Table and Index Design: DB2 9 Strategies

Susan Lawson

YL&A

Yevich, Lawson & Assoc. Inc.
3309 Robbins Road PMB 226
Springfield, IL 62704

www.ylassoc.com
www.db2expert.com

Disclaimer PLEASE READ THE FOLLOWING NOTICE

- The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.
- The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an “As is” basis without any warranty, either expressed or implied.
- The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client’s ability to evaluate and integrate them into the client’s operational environment.
- While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.
- Clients attempting to adapt these techniques to their own environments do so at their own risks.
- Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.

Abstract

This presentation takes a look at table and index design strategies we have practiced over the years and see how they are impacted by DB2 9 for z/OS. We also look at how we can move forward with current designs and also be able to take advantage of the new features.

Challenges with Table and Index Design

- Tables
  - Managing growth
  - Providing availability
  - Improving mass processes
- Indexes
  - Managing growth
  - Providing availability
  - Controlling splits
  - Minimizing the number of indexes

Table and Index Design in DB2 9 - Outline

- Append processing
- No log tables
- Truncate Table
- Clone Tables
- Index on Expression
- Index Compression
- Larger Index Page Sizes
- Universal Tablespaces
  - Partition by Range
  - Partition by Growth
- Reordered Record Format

Logging Issues

- As databases growth and application volumes increase, logging increase
- Can spread logging over multiple data sharing members
- Can minimize amount of data logged via table designs
- Some tables do not necessarily require logging
  - Materialized Query Table
  - Rebuilt Summary Tables
- Logging can be an issue for mass insert/update processing
  - Logs can quickly fill up an become a bottleneck

NOT LOGGED Tablespaces

- Logging can be suppressed for
  - Base tablespaces (applies to all partitions)
  - XML tablespaces
  - Including associated indexes
    - Inherit attribute of tablespace
- Changed via ALTER or CREATE
- Defined at the tablespace level
  - LOGGED (or LOG YES)
    - Modifications are recorded on log
    - Default, except for workfile tablespaces and LOBs >1 GB
  - NOT LOGGED (or LOG NO)
    - Modifications are not recorded on log
  - Tablespace option, not a table option
  - Can be defined for each partition
  - Switching between LOGGED/NOT LOGGED will result in COPY pending

```
CREATE TABLESPACE
….LOG NO

ALTER TABLESPACE
…..LOG NO
```

NOT LOGGED Tables (Cont..)

- NOT LOGGED will not necessarily improve performance of subsystem
  - Will be the exception
  - Do not sacrifice recoverability
    - Use if data is duplicated or can be regenerated from source
- XML tablespaces will have same attributes as base
- LOB tablespaces could be different (base is logged, LOB is not)
- Updating a NOT LOGGED table space will place tablespace in Informational Copy Pending (ICOPYP) status
- If uncommitted work in a NOT LOGGED table space needs rolled back
  - Tablespace will be placed in Recovery Pending (RECP) status
  - Data can be recovered, but changes are not
  - Will need to recover to a recoverable point
- Recommended to commit often
- Performance
  - In some cases parallel INSERTs in a not logged tablespace may see a performance improvement of about 10%

NOT LOGGED Tables ..When to Use

- Data can be regenerated from source
- Massive updates
  - Could turn logging off for the mass updates, then turn back on for infrequent updates
- Before an online load resume
  - If load fails, could recover from previous image copy
- Possibly used during modifications after a load
  - Do load and modifications without logging
- Materialized Query Tables
  - Not log changes to MQTs
  - They can be easily rebuilt with a REFRESH
- Summarized Tables
  - Maybe not log these changes since the data can be derived from other source
- Temporary Table for BI
  - Are built, used and destroyed...no need to log

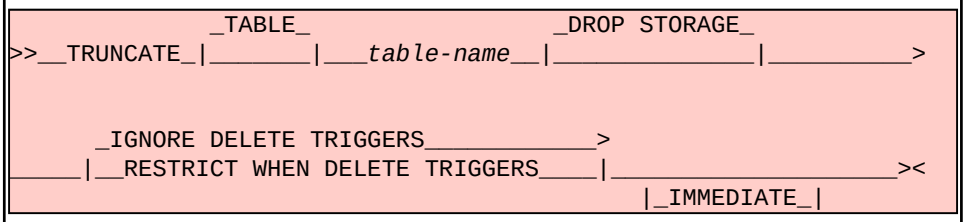
NOT Logged Tables - Considerations

- PCLOSEN and PCLOSET will be set to 1
  - Influences when to switch the tablespace from read-write to read-only
  - PCLOSEN with 1
    - When a checkpoint occurs all read-write no logged tablespaces that are not currently in use are converted to read-only
  - PCLOSET with 1
    - 1 minute after the commit of the last update it is converted from read-write to read-only (if checkpoint has not already done this)
  - DB2 externalized unprotected modifications to data from bufferpool
- Recommended to commit often
- Performance
  - In some cases parallel INSERTs in a not logged tablespace may see a performance improvement of about 10%
  - You must test this to see if there is actual benefit!

Truncate Table

- Prior to DB2 9
  - To empty a table you have to either
    - Do a mass delete
    - Or use LOAD Utility with REPLACE REUSE and LOG NO NOCOPYPEND
- Issues
  - Delete triggers introduce problems with DELETEs
    - Requires a DROP and recreation of the delete trigger to empty the table
      - To avoid firing the trigger
  - LOAD REPLACE works on a table space level (not table)
    - If multi-table tablespace, then this is a problem
- In DB2 9
  - The TRUNCATE statement addresses these problems
    - Deletes all rows for either base tables or declared global temporary tables

TRUNCATE TABLE Syntax




```
TRUNCATE TABLE ACCOUNTS
REUSE STORAGE
IGNORE DELETE TRIGGERS
IMMEDIATE;
```

Above statement will remove all records from the Account table and will ignore all delete triggers and will reserve the storage for immediate usage. A rollback would not have any effect on this table.

Truncate Table

- Effective way to delete all rows
  - More flexibility and shorter path length than mass DELETE
  - Will not fire delete triggers
  - Will not alter definitions in catalog
  - Will no go through current commit phase
  - Allows for option to reuse deallocated storage
- Tablespace can be
  - Simple or segmented
  - Partitioned
  - Universal
  - If LOB or XML columns, corresponding tablespaces and indexes are also emptied
- Benefits will really only be seen on tables with DELETE triggers
  - Else, performance will be the same as mass delete is today
  - But does have IMMEDIATE option to allow space to be immediately reused by inserts in same unit of work
- Indexes are also truncated

```
TRUNCATE TABLE PARTS IGNORE
DELETE TRIGGERS
DROP STORAGE;
```

Universal Tablespace

- Combination of segmented and partitioned tablespace schemes
- Brand new tablespace type
  - Others will still exist and be supported
  - Based upon partition ranges
  - Always defined as large
- Benefits include
  - New partition-by-growth functionality
  - Better space management for varying length rows
    - Segment space map has more information about freespace
  - Improved mass delete performance
  - Tablespace scans localized to segments
  - Immediate reuse of segments after DROP or mass delete
  - Max table size of 128TB and supports DSSIZE
  - Partition level operations and parallelism
- Performed by specifying both SEGSIZE and NUMPARTS on CREATE
  - Will contain a single table
- Not compatible with MEMBER CLUSTER

Partition By Growth Universal Tablespaces - Benefits

- Better space management
- Improved delete performance than traditional partitioned table spaces
  - Due to their segmented space organization
- No need to define a partitioning key to bound the data within a table space
- Partitioning is managed by DB2
  - Depending on the DSSIZE and MAXPARTITIONS
- Space must be STOGROUP defined
  - Only non-partitioning indexes (NPIs) are allowed to be defined on PBG UTS
- Free space, caching, define, logging, and TRACKMOD
  - Same for each partition
- Each partition will have the same compression dictionary

```
CREATE TABLESPACE TS01TS IN
TS01DB USING STOGROUP SG1
DSSIZE 2G
MAXPARTITIONS 24
LOCKSIZE ANY
SEGSIZE 4;
```

Partition By Growth Universal Tablespaces

- When you define a PBG UTS, DB2 starts out with one partition
  - When a partition is filled
    - DB2 automatically defines the next partition and starts using it (similar to extending a non-partitioned table space)
  - Table space acts like a non-partitioned table space but with the added benefit of being able to grow to a maximum of 128 TB in size (for 32 KB page size)
- Utility benefits
  - Running in parallel (at the partition level)
  - Except LOAD, which must be run at the table space level
  - Running a REORG will eliminate holes (deleted rows) and condense the entire table space
    - It is possible to have empty partitions at the end of the table space
- No need to ALTER ADD PART, ALTER ROTATE PART, or DROP PART to manage the partitions
  - Not allowed to issue these statements against a PBG UTS

Size Limitations with Increased Number of Partitions

- While 4096 partitions gives us many opportunities to support better designs, there are some limitations
- The maximum number of partitions a tablespace can have is dependent on
  - The DSSIZE, the page size and the total tablespace size
  - Page size affects table size because it affects the number of partitions allowed

Max Part	PageSize	DSSIZE	Max TS Size
4096	4KB	4GB	16TB
256	4KB	64GB	16TB

  - Page size and DSSIZE are still not ALTERable
- Can only be up to 5 LOBs on a table if 4096 partitions are to be supported
  - One LOB would take 12,288 objects to support on 4096 part table – only total of 65,535 object allowed in a database

Still an issue in DB2 9!

Clone Table Support

- Ability to generate a table with the same attributes of one that already exist on the current server

```
ALTER TABLE base-name ADD CLONE clone-name
```

- Created in the same tablespace
  - Will be structurally the same (column names, data types etc)
  - Will be referred to by different names and may have different data
  - Will have same internal object descriptors
- Created with same indexes, before triggers, check constraints, LOB objects etc
  - Referred to as clone objects
  - Referred to by the same names as base objects
- Creation or dropping of clone table does not effect the applications using the base table
  - No quiesce or invalidations

Clone Table Support - Definitions

- Table and index are created in different datasets using an instance number

catname.DSNDBx.dbname.psname.x0001.A001
catname.DSNDBx.dbname.psname.x0002.A001
Instance numbers

- Can easily exchange base and clone
  - Names remain the same, only underlying data and instance numbers change

```
EXCHANGE DATA BETWEEN TABLE
base-table AND clone-table
```

- Can have different authorities and views
- Can be created on existing base table
- Can only be created in a single tablespace that's DB2 managed
  - Single Table UTS
- Two ways to drop
  - DROP base or ALTER/DROP base
    - ALTER/DROP CLONE
      - x0002 instance remains

```
ALTER TABLE base-table
DROP CLONE
```

Clone Table – EXCHANGE DATA Command

- Switches underlying datasets associated with base and clone table
- Only the data instances will change
  - No data will be copied
- Transparent to the application
  - Does not require any application changes
- EXCHANGE DATA
  - Clone assumes the base objects' statistics
  - Allows bound static SQL to function without a rebind
    - No invalidations
- EXCHANGE DATA puts entry in SYSCOPY
  - ICTYPE='A' and STYPE='E'
- EXCHANGE DATA requires one of the following:
  - Ownership of both tables
  - Insert and delete privileges for both tables
  - DBADM authority on the database
  - SYSADM authority

EXCHANGE DATA BETWEEN
TABLE base-table AND clone-table

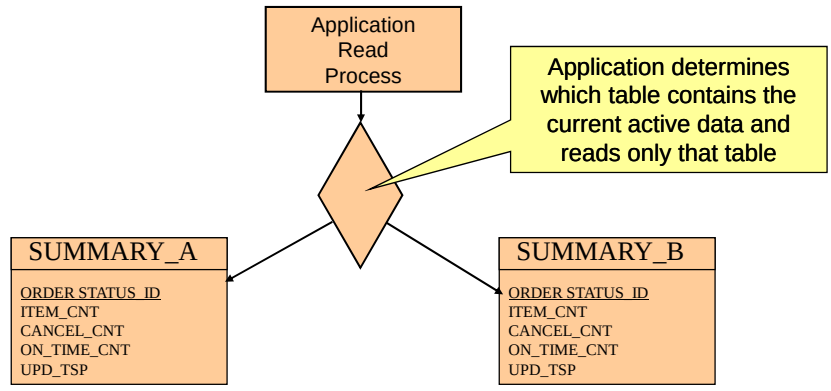
Will perform a drain!

Length depends on number… not size

Must be committed before use!!!

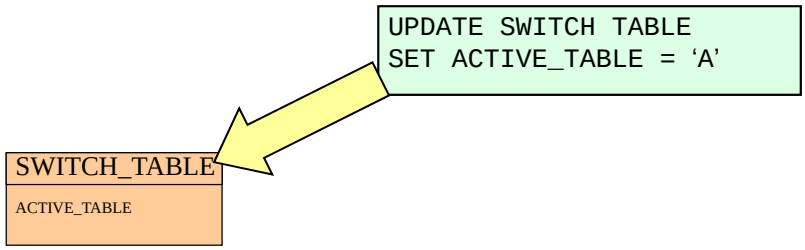
Table Switching Example Without Clone Table Support

- Table switching is an application based high availability solution
  - Very flexible
  - Has to be coded in application
  - Can use DB2 database features to simplify application design



Switch Table

- The Switch Table facilitates the table switching
  - One column, two values
    - “A” – Table A is active
    - “B” – Table B is active
- This allows for flexible control of active table
  - Column can be set by application or manually



Switching Inside a Query

- Switching Inside a Query
  - Utilizes a During-Join predicate
  - Reduces application logic
  - Reduces potential outages
    - Read of Switch and read of Active Table in one statement instead of two

```
SELECT COALESCE(A.ORDER_STATUS_ID, B.ORDER_STATUS_ID)
      ,COALESCE(A.ITEM_CNT, B.ITEM_CNT)
      ,COALESCE(A.CANCEL_CNT, B.CANCEL_CNT)
      ,COALESCE(A.ON_TIME_CNT, B.ON_TIME_CNT)
FROM   SWITCH_TABLE AS SW
LEFT JOIN
       SUMMARY_A AS A
ON SW.ACTIVE_TABLE = ‘A’
LEFT JOIN
       SUMMARY_B AS B
ON SW.ACTIVE_TABLE = ‘B’;
```

The first non-null value is returned. The outer join will supply nulls for the table that is not “active”

SUMMARY_A will only return the summary data when this join predicate is true

SUMMARY_B will only return the summary data when this join predicate is true

Table Switching Using Clone Tables

- DB2 9 simplifies this type of high availability applications
  - A clone can be created for a table
  - The EXCHANGE statement does the switch
- The refresh process is simplified
  - The clone table is truncated
  - Then fresh data is inserted
  - Then the exchange – online LOAD REPLACE!

```
ALTER TABLE SUMMARY_A
ADD CLONE     SUMMARY_B;
```

```
SELECT  A.TOT_AMT
FROM    SUMMARY_A AS A;
```

Returns the data from the base table. No complex SQL required!

```
TRUNCATE SUMMARY_B;
INSERT INTO SUMMARY_B
SELECT ….;
EXCHANGE DATA BETWEEN
SUMMARY_A AND SUMMARY_B;
COMMIT;
```

This script refreshes the data in the clone table, and then exchanges the clone and base tables. Now the base has become the clone and the clone the base! No outage!

Sequential Insert Design in V8

- Design key for sequential inserts at end
- No Freespace or Pctfree on data or sequential index
  - With no freespace you can reduce number of index levels and pages
- Can eliminate read I/Os
- No data read time needed
- Efficient use of IPROCs
- Index Lookaside on Clustering Index
- Less writes needed (less pages to be written)
- Will utilize deferred writes
- Minimizes read/write I/O, Getpages, and Locking
- MEMBER CLUSTER Setting Impacts Performance
- Index Design Affects Performance
- PQ87381 Needs to be Applied
  - Improved insert algorithm

New inserts at end together, Very efficient deferred write

Date	Acctno
12/01/02	1234
12/02/02	1234
12/05/02	5893
12/25/02	7892
01/01/03	7892
01/20/03	0414

APPEND on Insert

- Challenges still remain in V8
  - Clustering may not be beneficial
  - Maybe requirements of MEMBER CLUSTER and zero freespace cannot happen
- DB2 9 allows for fast inserts at the end of the table or partition
  - At the expense of organization
  - May account for some faster table growth
- Clustering can still be achieved by a reorg
  - With an explicit clustering index defined
- Reduces longer chain of spacemap page search as table grows

CREATE TABLE …APPEND YES/NO

ALTER TABLE…APPEND YES/NO

APPEND on INSERT (cont..)

- The APPEND processing option instructs DB2 to ignore clustering during SQL insert and online LOAD processing
- Rather than attempting to insert rows in cluster-preserving order
  - Rows are appended at the end of the table or appropriate partition
- For range-partitioned table spaces
  - The appropriate partition is the one dictated by the value in the row's partitioning column
- For partition-by-growth table spaces
  - The appropriate partition is any partition with space at the end
- Can benefit from the APPEND processing option even if you still rely on having a clustering index defined on your table
  - Can insert or LOAD your data quickly and regain the clustering order later by running REORG

APPEND on INSERT (cont..)

```
CREATE TABLE MYTB
(COL1 INTEGER,
COL2 DECIMAL(6,2),
COL3 CHAR(10))
APPEND YES
IN MYDB.MYTS;
```

Additional Index Page Sizes

- Prior to DB2 9
  - Size of an index page is limited to 4 KB
- Size of an index page limits the number of index keys that the index page can accommodate
  - Can cause contention in indexes that split frequently
- In DB2 9
  - Offers expanded index page sizes of 8 KB, 16 KB, and 32 KB
- An index page size greater than 4 KB accommodates more index keys per page
  - Can reduce the frequency of index page splits
  - Create ‘shallow’ indexes (less getpages and faster index scans)
- The larger page sizes are required with index compression

Additional Index Page Sizes (cont..)

- Can use the INDEXBP option on CREATE DATABASE
  - Can also be ALTERed via ALTER BUFFERPOOL
- Specify 4 KB, 8 KB, 16 KB, or 32 KB index buffer pools
  - If done via ALTER, the index is set REBUILD PENDING
- There are restrictions for indexes that use non-DB2 Managed datasets with incompatible CISIZEs:
  - The CISIZE must be either 4K or equal to the new page size
  - If you create an index in a dataset with a CISIZE of 8K and later try to alter to use 16K pages, it will fail with a -676
- Page size recommendations:
  - 4K for random inserts
    - Large page sizes are not good for index buffer pool hit ratio for random access
  - >4K for more sequential inserts

```
ALTER INDEX INDX1
BUFFERPOOL BP8K0
```

Larger Index Page Size Advantages and Cautions

- Index page > 4KB accommodates more index keys per page
  - Can reduce the frequency of index page splits
  - Create ‘shallow’ indexes
    - A larger page can result in a reduction in index levels
    - Fewer getpages and faster index scans
- A larger CI size can mean fewer index I/O’s
  - The DB2 linear VSAM dataset can have a CI size that matches the index page size
  - Example: an 8K CI size has the potential reduce the number of index I/O’s
    - Highly dependent on index access patterns of queries
- Caution
  - Could use more buffer storage for large pages
  - Larger I/O’s could mean longer I/O response
  - Always test or do a ‘proof of concept’ before changing index page sizes

Index Compression

- Indexes can be compressed in DB2 9
- Does not require a compression dictionary
  - Newly created indexes may begin compressing their contents immediately (first insert/update)
- Only compresses the data in the leaf pages
- Compression is based on eliminating repetitive strings (compaction)
  - Similar to what VSAM does with its indexes
- Index pages are stored on disk in their compressed format
  - Physical 4 KB index page on disk
  - Will be expanded when read from disk into 8 KB, 16 KB, or 32 KB pages
- To turn on compression for an index
  - Must be defined in an 8 KB, 16 KB, or 32 KB buffer pool
  - Can not be in a 4K bufferpool
- 25-75% compression is average

CREATE INDEX and ALTER INDEX COMPRESS YES/NO

Index Compression (cont..)

- ALTER INDEX COMPRESS YES/NO
  - Index is placed in Rebuild Pending status
- Compression takes place for the whole index (not at partition level)
- Use DSN1COMP to estimate the effectiveness
  - Do not choose a larger index page size than what is recommended
    - Can end up wasting valuable buffer pool space if you do
- Stop below 50% unused space
  - When choosing compressed index page size
- Recommended for applications ..
  - Who perform sequential insert operations with few or no delete operations
  - Where the indexes are created primarily for scan operations
- Random inserts and deletes can adversely affect compressions
- Can increase CPU time (Class1 CPU + DBM1 SRB)
  - Best used for indexes residing in the bufferpool

Index Compression – DSN1COMP

```
DSN1944I DSN1COMP INPUT PARAMETERS
PROESSING PARMS FOR INPUT DATASET
NO LEAFLIM WAS REQUESTED
DSN1940I DSN1COMP COMPRESSION REPORT
4,220 Index Leaf Pages Processed
1,000,000 Keys Processed
1,000,000 RIDS Processed
16,602 KB of Key Data Processed
7,837 KB of Compressed Keys Processed
8 K Buffer Page Size yields a
51 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
49 % of the original index's Leaf Page Space
No Bufferpool Space would be unused
------------------------------------------------
16 K Buffer Page Size yields a
53 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
47 % of the original index's Leaf Page Space
46 % of Bufferpool Space would be unused to
ensure keys fit in compressed buffers
------------------------------------------------
32 K Buffer Page Size yields a
53 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
47 % of the original index's Leaf Page Space
73 % of Bufferpool Space would be unused to
ensure keys fit in compressed buffers
```

Best reduction and no unused buffer space

Random Index Keys

- Prior to DB2 9
  - Only ASCending and DESCending indexes were supported
- In DB2 9
  - New RANDOM column ordering specification
  - Internally, DB2 ‘scrambles’ the values in these key columns so they appear quasi-random
  - Random key columns are scrambled on insertion into the index
  - The index becomes randomly ordered on that key column
  - Randomized index values can still be used for equality lookups
  - The keys can be decoded to get the original values
- Can be beneficial for data sharing because of index page P-lock contention
- Trade-off between contention relief and additional Getpage, read/write I/O, and lock request
  - Best for indexes resident in buffer pool

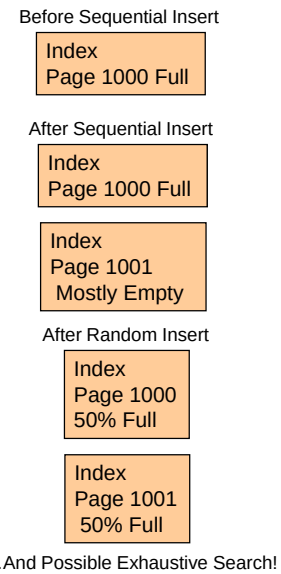
```
CREATE INDEX IX1 ON TB1
(COL1 RANDOM)
```

Random Index Keys (cont..)

- Index only access still possible
- Range scans are not supported
- Useful for avoiding some hotspots
  - Page split hotspots
  - P-Lock contention in data sharing
- Only use RANDOM if key ordering is unimportant
- Not Supported for NOT PADDED varying length columns
- Cannot use RANDOM order index columns as part of a sort merge join
  - If a join is between one table with that has an ASC index on the join column
    - And a second table that has a RANDOM index
    - The indexes are in completely different orders
    - And cannot be merged

Inserts and Indexes – Challenge with Splits

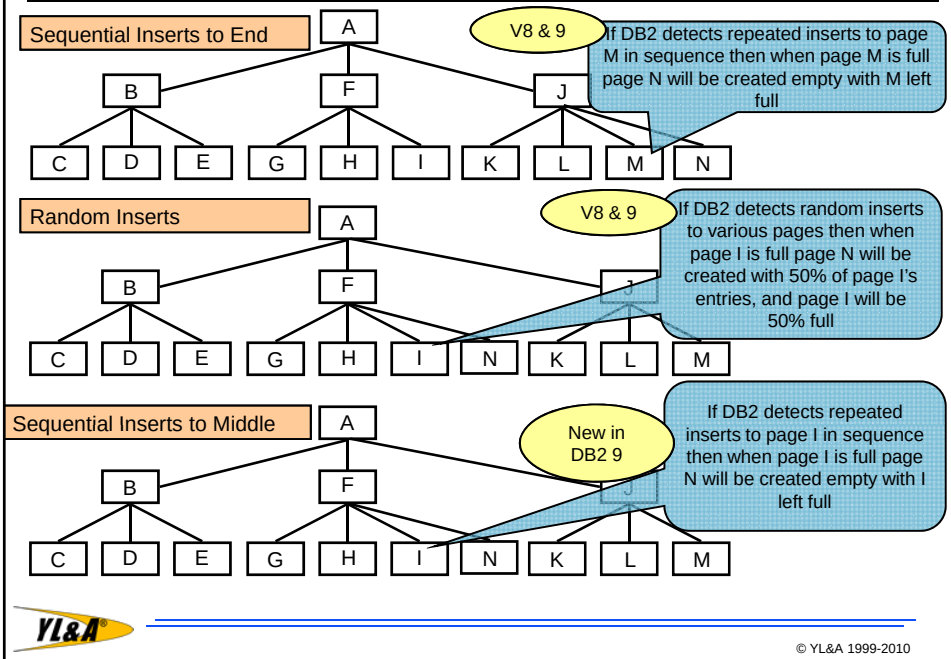
- If Purely Sequential Index Inserts
  - DB2 will not split pages at end
- If Not Purely Sequential Index Inserts
  - DB2 will split pages 50/50
  - Exhaustive search for available page
- Splits and Exhaustive Searches
  - DB2 will look for nearby page
  - No nearby page
    - Exhaustive search via spacemap chain
    - Huge cost!
- Bottom line on indexes and inserts
  - Pure sequential
    - No pctfree or freepage
  - Random
    - Set freepage
    - Better yet, large pctfree
      - So you never ever split!



Improved Index Page Splits

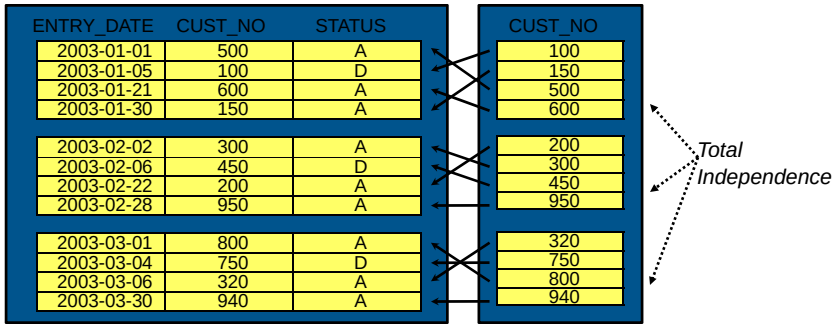
- Sequential inserts can cause index splits leaving 50% of the page empty and not immediately used
  - Caused by inserts into the middle of the index
  - With 4K index pages this can start to cause more frequent splits and increased contention on the index tree
- In DB2 9 there will be a asymmetric split of the index pages
  - Will better accommodate insert patterns into the index
  - Will allow for better space allocation
- Larger than 4K page sizes are now allowed for indexes
  - Larger number of index keys can be held on a page
  - Will help to reduce the amount of splitting necessary
    - Relieving contention on the index pages
- DB2 will detect insert pattern
  - Will select from different index page split algorithms
    - If random, DB2 will do a 50/50 page split
    - If sequential, DB2 will do an asymmetric split
      - Existing keys will move to new page to make room for new inserts

Splitting Index Pages in DB2 9



Some Pros and Cons of DPSIs

- Some of the benefits that this provides include:
  - Clustering by a secondary partitioned index
  - Ability to rotate partitions easily
  - Allow for reducing overhead in data sharing(affinity routing)
  - Further partition independence
  - Many utility benefits (discussed later)



DPSI Queries…more issues…

May also have issues with index lookaside and sequential detection resulting in more getpages

DPSI on CUST_ORDER

DPSI on CO.CUSTID
PARTITIONED BY CO.ORDER_DATE

```
SELECT *
FROM CUSTOMER C, CUST_ORDER CO
WHERE C.CUSTID = CO.CUST_ID
AND ORDER_DATE = ?
```

Local Predicate on limit key
- Will have partition elimination

Partition elimination only works if …

1. There is a local predicate (literal value, host variable, parameter marker, special register), on the leading partition limit key(s)

2. Or, a local predicate can be transitively closed against the leading partition limit key(s).

```
SELECT *
FROM CUSTOMER C, CUST_ORDER CO
WHERE C.CUSTID = CO.CUST_ID
AND C.CUST_DATE = CO.ORDER_DATE
```

Join predicate on limit key
- No partition elimination

DPSI and RI

- Parent table (MEFTBL) with a unique, clustered, partitioning index
  - SEG, SUBSEG, SSN
- Child table (DSCREP1) with a non-unique, clustered, partitioning index
  - SSN, SEQ_NUM
  - DELETE CASCADE

```
Select * from meftbl m
  INNER join dscrep1 d
  on m.seg = d.seg
  and m.subseg = d.subseg
  and m.ssn = d.ssn
  where m.seg = 1
  and m.subseg = 'A'
  and m.ssn = '001010001'
```

DB2 generated redundant predicates for all three columns of the join

where d.seg = 1
and d.subseg = 'A' } *Allowed for partition elimination against child*

and d.ssn = '001010001' ← *matched DSPI*

DPSI and RI – DELETE Issues

- Parent table (MEFTBL) with a unique, clustered, partitioning index
  - SEG, SUBSEG, SSN
- Child table(DSCREP1) with a non-unique, clustered, partitioning index
  - SSN, SEQ_NUM
  - DELETE CASCADE

```
DELETE FROM MEFTBL
   WHERE SEG = 1
   AND SUBSEG = 'A'
   AND SSN = '001012800'
```

NO partition elimination
And it attempted to read through all the partitions, instead of just the one part it needs

- Would either need the SEG and SUBSEG as part of the child key or cannot use DB2 enforced RI
- This would be a challenge for some designs, especially those produced by modeling tools

DPSI Improvements

- More parallelism
- More index lookaside
- Unique DPSI support when DPSI columns are superset of partitioning columns
  - Partitioning: C1.C2
  - DPSI: C1.C2.C3
  - Else the index can only be non-unique
- Enhanced page range screening/partition elimination
  - Non-matching predicates

```
PARTITIONED BY ORDER_DATE  AND ORDER_TYPE
```

```
SELECT * FROM CUSTOMER WHERE
ORDER_TYPE = 'DISCOUNT'
```

Can now be used for partition elimination even though ORDER_DATE is missing

DSPI APAR - PK41878

- Poor performance may occur for queries on a partitioned table space that uses DPSI index access

- Table partitioned: T1 (C4,C5)
- Unique index on T1 (C1,C2)
- DPSI on T1 (C1,C2,C3)

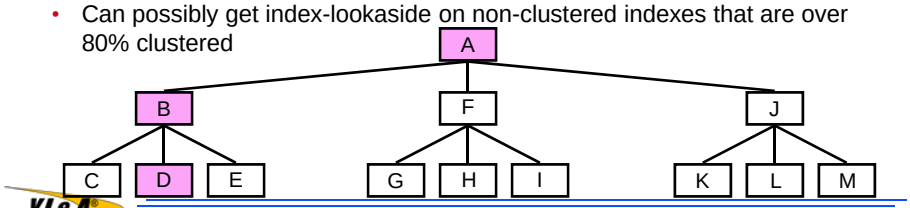
```
SELECT C1, C2,...
FROM T1 WHERE C1 = ?
AND C2 = ?
AND C3 = ?
AND C4 = ?
```

No screening because C1 and C2 are subset of unique index

- DPSI was chosen as index access to T1
  - However page range screening was not applied correctly
    - Because the DPSI column set is a superset of a unique index
- With APAR, the page range screening work properly

More Index Lookaside

- The objective of the index look-aside technique is to minimize the number of getpage operations
  - When an individual SQL statement or DB2 process is executed repeatedly and makes reference to the same or nearby pages
  - Results in a significant reduction in
    - Index and data page getpage requests when index is accessed in a sequential, skip-sequential, or hot spot pattern
- Prior to DB2 9
  - Used for clustering index only in insert, and not for delete
- In DB2 9
  - Possible for more indexes in both insert and deletes
  - More possibilities to use with DPSIs
  - Can possibly get index-lookaside on non-clustered indexes that are over 80% clustered



Index on Expression

- V8: Queries that include expressions cannot be index only

```
SELECT NAME
        FROM EMP
        WHERE SALARY + COMM < 20000
```

- DB2 9: Can now have indexes created on expressions

```
CREATE INDEX TOTALSAL ON EMP
( SALARY + COMM)
```

- New type of index to help have more efficient use of column expressions
- Some restrictions
  - Cannot be clustered
  - Cannot be primary or foreign key
  - Cannot be DESC

Index on Expression Example

```
CREATE TYPE 2 INDEX DSN8910.XEMP3
      ON DSN8910.EMP

      (cast(soundex(lastname) as char(4) ccsid ebcdic)  ASC  )
      USING STOGROUP DSN8G910
           PRIQTY 12
           ERASE NO
      FREEPAGE 0
      PCTFREE 10
      BUFFERPOOL BP0
      CLOSE NO
      PIECESIZE 2097152  K;
```

```
select * from dsn8910.emp a where
cast(soundex(lastname) as char(4) ccsid ebcdic) = 'R300'
```

Index on Expression

- Index on expression allows us to expand the indexability and control of our tables within the database
  - Create indexes on derived columns and formulas
  - Enforce uniqueness on derived columns

I store a date in my table, but need to search on a collection of years

```
CREATE INDEX NAME_IDX1 ON NAME_TBL
(YEAR(BRTH_DTE), LAST_NME);
```

```
SELECT FRST_NME, LAST_NME, BRTH_DTE
FROM    NAME_TBL
WHERE YEAR(BRTH_DTE) IN ('2000', '2002', '2006')
AND      LAST_NME = 'RADY';
```

This is a two column match against the NAME_IDX1 index. Prior to DB2 9 we needed a redundant table column to do this

Index on Expression

- Allows for a search on a derived value without having to store that value in a table

List all the accounts that will have a balance under $10,000 in 12 months

```
CREATE INDEX ACCT_IDX_3 ON ACCOUNT
(ACCT_BAL – (PAY_AMT * 12) , ACCT_ID);
```

```
SELECT  ACCT_ID
FROM    ACCOUNT
WHERE ACCT_BAL – (PAY_AMT * 12) < 10000;
```

Prior to DB2 9 this query would scan the entire ACCOUNT table. With DB2 9 an index can be built to directly answer this question with an index only scan

Column Ordering Strategy

- Depending on the process, the ordering of the columns can have a significant impact on performance
  - Logging overhead (more on this topic later…)
- INSERT and DELETE processes → no options
  - Entire records will be logged
- UPDATE
  - DB2 must log both the before and after image
  - Fixed length row
    - Log the first byte updated through the last byte updated
  - Variable length row (VARCHAR column(s) or DB2 Compression)
    - Log the first byte updated in the row through the end of the row
      - EXCEPT V7 where the first to last if no row length changes
- Physical table column ordering can be very critical

Infrequent Updated	VARCHAR		Regularly Updates
	Non-updated	Updated	

Reordered Row Format - DB2 9

- Improvement to access to data stored in varying length columns
  - No change for fixed length rows
- Format for which the data is stored in the table has changed
  - To facilitate locating columns within the row for data retrieval and predicate evaluation
- Prior to DB2 9
  - Varying length columns are not padded to full length
    - Columns after a varying length column are variable offsets
    - Therefore to get a column after a varying length column a sequential scan of the columns needed to be performed
- In DB2 9, with Reordered Row Format
  - No longer necessary to scan columns to find the column of interest
  - DB2 is enabled to find columns quickly after the first varying length column
- NOT optional (except for a new hidden ZPARM – SPRMRRF)

Reordered Row Format

- Basic Row Format (prior to DB2 9)
  - Varying length columns have a 2 byte field prior to the column that holds the length
    - In the following example NAME is a CHAR(10), ADDRESS is a VARCHAR(20), PHONE is CHAR(10) and JOB_DESC of VARCHAR(15)

NAME	ADDRESS		PHONE	JOB_DESC	
Susan	12	1234 YLA Way	217-698-1162	8	The Boss

- Reordered Row Format(DB2 9)
  - All fixed length columns are placed at the beginning of the row followed by the offset (in hex) to the varying length columns, then the values of the varying length columns

NAME	PHONE	OFF SET2	OFF SET4	ADDRESS	JOB_DESC
Susan	217-698-1162	18	30	1234 YLA Way	The Boss

Summary

DB2 9 provides several new features for enhanced table and index design

- Append processing
- No log tables
- Truncate Table
- Clone Tables
- Index on Expression
- Index Compression
- Larger Index Page Sizes
- Universal Tablespaces
  - Partition by Range
  - Partition by Growth
- Improved Index Page Splitting

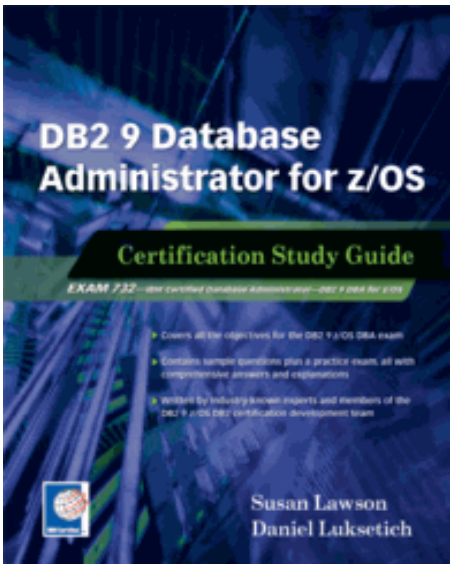
But many challenges still remain!

DB2 9 Manuals

- DB2 V9.1 for z/OS Administration Guide, SC18-9840-00
- DB2 V9.1 for z/OS Application Programming and SQL Guide, SC18-9841-00
- DB2 V9.1 for z/OS Application Programming Guide and Reference for JAVA SC18-9842-00
- DB2 V9.1 for z/OS Codes, GC18-9843-00
- DB2 V9.1 for z/OS Command Reference, SC18-9844-00
- DB2 V9.1 for z/OS Data Sharing: Planning and Administration, SC18-9845-00
- DB2 V9.1 for z/OS Diagnosis Guide and Reference, LY37-3218-00
- DB2 V9.1 for z/OS Diagnostic Quick Reference, LY37-3219-00
- DB2 V9.1 for z/OS Installation Guide, GC18-9846-00
- DB2 V9.1 for z/OS Introduction to DB2, SC18-9847-00
- DB2 V9.1 for z/OS Licensed Program Specifications, GC18-9848-00
- DB2 V9.1 for z/OS Messages, GC18-9849-00
- DB2 V9.1 for z/OS ODBC Guide and Reference, SC18-9850-00
- DB2 V9.1 for z/OS Performance Monitoring and Tuning Guide, SC18-9851-00
- DB2 V9.1 for z/OS RACF Access Control Module Guide, SC18-9852-00

DB2 9 for z/OS DBA Certification Guide

- "DB2 9 for z/OS DBA Certification Guide"
- McPress
- October 2007
- Authored By Susan Lawson and Dan Luksetich



DB2 Information on the Web

- DB2 9 for z/OS
  - Ibm.com/software/db2zos/db2zosv91.html
- IBM Software
  - Ibm.com/software
- DB2 Family
  - Ibm.com/software/db2
- DB2 Solutions Directory Applications
  - Ibm.com/developerworks/db2
- "Red Books"
  - ibm.com/ redbooks
- DB2 for z/OS
  - Ibm.com/software/db2zos
- DB2 Support
  - Ibm.com/software/db2zos/support.html
- DB2 for z/OS Papers
  - ftp://ftp.software.ibm.com/software/data/db2zos
- DB2 Magazine
  - http:// www. db2mag.com
- DB2 Certification
  - http://www.ibm.com/certify
- DB2 Experts
  - www.db2expert.com

Courses by YL&A - Taught by skilled instructors world-wide!

- DB2 9 for z/OS Transition
  - Application or DBA or both
- SQL (z/OS and LUW)
  - Basic SQL
  - Advanced and Complex SQL
  - SQL Performance Tuning and Optimization
- Application Development
  - Stored Procedure Development and Implementation
  - UDFs and Triggers Development
- Database Design
  - Physical Design, Logical Design
- Data Sharing
  - Implementation, Performance, Recovery
- High Performance Design and Tuning
  - Application, Database, Systems
- DB2 9 for z/OS Certification Crammer Course

We customize all classes based upon customer requirements

ONLY YL&A offers you the DB2 9 DBA Certification Crammer Course to help you become certified!!

CPU Reduction Through Performance Audits

- DB2 Performance Audits
  - Existing or new database designs and applications
  - Certification of design and implementation acceptance
  - Evaluation of all the performance 'points' in a DB2 environment
    - Physical Design
    - Subsystem
    - Application Code and SQL
  - Help with bringing legacy application to an e-business environment – the rules have changed!
    - What was acceptable performance in the past is NOT acceptable in an e-business environment
  - Experienced in 'fighting fires' – many performance problems do not become reality until production
  - *Results*: problems identified, solutions proposed (many implemented immediately), continual knowledge transfer during the process

Cost Avoidance Through Performance Tuning!!!!