scale well due to increased lock contention.

Application semantics requires committed and **available** rows only.

An example of such an application is a messaging system:

- only committed and available messages can be processed
- those locked at the time will be processed later.

## V9

New SQL clause: SKIP LOCKED DATA

Applies to:

- select-statements, SELECT INTO, PREPARE, searched UPDATE, searched DELETE, UNLOAD utility
- Isolation levels CS or RS
- Row or page level locking

It allows a transaction to skip over rows that are incompatibly locked by other transactions, without being blocked.

An interesting usage scenario is serializing access to any kind of object:

- Create a table and insert a row for each object to be controlled
- Code: SELECT … FOR UPDATE OF … SKIP LOCKED DATA
- The object unavailability is identified by return code +100 (without any wait)

# APPEND

## V8

All of the following applies:

- Critical, high insert rate workload needs better performance and all the conventional tuning steps have already been applied.
- Clustering is either not beneficial or more frequent reorganizations are acceptable
- *MC00* insert algorithm is still not fast enough or the prerequisites cannot be satisfied:
  - MEMBER CLUSTER
  - FREEPAGE=PCTFREE=0

## V9

CREATE TABLE … APPEND YES | NO

ALTER TABLE … APPEND YES | NO

- APPEND YES avoids CPU cost of space search at the cost of DASD space
- After populating with the APPEND option, space may be reclaimed or reused by Reorg or altering APPEND off
- Make sure PK81471 is applied
- Note that MC00 is still valid, but make sure that PK81470 is applied
- Apply UK41212 for LOB Append

# TRUNCATE TABLE

## V8

Alternative way of deleting the entire table is needed for any of these reasons:

- DELETE without WHERE clause is not fast enough as the table includes delete triggers
- Using LOAD REPLACE with empty input data set (even when called via DSNUTILS) is not DBMS agnostic
- Storage occupied by deleted rows should be released faster

## V9

New DML statement:

**TRUNCATE** table
DROP | REUSE STORAGE
IGNORE | RESTRICT DELETE TRIGGERS
IMMEDIATE

Under the cover it's DELETE without WHERE clause, but without delete triggers processing overhead.
Therefore it is fast for tables in segmented and universal table spaces for which there are no CDC, MLS and VALIDPROC enabled attributes.

# MERGE Statement

## V8

For a set of input rows update the target table when the key exists and insert the rows for which keys do not exist.
E.g.

- For activities whose description has been changed, update the description in table **archive**.
- For new activities, insert into **archive**.

Prior to V9 this has been coded as a loop over conditional INSERT and UPDATE statements

## V9

```
MERGE INTO archive AR
  USING VALUES (:hv_activity, :hv_description) FOR :hv_nrows ROWS
    AS AC (ACTIVITY, DESCRIPTION)
  ON (AR.ACTIVITY = AC.ACTIVITY)
  WHEN MATCHED THEN UPDATE SET DESCRIPTION = AC.DESCRIPTION
  WHEN NOT MATCHED THEN INSERT (ACTIVITY, DESCRIPTION)
                                VALUES (AC.ACTIVITY, AC.DESCRIPTION)
  NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

# MERGE Example

```
MERGE  INTO account AS T
USING VALUES (:hv_id, :hv_amt) FOR 5 ROWS AS S (id, amt)
ON T.id = S.id
WHEN MATCHED          THEN UPDATE SET balance = T.balance + S.amt
WHEN NOT MATCHED   THEN INSERT (id, balance) VALUES (S.id, S.amt)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

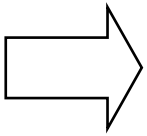
MERGE

VALUES AS S

| id | amt |
|---|---|
| 1 | 30 |
| 5 | 10 |
| 10 | 40 |
| 5 | 20 |
| 1 | 50 |

account AS T

| id | balance |
|---|---|
| 1 | 1000 |
| 10 | 500 |
| 200 | 600 |
| 300 | 300 |
| 315 | 100 |
| 500 | 4000 |



account AS T

| id | balance |
|---|---|
| 1 | 1080 |
| 5 | 30 |
| 10 | 540 |
| 200 | 600 |
| 300 | 300 |
| 315 | 100 |
| 500 | 4000 |

# DB2 9 for z/OS at a glance

| | |
|---|---|
| Application Enablement | • pureXML<br>• Optimistic locking for WebSphere<br>• LOB performance, usability<br>• Native SQL procedural language<br>• SQL improvements that simplify porting |
| RAS, Performance, Scalability, Security | • More online schema changes<br>• Online REBUILD index, Online REORG improvements<br>• Clone tables<br>• Trusted contexts and ROLES<br>• Parallel sysplex clustering improvements<br>• 64-bit virtual storage enhancements |
| Simplification, Reduced TCO | • Index compression<br>• Partition by growth tables<br>• Package stability<br>• Volume based backup / recovery<br>• Automatic object creation |
| Dynamic Warehousing | • Many SQL improvements<br>• Dynamic index ANDing<br>• Histogram statistics<br>• New built-in OLAP expressions<br>• Optimizer service center |

# References

- DB2 9 for z/OS Migration Planning and Experience, John Campbell and Florence Dubois
  - ftp://ftp.software.ibm.com/software/systemz/pdf/db2techconf2009/DB2Conf_DB2_9_for_zOS_Migration_Planning_and_Experience.pdf

# Acknowledgements

Namik Hrle, IBM Distinguished Engineer

# DB2 9 in IBM Redbooks Publications

1. DB2 9 Technical Overview SG24-7330
2. DB2 9 Performance Topics SG24-7473 updated Dec. 2009
3. DB2 9 Stored Procedures SG24-7604
4. Index Compression with DB2 9 for z/OS redp4345
5. SQL Reference for Cross-Platform Development
6. Enterprise Database Warehouse, SG24-7637
7. 50 TB Data Warehouse on System z, SG24-7674
8. New Tools for Query Optimization SG24-7421
9. LOBs with DB2 for z/OS SG24-7270
10. Deploying SOA Solutions SG24-7663
11. Enhancing SAP - DB2 9 SG24-7239
12. SAP Application on Linux z SG24-6847
13. Best practices SAP BI - DB2 9 SG24-6489-01
14. Data Sharing in a Nutshell, SG24-7322
15. Securing DB2 & MLS z/OS SG24-6480-01
16. Data Sharing: Distributed Load Balancing & Fault Tolerant Configuration redp4449
17. Considerations on Small & Large Packages redp4424
18. Backup and Recovery Considerations redp4452
19. Powering SOA with IBM Data Servers SG24-7259
20. Packages Revisited, SG24-7688
21. Data Studio V2.1 Web Services redp4510
22. Ready to Access Solid-State Drives redp4537
23. Distributed Functions SG24-6952
24. Buffer Pool Monitoring & Tuning redp4604
25. Securing & Auditing Data SG24-7720
26. Serialization and Concurrency SG24-4725-01 new
27. Utilities SG24-6289-01 draft