Business-critical real-time analytics—IBM DB2 11 for z/OS

By Meg Bernal

Contents

Overview

Analytics can provide greater insight into your business and increase your business performance. When you store data on IBM® DB2® for z/OS® and then collect, report, and analyze it directly from DB2 for z/OS, you can achieve better outcomes at lower cost and with less risk. Typically, analytics are performed on large amounts of data that need to be stored for an extended time, yet real-time analytics are becoming more important. DB2 11 for z/OS introduces several enhancements to support business analytics. This white paper introduces the concepts and capabilities to use the new business analytic enhancements. The new enhancements to be discussed include: more efficient performance for query workloads, temporal enhancements, transparent archiving, SQL improvements, and IBM DB2 Analytics Accelerator enhancements. Several examples will help you understand how easy it will be to take advantage of the enhancements.

More efficient performance for query workloads

IBM has invested significantly in DB2 11 to improve performance for query workloads, and the results have been promising. Up to 40 percent CPU improvements have been seen by the DB2 development team at IBM's Silicon Valley Laboratory. Customers who participated in the DB2 11 early support program have reported similar results. Some of the enhancements might be observed without any application or system change, while other enhancements naturally occur after a rebind. Yet other improvements will be realized after modifying applications. Query performance workload improvements have been addressed among the following items:

- ODBC and Java Database Connectivity (JDBC) drivers
- Query execution time optimization
- Large numbers of partitions
- Null indexes
- Missing statistics
- Query transformations

- Index duplicate skipping
- Expression evaluations
- Plan management
- Selectivity overrides
- In-memory techniques
- RID overflows
- Stage 2 (residual) predicate pushdowns
- Sorts
- Work files
- DPSIs (data-partitioned secondary indexes)

A few of the features are discussed in this white paper, and a table in the "Query workload performance summary" paragraph describes when the optimization becomes available.

Access path improvements

Predicate indexability improvements

Several enhancements have been introduced in DB2 11 to improve predicate performance. This is done by improving the indexability of commonly used stage 2 predicates with expressions, as well as common ERP predicate patterns. Other methods include removing "always true" or "always false" literal expressions (for example, where 1=1); reducing the overhead of SELECT list expressions; and converting correlated sub-queries to non-correlated subqueries, when possible.

The following types of stage 2 predicate expressions are heavily used and have been the focus of these indexability improvements: YEAR(DATE_COLUMN), DATE(TIMESTAMP_COLUMN), value BETWEEN COLUMN1 and COLUMN2, and SUBSTR(COLUMN1, 1,10). DB2 might choose to rewrite your queries so that the predicate becomes a stage 1 predicate. Before we take a look at a few examples, recall that rows retrieved for queries go through a two-stage process. Certain predicates can be processed during stage 1, where others can be processed only in stage 2. Predicates that can be performed during stage 1 provide better performance.

In this first example, the YEAR(DATE_COLUMN) expression in the predicate makes the predicate a stage 2 predicate. Therefore, DB2 will rewrite the query so that the stage 2 predicate becomes a stage 1 predicate:

```
-- Original SQL, predicate is stage 2
SELECT * FROM TABLE1
WHERE YEAR(DATE_COLUMN) = 2013;
-- Corresponding rewrite courtesy of DB2;
predicate now indexable stage 1 predicate as a
BETWEEN predicate
SELECT * FROM TABLE1
WHERE DATE_COLUMN BETWEEN '2013-01-01' AND '2013-
12-31';
```

Here's another example, but this time the predicate uses the YEAR(DATE_COLUMN) in an IN predicate. DB2 will rewrite the query to use two BETWEEN predicates separated by an OR condition, therefore transforming the original predicate into two stage 1 predicates:

```
-- Original SQL, predicate is stage 2
SELECT * FROM TABLE1
WHERE YEAR(DATE_COLUMN) IN (2012, 2013);

-- Corresponding rewrite courtesy of DB2,
predicate now indexable stage 1 predicate as a
BETWEEN predicate
SELECT * FROM TABLE1
WHERE (DATE_COLUMN BETWEEN '2012-01-01' AND
       '2012-12-31') OR
       (DATE_COLUMN BETWEEN '2013-01-01' AND
       '2013-12-31');
```

This next example uses a value BETWEEN column1 and column 2 predicate. DB2 will rewrite the query to use basic predicates, which qualify as stage 1 predicates:

```
-- Original SQL, predicate is stage 2
SELECT * FROM TABLE1
WHERE :host_variable BETWEEN START_DATE_COLUMN
AND END_DATE_COLUMN;

-- Corresponding rewrite courtesy of DB2,
predicate now indexable stage 1 predicate as a
basic predicate
SELECT * FROM TABLE1
WHERE START_DATE_COLUMN <= :host_variable AND
      END_DATE_COLUMN >= :host_variable;
```

DB2 11 removes many of the remaining limitations on correlated subquery processing. DB2 now provides support for non-Boolean term subqueries, supports all legs of correlated subqueries with UNION and UNION ALL, and provides support for different data types and lengths. When DB2 rewrites the query to use non-correlated subqueries, the predicates are eligible to be stage 1 predicates.

The example that follows uses a non-Boolean term in the correlated subquery. DB2 will rewrite the subquery by replacing the correlated reference of T1.COLUMN1 with the literal value 5 in order to make the subquery non-corrrelated and make the predicate a stage 1 predicate:

```
-- Original SQL, correlated name T1.COLUMN1
referenced in subquery
SELECT * FROM TABLE1 T1
WHERE T1.COLUMN1 = 5
AND  (T1.COLUMN2 IS NULL OR
     (T1.COLUMN2 = (SELECT MAX(T2.COLUMN2) FROM
     TABLE1 T2 WHERE T2.COLUMN1 = T1.COLUMN1);

-- Corresponding rewrite courtesy of DB2,
subquery becomes non-correlated by replacing
T1.COLUMN1 with 5
SELECT * FROM TABLE1 T1
WHERE T1.COLUMN1 = 5
AND (T1.COLUMN2 IS NULL OR
     (T1.COLUMN2 = (SELECT MAX(T2.COLUMN2) FROM
     TABLE1 T2 WHERE T2.COLUMN1 = 5);
```

If you have a similar query that uses the UNION ALL operator, DB2 will also qualify the query as eligible for a rewrite and transform the correlated subquery into a non-correlated subquery and replace the correlated reference with the corresponding literal value.

```
SELECT * FROM TABLE1 T1
WHERE T1.COLUMN1 = 5
AND T1.COLUMN2 = (SELECT MAX(T2.COLUMN2)
           FROM (SELECT MAX(T2.COLUMN2)
                 FROM TABLE1 T2
                 WHERE T2.COLUMN1 = T1.COLUMN1
                 UNION ALL
                 SELECT MAX(T2.COLUMN2)
                 FROM TABLE3 T3
                 WHERE T3.COLUMN1 = T1.COLUMN1);

-- Corresponding rewrite courtesy of DB2,
subqueries become non-correlated by replacing
T1.COLUMN1 with 5
SELECT * FROM TABLE1 T1
WHERE T1.COLUMN1 = 5
AND T1.COLUMN2 = (SELECT MAX(T2.COLUMN2)
           FROM (SELECT MAX(T2.COLUMN2)
                 FROM TABLE1 T2
                 WHERE T2.COLUMN1 = 5
                 UNION ALL
                 SELECT MAX(T2.COLUMN2)
                 FROM TABLE3 T3
                 WHERE T3.COLUMN1 = 5);
```

Executing access paths more efficiently

Query execution time optimization

Optimizing query execution time has been provided for several common operators: moving data within DB2, casting of numeric data, returning data to calling applications, evaluating the CASE and SUBSTR scalar functions, and performing DECFLOAT data type operations. DB2 might generate customized machine code that is specific to the SQL statement that you write. By generating this machine code, DB2 execution time and CPU utilization could improve. These improvements, introduced by generated machine code, can be realized for dynamic SQL as well as static SQL after a BIND or REBIND. The DECFLOAT data type improvements may be seen ready for immediate use with up to 23 percent CPU reduction for DECFLOAT conversions and roughly 50 percent CPU reduction in the INSERT and FETCH of DECFLOAT columns. Further DECFLOAT improvements might be observed when DB2 is run on the zEC12 hardware.

Reusable work file enhancement

Depending on the complexity of your queries, DB2 might choose to create work files to evaluate the result table. DB2 11 has provided a method to reuse work files for certain types of queries. By reusing work files, scalability improvements and CPU reduction might be observed. If your query contains a correlated subquery with an aggregate function, DB2 might choose to create a work file and possibly to reuse that work file. Let's take a look at two examples where DB2 could choose to reuse a work file. The first example creates a view with the UNION ALL operator. The query against the view references the MAX aggregate function in a correlated subquery. The MAX aggregate function needs to be evaluated before the predicate can be evaluated; therefore, DB2 creates a work file to store the result of the MAX aggregate function. Because the query contains a correlated subquery in the predicate that references a view with UNION ALL, the aggregate function would be evaluated for each row of each leg of the UNION ALL.

```
CREATE TABLE CUSTOMER_INFO (CUSTOMER_ID INTEGER,
BALANCE DECIMAL(8,2), ...);
CREATE TABLE CUSTOMER_INFO2 (CUSTOMER_ID INTEGER,
BALANCE DECIMAL(8,2), ...);

CREATE VIEW CUSTOMER_VIEW AS
          SELECT * FROM CUSTOMER_INFO
          UNION ALL
          SELECT * FROM CUSTOMER_INFO2;

SELECT *
FROM CUSTOMER_VIEW AS CV
WHERE BALANCE = (SELECT MAX(BALANCE)
          FROM CUSTOMER_VIEW AS CV2
          WHERE CUSTOMER_ID = CV.CUSTOMER_ID);
```

The next example uses a join between a table and nested table expression, where the table expression contains a correlated subquery. Once again, DB2 can reuse the work file for the inner table for each outer table row.

```
CREATE TABLE EMPLOYEE (LASTNAME VARCHAR(100),
ASSIGNMENT INTEGER,...);
CREATE TABLE DEPT (DEPTNAME VARCHAR(10), DEPTNO
INTEGER, ...);

SELECT DEPTNAME, EMPLOYEE_COUNT
FROM DEPT, TABLE (SELECT COUNT(*)
          FROM EMPLOYEE EMP
          WHERE EMP.ASSIGNMENT = DEPT.DEPTNO) T1
          (EMPLOYEE_COUNT);
```

Query workload performance summary

The table below summarizes when each of the query workload optimization features may be leveraged.

Feature	Improvement realized
Limited Block Fetch for the ODBC and JDBC drivers	Enabling limited block fetch
Customized machine code	Dynamic SQL or Static SQL after BIND or REBIND
DECFLOAT improvements	Existing plans or packages or Dynamic SQL
Large number of partitions	Existing plans or packages or Dynamic SQL
Null indexes	Dynamic SQL or Static SQL after BIND or REBIND
Missing statistics	Dynamic SQL or Static SQL after BIND or REBIND
Query transformations	Dynamic SQL or Static SQL after BIND or REBIND
Index duplicate skipping	Dynamic SQL or Static SQL after BIND or REBIND
Expression evaluations	Dynamic SQL or Static SQL after BIND or REBIND
Plan management	Dynamic SQL or Static SQL after BIND or REBIND
Selectivity overrides	Dynamic SQL or Static SQL after BIND or REBIND
In-memory techniques	Existing plans or packages or Dynamic SQL or Static SQL after BIND or REBIND
RID overflows	Existing plans or packages or Dynamic SQL
Stage-2 (residual) predicate pushdowns	Dynamic SQL or Static SQL after BIND or REBIND
Sort	Existing plans or packages or Dynamic SQL or Static SQL after BIND or REBIND
Work files	Dynamic SQL or Static SQL after BIND or REBIND
DPSI	Dynamic SQL or Static SQL after BIND or REBIND

Temporal enhancements

With regulatory, auditing, and compliance pressures affecting businesses more frequently, businesses often need to analyze historical, current, and future data. Temporal data management provides the technology that businesses need to keep track of time-oriented data. DB2 10 for z/OS delivered temporal data management with the introduction of SYSTEM_TIME and BUSINESS_TIME period concepts. SYSTEM_TIME period refers to the *audit time* recording of data changes based on when the data in the database changed. BUSINESS_TIME period pertains to the *valid time* of data changes based on business conditions (for example, insurance policy terms or terms of a loan). DB2 11 provides enhancements to temporal data management with support for views and the introduction of two new special registers that enable users to obtain time-sensitive data without changing existing applications.

Views are often used for security purposes in that they enable only certain individuals to see certain data. One such example of sensitive data would be a customer's government-issued identification. Instead of returning the entire ID, a view can be created to return a portion of the ID. DB2 10 delivered temporal support for base tables, whereas DB2 11 extends the temporal support to views. Temporal queries, as well as temporal UPDATE and DELETE operations, can be executed against views that are based on temporal tables. The following example shows an application-period temporal table that uses a BUSINESS_TIME period and a view that is based on that application-period temporal table.

```
CREATE TABLE MASTER_POLICY_INFO (
  POLICY_ID CHAR(4) NOT NULL,
  CUSTOMER_ID CHAR(11) NOT NULL,
  TYPE CHAR VARYING(10) NOT NULL,
  COVERAGE INTEGER NOT NULL,
  BUS_START DATE NOT NULL,
  BUS_END DATE NOT NULL,
  PERIOD BUSINESS_TIME(BUS_START, BUS_END)
);

CREATE VIEW POLICY_INFO AS
  SELECT POLICY_ID,
  CONCAT('XXX-XX-',SUBSTR(CUSTOMER_ID,8:11)),…
  FROM MASTER_POLICY_INFO
  WHERE TYPE = 'PRIMARY';
```

Similar to writing temporal queries against base tables, you can now write temporal queries against views that are based on temporal tables.

```
SELECT...FROM POLICY_INFO FOR BUSINESS_TIME AS
OF business_value;
```

DB2 will implicitly apply the period specification to the appropriate type of temporal tables inside the view that is referenced by the query. Notice how the POLICY_INFO view has pulled in the application-period temporal table, MASTER_POLICY_INFO, into the query.

```
SELECT...FROM (SELECT...FROM MASTER_POLICY_INFO
FOR BUSINESS_TIME AS OF business_value
           WHERE TYPE = 'PRIMARY') POLICY_INFO ;
```

DB2 will then rewrite the query to include the period specification as a predicate.

```
SELECT...FROM (SELECT...FROM MASTER_POLICY_INFO
           WHERE TYPE = 'PRIMARY' AND
           BUS_START <= business_value AND
           BUS_END > business_value) POLICY_INFO ;
```

Similar to modifying temporal tables, now you can also modify temporal views.

```
UPDATE POLICY_INFO FOR PORTION OF BUSINESS_TIME
FROM business_value1 to business_value2
           SET COVERAGE = 10000
WHERE POLICY_ID = 'B001';
```

DB2 will rewrite the above UPDATE statement to update the underlying application-period temporal table for a portion of time (for example, from business_value1 to business_value2.)

```
UPDATE MASTER_POLICY_INFO
  SET COVERAGE = 10000,
           BUS_START = MAX(BUS_START,business_
           value1),
           BUS_END = MIN(BUS_END,business_value2)
WHERE POLICY_ID = 'B001' AND
           TYPE = 'PRIMARY' AND
           BUS_START < business_value2 AND
           BUS_END > business_value1;
```

The two new special registers that are introduced in DB2 11, CURRENT TEMPORAL SYSTEM_TIME and CURRENT TEMPORAL BUSINESS_TIME, make it easier to query data with an implicit period specification and to modify data with an implicit temporal predicate. In other words, after users set the special registers, any query or data modification against a temporal table or temporal view is automatically transformed to include the special register in a predicate of the query or data modification. By simply modifying the special registers, users are able to see data from different perspectives.

Let's look at an example. The following example creates a bitemporal table where both SYSTEM_TIME and BUSINESS_TIME periods are defined. A history table is also created to store historical data. Storing historical data in the history table is enabled by associating the history table to a base table.

```
CREATE TABLE MASTER_POLICY_INFO (
    POLICY_ID CHAR(4) NOT NULL,
    COVERAGE INTEGER NOT NULL,
    BUS_START DATE NOT NULL,
    BUS_END DATE NOT NULL,
    SYS_START TIMESTAMP(6) NOT NULL GENERATED
    ALWAYS AS ROW BEGIN,
    SYS_END TIMESTAMP(6) NOT NULL GENERATED ALWAYS
    AS ROW END,
    CREATE_ID TIMESTAMP(6) GENERATED ALWAYS AS
    TRANSACTION START ID,
    PERIOD BUSINESS_TIME(BUS_START, BUS_END),
    PERIOD SYSTEM_TIME(SYS_START, SYS_END)
);

CREATE TABLE HISTORY_POLICY_INFO (
    POLICY_ID CHAR(4) NOT NULL,
    COVERAGE INTEGER NOT NULL,
    BUS_START DATE NOT NULL,
    BUS_END DATE NOT NULL,
    SYS_START TIMESTAMP(6) NOT NULL,
    SYS_END TIMESTAMP(6) NOT NULL,
    CREATE_ID TIMESTAMP(6)
);
ALTER TABLE MASTER_POLICY_INFO ADD VERSIONING USE
HISTORY TABLE HISTORY_POLICY_INFO;
```

After populating the table, you might want to know what your policy information will look like after the start of the new year. Set the CURRENT TEMPORAL SYSTEM_TIME special register to New Year's Day and execute your query:

```
SET CURRENT TEMPORAL SYSTEM_TIME = '2014-01-01-
08.00.00.000000';
SELECT...FROM MASTER_POLICY_INFO WHERE POLICY_ID =
'A001';
```

DB2 will implicitly generate a period specification to your query:

```
SELECT...FROM MASTER_POLICY_INFO
FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME
WHERE POLICY_ID = 'A001';
```

DB2 will then rewrite the query to use the CURRENT TEMPORAL SYSTEM_TIME in evaluating the result table:

```
SELECT...FROM (SELECT...FROM MASTER_POLICY_INFO
      WHERE SYS_START <= CURRENT TEMPORAL
      SYSTEM_TIME AND
           SYS_END > CURRENT TEMPORAL SYSTEM_TIME
      UNION ALL
      SELECT...FROM HISTORY_POLICY_INFO
      WHERE SYS_START <= CURRENT TEMPORAL SYSTEM_
      TIME AND
           SYS_END > CURRENT TEMPORAL SYSTEM_TIME
      ) MASTER_POLICY_INFO
WHERE POLICY_ID = 'A001';
```

You can also look at your policy information as of tomorrow by simply modifying the special register and executing the same query.

```
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT
TIMESTAMP + 1 DAY;
SELECT … FROM MASTER_POLICY_INFO WHERE POLICY_ID =
'A001';
```

Let's say you want to review your policy information as of a certain time in addition to your system time. Set both the CURRENT TEMPORAL BUSINESS_TIME and CURRENT TEMPORAL SYSTEM_TIME special registers, and execute the same query.

```
SET CURRENT TEMPORAL BUSINESS_TIME = '2013-07-01-
08.00.00.000000';
SET CURRENT TEMPORAL SYSTEM_TIME = '2013-01-01-
08.00.00.000000';
SELECT...FROM MASTER_POLICY_INFO WHERE POLICY_ID =
'A001';
```

Once again, DB2 will rewrite your query to include both SYSTEM_TIME and BUSINESS_TIME period specifications.

```
SELECT...FROM (SELECT...FROM MASTER_POLICY_INFO
      WHERE SYS_START <= CURRENT TEMPORAL SYSTEM_
      TIME AND
           SYS_END > CURRENT TEMPORAL SYSTEM_TIME
           AND
           BUS_START <= CURRENT TEMPORAL BUSINESS_
           TIME AND
           BUS_END > CURRENT TEMPORAL BUSINESS_
           TIME
      UNION ALL
      SELECT...FROM HISTORY_POLICY_INFO
      WHERE SYS_START <= CURRENT TEMPORAL SYSTEM_
      TIME AND
           SYS_END > CURRENT TEMPORAL SYSTEM_TIME
           AND
           BUS_START <= CURRENT TEMPORAL BUSINESS_
           TIME AND
           BUS_END > CURRENT TEMPORAL BUSINESS_
           TIME
      ) MASTER_POLICY_INFO
WHERE POLICY_ID = 'A001';
```

As you can see from the preceding examples, DB2 11 provides greater flexibility in writing temporal-based applications, while simplifying the applications.

Transparent archiving

Archiving data, either old data or less frequently used data, is simplified with the introduction of transparent archiving in DB2 11. Data can be archived transparently by using a DELETE SQL statement in combination with a new built-in global variable SYSIBMADM.MOVE_TO_ARCHIVE. In addition, the new archive transparency support in DB2 11 provides easy access to both warm (current) and cold (archived) data within a single query. Archive transparency support for viewing both current and archived data is controlled by a new BIND/REBIND option and a new built-in global variable SYSIBMADM.GET_ARCHIVE. Transparent archiving can be achieved in five steps:

```
CREATE TABLE POLICY_ACTIVE (POLICY_ID CHAR(10) NOT
NULL, COVERAGE INTEGER NOT NULL);
```

Step1: Create a base table.

```
CREATE TABLE POLICY_ARCHIVE (POLICY_ID CHAR(10)
NOT NULL, COVERAGE INTEGER NOT NULL);
```

Step 2: Create a corresponding archive table.

```
ALTER TABLE POLICY_ACTIVE ENABLE ARCHIVE USE
POLICY_ARCHIVE;
```

Step 3: Enable transparent archiving by specifying the new ENABLE ARCHIVE clause on the ALTER TABLE statement.

```
Policy_Updater: PROC(...)
    ...
    DELETE FROM POLICY_ACTIVE...;
    ...
END Policy_Updater;

Policy_Count_Getter: PROC(...)
    ...
    SELECT COUNT(*) INTO ... FROM POLICY_ACTIVE
    ...;
    ...
END Policy_Count_Getter;
```

Step 4: Write an application that deletes and retrieves data from the base table.

```
Policy_Driver: PROC(..)
    ...
    SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y';
    CALL Policy_Updater;   /* data will be
    archived */
    ...
    SET SYSIBMADM.GET_ARCHIVE = 'Y';
    CALL Policy_Count_Getter (...);    /* get the
    total count of both current and archived
    policies  */
    ...
END Policy_Driver;
```

Step 5: Tell DB2 to transparently archive the base data by specifying the new DB2 11 built-in global variable, SYSIBMADM.MOVE_TO_ARCHIVE. To retrieve both current and archived data, you can use the ARCHIVESENSITIVE BIND/REBIND OPTION and the new built-in global variable SYSIBMADM.GET_ARCHIVE.

When the Policy_Driver application invokes the Policy_Updater application and the DELETE statement inside the Policy_Updater application is executed, the deleted data from the POLICY_ACTIVE table will be inserted into the POLICY_ARCHIVE archive table. The query on the POLICY_ACTIVE table in Policy_Count_Getter application will retrieve data from both current and archive table as a result of implicit UNION ALL query transformations.

If, at some point, you no longer want to archive your base data, simply modify your driver to tell DB2 that transparent archiving is no longer needed by modifying the SYSIBMADM.MOVE_TO_ARCHIVE global variable. When only current data is expected, you can use the same query to obtain your results but either modify the SYSIBMADM.GET_ARCHIVE global variable or turn off the ARCHIVESENSITIVE BIND/REBIND OPTION.

```
Policy_Driver: PROC(..)
    ...
    SET SYSIBMADM.MOVE_TO_ARCHIVE = 'N';
    CALL Policy_Updater;                /* data will
    no longer be archived */
    ...
    SET SYSIBMADM.GET_ARCHIVE = 'N';
    CALL Policy_Count_Getter (...);    /* get the
    count of current policies only */
    ...
END Policy_Driver;
```

```
BEGIN
  -- declare local variable
  DECLARE UserId CHAR(5);
  -- declare array variable
  DECLARE ResultArray IntArray;

  -- set local variable to 1st 5 characters of
proc caller id
  SET UserId = SUBSTR(CALLER_ID,1,5);
  -- populate array variable with result based on
user role
  IF (UserId = 'Sales') THEN
    SELECT ARRAY_AGG(CUSTOMER_ID) INTO ResultArray
    FROM POLICY_INFO
    WHERE COVERAGE > 100000;
  ELSE
   SELECT ARRAY_AGG(CUSTOMER_ID) INTO ResultArray
    FROM POLICY_INFO;
  END IF;
END;

-- create caller
CREATE PROCEDURE POLICY_PROC (IN ID CHAR(10), OUT
CODE INT)
BEGIN
  -- declare array variable
  DECLARE CustomerArray IntArray;
  CALL COVERAGE_PROC(ID, CustomerArray, CODE);
  …
END;
```

SQL improvements

Analytics are often used against large amounts of data, and aggregations are performed against those large amounts of data in an effort to summarize the data. DB2 for z/OS aggregation support entails basic aggregations, such as MAX and MIN, as well as basic grouping with the GROUP BY clause. DB2 9 for z/OS introduced a set of moving aggregates, namely moving SUM and moving AVG; these aggregates calculate cumulative sums and moving averages. DB2 10 for z/OS introduced aggregation groups, which enable you to specify which rows of a partition, in relation to the current row, should participate in the calculation. With DB2 11, new aggregations that use grouping sets have been introduced, with CUBE and ROLLUP. Grouping sets enables you to specify multiple groups in your queries, so that querying and reporting are made easier and faster. This is because you produce a single result table by performing a UNION ALL of two or more groups of rows.

Before DB2 11, when you specified GROUP BY on your query, you received groups of rows in which all rows in a group had the same value. Let's look at an example of GROUP BY and the DB2-supplied table DSN8B10.PROJ. In the following example, we group by the DEPTNO and RESPEMP columns.

```
Policy_Driver: PROC(..)
    ...
    SET SYSIBMADM.MOVE_TO_ARCHIVE = 'N';
    CALL Policy_Updater;                /* data will
    no longer be archived */
    ...
    SET SYSIBMADM.GET_ARCHIVE = 'N';
    CALL Policy_Count_Getter (...);    /* get the
    count of current policies only */
    ...
END Policy_Driver;
```

In the picture that follows, the table on the left contains the source data, and the table on the right shows the result table. Notice how departments C01, D01, and E01 each have two individuals reporting to two separate prohjects, whereby the GROUP BY collapses each of those departments to report an overall sum of persons needed to complete the projects for their respective departments.

Original data

DEPTNO	RESPEMP	PRSTAFF
B01	000020	1.00
C01	000030	2.00
C01	000030	1.00
D01	000010	12.00
D01	000010	6.50
D11	000060	9.00
D11	000220	2.00
D11	000150	3.00
D11	000160	3.00
D21	000070	6.00
D21	000230	2.00
D21	000250	1.00
D21	000270	2.00
E01	000050	6.00
E01	000050	5.00
E11	000090	5.00
E21	000100	4.00
E21	000320	1.00
E21	000330	1.00
E21	000340	1.00

Result data

DEPTNO	RESPEMP	PRSTAFF
B01	000020	1.00
C01	000030	3.00
D01	000010	18.50
D11	000060	9.00
D11	000150	3.00
D11	000160	3.00
D11	000220	2.00
D21	000070	6.00
D21	000230	2.00
D21	000250	1.00
D21	000270	2.00
E01	000050	11.00
E11	000090	5.00
E21	000100	4.00
E21	000320	1.00
E21	000330	1.00
E21	000340	1.00

With the introduction of grouping sets, multiple groups can be specified in a single statement, which enables groups to be computed with a single pass over the data. Grouping sets can be used to determine subtotals and grand totals. The pre-defined grouping set, ROLLUP, creates subtotals that "roll up" from the most detailed level to a grand total. Let's look at a ROLLUP example with the same DB2-supplied table DSN8B10.PROJ, but this time we use ROLLUP on the DEPTNO and RESPEMP columns.

A ROLLUP on DEPTNO and RESPEMP is equivalent to three grouping sets:

```
SELECT DEPTNO, RESPEMP, SUM(PRSTAFF)
FROM DSN8B10.PROJ
GROUP BY ROLLUP (DEPTNO, RESPEMP);
```

1. DEPTNO, RESPEMP
2. DEPTNO
3. Grand total

You could write the above query to use the new GROUPING SETS clause to do the same evaluation as ROLLUP:

```
SELECT DEPTNO, RESPEMP, SUM(PRSTAFF)
FROM DSN8B10.PROJ
GROUP BY GROUPING SETS ((DEPTNO, RESPEMP),
(DEPTNO),());
```

Using the ROLLUP on the DEPTNO and RESPEMP columns results in the sum of persons needed to complete the projects for each respective employee for each respective department, the subtotal of the sum of persons needed to complete the projects for each department , and finally the grand total of the sum of all persons needed to complete all projects for all departments.

Results

DEPTNO	RESPEMP	PRSTAFF
B01	000020	1.00
C01	000030	3.00
D01	000010	18.50
D11	000060	9.00
D11	000150	3.00
D11	000160	3.00
D11	000220	2.00
D21	000070	6.00
D21	000230	2.00
D21	000250	1.00
D21	000270	2.00
E01	000050	11.00
E11	000090	5.00
E21	000100	4.00
E21	000320	1.00
E21	000330	1.00
E21	000340	1.00
B01	?	1.00
C01	?	3.00
D01	?	18.50
D11	?	17.00
D21	?	11.00
E01	?	11.00
E11	?	5.00
E21	?	7.00
?	?	73.50

Subtotals for each respemp in each deptno

Subtotals for each deptno

Grand total for all deptno

The pre-defined grouping set, CUBE , creates subtotals for all permutations of the grouping set. Let's look at a CUBE example with the same DB2-supplied table DSN8B10.PROJ, but this time we use CUBE on the DEPTNO and RESPEMP columns.

```
SELECT DEPTNO, RESPEMP, SUM(PRSTAFF)
FROM DSN8B10.PROJ
GROUP BY CUBE (DEPTNO, RESPEMP);
```

A CUBE on the DEPTNO and RESPEMP columns is equivalent to the following four grouping sets:

1. DEPTNO, RESPEMP
2. DEPTNO
3. RESPEMP
4. Grand total

Using the CUBE on the DEPTNO and RESPEMP columns results in the four grouping sets listed above:

1. The sum of persons needed to complete the projects for each respective employee for each respective department,
2. The subtotal of the sum of persons needed to complete the projects for each department,
3. The subtotal of the sum of persons needed to complete the projects for each respective employee; and, finally
4. The grand total of the sum of all persons needed to complete all projects for all departments

DEPTNO	RESPEMP	PRSTAFF
B01	000020	1.00
C01	000030	3.00
D01	000010	18.50
D11	000060	9.00
D11	000150	3.00
D11	000160	3.00
D11	000220	2.00
D21	000070	6.00
D21	000230	2.00
D21	000250	1.00
D21	000270	2.00
E01	000050	11.00
E11	000090	5.00
E21	000100	4.00
E21	000320	1.00
E21	000330	1.00
E21	000340	1.00
B01	?	1.00
C01	?	3.00
D01	?	18.50
D11	?	17.00
D21	?	11.00
E01	?	11.00
E11	?	5.00
E21	?	7.00

Subtotals for each respemp in each deptno

Subtotals for each deptno

DEPTNO	RESPEMP	PRSTAFF
?	000010	18.50
?	000020	1.00
?	000030	3.00
?	000050	11.00
?	000060	9.00
?	000070	6.00
?	000090	5.00
?	000100	4.00
?	000150	3.00
?	000160	3.00
?	000220	2.00
?	000230	2.00
?	000250	1.00
?	000270	2.00
?	000320	1.00
?	000330	1.00
?	000340	1.00
?	?	73.50

Subtotals for each respemp

Grand total for all deptno

IBM DB2 Analytics Accelerator

IBM DB2 Analytics Accelerator is an appliance built on IBM Netezza® technology that has the capability to accelerate query execution. This capability enables businesses to gain insight into their data quickly and predictably. Even though the accelerator is separately licensed from DB2, it is tightly integrated with DB2 and is transparent to DB2 applications. By integrating the accelerator on DB2 for z/OS, users get the qualities of service that they expect from DB2 and IBM System z®: availability, reliability, and security. The accelerator has quickly evolved from a query accelerator to a storage saver and is moving toward being an ELT accelerator with advanced analytics and transparent access to both your transactional and analytic data.

DB2 Analytics Accelerator Version 3 came equipped with the High Performance Storage Saver (HPSS). This enables users to offload historical data directly to the accelerator, thus eliminating the need to store the data directly on System z. This enhancement with HPSS can significantly reduce the cost for storage and provides an option to store the data only once, namely on the accelerator. Another key feature of Version 3 of the Accelerator is support for incremental update, which enables data changes to be propagated to the accelerator as they happen. The z/OS Workload Manager was integrated into the product to help you efficiently manage your accelerator resources. Additional optimization has facilitated much faster and less resource-intensive refreshes of tables and partitions. Also added to Version 3:

- More query routing control
- Support for additional types of queries that can be accelerated
- Support for DB2 OLAP functions
- Support for multiple-byte EBCDIC encoding
- Support for additional DB2 scalar functions TIMESTAMPDIFF and for bit manipulation functions such as BITAND, BITANDNOT, BITOR, BITXOR, and BITNOT
- Support for DECFLOAT for implicit casting

Version 4 of the Accelerator provides even more accelerator opportunities and enterprise features. The Accelerator's Version 4 supports the DB2 11 generally available code base. More queries are eligible to be accelerated with the additional support for the following items:

- Static SQL (the most requested feature of the Accelerator)
- Multiple-row fetch
- Implicit casting when comparing VARCHAR and numeric data
- Additional scalar function support (bringing the total number of scalar functions supported to over 80)
- Support for EBCIDC and Unicode encoding schemes in the same DB2 subsystem and accelerator

A major focus in developing Version 4 was to provide greater scalability and performance of incremental update operations. HPSS now provides better access control to the archived accelerators and enables archiving to multiple accelerators.

Version 4 has provided support for multiple accelerators with group workload balancing and improved workload management for high-priority work items. With Version 4, workload balancing across multiple accelerators is automated. Additionally, the accelerators can notify DB2 about their utilization status by regularly sending the information to all attached DB2 subsystems. DB2 can then check the accelerator's utilization and route the query to the optimal accelerator.

When Workload Manager was integrated in Version 3 of the Accelerator, it allowed DB2 to detect the WLM service class and importance levels and to send this information to the Accelerator with each query submitted from a remote application. The Accelerator would then map the importance level to Netezza's priority and would alter the accelerator session before the query execution. Version 4 extends this support of mapping Workload Manager importance levels to Netezza priorities to local applications, such as SPUFI, TEP3, IBM CICS®, and IBM IMS™.

If you are interested in the Accelerator and want to see how it might benefit you,IBM provides accelerator modeling. This modeling lets you see the potential CPU and elapsed time savings if your DB2 subsystem was connected to an accelerator—without an accelerator actually being connected to your DB2 box. After turning on the ACCELMODEL subsystem parameter, DB2 accounting records (IFCIDs 3 and 148) will include projected CPU and elapsed-time savings for your plans. For additional information, please see DB2 for z/OS APAR PM90886.

Summary

Businesses can gain greater insight into their data through analytics. Whether you run your existing applications, REBIND your applications, or create entirely new workloads, IBM DB2 11 provides greater support for your analytics. Analytics support in DB2 11 can simplify applications, reduce I/O, and improve CPU and elapsed times by means of query workload improvements, temporal enhancements, transparent archiving, and SQL enhancements. Using the DB2 Analytics Accelerator in conjunction with DB2 11 opens up even more opportunities for CPU and elapsed time improvements for your business analytics.

About the author

Meg Bernal is a DB2 for z/OS senior software engineer at IBM's Silicon Valley Laboratory. Meg has 16 years of experience as a developer on DB2 for z/OS where she has contributed to many SQL enhancements, including native SQL PL routines, multiple row fetch, scrollable cursors, and SELECT FROM INSERT/UPDATE/DELETE, to name a few.

Acknowledgments

Several individuals assisted in the writing of this paper. A warm thank you to the talent behind the DB2 for z/OS product: Ann Hernandez, Devon Yu, Jeff Josten, Maryela Weihrauch, Shengxi Suo, Terrie Jacopi, Terry Purcell, and Xiaohong Fu.

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
August 2014

IMW14746-USEN-03