Improved query performance in IBM DB2 11 for z/OS

By Terry Purcell, Senior Technical Staff Member, IBM DB2 development

Contents

The immediate performance improvement message in IBM® DB2® 10 for z/OS® (DB2 10) resonated well with DB2 clients. It also provided motivation for the DB2 for z/OS Optimizer development team to look for opportunities to continue this theme in DB2 11 for z/OS (DB2 11).

For the Optimizer, DB2 clients pointed out that performance was not immediate, considering that static SQL needs a REBIND option to take advantage of new access paths, runtime optimizations or both. However, minimal actions, such as using a REBIND command to achieve performance gains, are preferred over rewriting queries or creating extra indexes to achieve performance improvement. Although not all database administrator (DBA) involvement is eliminated in DB2 11 for z/OS, we listened to our clients and are giving you the "closer to immediate" performance you want in DB2 11.

Most business intelligence and business analytics workloads use dynamic SQL. For dynamic SQL, the first execution results in a new prepare statement so that dynamic SQL can achieve immediate optimization. Also, with the ad hoc nature of SQL requests in these environments, minimal effort to achieve performance improvements is preferred.

A major goal for the Optimizer development team was to increase focus on query performance improvements in DB2 11 for z/OS that required minimal action. Considering the amount of diversity in customer workloads, the first question was how do you identify what would benefit the broadest set of clients?

Fortunately, some operations are common to most workloads, but are not necessarily part of the DB2 Optimizer. As such, DB2 11 delivers performance enhancements to the decompression of data rows, sort, and numerous internal code path optimizations (not explained in this paper). Each of these enhancements can generally improve query performance.

The first task for the Optimizer development team was to identify common query patterns, which involved the following tasks:

- Investigating customer workloads that the DB2 development team obtained for performance testing
- Analyzing IBM and other enterprise resource planning (ERP) vendor packaged applications and query generators
- Gathering input from level 2 support on challenging query patterns
- Reviewing lessons learned from client proof of concepts (POCs) and migrations from other platforms

The results of the analysis were used to identify common patterns to target for DB2 11.

DB2 11 for z/OS offers several query performance enhancements that require the least DBA interaction to implement.

Predicate indexability

Over the years, IBM clients were taught that indexable predicates are the most efficient, and stage 2 predicates are the least efficient. In traditional applications, developers were instructed by their DBAs to write their predicates so that matching index access was achievable for their queries. However, query generators, ERP applications, rapid development or geographically dispersed users can all result in limited ability for a DBA to control the quality of the SQL. Therefore, the preference is for DB2 to internally perform these rewrites or to optimize predicate performance.

DB2 11 rewrites some of the more common stage 2 local predicates, including the following predicates, to an indexable form:

- YEAR(DATE_COL)
- DATE(TIMESTAMP_COL)
- value BETWEEN C1 AND C2
- SUBSTR(C1,1,10) SUBSTR from position 1 only

DB2 9 for z/OS delivered the ability to create an index on an expression, which required the developer or DBA to identify the candidate queries and create the targeted indexes. The DB2 11 predicate rewrites allow optimal performance without needing this intervention. In some cases, an index on expression can result in better performance for the query. For this reason, if an index on expression exists, the predicate is not rewritten to an indexable form in DB2 11. Index on expression carries other work, such as resolution of the expression during insert or update, and applicability only to the targeted expression. Coupled with the need for the DBA or user to identify the need for the index, the anticipation is that many of these use cases today do not already have an applicable index on expression.

CASE expressions are also enhanced to support indexability as shown in Figure 1. More common, complex resolution of code values to their business value are being included in a view or table expression to be used within a query, rather than using a code table or dimension table for this purpose. When used in predicates, DB2 11 can now use these expressions as indexable, rather than stage 2 predicates as in previous releases.

Indexability for local predicate

```
SELECT * FROM T1
WHERE COL =   CASE (CAST(? AS INT))
              WHEN 1 THEN 'A'
              WHEN 2 THEN 'B'
              ELSE 'C' END;
```

Indexability for a join predicate

- A CASE expression must be evaluated before the join.
- In the following example, the join predicate is indexable if T1 is accessed before T2.

```
SELECT * FROM T1, T2
WHERE T2.COL = CASE WHEN T1.COL = 'Y'
               THEN T1.COL2
               ELSE T1.COL3
               END;
```

Figure 1: CASE expression indexability

Other patterns that are optimized include OR and IN predicate combinations, which are common in ERP applications. In some cases, single matching index access is now possible where previously only multi-index access was available. In other cases, multi-index access is available, where matching index access was not possible.

In addition, query generators are known to add dummy WHERE clause predicates to simplify their generation framework, for example, WHERE 1=1. DB2 11 enhances these patterns by removing unnecessary "always true" and "always false" predicates in some instances. For clients who historically used such query tricks as OR 0=1 or OR 0<>0, DB2 still accepts these tricks.

Another enhancement is predicate pushdown. Before the release of DB2 11, simple predicates were pushed inside materialized views, such as views that contain DISTINCT or UNION clauses, and table expressions. DB2 11 extends predicate pushdown to include OR predicates, stage 2 predicates, and outer join ON clause predicates. This method allows the filtering to be applied before materialization. This method also benefits workloads that include views or table expressions with DISTINCT or GROUP BY clauses and that are used in outer joins.

These predicate patterns cover a broad array of ERP applications and are known customer concerns.

Duplicate removal

Duplicate removal by using DISTINCT or GROUP BY clauses is another common usage in query processing. DB2 11 is enhanced to improve performance of DISTINCT, GROUP BY, and non-correlated subqueries when an index exists to provide order. In previous releases, DB2 could avoid or minimize the sort overhead for duplicate removal by scanning a candidate index in sequence.

Figure 2 illustrates a simplified example of how DB2 11 can use the non-leaf key information to skip forward to find the next distinct key value. This technique is applicable for DISTINCT, GROUP BY, and non-correlated subqueries. Before DB2 11, DB2 would scan the leaf pages and internally remove duplicates before returning the data to the application. DB2 11 can remove the duplicates earlier by skipping them within the index, regardless of whether the distance between distinct entries is short or long. A greater performance benefit is gained when whole index leaf pages can be skipped.

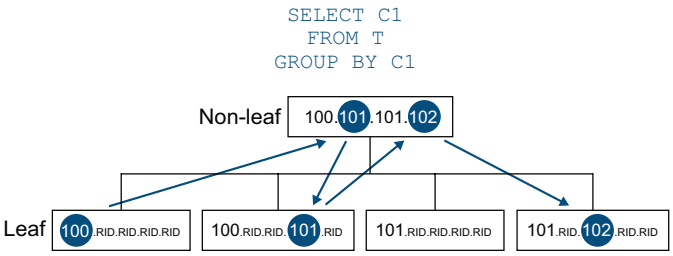

Figure 2: DB2 11 index key skipping

Optimization of DISTINCT and other duplicate removal patterns extends to join queries in DB2 11 where the join is coded as an existence check. In such queries, any duplicates that are introduced from the join are not required for the final result.

Figure 3 demonstrates two examples of the targeted query patterns. DB2 11 can select an early-out join for the inner table as soon as the first match is found rather than processing all matching rows. Before DB2 11, this type of early-out join was available only to correlated EXISTS subqueries that were transformed to a join.

In DB2 11, each inner table probe will stop after the first match is found:

```
SELECT DISTINCT T1.*
FROM T1, T2
WHERE T1.C1 = T2.C1
```

Early-out join also applies to non-Boolean term join conditions with an early-out table:

```
SELECT DISTINCT T1.*
FROM T1, T2
WHERE T1.C1 = 1
   OR T1.C1 = T2.C1
```

Figure 3: Early-out join

Correlated subqueries can also have improved performance in DB2 11 because of optimized usage of a subquery cache that has existed since DB2 V2. Figure 4 shows a simple example of a common query pattern that is used in temporal-based or time-based implementations where the most recent (or least recent) version is required by the query. DB2 11 optimizes this pattern by recognizing when order is provided and by adjusting the cache size as needed.

```
SELECT *
FROM POLICY P1
WHERE P1.POLICY_DATE =
(SELECT MAX(P2.POLICY_DATE)
 FROM POLICY P2
 WHERE P2.CUSTNO = P1.CUSTNO)
```

Figure 4: Correlated subquery pattern optimized in DB2 11

Hash join and sparse index

Most database management systems (DBMS) provide a hash join method for efficient join performance when a suitable join index does not exist or if a large percentage of the rows will be joined. In these cases, a hash join can be more efficient than a nested loop join or merge scan join. Sparse index support has existed since DB2 V4 for noncorrelated subqueries. It was used in DB2 V7 for star joins and was opened to nonstar joins in DB2 9. DB2 10 added hash support, which was limited to cases with no index to support a join.

DB2 11 extends its hash join support by allowing hash joins to be chosen in more cases. It also optimizes memory usage, including runtime validation of available system memory, and appropriately falls back to a sparse index when the result cannot be contained in memory.

These enhancements are available after the first REBIND action for static SQL or the next dynamic SQL execution. However, use of a hash join is controlled by the MXDTCACH zparm. This zparm is conservatively set to 20 MB, which you might consider increasing to gain further improvement. Although optimal setting of MXDTCACH might require some skill, the accounting and statistics reports provide a record of the number of sparse indexes where a work file was built. This information and the number of synchronous reads in the sort workfile buffer pool can be used as a guide to increase MXDTCACH or to reduce VPSEQ in the sort buffer pool.

Page-range screening and indexing for partitioned table spaces

When local predicates exist in the partitioning columns, DB2 can limit the access to only the qualified partitions. In DB2 11, this support is extended to join predicates. This support and other DB2 11 enhancements should increase scenarios where data partitioned secondary indexes (DPSIs) can be used in place of nonpartitioned secondary indexes (NPSIs) for improved utility performance without compromising query performance.

In a partitioning scheme where the table is partitioned by columns used in queries as join predicates, DB2 11 can use those predicates to filter out unnecessary partitions and probe only the qualified parts. This enhancement is most effective when the index for the join is a DPSI and the partitioning columns are excluded from the index or not the leading index columns. Before DB2 11, optimal join performance could be achieved for this partitioning scheme only in either of the following situations:

- The index was created as an NPI.
- The index was partitioned (a partitioning index (PI), not a DPSI), but the partitioning columns were the leading index columns.

Figure 5 demonstrates a join between T1 and T2, where the inner table of the join (T2) uses a DPSI (on C1) and a join predicate exists on the nonindexed YEAR column. In this example, each probe to the inner table can use the join predicate on the partitioning column to ensure that only the necessary partition is accessed for each inner table access.

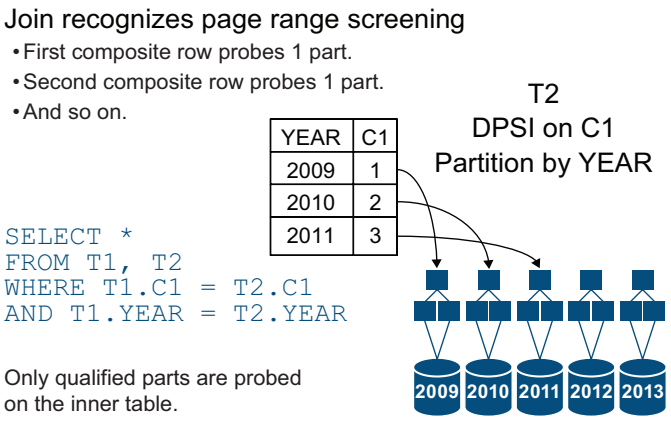

Figure 5: Page-range screening from a join

Before DB2 11, each inner table access probed all partitions, resolving the YEAR join predicate after the data page was accessed. The benefit of the page-range screening enhancement is that clients can convert to using a DPSI, which can ultimately enhance utility performance if fewer NPIs result in a table.

To further expand the usage of DPSIs in DB2 11, additional enhancements are available for workloads that already use DPSIs or are considering moving more of their NPIs to DPSIs. However, although DB2 11 increases the use of DPSIs, still several scenarios exist where having one index b-tree structure (as with NPIs) has considerably better query performance than having one b-tree per partition.

When a query contains an ORDER BY, GROUP BY, or DISTINCT clause, and the query requires a subset of the table rows, an index is more efficient to provide that order rather than introducing a sort. For a partitioned table space, both a PI and an NPI can provide order that can match an ORDER BY, GROUP BY or DISTINCT clause. However, a DPSI provides order only within a partition, not across the entire table space.

DB2 can use one of two methods to allow a DPSI to provide order without requiring a sort:

- Have parallelism provide order, where each child task processes one partition and the parent task merges the results to provide one ordered set.
- Use DPSI merge (also known as DPSI return in order), where DB2 processes each partition serially, but a merge process returns the result in order across all partitions.

The DPSI merge process is enhanced in DB2 11. First, index on expression can now use the DPSI merge, and second, DPSI merge has general performance enhancements such as improved index lookaside and, thus, getpage avoidance.

The next enhancement to DPSIs involves using parallelism for improved join performance when the partitioned table space is the inner table of a join. To benefit from this enhancement, the partitioning scheme must be based on columns that are not included as join predicates in a query. Figure 6 demonstrates the enhancement, which is referred to as a part-level join. In this example, the table is partitioned by YEAR (where each partition is numbered 2009—2013), although the query includes only join predicates on C1.

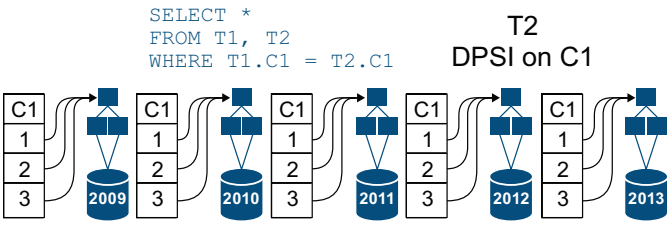

Figure 6: Part-level join

With the part-level join, each parallel child task processes only one partition, and the composite (outer) table is shared or replicated to each child task. This method allows each child task to act as though it is a two-table join (involving one partition) rather than a join to multiple partitions.

A common complaint with DPSI join performance is that the join results in a large amount of random I/O because each partition was probed on the inner table of the join. However, in DB2 11, both page-range screening from join predicates and part-level join should improve join performance for DPSIs. These enhancements might also potentially allow more workloads to convert NPIs to use DPSIs.

RUNSTATS enhancements

The RUNSTATS utility is crucial to the DB2 for z/OS Optimizer to ensure that accurate information is used for access path selection. Although running the RUNSTATS utility is not a feature of DB2 that is devoid of DBA or user involvement, many clients have automated or regular schedules for RUNSTATS collection. Therefore, any enhancements are important that simplify integration with their schedule, improve the ability to recognize important statistics to collect, or both.

DB2 10 delivered statistics profiles, so that clients could combine the complexity of individualized statistics into a stored profile, and subsequent RUNSTATS executions could use that profile to ensure consistency of statistics collection. DB2 11 supports integrating those profiles into a LISTDEF control statement. Therefore, the USE PROFILE keyword can be added to the LISTDEF control statement. Tables with a profile collect their specialized statistics, and those tables without continue to collect the basic statistics.

Simplified syntax in DB2 11 clears the statistics for a table and its associated index. The RESET ACCESSPATH keyword of the RUNSTATS command resets all statistics for the named objects back to -1s and clears any specialized frequency, cardinalities, or histograms from the catalog. After the statistics are cleared, you can collect the desired statistics again.

The RUNSTATS enhancement entails guidance that is provided by the DB2 11 for z/OS Optimizer about the statistics that were missing as part of a BIND or REBIND, dynamic PREPARE, or EXPLAIN action. While determining the access path, the Optimizer externalizes whether the statistics that could be used by the Optimizer are missing or conflicting. This information is externalized to the catalog (from the New Function Mode), a new explain table (if such a table exists), or both. The DBA can use this information to determine which RUNSTATS information to collect to potentially improve the Optimizer's chance at choosing an optimal performing access path. Alternatively, IBM Optim™ Query Workload Tuner can interpret the information and provide the RUNSTATS input.

Externalizing the statistics recommendations as part of general query processing is a significant step forward compared to individually analyzing a query or having tools collect a workload for analysis.

Additional performance improvements

Up to this point, the focus was on DB2 11 performance improvements that might or might not apply to your workload. DB2 11 also features performance improvements that apply generally to an entire workload.

In recent DB2 releases, sort performance was a high priority, considering that most workloads involve sorting. Workload sorting is an area of resource contention because all tasks converge on the same sort buffer pool and data sets. DB2 11 extends in-memory sort capabilities for smaller sorts and temporary storage of intermediate results for some correlated table expressions and subqueries. DB2 11 also provides general code path-length optimizations for sort and reduced workfile usage for final sort. Simplifying this task means reducing contention for workfile resources and improved performance for workloads that involve sorting.

Decompression performance is also improved in DB2 11. This enhancement benefits workloads that issue queries against compressed table spaces. Similar to the sorting enhancement, no DBA involvement is needed to benefit from the enhancement.

Similarly, DB2 11 includes numerous internal optimizations such as to the DECFLOAT data type. This data type is used extensively in XML and when workloads involve local or join predicates with mismatched data types such as character to numeric. In such cases, DB2 uses the DECFLOAT function as an intermediate data type to cast for the conversion. DECFLOAT users can see considerable performance improvements in DB2 11.

Summary

DB2 11 for z/OS is full of query optimization enhancements. Many of these enhancements require minimal involvement to use than in any recent previous DB2 release. The key enhancements in DB2 11 include predicate indexability, duplicate removal, and hash join and sparse index. They also include page-range screening and indexing for partitioned table spaces, RUNSTATS, and other performance improvements. DB2 11 also includes more query optimization enhancements, but that require greater DBA involvement.

For more information

For more information about IBM DB2 for z/OS, see **ibm.com**/software/data/db2/zos/family.

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
September 2013

SWW14016-USEN-00