Performance Enhancements for DB2 UDB for z/OS Version 8

Part One

Tom Moulder

October 7, 2004

LIGHTYEAR

Topics for Today

- Materialized Query Tables
- Index Changes
- RDS Changes
- UDF Changes

The Origin of MQT

Distributed Systems Function

- Known as AST (Automatic Summary Tables)
- System defined and maintained

- Mainframe
  - Manually created Summary Tables
  - Require administration
  - Users must know table names and schema

Benefit to Users

- Frequent joins, summarizations and aggregations are pre-computed
- Improves performance for repetitive queries that use these pre-computed values
- Very beneficial for data warehousing

What is an MQT?

- Table containing materialized data derived from one or more source tables specified by a fullselect
  - Source can be
    - Base Tables
    - Views
    - Table Expressions
    - User Defined Table Expression

User Access

- Directly by coding the table name
- Indirectly through Query Re-Write
  - SQL references the same
    - Base Tables
    - All or some of the columns
    - With same Group By
    - As the MQT
  - Only Supported by Dynamic SQL

Two Types of MQT

- User maintained
  - Maintained with
    - Load Utility
    - DML (Insert, Update and Delete)
    - Triggers
- System Maintained
  - Maintained with
    - Refresh Table SQL statement
    - DML will fail (Insert, Update and Delete)

Creation

Table Name

Columns

```
Create Table My_MQT (Store_Number, Customer_Number, Store_City,
    Store_State, Month, Store_Sales) as
(Select Store_Number, Customer_Number, Store_City, Store_State, Month,
    Sales
From Sales_Detail
Group By Store_Number, Customer_Number, Store_City, Store_State, Month)
Data Initially Deferred
Refresh Deferred
Maintained by System
Enable Query Optimization
In MyDBMQT.MyTSMQT;
```

Full Select

Mandatory and Identifies this as an MQT

Altering

- Add/Drop/Alter Materialized Query
  - Add converts existing base table with aggregated data to an MQT
  - Drop actually converts the MQT into a base table
  - Alter will manipulate the creation parameters
    - Switch between system and user maintained
    - Enable/Disable Query re-write option

Query Re-Write

- Additional Restrictions for Query Re-write
  - Fullselect must be a subselect
  - Avoid use of “External Action” or “Non-Deterministic” attributes
  - Cannot use “RAND” built-in function

Fullselect Restrictions

Cannot use

- Rowids
- LOB data types
- Host Variables
- Remote Objects
- “Previous Value for” or “Next Value For” expressions
- Created Global Temporary Table (GTT)
- Declared Global Temporary Table
- Another MQT

Subselect Restrictions

- Subselect Restrictions for Query Re-write
  - Cannot contain
    - Additional subselects
    - NTE or Views requiring Materialization
    - INNER JOIN
    - Special Register
    - Scalar Fullselect
    - Row Expression Predicate
    - Sideways References
    - Tables with Multiple CCSIDs

Keeping Data Current

- System Maintained using “Refresh Table”
  - Four Step Process in one UOW
    - Deletes all rows
    - Executes the Fullselect
    - Inserts Calculated rows into MQT
    - Updates Catalog with cardinality and time of refresh
  - DB2 logs all deletes and inserts
  - Measure the overhead in test before production

Keeping Data Current

- User Maintained Options
  - Load Replace
  - Insert/Update
  - DPROP Apply
- Logging can be reduced
- Multiple Step Process
- User managed data integrity and locking

Maintenance of MQTs

- Attributes stored in SYSIBM.SYSVIEWS
- Cannot have
  - Primary Keys
  - Unique Indexes
  - Triggers
- Dropped automatically when base tables are dropped

Index Changes

- Truly Varying Length Index Keys
- Backward Index Scans

Varying Length Index Keys

- Previous versions always padded variable length index keys to the full length
- V8 makes this padding an option
  - "Padded" and "Not Padded" expression of Create Index DML
- Index used for all access path choices
- Storage requirements are reduced

Administration

- No automatic conversion
- Alter Index to convert to “Not Padded”
- Default controlled through DSNZPARM
  - PADIX on DSNTIPE panel at install

Overhead Issues

- Comparison Overhead at SQL execution
  - Key columns are padded to the longest length
  - Equal length values are compared
  - Simple Example
    - Key Value 1 “ABC” x”C1C2C3”
    - Key Value 2 “ABCD” X”C1C2C3C4”
    - Value 1 is padded with “ “ or X”40”
    - Not Equal, Value 1 orders before Value 2
  - Padding only occurs in buffers, not externalized

Conversion Issues

- Impact of Alter Conversion
  - Alter Index Index_Name Not Padded
    - Index placed in RBDP status
    - Optimizer will NOT use the index
    - Rebuild Index clears the status and use begins

Performance Impact

- Fewer Index pages and potentially fewer levels
- Therefore Fewer I/O for Index scans
- More CPU to pad for comparisons
- More CPU to find the start of the column
- YMMV, therefore test to determine the benefit

Index Scan Changes

- Backward Index Scan
  - Sort Avoidance Technique of the Optimizer
    - Scan Ascending index backward if SQL requests data in descending order and vice versa
  - Index Order and SQL Order MUST be exact opposites
    - Index -- Acct_Num, Order_Date, Order_Time ASC
    - SQL -- Acct_Num =:HV Order By Order_Date DESC, Order_Time DESC

RDS Changes

- Mismatched Data Types
- Unknown Join Sequences

Mismatched Data Types

- More common because of C/C++ and JAVA data types
- Performance Problem for previous versions
- V8 improves performance by making Stage 1 and possible indexable
  - "Column" operation "expression"
  - "Expression" operation "Column"
  - "Column" between "Expression" and "Expression"
  - "Column" in ("List")

The Fine Print

- Operation can be =, <, <=, >, >= or <>
- <> is made Stage 1, but can not be indexable
- "List" items must be
  - Constants
  - Host Variables
  - Special Registers
  - Session Variables
  - Parameter Markers
- List cannot be in a "When" clause of a trigger
- All items in list must be Stage 1 and Indexable

Numeric Mismatch

- An Example

```
EMP( NAME CHAR (20),
SALARY DECIMAL (12,2),
DEPTID CHAR (3) );

SELECT * FROM EMP
WHERE SALARY > :HV_FLOAT ;
```

  - Stage 1 and indexable on column Salary in V8
  - No Change to source code or schema

String Mismatch

- An Example

```
EMP( NAME CHAR (20),
SALARY DECIMAL (12,2),
DEPTID CHAR (3) );

SELECT * FROM EMP
WHERE NAME = :HV_LENGTH_15 ;
```

  - Stage 1 and Indexable on column NAME
  - No Change to source code or schema

Transitive Closure

- Works also

```
SELECT DEPT.NAME, EMP.NAME
FROM EMP, DEPT
WHERE EMP.DEPTID = ? AND
EMP.DEPTID = DEPT.ID
AND DEPT.ID = ? ;
```

CHAR(3)

CHAR(4)

  - Stage 1 Predicate and Indexable on DEPT.ID

Unknown Join Sequences

- Both operands are columns from different tables
- Stage 1 Decision based on
  - Join Sequence
    - Table_One.Column_One = Table_Two.Column_Two + 1
  - Which column is really the column and which is the expression

Column Expression

- Example

```
SELECT E1.*
FROM EMP E1, EMP E2, DEPT
WHERE E1.DEPTID = DEPT.ID AND
DEPT.MGR = E2.NAME AND
E1.SALARY > E2.SALARY * 1.10
```

  - If E2 is the inner table
    - Stage 2 and table space scan
  - If E1 is the inner table
    - Stage 1 and indexable on E1.Salary

Between Predicates

- Example

```
SELECT EMP.*
FROM EMP, SALRANGE S
WHERE EMP.LEVEL = S.LEVEL AND
EMP.SALARY BETWEEN S.LOW AND S.MID;
```

  - If SALRANGE is the inner table
    - Stage 2 and table space scan
  - If EMP is the inner table
    - Stage 1 and indexable on EMP.Salary

Table UDF

- Example

```
BOOK( ID INTEGER,
TITLE CHAR(60),
AUTHOR CHAR(30) );
SELECT BOOK.ID, BOOK.TITLE
FROM BOOK, TABLE (tf_contain('Computer')) tf(id)
WHERE BOOK.ID = tf.id ;
```

  - If tf_contain is the inner table
    - Stage 2 and table space scan
  - If Book is the inner table
    - Stage 1 and indexable on Book.ID

Multiple CCSIDs

- Example

```
SELECT ...
FROM SUPPLIER, PRODUCT
WHERE SUPPLIER.ID = PRODUCT.SUPPLIERID ;
```

  - Supplier is Unicode and Product is EBCDIC
  - If Product is inner table
    - Stage 1 but not indexable
  - If Supplier is the inner table
    - Stage 1 and indexable on Supplier.ID

UDF Cardinality

- Previous Versions
  - Cardinality previously defined at create time
  - Only this one value for all uses of the UDF
- V8
  - Additional capability within each SQL statement
  - Cardinality and Cardinality Multiplier expressions
  - Non-Standard SQL syntax

UDF Cardinality

- Examples

  Select Columns from Table (UDF(1) Cardinality 1000 as XYZ;

  Select Columns from Table (UDF(1) Cardinality Multiplier 10 as XYZ;

  – Cardinality replaces the value in the catalog for this statement only

  – Cardinality Multiplier option multiplies the catalog value with the literal for this statement only

UDF Cardinality

- Each UDF reference has its own cardinality

```
SELECT *
FROM BOOKS B,
TABLE(CONTAINS(1,'cs') CARDINALITY 15000) AS
  X1(ID),
TABLE(CONTAINS(2,'database') CARDINALITY 2000)
  AS X2(ID),
TABLE(CONTAINS(3,'Clark') CARDINALITY 30) AS
  X3(ID)
WHERE B.ID = X1.ID AND B.ID = X2.ID AND B.ID =
  X3.ID;
```

UDF Cardinality

- Why and for What purpose
  - More information to the optimizer
  - Better access path selection
  - Improved performance
  - Reduced CPU Memory and I/O resources
  - Only as good as the accuracy of the cardinality

Table UDF Block Fetch

- Problem in previous versions
  - One UDF Fetch for each row returned from UDF
  - UDF runs in a separate WLM managed ASID
  - CPU overhead for context switch is non trivial
- V8
  - First UDF fetch materializes a result data set
  - Subsequent retrieval is by fetch no context switch is required

Table UDF Block Fetch

- Saves
  - CPU time
    - (Result_Set_Rows -1) * context switch overhead
- Costs
  - I/O to materialize the result set as a work file
- Good for CPU constrained systems
- No Coding Changes Required

Table UDF Block Fetch

- How do I know if DB2 chose to do this?
  - Plan_Table
    - Table UDF Access Type of “RW”
    - Referred to as “Materialized Fetch”
- Don’t confuse with “Block Fetch” for Distributed Units of Work

Summary

- Wow! A Lot of Information to Digest
- Points to Remember
  - NCCR – No Coding Changes Required
  - Significant Performance Improvements
  - Many changes are easy to implement in V8
- Questions Anyone

Next Steps with DB2 V8 and Lightyear

- Our series of detailed presentations on various DB2 V8 topics:

Pre-requisites	V7 to V8 Migration
Unicode	Utilities
Access Path Review	Highlights of New Functionality
Schema Evolution	In Depth Review of the “Top” 5 New Functions
Catalog Changes	Enhancements to SQL

Next Steps with DB2 V8 and Lightyear

- FREE presentations available NOW.
- These presentations will be given to just one customer at a time and will be tailored to that customer.
- Presentations will vary in length, dependent upon topic and tailoring.
- A 15 minute “prep” call will be required.

Next Steps with DB2 V8 and Lightyear

- Access Path Review is a performance factor
- We offer a free on-site two-day review
  - Your systems
  - Your SQL
  - Predictive Report of potential problem SQL
- Watch our website for details on both of these exciting and FREE offers and register your interest.

Services available from:

L I G H T Y E A R

Palo Alto – Fort Worth - Calgary - Laguna Beach

Scottsdale – Chicago – St. Louis – Boston

www.lightyr.com

- Database migration to *DB2*
- *VS/2* and *DL/2* software sales and related services *(sole North American distributor)*
- *CICS*, *DB2*, *IMS*, & *z/OS* software and tools, sales and upgrades
- Customized on-site technical seminars & education classes
- *WebSphere MQ* and application integration services
- *CICS & IMS* Web enabling design and implementation
- Database and online system performance analysis and tuning

Questions …