Interesting SQL Topics from Db2 12 for z/OS

Speaker - Judy Nall

Computer Business International, Inc. (CBI)

DB2 Education and Consulting

IBM Gold Consultant

IBM Data Champion

IBM DB2 DBA, System Administration and Programming Certifications

www.cbi4you.com

866.224.4968 Toll Free

Topics

- Advanced trigger support
- Pagination
- MERGE statement
- KEEPDYNAMIC (YES) after ROLLBACK
- UNION ALL and OUTER JOIN performance enhancements
- Predicate optimizations
- Optimizer cost model enhancements
- Runtime enhancements
- Sort space reductions and in-memory use
- SQL differences when Migration from Db2 11 to Db2 12

Advanced Trigger Support

- Starting in Db2 12 for z/OS, triggers can be
  - Written in SQL Procedural Language (SQL PL)
  - If a trigger is written with SQL PL, it is known as an advanced trigger

TRIGGER SUPPORT

```
CREATE TRIGGER BAS_TRG_BONUS
  AFTER INSERT ON MYEMP
  FOR EACH ROW MODE DB2SQL   -- MODE DB2SQL indicates a Basic Trigger
 BEGIN ATOMIC
   UPDATE MYEMP SET BONUS = BONUS + 5000;
 END!
```

(exclamation point (!) is used as the SQL terminator)

Advanced Trigger Support

```
CREATE TRIGGER ADV_TRG_SAL
  AFTER INSERT ON MYEMP
  FOR EACH ROW       -- no MODE DB2SQL indicates an Advanced Trigger
 BEGIN ATOMIC
   UPDATE MYEMP SET SALARY = SALARY + 1000;
 END!
COMMIT!
```

(exclamation point (!) is used as the SQL terminator)

TRIGGER – CATALOG

```
SELECT TYPE, VERSION, LENGTH(VERSION) AS LEN_VER, NAME
  FROM SYSIBM.SYSPACKAGE
  WHERE NAME = 'BAS_TRG_BONUS' OR NAME = 'ADV_TRG_SAL'!

SELECT SQLPL, HEX(SQLPL) AS HEX_SQLPL,
       DEBUG_MODE AS DBG, HEX(DEBUG_MODE) AS HEX_DBG,
       VERSION, LENGTH(VERSION) AS LEN_VER, NAME
  FROM SYSIBM.SYSTRIGGERS
  WHERE NAME = 'BAS_TRG_BONUS' OR NAME = 'ADV_TRG_SAL'!
```


PAGINATION

- Db2 12 for z/OS introduces two types of pagination support:
- Data-dependent
- Numeric-based

PAGINATION

- Numeric based pagination
  - Db2 12 introduces the OFFSET clause to define the number of rows to skip from the beginning of the query result table
  - **SELECT * FROM TAB *OFFSET 10 ROWS* FETCH FIRST 10 ROWS ONLY;**
- *Data-dependent* pagination, which uses row value expressions in a basic predicate, which enables an application to access part of a Db2 result table based on a logical key value:
  - **WHERE (LASTNAME, FIRSTNAME) > (SMITH, JOHN)**
- Prior to 12;
  - **WHERE (LASTNAME = 'SMITH' AND FIRSTNAME > 'JOHN') OR (LASTNAME > SMITH)**

PAGINATION

- New pagination syntax provides useful developer productivity gains while ensuring performance and operational costs are not impacted

```
SELECT NAME, PRICE
FROM MYPRODUCT
ORDER BY PRICE ASC OFFSET 3 ROWS
```

- OFFSET – Define a number of rows to skip from the beginning of the table

PAGINATION

- Data-dependent pagination is a method whereby an application developer can display pages of data at a time
  - The data returned in the last row of the page is the data used to fetch the next page of data
  - Application developers code this style of pagination in their applications, returning pages of data until no more rows qualify
- Ensure data-dependent pagination returns the correct and proper data:
- All of the columns specified in the ORDER BY clause must be in either ascending or descending order
- All of the columns specified in the ORDER BY clause must Generate a unique value
  - at least one column in that row must distinguish that row from all other rows

PAGINATION

- The basic predicate can now compare one row-value-expression with another row-value-expression with the greater than (>) operator:

```
SELECT VIN, MODEL, PRICE, MILEAGE
FROM myAUTOS
WHERE (PRICE, MILEAGE, VIN) > :PRICE_VAR,:MILEAGE_VAR,:VIN_VAR)
ORDER BY PRICE ASC, MILEAGE ASC, VIN ASC
FETCH FIRST 4 ROWS ONLY;
```


PAGINATION

- Return only items 5 - 9 from the original table, the application developer might code the cursor to use the offset-clause in conjunction with the fetch-clause

```
DECLARE CS1 CURSOR FOR
SELECT NAME, PRICE
FROM MYPRODUCT
ORDER BY PRICE ASC
OFFSET 4 ROWS
FETCH FIRST 5 ROWS ONLY;
```


MERGE

- Db2 12 delivers ANSI compliant MERGE capability
- Source can now include TABLE, VIEW and full Select
- Additional predicates on MATCHED/NOT MATCHED
- Can do DELETE
- Can do multiple UPDATE, INSERT and DELETE phrases
- But not on same row
- Can accept SIGNAL and IGNORE

MERGE

- Support for delete. MERGE did not previously allow rows to be deleted from the target table (only to be inserted or updated) thereby occasionally forcing developers to have to revert to other techniques.
  - This restriction has been removed in Db2 12, and MERGE can now delete target rows as well as insert and update them.
- The source for the MERGE can now be another table. Previously MERGE worked only with a host variable array or a list of values, which limited its usefulness in situations where the source data was in an intermediate staging table – a common situation in ETL processing.
- The MERGE statement can now include one or more SIGNAL clauses, to set specific SQL return codes when a condition is met. This will allow developers to more easily take appropriate action in the event of an unexpected condition being encountered (e.g. an invalid value in the target data).

MERGE

- **MERGE is powerful …**
- Input can be a SELECT (JOIN) returning many rows (millions, billions)
- # UPDATEs, INSERTs and DELETEs could explode
- Considerations
- No intermediate commit points
- Long rollback time
- Lock escalation and impact on concurrency
- No SQL pagination support

KEEPDYNAMIC(YES)

- KEEPDYNAMIC(YES) functionality is to avoid a trip from the application to re-prepare dynamic SQL statements after COMMIT, which improves performance
- This functionality does not apply to *ROLLBACK on DB2 11*

KEEPDYNAMIC (YES) after ROLLBACK

- Db2 12 extends the KEEPDYNAMIC(YES) functionality to ROLLBACK also, where prepared dynamic statements are retained at then end of a unit of work

```
PREPARE STMTID FOR INSERT INTO T1 SELECT * FROM T2 WHERE T2.C1 = ?
EXECUTE STMTID USING :H1
EXECUTE STMTID USING :H2
ROLLBACK                          the dynamic INSERT is not unprepared
EXECUTE STMTID USING :H3          successful because statement is still prepared
```

- The DBAT is also kept active so the next re-execution of the statement can resume immediately without the need for re-prepare
- This function is available for local application as well

UNION ALL and OUTER JOIN Performance Enhancements

- The query patterns UNION ALL and Outer Join have similar issues with materialization (workfile) usage and inability to apply filtering early
- Db2 12 introduces the following high-level solutions:
  - Reorder outer join tables to avoid materializations
  - UNION ALL and Outer Join predicate pushdown
  - Avoid workfile for outer materialization
  - Push predicates inside UNION ALL legs or outer join query blocks
  - Sort outer into table expression order
  - Enable sparse index for inner table/view expression
  - Pruning unused columns from materialized result
  - Extended LEFT JOIN table pruning

Reorder Outer Join Tables to Avoid Materializations

- Reorder outer join tables to avoid materializations
- NOW - Allowing Db2 to internally reorder the outer join tables within the query overcomes a limitation that can be exposed when combining outer and inner joins in the same query
- Users who have been exposed to this performance challenge have rewritten their queries to ensure all inner joins appear before outer joins in the query, if possible. Rewriting a query is often difficult

Predicate Optimizations

- Sort for stage 2 join expressions
- User-defined table function predicate optimizations
- VARBINARY data type indexability
- Row permission to correlated subquery indexability
- Additional IN-list performance enhancements

Predicate Optimizations

- Improvement to matching IN-list performance for poor clustering index, allowing the accumulation of 32 RIDs per list prefetch request
- Removes a limitation that IN-lists cannot be used with index screening predicate for range-list access

Sort for Stage 2 Join Expressions

SELECT . . . .

FROM T1,T2

WHERE T1.COL1 = T2.COL1 AND

T1.COL2 = SUBSTR (T2.COL2,1,10)

- Expressions as join predicates are often stage 2, unless the exception for expression on the outer (without sort) for nested loop joins
- Db2 12 allows resolution of expression before sort to support join support for expression on inner and sparse index without sort on outer

Optimizer Cost Model Enhancements

- Dynamic plan stability is considered to be one of the most significant SQL performance enhancements in Db2 12 because of the benefit to dynamic SQL OLTP applications and repeating analytics SQL

Runtime Enhancements

- The source statements in the DSC can be saved in those catalog tables through the new START DYNQRYCAPTURE command
- During subsequent prepared request of the same dynamic statement, Db2 looks in the DSC (Dynamic Statement Cache)
- If the statement is not found in the DSC (cache miss)
  - Db2 will look up the SYSDYNQRY for a stabilized statement using the same criteria for DSC look up
  - If a match is found on the catalog table, a running copy of the statement is made along with its dependency information to insert into the DSC and to be executed by the running thread
  - This process is called loading a stabilized dynamic statement into the DSC, which bypasses the full PREPARE process altogether

Sort Space Reductions and In-Memory Use

- Sort minimization for partial order with FETCH FIRST
- Sort avoidance for OLAP functions with PARTITION BY
- Reducing sort row length
- Improve GROUP BY/DISTINCT sort performance and In-memory sort exploitation
- Sort workfile impacts
- Sparse index improvements

Sort minimization for Partial Order with FETCH FIRST

- Db2 12 reduces the number of rows fetched or processed when there is no index that avoids the sort, if ordering by more than one column and only the leading column has an index
- In the previous DB2 versions, the scenario was fetching all rows and sort into C1, C2 sequence, PROCESSING millions of rows

```
SELECT * FROM T1
ORDER BY C1, C2
FETCH FIRST 10 ROWS ONLY

INDEX1 (C1)
Index entries: 1,1,1,2,2,2,3,3,3,4,4,4,5……999999
```

Improve GROUP BY/DISTINCT Sort Performance and In-memory Sort Exploitation

- Prior to Db2 12, the maximum number of nodes of the sort tree was 32,000, and less for longer sort keys that were limited by ZPARM SRTPOOL
- Db2 12 enables sort tree and hash entries growth to 512,000 nodes (non-parallelism sort) or 128,000 (parallel child task sort)
- In addition, default 10MB SRTPOOL can support 100,000 nodes for 100 byte rows
- Db2 12 performance improvements for GROUP BY/DISTINCT provides a CPU savings when sort can be contained in memory

SQL Differences when Migration to Db2 12

- Trimming columns from a materialized view or table expression if those columns are not required by the outer query
- Pruning unique LEFT OUTER JOIN views or table expressions if columns are not required in SELECT list
- Pushing join predicates into UNION ALL or outer join query blocks if it is cost effective for the optimizer to do so
- Push ORDER BY and FETCH FIRST into UNION ALL legs to allow each UNION ALL leg to avoid a sort (if possible) and fetch only the required rows for the outer query (for FETCH FIRST)
- Reorder outer join tables to avoid unnecessary materializations
- Reduce situations where workfile usage is required for materialized outer join query blocks or UNION ALL legs

Review and Questions

- Advanced trigger support
- Pagination
- MERGE statement
- KEEPDYNAMIC (YES) after ROLLBACK
- UNION ALL and OUTER JOIN performance enhancements
- Predicate optimizations
- Optimizer cost model enhancements
- Runtime enhancements
- Sort space reductions and in-memory use
- SQL differences when Migration from DB2 11 to Db2 12

Review

- Questions?
- Need to contact me?
  - judynall@cbi4you.com
  - www.cbi4you.com for classes and schedules of virtual instructor lead classes
- Thanks for coming to the ADUG meeting!!

- Check out CBI's upcoming classes at www.cbi4you.com
- Instructor led on-line classes with hands-on labs

