IDUG
The Worldwide DB2 User Community

IDUG DB2 Tech Conference
Prague, Czech Republic | November 2011

Table and Index Design Strategies in DB2 9 and 10 for z/OS

Susan Lawson
YL&A

Session: F12, Wednesday, Nov 16th 02:15-03:15
Platform: z/OS

YL&A®
Yevich, Lawson & Associates, Inc.

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 improvements in the area of performance and availability. We will look also at how this impacts the design strategies we have practiced over the years and see how they are impacted by DB2 9 and 10 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.

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 Table Spaces
  - Partition by Range
  - Partition by Growth
- Index Lookaside
- Reordered Record Format

Table and Index Design in DB2 10 - Outline

- UTS and Member Cluster
- On-the-Fly Compression
- Hash Key Access
- Online ALTERs
  - Page Size (Table and Index)
  - Segment Size
  - DSSIZE
- Table Space Conversions
- Parallel Update on Insert
- Prefetch and Disorganized Indexes
- INCLUDE Index Columns
- More Index Lookaside

Logging Issues

- As database 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 Table Spaces – DB2 9

- Logging can be suppressed for
  - Base table spaces (applies to all partitions)
  - XML table spaces
  - Including associated indexes
    - Inherit attribute of table space
- Changed via ALTER or CREATE
- Defined at the table space level
  - LOGGED (or LOG YES)
    - Modifications are recorded on log
    - Default, except for work file table spaces and LOBs >1 GB
  - NOT LOGGED (or LOG NO)
    - Modifications are not recorded on log
  - Table space 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

Quickly and Efficiently Removing Data From a Table

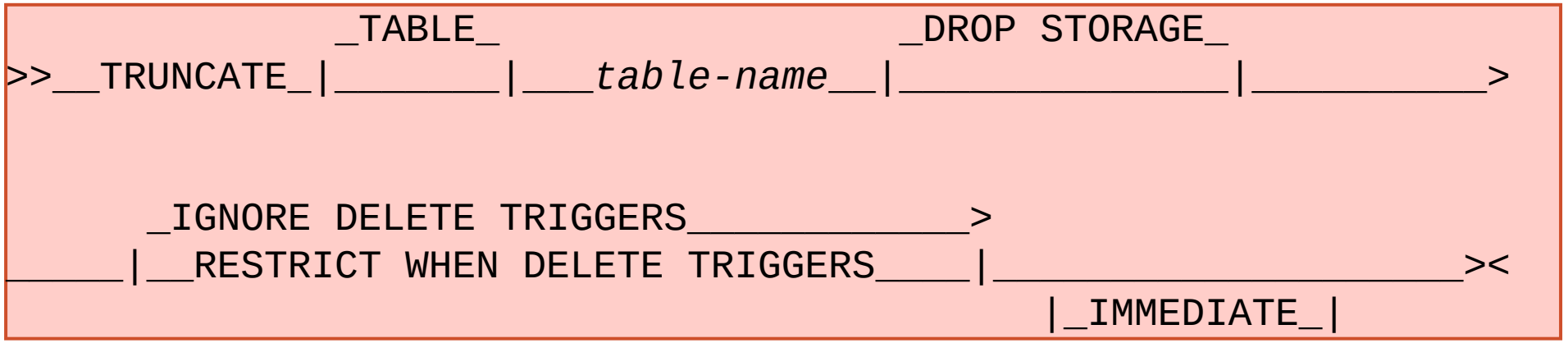
- Prior to DB2 9
  - To empty a table you have to either
    - Do a mass delete
    - Or use LOAD Utility with REPLACE REUSE and LOG NO NOCOPYPEND
    - DROP and RECREATE table
- Issues
  - Delete triggers introduce problems with DELETEs
    - Requires a DROP and recreation of the delete trigger to empty the table
      - To avoid firing the trigger
  - LOAD REPLACE works on a table space level (not table)
    - If multi-table table space, then this is a problem
      - All tables get replaced
    - A DROP and recreate is a lot of work and structures and dependencies are gone

Truncate Table – DB2 9

- In DB2 9
  - The TRUNCATE statement address issues on previous slide
    - Deletes all rows for base or declared global temporary tables
- 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
- 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;
```

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.

Universal Table Space – DB2 9

- 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 (until 10…)

Universal Table Space - Migration

- Not an easy migration
- Will need
  - A LOAD/REPLACE
  - A REORG
  - DROP/RECREATE
- In DB2 10
  - Migration will be available with an ALTER and deferred REORG

Where is Universal Table Space Required?

DB2 9

- Clone Tables
- Partition by Growth

DB2 10

- Hash Access
- Inline Lobs
- XML Document Version
- CURRENTLY COMMITTED
- Deferred ALTERs
  - MEMBER CLUSTER, DSSIZE, Page Size, SEGSIZE

Partition By Growth Universal Table Spaces - 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;
```

Size Limitations with Increased Number of Partitions

- While 4096 partitions gives us many opportunities to support better designs, there are some limitations
- Maximum number of partitions a table space can have is dependent on
  - The DSSIZE, the page size and the total table space size
  - Page size affects table size because it affects 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 until DB2 10
    - And still requires a REORG!
    - Must be a UTS first!

Still a limitation in DB2 9 and 10!

- Can only be up to 5 LOBs on a table if 4096 partitions are to be supported

DB2 10 Deferred ALTERs

- Deferred alterations
  - Page size
    - Table spaces and index
  - DSSIZE
  - SEGSIZE
  - MEMBER CLUSTER
  - Conversions
    - Single table (simple or segmented) into UTS PBG
    - Partition table space to UTS PBR
    - PBG to hash(also sets RBDP status on index)

Must be a UTS!

Online Schema - SYSPENDINGDDL

- SYSIBM.SYSPENDINGDDL
  - DB2 catalog table
  - Holds the information about the change

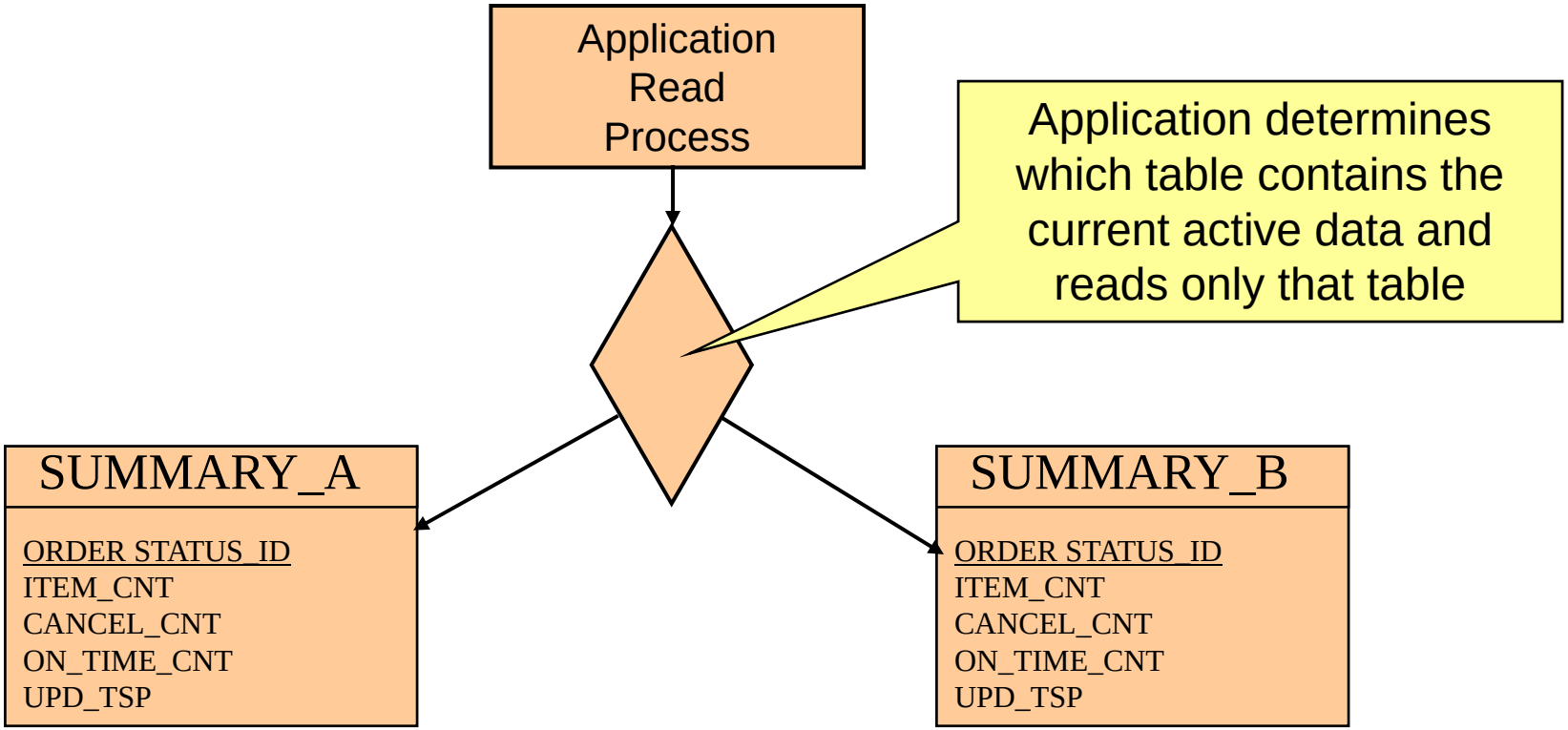
- DBNAME
- TSNAME
- DBID
- PSID
- OBJSCHEMA
- OBJNAME
- OPTION_KEYWORD
- OPTION_VALUE
- STATEMENT_TEXT

AREOR

- REORG SHRLEVEL CHANGE/REFERENCE will materialize

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



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

Clone Table Support – DB2 9

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

EXCHANGE DATA BETWEEN TABLE base-table AND clone-table

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 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, except drain!

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

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

New inserts at end together,
Very efficient deferred write

APPEND on Insert – DB2 9

- 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 … Issues and APARs

- DB2 9 Issue with Member Cluster (MC00)
  - Does not work in DB2 9
  - Caused performance problems with those using this technique
- APAR PK81470
  - Re-establishes MC00 in DB2 9
  - Some possible regression
  - Will allow for reuse of deleted space
- APAR PK81471
  - Delivers proper APPEND
  - Does not reuse space from deletes
- Could use both
  - Append processing
  - Space map separation

MEMBER CLUSTER Support for UTS

- DB2 9
  - Universal table space does not support member cluster
    - Needed for SQL INSERT performance
      - Removes hot spots in concurrent sequential inserts in data sharing by dividing up the space map (199 pages)
      - Does not maintain clustering during INSERT
      - Helps support fast INSERT processing with
        - FREEPAGE 0 , PCTFREE 0 append strategy
- DB2 10
  - MEMBER CLUSTER is compatible with a universal table space
  - Can be ALTERed
    - Once the table space is a UTS
    - Will need a REORG to materialize
  - SYSTABLESPACE
    - MEMBER_CLUSTER column
  - Space map also now covers 10 segments

```
ALTER TABLESPACE TBS1
MEMBER CLUSTER YES
```

Index Compression – DB2 9

- 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 buffer pool
- 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 buffer pool

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 – DB2 9

- 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
      - Cannot be merged

Hash Key Access – DB2 10

- A hash key is defined on the table
  - Key must be unique
  - No duplicates are allowed
  - Can have only a single hash key
- Will benefit…
  - Singleton Select/Update or Fetch
    - With equal predicates on all hash key columns
- Candidate Tables
  - Large tables with random access through a unique index
  - Potential degradation if used in sequential access
  - Better with larger tables with small rows

```
ORGANIZE BY HASH UNIQUE (CUSTID)
```

```
Select NAME
From CUSTOMERS
WHERE CUSTID = 5
```

More on Hash Access

- The primary purpose is to improve single row access
  - Can be used to enforce uniqueness
  - Hash key can be used for RI primary key
- Space Utilization
  - Hash area required for table
    - Up to 100% more table space
  - Hash space must be maintained by a DBA
    - There is an overflow index created by DB2 and used if hash space fills
    - Hash real-time statistics are available to determine REORGs
    - REORG can organize has space
- Can we eliminate an index?
  - Hash access may not be chosen always
    - So, no…
- Requires
  - Universal table space
  - Reordered record format
- Evaluate hash access versus index-only access

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!

Before Sequential Insert

Index
Page 1000 Full

After Sequential Insert

Index
Page 1000 Full

Index
Page 1001
Mostly Empty

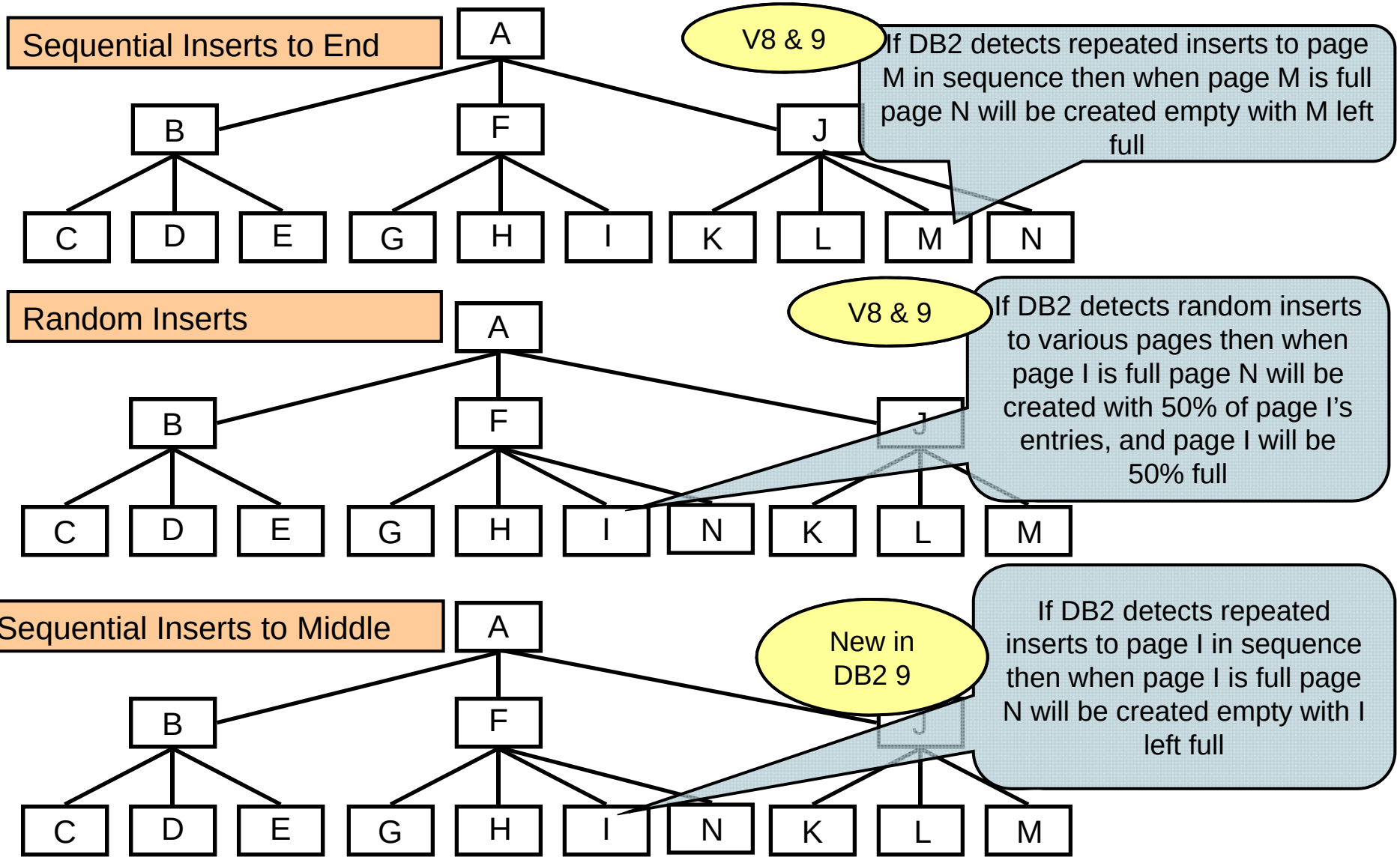
After Random Insert

Index
Page 1000
50% Full

Index
Page 1001
50% Full

…And Possible Exhaustive Search!

Splitting Index Pages in DB2 9



IFCID Support for Index Page Splits – DB2 10

- DB2 10
  - Support for IFCID 359
- IFCID 359 records an index page split
  - Makes monitoring index page splits easier
- IFCID 359 stores the following:
  - Database ID
  - Page set ID
  - Partition number of the splitting index
  - Splitting page number
  - Start timestamp of the split
  - End timestamp of the split

DPSI Queries…issues in DB2 9 (and 10)…

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 Improvements in DB2 9

- 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

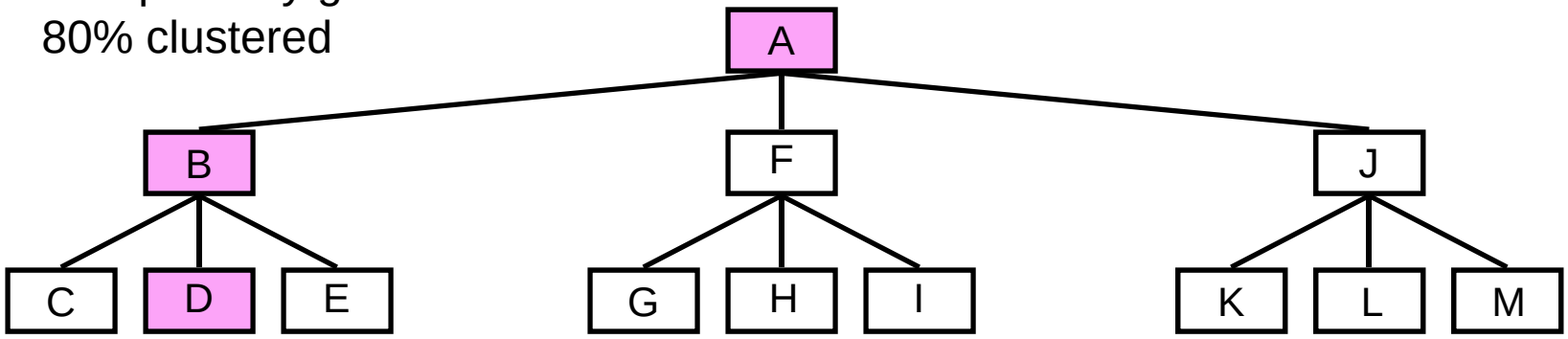
SELECT * FROM CUSTOMER WHERE ORDER_TYPE = 'DISCOUNT'

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

More Index Lookaside – DB2 9

- 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

More in DB210
For RI support



Reduce Need for Reorganization of Indexes in DB2 10

- Prior to DB2 10
  - A disorganized index results in sequential prefetch failure
    - Due to larger and more frequent gaps between successive leaf pages
    - Result:
      - More sync I/O
      - Potentially more index levels
      - More efforts to traverse the tree
- DB2 10
  - List prefetch can be done for index leaf pages during query processing
    - Based on index non-leaf page information
  - List prefetch usage decision is done at execution
    - Use list prefetch on index leaf pages with range scan
    - Reduces CPU time spent on index root page GETPAGE requests
  - Reduce synchronous I/O waits for queries accessing disorganized indexes
  - Throughput improvement in Reorg, Runstats, Check Index
  - Limited to forward index scan

Index on Expression – DB2 9

- 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'
```

Additional Non-Key Columns in Unique Index – DB2 10

- DB2 9
  - An index can be defined to enforce uniqueness
    - Additional indexes may be needed to achieve index only access
      - On columns which are not part of the unique index
  - Additional indexes…
    - Higher insert /delete CPU time
      - Approximately 30% for each index
    - Increased storage
    - Maintenance and availability issues
- DB2 10
  - Ability to add non-key columns in a unique index
  - Can help reduce overall number of indexes
  - 30% CPU reduction in insert
    - IF additional index was removed

```
CREATE UNIQUE INDEX IX1 ON TAB1(COL1,COL2) INCLUDE (COL3)
or
ALTER INDEX IX1 ADD INCLUDE (COL3)
```

Reordered Row Format – DB2 9

- 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

Reordered Row Format – DB2 9

- In DB2 9
  - 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
- Typically is the default
  - Hidden ZPARM in DB2 9

SPRMRRF

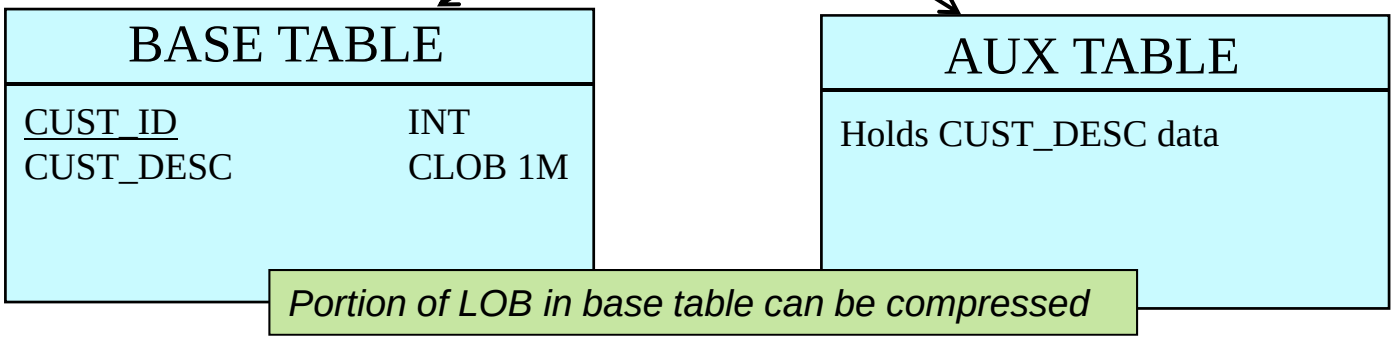
  - Or, external DSNZPARM RRF in DB2 10
    - Can turn off the automatic RRF
    - Data will not be converted
  - Can also be specified on LOAD or REORG
    - ROWFORMAT (RRF or BRF)

In line LOBs for Reduced LOB Access Times – DB2 10

- Prior to DB2 10
  - LOBs (CLOB, DBCLOB, BLOB) are only stored in auxiliary tables with indexes
  - One additional table and index per LOB column per partition
  - This results in additional I/O to retrieve LOB data

```
SELECT CUST_DESC
FROM BASE_TABLE
WHERE CUST_ID = 100
```

This statement will result in the reading of data from both tables



In line LOBs for Reduced LOB Access Times – DB2 10

- In DB2 10
  - A portion of LOBs can be stored “inline” with the base table data
  - Additional I/O’s not required for smaller LOBs

```
CREATE TABLE BASE_TABLE
(CUST_ID INT NOT NULL,
 CUST_DESC CLOB 1M INLINE LENGTH 500)
```

```
SELECT CUST_DESC FROM BASE_TABLE WHERE CUST_ID = 100
```

Will result in the reading of data from both tables only if the length of CUST_DESC for CUST_ID = 100 exceeds 500 bytes

Maintaining Current and History Data

- We have always promoted keeping current and historical data separate for most table designs
  - There are some exceptions, however!
- You can achieve higher performance with a multi-table design

CUST_TABLE	
CUST_ID	INT
START_TSP	TIMESTAMP
END_TSP	
data	

CUST_TABLE	
CUST_ID	INT
data	

CUST_HIST_TABLE	
CUST_ID	INT
START_TSP	TIMESTAMP
END_TSP	
data	

With a single table:
- Table can get large
- Prefetch can get disabled
- All queries need a timestamp, even for current
- Joins become more complicated
- Table maintenance difficult due to high availability requirements for current data

With current and history table:
- Current table remains smaller
- Prefetch less likely to get disabled
- Only history queries need a timestamp
- Joins for current data are easier
- History table easier to take off line for mass deletes or archiving

Reading Current and Historical

- For highest performance of current data you read the current table only
- For highest performance of historical reads you read the history table only
- If you do not know which data you need, but have an effective time then read both
  - This can be done with a UNION ALL

```
SELECT CUST_ID, DATA_COL, UPD_TS
FROM CUST_TABLE BASETB
WHERE CUST_ID = ?
AND BASETB.UPD_TS <= ?
UNION ALL
SELECT CUST_ID, DATA_COL , STRT_TS
FROM CUST_HIST_TABLE AUDITB
WHERE CUST_ID = ?
AUDITB.STRT_TS <= ?
AND AUDITB.END_TS > ?;
```

System-Maintained Temporal Tables – DB2 10

- Two types
  - System-Maintained
  - Business-Maintained (not discussed in this presentation)
  - Both can be combined to create a bi-temporal table
- System-Maintained temporal tables automate our two table history design
  - A base table is created
  - A versioned (history) table is created
  - DB2 automates the data movement between the tables!
  - DB2 automates the selection of data based upon a timestamp!
- Special column designations control system-maintained temporal table
  - Begin and end timestamp columns for the time period of the data
  - Columns designated as TIMESTAMP
    - GENERATED ALWAYS AS ROW BEGIN for start timestamp
    - GENERATED ALWAYS AS ROW END for end timestamp
    - PERIOD designation identifies the timestamp columns for a time period
    - VERSIONING designation related current to history table

System-Maintained Temporal Tables

- Use table definitions to set up the period columns and relationship to the history table

```
CREATE TABLE CUST_TABLE
(CUST_ID INT NOT NULL,
 START_TSP TIMESTAMP NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 END_TSP TIMESTAMP NOT NULL GENERATED ALWAYS AS ROW END,
 DATA_COL CHAR(10),
 PERIOD SYSTEM_TIME(START_TSP,END_TSP));

CREATE TABLE CUST_HIST_TABLE
(CUST_ID INT NOT NULL,
 START_TSP TIMESTAMP NOT NULL ,
 END_TSP TIMESTAMP NOT NULL ,
 DATA_COL CHAR(10));

ALTER TABLE CUST_TABLE ADD VERSIONING USE HISTORY TABLE
CUST_HIST_TABLE;
```

Reading System-Maintained Temporal Tables

- Current table can be read directly
- History table can be read directly
- FROM clause has been extended to read time based data
- SELECT FROM CUST_TABLE
  - FOR SYSTEM TIME AS OF ?
    - Will return data “as of” the specific timestamp from either current or history
    - Begin value less than or equal to input value
    - End value greater than input value
  - FOR SYSTEM TIME FROM ? TO ?
    - Begin value less than second value
    - End value greater than the first value
  - FOR SYSTEM TIME BETWEEN ? AND ?
    - Begin value less than or equal to second value
    - End value greater than the first value
- DB2 will automatically rewrite the query with a UNION ALL to retrieve data from both tables if needed

```
SELECT CUST_ID, DATA_COL
FROM   CUST_TABLE BASETB
WHERE CUST_ID = ?
FOR SYSTEM TIME AS OF ?
```

In Memory Tables – DB2 10

- Prior to DB2 10
  - Difficult to ensure a table would be placed in memory and remain there
  - Could define a buffer pool with VPSEQT(0) and PGSTEAL(FIFO)
    - And enough pages to hold the entire object
- DB2 10
  - Can cache entire table space into a buffer pool
    - Data will be preloaded into buffer pool
      - When object is opened and will remain until closed

PGSTEAL=NONE

- Excellent for lookup tables and indexes
  - Small tables that have high I/O rates
  - Avoid LRU chain maintenance and LC14 contention
  - Avoid unnecessary prefetch and LC24 contention
- IFCID 201 & 202 will be updated to signify this condition

DB2 9 and10 both provide 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 Table spaces
  - Partition by Range
  - Partition by Growth
- Index Lookaside
- Reordered Record Format

- UTS and Member Cluster
- On-the-Fly Compression
- Hash Key Access
- Online ALTERs
  - Page Size (Table and Index)
  - Segment Size
  - DSSIZE
- Table space Conversions
- Parallel Update on Insert
- Prefetch and Disorganized Indexes
- INCLUDE Index Columns
- More Index Lookaside

But many challenges still remain!

IDUG
The Worldwide DB2 User Community

IDUG DB2 Tech Conference
Prague, Czech Republic | November 2011

Table and Index Design Strategies in DB2 9 and 10 for z/OS

Susan Lawson
YL&A