Leverage DB2 BLU Acceleration technology for tables involving unsupported data types

A step-by-step guide

Masahiro Ohkawa (mohkawa@jp.ibm.com)  
Yamato Software Development Laboratory (YSL)  
IBM

Abhishek Kumar Chourasia (abkchour@in.ibm.com)  
Software Developer  
IBM

Introduction

BLU Acceleration is a dynamic in-memory columnar technology. Each column is physically stored in a separate set of data pages. The BLU Acceleration feature is intended for analytics or data mart workloads, where such workloads typically involve regular reporting queries, as well as ad-hoc business intelligence queries that can't be tuned in advance. If your workload is primarily transaction processing, you may want to consider using row-organized tables. Having the ability to store row and column-organized tables in the same database. This allows users to apply BLU Acceleration even in database environments where mixed OLTP and OLAP workloads are required. Shadow tables use Materialized Query Tables (MQTs), where the queries can be automatically processed by effectively using row-organized tables or column-organized tables. But there are some restrictions in column-organized tables. Column-organized tables support the following subset of DB2 data types:

- SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, DECFLOAT

DB2® BLU Acceleration can provide performance improvements ranging from 35x to 73x and beyond for analytical queries with minimal tuning. In the mixed environment between OLTP and OLAP called OLTAP, DB2 shadow tables can be used to leverage BLU Acceleration without sacrificing OLTP performance. On the other hand, there are restrictions, such as data types. Though most data types are supported, LOB and XML, for example, are not. For example:

If your fact table contains invoice PDF data stored in a BLOB column, you cannot use BLU Acceleration as it is, which means you cannot create tables like a column-organized table. But there are ways to leverage BLU Acceleration even in such cases.
• CHAR (including FOR BIT DATA), VARCHAR (including FOR BIT DATA), GRAPHIC, VARGRAPHIC
• DATE, TIME, TIMESTAMP(n)
• Distinct types of a supported type

Let’s suppose we have the following table.

```sql
CREATE TABLE ORDER (ORDER_ID INT NOT NULL PRIMARY KEY,
ORDERPRIORITY VARCHAR(15),
PRODUCT_ID INT,
SHOP_ID INT,
ORDERDATE TIMESTAMP,
QUANTITY INT,
REVENUE INT,
PROFIT INT,
INVOICE BLOB(1M));
```

This cannot be a column-organized table because the data type involved is BLOB.

This tutorial guides you in two different ways of handling this to leverage BLU Acceleration. One method is to split the table between the table that contains all supported data types and the table that contains unsupported data types and create a view to combine them. The other is to use a shadow table, where the shadow table contains only supported data types. Which is preferable depends on the way you want to use the database system. If you use the database system as OLAP, we recommend the former one (Solution 1: Split table and combine them as view below). If you use the database system as OLTAP, we recommend the latter one (Solution 2: Configure shadow table that contains only supported data types below).

**Solution 1: Split table and combine them as view**

Split the table and combine them as a view by joining with a key. For effectively processing, the key is better to be a numeric data type (e.g., INT). The above-mentioned table’s key is ORDER_ID. So the table can be split into the following two tables, where both tables have ORDER_ID column as key. One split table contains all columns except for columns with unsupported data types. So it can be column-organized table. The other table is row-organized table, which contains the key and columns with unsupported data types.

```sql
CREATE TABLE ORDER1 (ORDER_ID INT NOT NULL PRIMARY KEY,
ORDERPRIORITY VARCHAR(15),
PRODUCT_ID INT,
SHOP_ID INT,
ORDERDATE TIMESTAMP,
QUANTITY INT,
REVENUE INT,
PROFIT INT) ORGANIZE BY COLUMN;
CREATE TABLE ORDER2 (ORDER_ID INT NOT NULL PRIMARY KEY,
INVOICE BLOB(1M)) ORGANIZE BY ROW;
```

Then create the following view.

```sql
CREATE VIEW ORDER AS
(SELECT ORDER1.*, ORDER2.INVOICE FROM ORDER1, ORDER2 WHERE
ORDER1.ORDER_ID=ORDER2.ORDER_ID);
```

Leverage DB2 BLU Acceleration technology for tables involving unsupported data types
The following image shows this overview.

**Figure 1. Split table and combine them as view**

We have the following two queries for the above scenario:

- \( \text{SELECT PRODUCT_ID, SUM(PROFIT) FROM ORDER GROUP BY PRODUCT_ID;} \)
- \( \text{SELECT INVOICE FROM ORDER WHERE ORDER_ID=123;} \)

To look at the explanation of the first query, the optimized statement contains `ORDER2` table, which is not good.

**Original Statement:**
```
SELECT
  PRODUCT_ID,
  SUM(PROFIT)
FROM
ORDER
GROUP BY
  PRODUCT_ID
```

**Optimized Statement:**
```
SELECT
  Q4."PRODUCT_ID" AS "PRODUCT_ID",
  Q4.$C1
FROM
( SELECT
    Q3."PRODUCT_ID",
    SUM(Q3."PROFIT")
  FROM
    ORDER
GROUP BY
  Q3."PRODUCT_ID"
) Q3
  JOIN
  ( SELECT
    Q4."PRODUCT_ID",
    SUM(Q4."PROFIT")
  FROM
    ORDER
GROUP BY
  Q4."PRODUCT_ID"
) Q4
ON
  Q3."PRODUCT_ID" = Q4."PRODUCT_ID"
```
To look at the explanation of the second query, the optimized statement contains `ORDER1` table, which is also not good.

**Original Statement:**

```
SELECT INVOICE
FROM ORDER
WHERE ORDER_ID=123
```

**Optimized Statement:**

```
SELECT Q2."INVOICE" AS "INVOICE"
FROM MOHKAWA.ORDER1 AS Q1,
     MOHKAWA.ORDER2 AS Q2
WHERE (123 = Q2."ORDER_ID") AND
       (Q1."ORDER_ID" = 123)
```

This is because we haven't given enough information to the DB2 optimizer. In this scenario, `ORDER_ID` columns of both `ORDER1` and `ORDER2` table are identical. `ORDER_ID` of `ORDER1` can be considered as referring to `ORDER_ID` of `ORDER2` and vice versa. So create the following referential (foreign key) constraints.

```
ALTER TABLE ORDER1 ADD CONSTRAINT FK1 FOREIGN KEY (ORDER_ID)
    REFERENCES ORDER2(ORDER_ID) NOT ENFORCED ENABLE QUERY OPTIMIZATION;

ALTER TABLE ORDER2 ADD CONSTRAINT FK1 FOREIGN KEY (ORDER_ID)
    REFERENCES ORDER1(ORDER_ID) NOT ENFORCED ENABLE QUERY OPTIMIZATION;
```

Figure 2 shows this relationship.

**Figure 2. Foreign key constrains in both directions**

Looking at the explanation of the first query, the optimized statement doesn't contain the `ORDER2` table, which looks good.
Leverage DB2 BLU Acceleration technology for tables involving unsupported data types

Original Statement:
------------------
SELECT
  PRODUCT_ID,
  SUM(PROFIT)
FROM
ORDER
GROUP BY
  PRODUCT_ID

Optimized Statement:
-------------------
SELECT
  Q3."PRODUCT_ID" AS "PRODUCT_ID",
  Q3.$C1
FROM
(SELECT
  Q2."PRODUCT_ID",
  SUM(Q2."PROFIT")
FROM
  (SELECT
    Q1."PRODUCT_ID",
    Q1."PROFIT"
    FROM
     MOHKAWA.ORDER1 AS Q1
  ) AS Q2
GROUP BY
  Q2."PRODUCT_ID"
) AS Q3

Also, if you look at the explanation of the second query, the optimized statement doesn't contain the ORDER1 table, which also looks good.

Original Statement:
------------------
SELECT
  INVOICE
FROM
ORDER
WHERE
  ORDER_ID=123

Optimized Statement:
-------------------
SELECT
  Q1."INVOICE" AS "INVOICE"
FROM
  MOHKAWA.ORDER2 AS Q1
WHERE
  (123 = Q1."ORDER_ID")

Solution 2: Configure shadow table that contains only supported data types

This can be used in an environment involving OLTP, as well as OLAP, called the OLTAP environment. In this environment the INSERT/UPDATE/DELETE statements are issued quite often. In Solution 1: Split table and combine them as view, you cannot issue INSERT/UPDATE/DELETE statements against view. DB2 can have INSTEAD OF triggers for such purpose, but currently it's not supported against view involving column-organized table. In Solution 2: Configure
shadow table that contains only supported data types, all statements can be issued against the base table (the original table), which includes INSERT/UPDATE/DELETE statements.

In this solution, the original table is created as row-organized table as is.

```sql
CREATE TABLE ORDER
  (ORDER_ID INT NOT NULL PRIMARY KEY,
   ORDERPRIORITY VARCHAR(15),
   PRODUCT_ID INT,
   SHOP_ID INT,
   ORDERDATE TIMESTAMP,
   QUANTITY INT,
   REVENUE INT,
   PROFIT INT,
   INVOICE BLOB(1M)) ORGANIZED BY ROW;
```

Then create a shadow table for the above table, where the unsupported data type columns are removed.

```sql
CREATE TABLE ORDER_SHADOW
  AS
  (SELECT ORDER_ID, ORDERPRIORITY, PRODUCT_ID, SHOP_ID, ORDERDATE,
   QUANTITY, REVENUE, PROFIT FROM ORDER)
  DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY REPLICATION ORGANIZE BY COLUMN;

SET INTEGRITY FOR ORDER_SHADOW ALL IMMEDIATE UNCHECKED;

ALTER TABLE ORDER_SHADOW ADD CONSTRAINT ORDER_SHADOW_PK PRIMARY KEY (ORDER_ID);
```

As for replication, just replicate existing columns in the shadow table (ORDER_SHADOW) from the base table (ORDER).

The following image shows this overview.
To make shadow tables work, there are several settings required. Refer to "Improve performance of mixed OLTAP workloads with DB2 shadow tables" (see Resources). To check how it works, apply the following settings.

```sql
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION REPLICATION;
SET CURRENT REFRESH AGE ANY;
```

Then, let's suppose we have the following queries.

- SELECT PRODUCT_ID, SUM(PROFIT) FROM ORDER GROUP BY PRODUCT_ID;
- SELECT INVOICE FROM ORDER WHERE ORDER_ID=123;

Looking at the explanation of the first query, you see the shadow table (MQT) is used instead of the original table.

**Original Statement:**
```
SELECT
  PRODUCT_ID,
  SUM(PROFIT)
FROM
ORDER
GROUP BY
  PRODUCT_ID
```

**Optimized Statement:**
```
SELECT
  Q3."PRODUCT_ID" AS "PRODUCT_ID",
```
Q3.$C1
FROM
(SELECT
    Q2."PRODUCT_ID",
    SUM(Q2."PROFIT")
FROM
(SELECT
    Q1."PRODUCT_ID",
    Q1."PROFIT"
FROM
    MOHKAWA.ORDER_SHADOW AS Q1
) AS Q2
GROUP BY
    Q2."PRODUCT_ID"
) AS Q3

To look at the explanation of the second query, the original table is used, which is expected.

Original Statement:
-------------------
SELECT
    INVOICE
FROM
    ORDER
WHERE
    ORDER_ID=123

Optimized Statement:
-------------------
SELECT
    Q1."INVOICE" AS "INVOICE"
FROM
    MOHKAWA.ORDER AS Q1
WHERE
    (Q1."ORDER_ID" = 123)

Summary

This tutorial shows two ways to leverage BLU Acceleration for tables involving unsupported data types. One is to split the table between the table that contains all supported data types and the table that contains unsupported data types, creating a view to combine them. The other is to use a shadow table, where the shadow table only contains supported data types. Consider choosing either of these tables depending on your environment.

Acknowledgements

Thanks to Berni Schiefer, Sam Lightstone, and Abhinav Goyal in IBM Toronto Lab for their valuable advice.
Resources

- Learn more about BLU Acceleration.
- Read the developerWorks article "DB2 with BLU Acceleration: A rapid adoption guide."
- Read the developerWorks article "Improve performance of mixed OLTAP workloads with DB2 shadow tables."
- Read the IBM Redbooks® publication titled "Architecting and Deploying DB2 with BLU Acceleration."
- For product documentation, visit the IBM Knowledge Center for DB2 10.5 for Linux, UNIX, and Windows.
- Visit IBMBLUhub.com.
- Refer to additional best practices for BLU Acceleration: Optimizing analytic workloads using DB2 10.5 with BLU Acceleration.
- The Information Management area on developerWorks provides resources for architects, developers, and engineers.
- Stay current with developer technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Follow developerWorks on Twitter.
- Watch developerWorks demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Get involved in the developerWorks Community. Connect with other developerWorks users while you explore developer-driven blogs, forums, groups, and wikis.
About the authors

Masahiro Ohkawa

Masahiro Ohkawa is a senior technical consultant with the IBM Analytics division at Tokyo Software & Systems Development Lab. He works with various database related product developments. He currently works on IBM Analytics products such as DB2 and InfoSphere BigInsights, and helps business partners and clients implement database and Big Data solutions involving such products.

Abhishek Kumar Chourasia

Abhishek Kumar Chourasia is a software developer in the IBM DB2 Pattern Team. He has been with IBM since December 2009 and has experience across the breadth of IBM’s DB2 product, combined with functional verification testing.

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)