DB2 10 for z/OS Performance Opportunities

Susan Lawson
YL&A

Session: F01, Monday, Nov 14th 11:30-12:30
Platform: z/OS

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 looks at the features of DB2 10 for z/OS with an emphasis on the features focusing on performance. DB2 10 promises opportunities for CPU savings in various areas. We will review those enhancements and look at all available features in the database, system and SQL and their potential usage. We will look at where the performance opportunities are, what we need to do to take advantage of them, and any additional considerations when exploiting these features. We will also look at experience with these features and performance realizations.

- Brief discussion of overall performance objectives of DB2 10
- Discuss details on database performance features and usage
- Discuss SQL, application and optimization performance enhancements
- Discuss considerations for implementing new features and enhancements
- Review goals and strategies for migration and feature exploitation

Key Points

DB2 10 Migration
- Goals and highlights
- CPU improvements
- Migration considerations (from 8 or 9)

DB2 10 Database and System Performance Features
- LOB streaming
- Compress on the fly during INSERT
- In-memory objects
- IFCIDs for performance
- Row-level sequential detection
- Additional index lookaside
- Sequential insert improvements
- Work file enhancements
- Virtual storage relief
- LRSN spin elimination

Key Points

DB2 10 Application and SQL Performance Enhancements
- Query parallelism enhancements
- Currently committed data
- SQL table functions
- Enhanced native SQL procedures
- Access to currently committed data
- Literal Reuse
- Optimistic locking for prepare
- Improved RID processing
- Predicate transitive closure improvements
- Earlier predicate processing
- Merge vs materialization
- Query sort enhancements
- Correlated to non-correlation rewrite
- High performance DBATs
- Improved parallelism

Migration Objectives

- Whether migration from DB2 8 to 10 or 9 to 10 objectives are relatively the same
  - Achieve the best possible performance!
- Many performance gains will be seen during migration without change
  - Some will need rebinds
  - Some may need change to old code
  - Some objects may need to change
    - Could mean small outages are needed
- Most performance opportunities come from using new features
  - Have goals and objectives
- Migrating from 9 to 10
  - Plan to rebind
- Migrating from 8 to 10
  - Plan to rebind
  - Don’t forget to review what is in 9 and take advantage of those features

Performance 'Opportunities'

- When it comes to achieving the best performance possible in DB2 we have to consider the following
  - Expectations
    - What new features look promising?
    - What problem are you looking to resolve with a new feature?
    - Is it a better option than what you are doing today?
  - Reality
    - What effort is required to take advantage of new features?
    - Will the usage achieve my goals?
    - What features will be automatic and did their implementation hurt or harm my current performance?
  - Usage
    - To use some new features there may be large changes needed
      - Rebinds, code changes, database changes
    - Plan for efforts needed
    - Evaluate effectivness

Compress on-the-fly during INSERT

- Prior to DB2 10
  - In order for compression on a table to take place, a REORG or LOAD must occur
    - To rebuild the compression dictionary
  - LOAD COPYDICTIONARY was introduced late in DB2 9 to help
- DB2 10
  - Tables with COMPRESS YES with no compression dictionary available
    - Are directly compressed by DB2
    - INSERT and MERGE triggers creation of compression dictionary
      - Applies also to LOAD REPLACE, RESUME NO or RESUME YES SHRLEVEL CHANGE
        - Without KEEPDICTIONARY
      - Table space needs to contain enough data
    - Compression dictionary is built asynchronously
      - A DB2 service task reads(UR) all tables rows
      - Dictionary is stored in chained page
        - Can span whole table space

Sequential Inserts Improvement in Clustering Index

- Prior to DB2 10
  - When inserting rows into a table
    - Space management algorithms are used to find the candidate page where the row is to be inserted according to the clustering index
  - To select the initial candidate page
    - DB2 selects the data page where the row that contains the next highest key to the row being inserted resides
  - If there is available space
    - DB2 inserts the row into that page
      - If there is not enough space
        - DB2 searches for another candidate page and eventually finds space to insert the row
      - If a second row is inserted that has a key higher than the row just inserted but lower than the next highest existing row
        - DB2 selects the same initial candidate page again, only to find it is still full
      - DB2 must repeat the process to find another candidate page

Sequential Inserts Improvement in Clustering Index (cont)

- In DB2 10 (CM – No rebind)
  - Improves way the first candidate page is selected
  - Instead of choosing the page pointed to by the next highest key as the initial candidate page
    - DB2 chooses the first candidate page based on the next lower key
      - Page where previous row was inserted
        - Chances are reasonable that the page still contains enough space for this new row
        - No search for another candidate page
- Helps sequential inserts into middle of table based on clustering index
  - On the second and subsequent sequential insert
    - DB2 does not have to repeatedly find the first candidate page as full
    - Higher chance to find open space and avoid a search
      - Provides CPU and getpage savings
        - Due to the fact that fewer candidate pages need to be searched for sequential insert workloads
      - Reduced latch class 006 and 245

I/O Parallelism for Index Updates

- DB2 9
  - DB2 updates indexes serially during inserts
  - Tables with several NPIs may suffer high synchronous read I/O wait
- DB2 10
  - I/O parallelism invoked
    - Prefetches index pages
      - Overlap I/Os against non-clustering indexes
      - Performed in one processing task
  - Enabled for three or more indexes
    - One cluster index enforced, two additional indexes non-clustered
  - Reduces I/O wait for large indexes
    - Which do not fit in the buffer pools
    - No improvement if all indexes are resident in the buffer pools
  - Partitioned table space
    - Classic and UTS PBG/PBR - not segmented
- Possible elapsed time improvements of up to 50%
  - Some CPU overhead for scheduling prefetch (zIIP eligible for read SRB time)

INDEX_IO_PARALLELISM

Row Level Sequential Detection

- Prior to DB2 10
  - Dynamic prefetch sequential works poorly when the number of rows per page is large
- DB2 10 (CM)
  - Row Level Sequential Detection (RLSD)
  - Count rows, not pages to track the sequential detection
  - Since DB2 10 will trigger prefetch more quickly, it will use progressive prefetch quantity:
    - For example, with 4K pages the first prefetch I/O reads 8 pages, then 16 pages, then all subsequent I/Os will prefetch 32 pages (as today)
  - Also applies to indexes
  - Row level sequential detection (RLSD) preserves good sequential performance for the clustered pages

Sequential Detection and Index Lookaside for RI

- Prior to DB2 10
  - When inserting into a dependent table
    - DB2 must access the parent key for referential constraint checking
  - No ability to utilize index lookaside or sequential detection for this
- DB2 10
  - CPU overhead of RI checking can be reduced by minimizing index probes for parent keys
    - Allows sequential detection to trigger dynamic prefetch for parent key referential integrity checking
    - Enables index look-aside for parent key referential integrity checking
    - Can also avoid index lookup for referential integrity checking, if the non-unique key to be checked has been checked before
      - Only 1st INSERT checks parent index, subsequent INSERT will not
    - Must be an index on the child table
      - Else, only benefit is index look-aside on parent table
  - Reduces CPU time of insert work loads involving referential constraints
- RI checking can take advantage of index enhancements and hash access can be used for parent key checking

Optimistic Locking Enhancement

- Prior to DB2 10
  - Could not specify optimistic locking options on dynamic SQL
- DB2 10
  - In dynamic embedded SQL applications
    - Can specify attributes in the PREPARE statement to add the columns that are required for optimistic concurrency control to query result tables
    - Without making any changes to the SQL statements

WITHOUT ROW CHANGE COLUMNS

(Default)Row change columns are not added to the result table

Workfile Database Performance

- DB2 10 CM
  - Spanned pages
    - Records of work files created for joins and large sorts can span multiple pages
      - Accommodates larger record and sort key lengths for sort records
    - Maximum length of a work file record is 65529 bytes
  - Reduced CPU time
    - For workloads executing queries requiring use of small work files
  - WORKFILE database now UTS/PBG table spaces
  - In-memory work file enhancements
    - More opportunity to use in-memory work files than in DB2 9
    - DB2 supports simple predicate evaluation for work files

LOB Streaming

- Prior to DB2 10
  - LOB streaming (available first in DB2 V8)
    - Allowed programs to read LOB objects from applications without knowing their size first
    - Streaming occurs to DDF (out), not to the engine (in)
    - Had to materialize it entirely before sending to DB layer
  - DB2 9
    - Can stream LOB data to the engine from DDF without needing to materialize it in memory
    - Can stream LOB data to traditional programming languages via FETCH CONTINUE
- DB2 10
  - LOB and XML objects will use fewer resources
  - Eliminates materialization for
    - LOB (>2MB)
    - XML objects (>32KB)
  - Reduced class 2 CPU time
  - Reduced virtual storage requirements

XML Performance Improvements

- XML Document Versioning
  - Can lead to improved concurrency through lock avoidance
    - Readers do not need to lock XML documents
      - Improving concurrency for update operations
- XML Update
  - No more full document replace

```
UPDATE DSN81010.CUSTOMER
SET INFO=XMLMODIFY('replace node /customerinfo/phone with $x',
       XMLPARSE('phone type="work">416-555-1358</phone>') AS "X")
WHERE CID=1000;
```

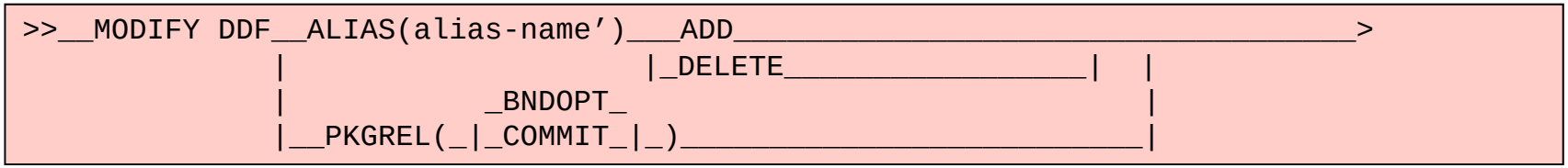
- Binary XML support
  - Avoid the cost of XML parsing during insert and reduces the size
    - 10-30% CPU and elapsed time improvement
- Schema validation in engine (built-in function)
  - No UDF call for validation
  - Utilizes XML System Service Parser
    - 100% zIIP / zAAP eligible

High Performance DBATs

- Prior to DB2 10
  - DRDA DBATs always allocated packages on with RELEASE(COMMIT)
    - To allow DDL and BIND to break into thread
- DB2 10
  - Allows RELEASE(DEALLOCATE) in distributed packages
  - Reduce CPU consumption by
    - Avoids repeated package allocation/deallocation
    - Avoids processing to go inactive and then back to active
    - Higher CPU reduction for short transactions
      - With highly active threads (protected threads)
    - xPROCs, CTs and PTs, lookaside and prefetch not re-initialized
  - Benefits thread pooling and CMTSTAT=INACTIVE
  - Real memory savings in z/OS by reducing number of DBATs
    - As well as virtual storage savings in DBM1
  - If at least one DEALLOCATEpackage exists, thread stays active
    - Will turn inactive after 200 times to free up DBAT
    - Normal idle thread time-out detection applied to these DBATs

High Performance DBATs – MODIFY DDF

- -MODIFY DDF PKGREL command
  - Alters DDF's inactive connection processing
    - Applies only to CMTSTAT=INACTIVE
  - PKGREL(BNDOPT) honors package bind option
  - PKGREL(COMMIT) forces RELEASE(COMMIT)
    - Allow BIND and DDL concurrent with distributed work
  - PKGREL(DEALLOC) forces RELEASE(DEALLOCATE)
    - Provides better performance behavior
    - BIND and DDL be concurrent distributed work runs



-MODIFY DDF PKGREL (BNDOPT)

Access to Currently Committed Data

- Prior to DB2 10
  - Read applications acquire locks on data
    - Resulting in overhead, contention, concurrency issues
  - UR avoids contention
    - But can return uncommitted data as well as committed
  - Could result in timeouts
- DB2 10 NFM
  - Allows access to version of data that was last committed
    - Version of data that existed before blocking unit-of-work changed the row
      - But not yet committed the change
  - Ability to return currently committed data without waiting for locks
  - For uncommitted inserts or deletes
    - Not uncommitted updates
  - Referenced data must be in a UTS

Currently Committed Options

- BIND

```
CONCURRENTACCESSRESOLUTION
(USECURRENTLYCOMMITTED | WAITFOROUTCOME)
```

- PREPARE

```
USE CURRENTLY COMMITTED | WAIT FOR OUTCOME
```

- CREATE/ALTER of PROCEDURE or FUNCTION

```
CONCURRENT ACCESS RESOLUTION
USE CURRENTLY COMMITTED / WAIT FOR OUTCOME
```

Currently Committed – Prepare Options Comparison

- SKIP LOCKED DATA
  - Skip data on which incompatible locks are held by other transactions

- USE CURRENTLY COMMITTED
  - Use currently committed version of the data for applicable scans when the data is in the process of being updated or deleted
  - Rows that are in the process of being inserted can be skipped
  - Applies when the isolation level CS. Else, ignored
  - When specified, the setting of the EVALUNC applies
  - If the row qualifies, this determines if the row is accessed or skipped
  - If SKIPUNCI is in effect, PREPARE uses the specification of this clause

- WAIT FOR OUTCOME
  - DB2 waits for the commit or rollback when encountering data that is in the process of being updated or deleted
  - Rows that are in the process of being inserted are not skipped

Parallelism - Workfile Usage

- DB2 10 CM
  - Allow parallelism if a parallel group contains a work file
  - View or table expression is materialization results in a work file
    - Previously the work file was not shared among child tasks and would disable parallelism
  - Work file is shareable
    - Applies to CP parallelism
    - Not supported for full outer join

Parallelism – Data Distribution

- Prior to DB2 10
  - DB2 divide the number of keys or pages by the number representing the parallel degree
  - One task is allocated per degree of parallelism
  - Range is processed and the task ends
  - Tasks may take different times to process due to uneven distribution/skew
- DB2 10
  - Straw Model workload distribution method may be used
  - More key or page ranges will be allocated than the number of parallel degrees
  - Same number of tasks as before are allocated (same as degree)
  - Once a task finishes its smaller range it will process another range
  - Skewed data has the opportunity to be divided into a smaller number of pieces
  - IFCID 363
    - Used to monitor straw mode parallelism

Dynamic SQL Cache and Literals

- Prior to DB2 10
  - To allow for reuse in the cache
    - SQL string had to be identical and the SQL had to be executed by the same user
    - Programs had to be coded with parameter markers (“?”)
      - Values are similar to using host variables in static SQL.
      - SQL then would always be identical
        - Even if the values in the parameter markers changed
    - Some dynamic SQL uses literals rather than using parameter markers
      - Literals are likely to be different with every execution of the SQL
        - No reuse of the SQL in the cache
        - A prepare must take place for each unique piece of SQL
- DB2 10
  - Provides literal replacement
    - Dynamic SQL with literals can now be re-used in the cache
    - Literals replaced with ‘&’
      - Similar to parameter markers but not the same

Dynamic SQL Cache and Literals - Usage

- Original SQL with literals is looked up in the cache
  - If not found, literals are replaced and new SQL is looked up in the cache
  - Additional match on literal usability
- Can only match with SQL stored with same attribute, not parameter marker
  - If not found, new SQL is prepared and stored in the cache
- Reuse of prepared statements in the dynamic statement cache
  - CONCENTRATE STATEMENTS WITH LITERALS
  - To be eligible for reuse of constant in new and cached statement
    - Immediate usage context
    - Data type
    - Data type length and size

    } *Must be the same*
- If both instances of the constant meet the criteria for reuse
  - CONCENTRATE STATEMENTS WITH LITERALS option
    - Can be shared by same SQL statement with different constants
  - New dynamic SQL statement will share the cached statement
    - New statement will use it's own literal constants executed, not constants of cached statement

Improved RID Processing

- Prior to DB2 10
  - At run time DB2 builds a RID list if necessary for query processing for such processes as list prefetch
  - If the RID pools is full DB2 falls back to a table space scan
  - Another problem is that if DB2 determines that the predicate will qualify more than 25% of the rows then it also fallback to table space scan
- DB2 10
  - During runtime RID overflows now use a work file to continue processing
    - Avoids fallback to table space scan
  - May increase work file usage
    - MAXTEMPS_RID
      - DSNZPARM for maximum work file usage for each RID list
    - Sort type work file usage (as opposed to DGTT-type)
    - Increase is expected to be small
    - RID Pool default has increased from 8MB to 400MB

MAXTEMPS_RID

Predicate Transitive Closure Improvements

- Prior to DB2 10
  - Predicate transitive closure supported for the following
    - =, BETWEEN, <, <=, >, >=
- DB2 10 (CM)
  - Predicate transitive closure supported for IN Lists
  - IN Lists can be generated for columns that are also referenced by another predicate

```
SELECT COL1
FROM TAB1, TAB2
WHERE TAB1.COL1 = TAB2.COL1
AND TAB1.COL1 IN ('A', 'B', 'C')
```

```
SELECT COL1
FROM TAB1, TAB2
WHERE TAB1.COL1 = TAB2.COL1
AND TAB1.COL1 IN ('A', 'B', 'C')
AND TAB2.COL1 IN ('A', 'B', 'C')
```

  - With the additional predicate
    - Optimizer may choose to access table in a different order
    - Usually the table that qualifies fewer rows first

IN List Direct Table Access and List Prefetch

- IN List Direct Table Access
  - Uses in-memory tables to process one or more IN-list predicates as matching predicates

```
SELECT COL1
FROM TAB1
WHERE TAB1.COL1 IN ('X', 'Y', 'Z'):
```

  - Supports matching on multiple IN-list predicates
    - If indexes exist on the necessary columns

```
SELECT COL1, COL2
FROM TAB1
WHERE TAB1.COL1 IN ('X', 'Y', 'Z')
AND TAB1.COL2 IN ('P', 'D', 'Q')
```

(X,P),(X,D),(X,Q), (Y,P), (Y,D) etc…

No limit to the number of IN-list predicates that DB2 builds into an in-memory table

  - Takes each of the IN-lists and builds in-memory tables
    - Then uses nested loop joins to join in-memory tables to target table
      - With the potential for matching on multiple columns of the index
- If IN-list predicate is selected as matching predicate and list prefetch is chosen
  - The IN-list predicates are accessed as an in-memory table
  - Otherwise you get normal In-list access without list prefetch

IN List Direct Table Access and List Prefetch (cont..)

- Can be viewed in PLAN_TABLE
  - TNAME = DSNINnnn(QB)
    - nnn=predicate #
    - QB=Query Block
  - TABLE_TYPE= 'I'
  - ACCESSTYPE ='IN'

If IN-list direct table access is not used, then normal access ACCESSTYPE = 'N' with no list prefetch

```
SELECT COL1
FROM TAB1
WHERE TAB1.COL1 IN (?, ?, ?):
```

QBNO	PLANNO	METHOD	TBNAME	ACCESSTYPE	MCLS	ACCESSNAME	TBTYP	PFETCH
1	1	0	DSNIN001(01)	IN	0		I	
1	2	1	TAB1	I	1	IX1	T	L

Shows access to in-list table

Range List Access

- Prior to DB2 10
  - OR predicates often result in multi-index access with list prefetch
  - Not as efficient as single index access
  - Multi-index access retrieves all RIDs that qualify from each OR condition and then unions the result
- DB2 10 (CM)
  - Range-list index scans
    - Simplifies processing of OR predicates that can be mapped to a single index
    - Improves the performance of applications with data-dependent pagination
      - Or in situations where tables hold a denormalization of several different tables providing the same functionality
  - DB2 logically breaks down OR predicate into a range list of two ranges
    - Then performs two index probes:
    - Scans the index once
      - Consuming fewer RID list resources than multiple index scans
  - Supports single index access without list prefetch

ACCESSTYPE="NR"

Range List Access (cont..)

- Requires every OR predicate references the same table and has at least one matching predicate mapped to same index
- Ordering of the access path steps will be determined at run time
  - PLAN_TABLE represents the order coded in the query

Index: LNAME, FNAME

```
SELECT LNAME, FNAME, ADDRESS
FROM CUSTOMER
WHERE
   (LNAME = 'Lawson'  AND FNAME > 'Susan')
    OR (LNAME > 'Lawson')
ORDER BY LNAME, FNAME
FETCH FIRST 5 ROWS ONLY;
```

First probe

(LNAME = 'Lawson' AND FNAME > 'Susan')

All remaining 'Lawson' rows are scanned until no more rows qualify the first probe predicate or until the application stops fetching rows
No need to sort the result set because rows are retrieved in required order through the index

If fetching 5 rows is not satisfied by first probe, then a second probe of the index is executed with (LNAME > 'Lawson')
A new position is established in the index and scanning continues until either all rows are processed or until the required number of rows are returned to the application

Second probe

(LNAME > 'Lawson')

QBNO	PLANNO	METHOD	TBNAME	ACCESSTYPE	MC	ACCESSNAME	QBTYPE	MIXOPSEQ
1	1	0	CUSTOMER	NR	2	IX1	SELECT	1
1	1	0	CUSTOMER	NR	1	IX1	SELECT	2

Early Application of Stage 2 Predicates

- Prior DB2 10
  - All stage 1 predicates processed first
  - Data passed from stage 1 to stage 2
  - Stage 2 predicates processed after all stage 1 predicates
  - No possibility to use an index to reduce data retrieved from a table with a stage 2 predicate
    - Unless you used index on expression
- DB2 10 (CM)
  - Some stage 2 predicates can be processed during stage 1
  - Call made from stage 1 to stage 2
  - Data can be eliminated earlier in the process
  - Indexes can be utilized (great improvement)
    - A stage 2 predicate can be used as index screening
  - Access is indicated in the DSN_FILTER_TABLE EXPLAIN table
    - PUSHDOWN column
      - I = Index manager evaluated the predicate
      - D = data manager evaluated the predicate
      - Blank = predicate was NOT pushed down

More Aggressive View and Expression Merge

- In DB2 9
  - DB2 performed materialization on some queries utilizing views and table expressions
  - Merge - view or nested table expression syntax physically merged with referencing portion of statement
  - Materialization - view or table expression materialized into a work file then work file read by referencing portion of statement
- DB2 10 (CM)
  - Can merge views and table expressions in outer joins more often
    - Queries involving IFNULL, NULLIF, COALESCE, CASE, VALUE expressions in the preserved row table of an outer join
    - When there is a subquery predicate in a table expression or single table view on the null supplying table
    - Can merge some correlated nested table expressions
      - If query contains a correlated table expressions without a GROUP BY, DISTINCT, or column function
    - If a subquery in the view or table expression is on the preserved side of an outer join

More Aggressive View and Expression Merge

- Can merge views and table expressions in outer joins more often
  - In some situations involving IFNULL, NULLIF, COALESCE, CASE, VALUE expressions in the preserved row table of an outer join
    - Except when CASE expression merges to become a join predicate

```
SELECT A.COL1, B.COL1, A.COL2, B.COL2
FROM TAB1, (SELECT COALESCE(COL1, 0) AS
COL1 ,COL2
FROM TAB2 ) A
LEFT OUTER JOIN
(SELECT COALESCE(COL1, 0) AS COL1 ,COL2
FROM TAB3 ) B
ON A.COL2 = B.COL2
WHERE TAB1.COL2 = A.COL2;
```

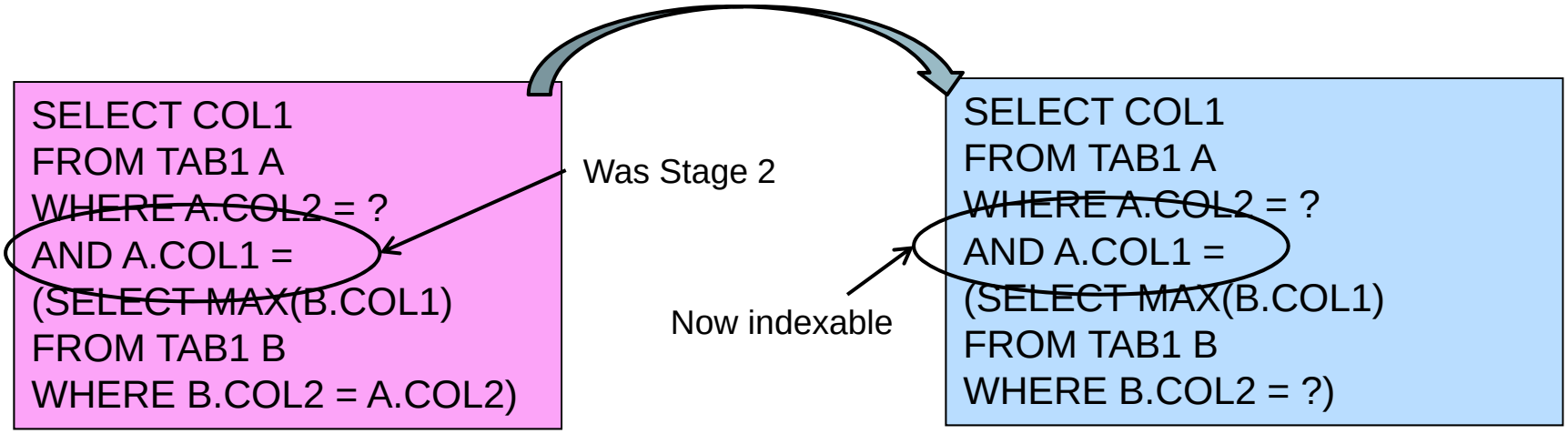
Pre-DB2 10 this is materialized

DB2 10 will rewrite query to avoid materialization

```
SELECT COALESCE(A.COL1, 0) AS COL1,
B.COL1, A.COL2, B.COL2
FROM TAB1, TAB2 A
LEFT OUTER JOIN
(SELECT COALESCE(COL1, 0) AS COL1
,COL2 FROM TAB3 ) B
ON A.COL2 = B.COL2
WHERE TAB1.COL2 = A.COL2;
```

Correlated to Non-Correlated Query Rewrite

- Prior to DB2 10
  - DB2 had limited ability rewrite correlated queries to be non-correlated
- DB2 10
  - The optimizer can rewrite correlated subqueries to be non-correlated
    - If the correlation predicates are covered by local predicates in outer query
  - This may potentially result in additional index matching predicate



Query Sort Enhancement (cont..)

- DB2 10
  - Allows for larger results to avoid work file sorts
    - (N * (sort key + data)) < 128 KB
  - Sort avoidance for small sorts to intermediate sorts
    - Except for parallelism or SET functions

```
SELECT  ACCT_ID,  ACCT_STATUS
FROM    ACCOUNT
WHERE  ACCT_ID  IN
          (SELECT ACCT_ID
            FROM  BAD_ACCOUNTS
            ORDER BY AMOUNT_OWED DESC
            FETCH FIRST 5 ROWS ONLY)
```

Sorted in memory
*if (N * (sort key + data)) < 128 KB*

  - Hashing technique(new) for managing large sorts
    - Potentially reduces number of merge passes needed to complete sorts
    - Can be used to identify duplicates on input to sort with functions
      - Such as DISTINCT or GROUP BY

Query Sort Enhancement

- Prior to DB2 10
  - Tournament tree sort is avoided
    - For small number of rows for FETCH FIRST N ROWS with ORDER BY
      - No need to sort the entire answer set
      - Only the top N are sorted
      - ORDER BY exploits FETCH FIRST n ROWS
        - Work files are not created (less I/O)
        - Only if (N * (sort key + data)) < 32 KB
  - Avoids allocating work files for small sorts for final sorts only
    - If no greater than 255 rows are to be sorted
      - And the result is < 32 KB (sort key + data)

```
SELECT  ACCT_ID,  ACCT_STATUS
FROM    ACCOUNT
WHERE  ACCT_ID  IN
          (SELECT ACCT_ID
            FROM  BAD_ACCOUNTS
            ORDER BY AMOUNT_OWED DESC
            FETCH FIRST 5 ROWS ONLY)
```


Sorted in memory
*if (N * (sort key + data)) < 32 KB*

Native SQL Procedure Performance

- DB2 9
  - Native SQL Procedures were introduced
  - Procedures execute in DBM1, not WLM address space
    - Performance improvements due to less cross-memory calls
- DB2 10 (CM)
  - Additional performance optimization
  - Specific CPU reduction in commonly used area
  - Section load avoidance with SET statements with function
  - Path length reduction in IF statement
  - Optimization in SELECT x from SYSDUMMY1
  - Chained SET statement support (NFM)
  - Some IBM OLTP workload saw performance improvements
    - 20% CPU reduction with V10 CM
    - 89% DBM1 Below the Bar usage reduction
    - 5% response time improvement due to latch contention relief

Will need to recreate or regenerate SQL PL procedures to realize the performance enhancements

SQL Table Functions

- An user defined SQL table function is
  - A function that is written exclusively in SQL statements
  - Returns a single result table
  - A function that behaves as a parameterized view
- Usage
  - Define a parameter for a transition table
  - Define a parameter as a built-in type or a distinct type
  - Include a single SQL PL RETURN statement that returns a result table
- Capabilities
  - Take common SQL statements and make available as table functions
  - Parameter input and use of parameters in the SELECT statement coded in the RETURN clause
    - Allows control over the predicate applied in the statement
      - Allows for more rigid control over the statement than a view offers
  - Best for SQL statements that process little or no data
    - Primarily used to limit returned result via a predicate
      - Best utilized for probing data, in transaction based applications

SQL Table Functions Creation

- CREATE FUNCTION…
  - Used to define a user-defined SQL table function
  - The CREATE FUNCTION statement contains the source code
  - Source code is a SQL statement
  - Function returns a table – it's a parameterized view!!
  - Good for transaction processing applications

```
CREATE FUNCTION DEPT_INFO (IN_WORKDEPT CHAR(3))
RETURNS TABLE (AVGSAL DECIMAL(9,2) , HDCOUNT INTEGER)
LANGUAGE SQL
SPECIFIC DEPTINFO
NOT DETERMINISTIC
READS SQL DATA
RETURN SELECT AVG(SALARY), COUNT(*)
FROM   DSN81010.EMP
WHERE  WORKDEPT = IN_WORKDEPT;
```

Example accepts a department number as input and returns a table containing aggregated employee information for that department.

Invoking the SQL Source Table Function

```
SELECT *
FROM TABLE(DANL.DEPT_INFO(CAST('E01' AS CHAR(3))))
AS TAB1(AVGSAL, HDCOUNT)
```

Table Specification Names the Columns

- Invocation with a correlated reference as Input

Column Names From Function Definition

```
SELECT TAB1.EMPNO, TAB1.SALARY,
       TAB2.AVGSAL,TAB2.HDCOUNT
FROM   DSN81010.EMP TAB1
, TABLE(DANL.DEPT_INFO(TAB1.WORKDEPT)) AS TAB2
WHERE  TAB1.JOB = 'SALESREP';
```

Parameterized View!

Can be used not only to centralize functionality, but also access paths

Works similar to a correlated nested table expression and thus good for transactions

The function is used as any other table would be used

The SQL within the function is ultimately merged with the SQL in the outer statement

Virtual Storage Improvements

- Prior to DB2 10
  - Amount of available virtual storage below the bar limited number of concurrent threads for a single DB2 subsystem
  - DB2 8 moved many DB2 control blocks and work areas (buffer pools, castout buffers, compression dictionaries, RID pool, EDM pool) above the bar
  - DB2 9 provides 10-15% more DB2 threads but some had to expand their environments horizontally to support workload growth in DB2, by activating data sharing or by adding further data sharing members
- DB2 10
  - Provides a dramatic reduction of virtual private storage below the bar
  - Moves 50-90% of the current storage above the bar by exploiting 64-bit virtual storage
  - More threads per image allowed (approximately 2500)
  - Allows as much as 10 times more concurrent active tasks in DB2
  - Less detailed virtual storage monitoring needed
  - Can potentially have fewer DB2 members
    - Can reduce the number of logical partitions (LPARs)
  - Provides cost reductions, simplified management, and easier growth

Consolidation of Data Sharing Members

- DB2 10 can provide significant virtual storage constraint relief
  - Consider consolidating data sharing groups to have fewer members
    - Especially if members were added to provide virtual storage constraint relief
- In DB2 10
  - Most thread-related storage is above the bar in DBM1
    - Which can provide significant virtual storage constraint relief
  - Need rebind static packages after migration
    - Some of this storage for static SQL is moved above the bar immediately after migrating to DB2 10
      - Before rebinding any packages
  - For dynamic SQL
    - Virtual storage constraint relief happens automatically when statements are prepared
    - Location of the thread storage allows potential to run more concurrent active threads for each DB2 subsystem or member
    - Provides the opportunity to consolidate to fewer members

1 MB Page Size

- 1 MB page size (frame size) supported
  - DB2 can use the new 1 MB page size on the Z10
    - DB2s pages are still 4-32K
  - Need to specify PGFIX=YES
    - To get 1 MB page size in buffer pool
      - Must be backed by real storage
      - Must allocate space above the bar
        - With LFAREA parm in IEASYSxx in Parmlib
    - Potential for significant performance improvements
      - Manipulating in 1MB vs KB chunks reduces overhead in memory
      - Increases hit ratio of hardware translation lookaside buffer
  - Using the 1 MB page size will enable efficiencies in the hardware

Data Sharing – LRSN Assignment Improvement

- Prior to DB2 10
  - LRSN had to be unique on a page
  - Could cause latching contention
  - DB2 9 provides a function called LRSN spin avoidance
    - Allows duplicate LRSN values for consecutive log records on a member
    - Consecutive log records that are written for updates to different pages (i.e. data and index page)
      - Can share the same LRSN value
  - However consecutive log records for same index/data page must still be unique
- DB2 10 NFM
  - LRSN in data sharing does not have to be unique within a page
  - Consecutive log records for inserts to the same data page
    - Can have the same LRSN value
  - Performance improvement for data sharing
    - Especially for high volume environments
      - No wait(spin) to get unique LRSN
    - Will help with multi-row insert application performance

Only for INSERT!

DELETE and UPDATE still require unique LRSN

Summary – DB2 10 Performance Opportunities

- DB2 10 for z/OS offers many opportunities for improving performance
  - Database Features
    - Member cluster on UTS
    - INCLUDE index columns
    - Inline LOBs
  - SQL and Application Features
    - Improved access paths
    - LOB streaming
    - Native SQL procedures
    - XML versioning
  - System Features
    - Virtual storage relief
    - In-memory tables
    - Work file enhancements
- All features are designed to provide better performance
  - Must considered what it may take to implement and future maintainability
  - Must consider true usage capabilities

Always plan…
Always test…

DB2 10 for z/OS Performance Opportunities

Susan Lawson
YL&A

Session: F01, Monday, Nov 14th 11:30-12:30
Platform: z/OS