Using DB2 for LUW SQL PL to make it faster, better, cheaper and simpler

Frank McEwen
Independent Consultant

Session Code: D5
15 October 2013 Time: 11:00 - 12:00 | Platform: LUW

Agenda

- DB2 SQL PL features
  - Introduction
  - Native DB2 LUW SQL PL
  - Anchored Data Types
  - Arrays
- Techniques
  - Synchronizing data
    - Performance, MERGE, federation and concurrency considerations
    - Dynamic generation and execution of SQL and SQL PL code
    - CURSOR loop with MERGE to synchronize data applying deltas only
  - Checking partition ranges
    - Stored proc returning rows, Session tables
- z/OS differences
- Please feel free to interrupt with questions

Introduction

- SQL PL features extended over the years
- Rich programming language
- Alternative to shell scripts, javascript, java, REXX, perl, …
- Consider what you can do with a stored proc before scripting
- Deploy SQL PL across different operating systems
- Recent features of SQL PL

DB2 SQL PL features: Native DB2 LUW SQL PL

- Can be unclear as to what is SQL PL and what is PL/SQL
- Only SQL PL syntax in this presentation
- No PL/SQL
- Presentation won't be covering DB2 for z/OS SQL PL syntax
  - Some code won't work on DB2 for z/OS
  - Highlight what is not available on DB2 for z/OS
  - Touch on DB2 for z/OS with federation

SQL PL features: Anchored Data Types

- Declare SQL PL working storage variable or input variables as having the data type of a column
  - Can also anchor to a row
  - Row data type
- Declare working storage variable:

```
declare WS_ISOLATION ANCHOR SYSCAT.PACKAGES.ISOLATION;
```

- Avoids hard coding types
- Reduces potential for errors wrong definition
- Reduces maintenance effort if a column type is altered
  - No rework of SQL PL code necessary
- Future proof your SQL PL code

SQL PL features: Array Processing (1)

- CREATE ARRAY TYPE:

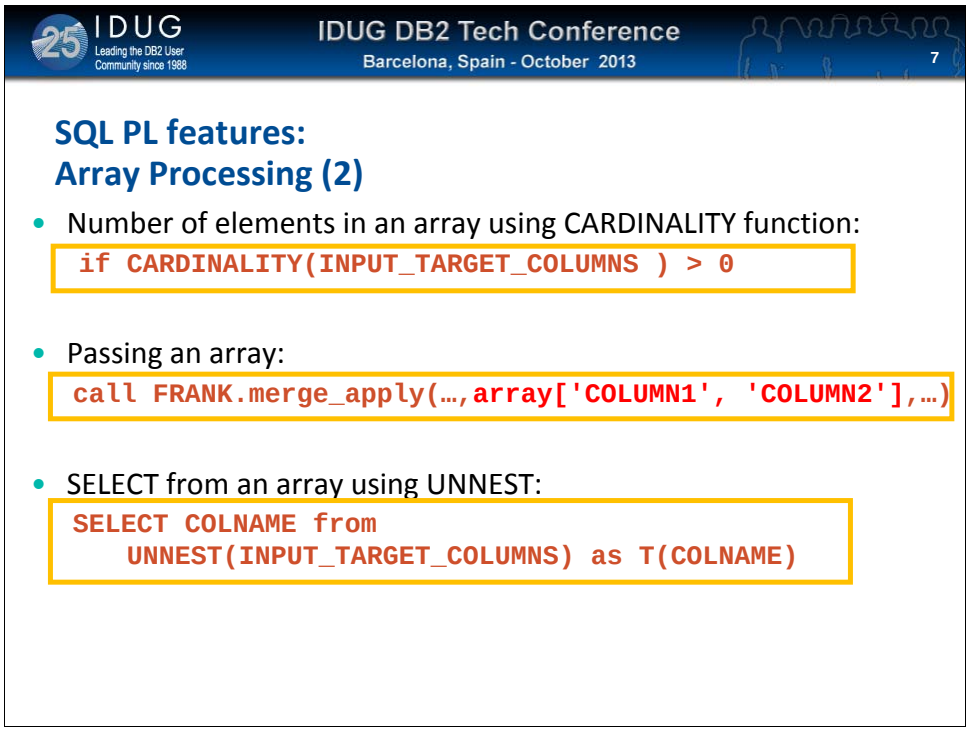
```
CREATE TYPE FRANK.ColumnNameArray
as ANCHOR SYSCAT.COLUMNS.COLNAME ARRAY[]
```

  - Unfortunately not possible to declare a type local to a SQL PL Stored procedure
    - It would be nice to be able to have inline or local type declarations
    - This is common in other languages
- Declaring an Array:
  - Input array

```
(IN INPUT_TARGET_COLUMNS FRANK.ColumnNameArray)
```

  - Working Storage Variable

```
declare WS_KEY_COLUMNS      FRANK.ColumnNameArray;
```



LISTAGG aggregate function could be used to produce a simple list of columns as an alternative to using arrays. This could be used on DB2 for z/OS before V11 when arrays are introduced.

SQL PL features: Array Processing (3)

Populate an array from a SELECT

```
FOR PK_CURSOR AS PK_CURSOR CURSOR FOR
 SELECT TGTIXCOL.COLNAME
   FROM SYSCAT.INDEXES TGTIX, SYSCAT.INDEXCOLUSE TGTIXCOL
  WHERE TGTIX.TABSCHEMA        = INPUT_TARGET_SCHEMA
    AND TGTIX.TABNAME          = INPUT_TARGET_TABLE
    AND TGTIX.UNIQUERULE       = 'P'
    AND TGTIX.INDSCHEMA        = TGTIXCOL.INDSCHEMA
    AND TGTIX.INDNAME          = TGTIXCOL.INDNAME
  ORDER BY TGTIXCOL.COLSEQ
do  set WS_COUNTER = WS_COUNTER + 1;
    set WS_KEY_COLUMNS[WS_COUNTER] = PK_CURSOR.COLNAME;
end for;
```

Here we are populating an ARRAY for the columns of the Primary Key. These columns are important for building our MERGE statement as we will see in later foils.

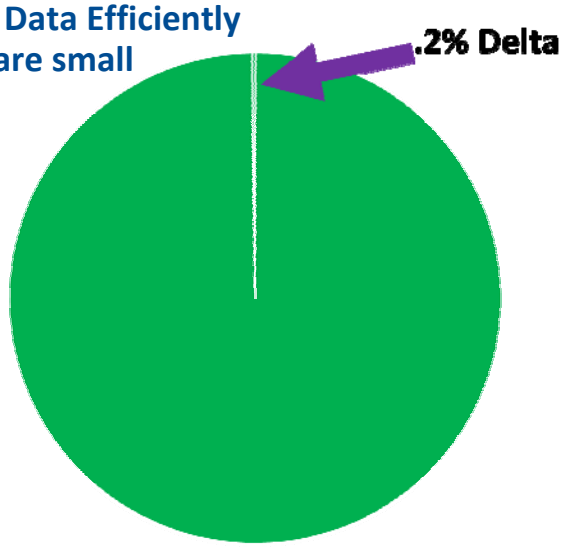
Note that you could potentially use the aggregation function ARRAY_AGG instead of the code above. I tried this but could not get it to work.

SQL PL features:
Dynamic Generation and Execution of SQL PL

- Build block of SQL PL code dynamically
- Just like dynamic SQL
- Delimit block with BEGIN and END
- Recommendation:
  - Insert generated SQL PL code into CLOB column
  - Useful for debugging
  - Put some line feeds in to make it readable!

Techniques 1: Synchronizing Data Efficiently

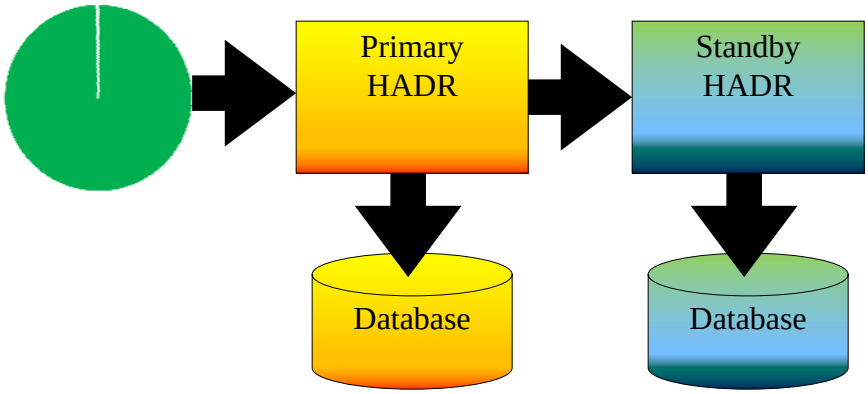
Synchronising Data Efficiently If differences are small



To synchronise Source and Target apply *only* the Delta to the Target

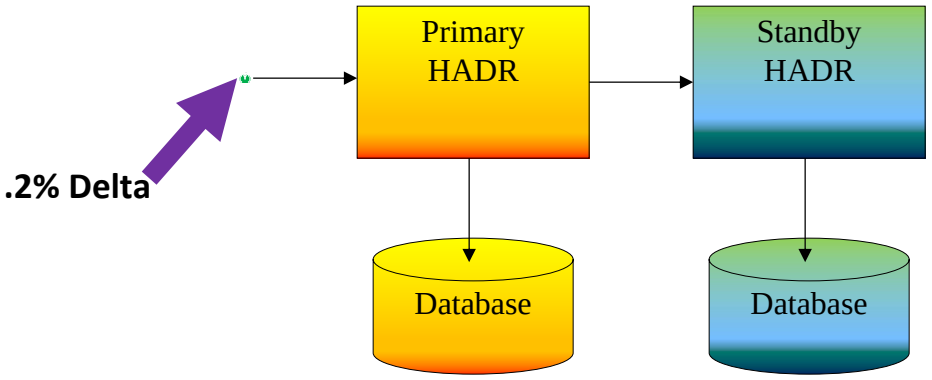
Synchronising Data Efficiently If differences are small

- Apply all the data



Synchronising Data Efficiently If differences are small

- Apply just the delta



Synchronising Data Efficiently If differences are small

- Compare source and target
  - Only INSERT/UPDATE/DELETE the delta
- Minimize processing time and CPU
  - Minimize UPDATE/INSERT/DELETE activity
  - Minimize logging
    - Avoid HADR lag
  - Use MERGE to minimize cost
    - Better than UPDATE/INSERT

Synchronising Data Efficiently If differences are small

- Maximize concurrency
  - Minimize locking
    - Avoid lock escalation
  - No large UOWs
  - Regular commits
  - Replacing entire table is a huge overhead
  - RI
    - Avoid check conditions from replacing parent tables

Synchronising Data Efficiently If differences are small

- If only a small delta
  - Data given as complete replacement
    - External data sources
      - Spreadsheets within the business
      - Other companies/regulators
  - Application Programmers are not able to provide the delta
  - Broken replication and need to resynchronize

Synchronising Data Efficiently
If differences are small

- Using DB2 Utilities
  - UNLOAD or EXPORT table and use diff, sort et cetera to generate the delta then use IMPORT
    - Inefficient and not online
  - IMPORT INSERT_UPDATE
    - Very easy
    - This is inefficient as this will update and log every row – no delta
    - Maybe have to work out how to DELETE rows no longer required
    - Input from CURSOR not supported
  - Use asntdiff then asntrep
    - http://www-01.ibm.com/support/docview.wss?uid=swg21229502
    - Involves multiple steps
    - Inefficient and not online
  - EXPORT to a pipe and INGEST

Synchronising Data Efficiently If differences are small

- EXPORT to a pipe and INGEST
  - INGEST from CURSOR is not supported
    - LOAD supports CURSOR as input
    - Use named pipe
  - Good option for Unix or Linux
    - Not so easy for Windows as pipe not supported at command line
      - Requires program code to create the pipe in Windows

```
mkfifo export_pipe
db2 -vtf export_to_pipe_ingest_from_pipe.sql
rm export_pipe
```

```
EXPORT TO export_pipe OF DEL
  SELECT * FROM FRANK.INVENTORY;
```

Synchronising Data Efficiently If differences are small

- EXPORT to a pipe and INGEST

```
INGEST FROM PIPE export_pipe FORMAT DELIMITED
   ($PID      CHAR
   ,$QUANTITY INTEGER EXTERNAL
   ,$LOCATION CHAR) RESTART OFF
MERGE INTO FRANK.INVENTORY_TWO ON $PID = PID
  WHEN NOT MATCHED THEN INSERT (PID ,QUANTITY ,LOCATION)
                        VALUES ($PID , $QUANTITY , $LOCATION)
  WHEN MATCHED AND (   $QUANTITY <> QUANTITY
                    OR $LOCATION <> LOCATION)
         THEN UPDATE SET QUANTITY = $QUANTITY
                        ,LOCATION = $LOCATION
  ELSE IGNORE
WITH UR;
```

Synchronising Data Efficiently If differences are small

- EXPORT to a pipe and INGEST
  - Add extra predicates on MERGE UPDATE clause
    - Only update matched rows that are different
  - Reasonable efficient
    - More efficient than IMPORT INSERT_UPDATE
  - Commit by count or elapsed time
    - Count restricted to 1000 multiple
    - Default elapsed time between commits is 1 second
  - Not online as using EXPORT
  - NICKNAME not supported for MERGE
  - No direct method of deleting rows in target table not in source
  - Rich set of controls available with INGEST

Synchronising Data Efficiently
If differences are small

- Could write a program
  - Simple
    - Read source file or table
    - compare with table data
    - MERGE/DELETE
  - Requires programming effort
    - Outsourced environment
    - Getting a program coded and tested is a large administrative task
    - Expensive
    - Error prone
    - Development and coding time
      - Suddenly, not so agile
      - Only able to use Java
    - Programmers not up to it
      - Even when provided MERGE code

In fact our COBOL programmers could do this, although they did not use MERGE.

Our Java programmers could not work it out.

Synchronising Data Efficiently
If differences are small

- SQL PL stored proc to generate the MERGE Statement
  - Parameters :- Source table and Target table
  - Use DB2 catalog to generate MERGE code to CLOB
  - 2 hour actual versus 2 week estimate from developers
    - $$$
- But…
  - Programmers had difficulty understanding the MERGE statement
  - Could not work out how to code the Java
- Why not…
  - Generate the whole program
  - SQL PL Stored procedure

Synchronising Data Efficiently If differences are small

- Finally…
- Why not generate *and* execute the SQL PL code
  - Write block of SQL PL code BEGIN .. END to a CLOB
  - Execute CLOB dynamically
- Generic synchronization
  - No further need of programmers!
  - 2 weeks estimated effort become 10 minutes
  - No design meetings
  - No Specification
  - No Red tape
  - No bespoke code
  - Test once, use multiple times

Synchronising Data Efficiently Call 1

- Simple Call
  - no columns or keys specified

```
call FRANK.merge_apply
('MERGE'
,'FRANK','INVENTORY'
,'FRANK','INVENTORY_TWO'
,2,'INSENSITIVE','UR',NULL,
NULL,NULL,NULL);
```

Synchronising Data Efficiently Call 2

- Specify columns and keys in call
  - NICKNAMES or Views where DB2 cannot determine Primary Key
  - Don't want all columns synchronized
  - Source table column names different from target table

```
call FRANK.merge_apply
('MERGE'
,'FRANK','INVENTORY'
,'FRANK','INVENTORY_TWO'
,2,'INSENSITIVE','UR',NULL
,ARRAY[PID]
,ARRAY[PID,QUANTITY,LOCATION]
,ARRAY[PID,QUANTITY,LOCATION]);
```

Synchronising Data Efficiently
Synchronization Options achieved with SQL PL

- MERGE
  - UPDATES or INSERTS, no DELETEs
- REPLACE
  - not recommended with RI
  - truncate and replace TARGET with SOURCE data via MERGE INSERT
- SYNCHRONISE
  - UPDATES or INSERTS then DELETE any rows in target but not in source
- UPDATE
  - MERGE UPDATES only, skip INSERTS
- INSERT
  - MERGE INSERTS only, skip UPDATES
- DELETE
    - Skip INSERTS, skip UPDATES, DELETE any rows in target but not in source

Synchronising Data Efficiently Parameters

- SOURCE_SCHEMA, SOURCE_TABLE
- TARGET_SCHEMA,TARGET_TABLE
- COMMIT_RATE
- CURSOR_SENSITIVITY
  - For generated CURSOR statement
  - Probably not required
- ISOLATION
  - Cursor ISOLATION. E.G. UR, CS
- GENERATE_ONLY *optional*
  - Y/N Generate SQL PL code but don't execute it

Synchronising Data Efficiently Parameters

- TARGET_MATCH_KEY_COLUMNS *optional*
  - By default use the DB2 catalog to get columns
  - Only required if
    - MERGE on a different key from the primary key
    - TARGET does not have a primary key
      - performance implications for the MERGE
    - TARGET is NICKNAME/VIEW and column names are different
  - Specify TARGET columns *only* as
    - SOURCE columns can be derived from TARGET + Column Arrays or DB2 catalog
- SOURCE_COLUMNS, TARGET_COLUMNS *optional*
  - SOURCE and TARGET columns have different names or to exclude columns

Synchronising Data Efficiently
Notes on MERGE usage 1

- MERGE on columns not primary key or unique leads to poor performance
  - For good performance, MERGE requires unique index on ON columns
  - Tablespace scan and sort to remove duplicates
  - If no UNIQUE index then minimize overhead by GROUP BY on MERGE "ON" columns
    - DB2 can then determine that the result sets are unique
    - Cheaper sort of MERGE "ON" columns rather than whole row
  - Best with Unique index

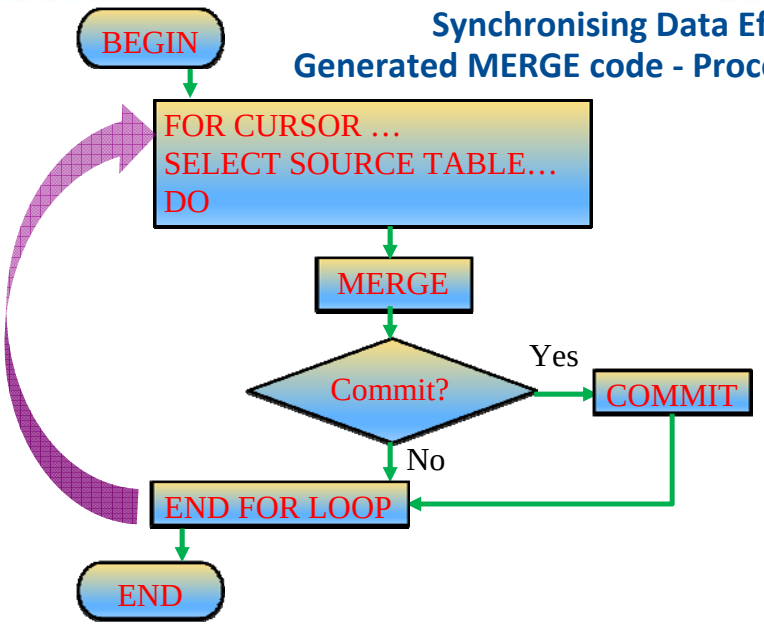
Synchronising Data Efficiently
Notes on MERGE usage 2

- Make sure RUNSTATS are up to date
- Check access paths
  - Look at index usage, scans, sorts
- Federation
  - Make sure that indexes on NICKNAMEs are known to DB2
    - SPECIFICATION ONLY indexes normally created automatically with CREATE NICKNAME
    - Sometimes does not occur
    - E.G. NICKNAME is on a VIEW
  - Make sure NNSTAT is up to date
  - Check what SQL is pushed down
  - Check what data is being shipped

Synchronising Data Efficiently
Generated MERGE code

- Cursor on source table
  - FOR loop on a CURSOR
  - Minimizes declarations of variables
- For each source table row
  - MERGE cursor row with the target ON the MERGE columns
    - Use VALUES statement to construct a row from the cursor
    - Order of MATCHED and NOT MATCHED is important to consider
  - WHEN MATCHED then UPDATE
    - if there is difference between the source and target
    - Minimise updates to those rows that are different
    - Minimise logging
  - WHEN NOT MATCHED then INSERT
- Keep a counter and COMMIT according to the commit rate parameter

Synchronising Data Efficiently
Generated MERGE code - Process Flow



Synchronising Data Efficiently
Generated MERGE loop code 1

```
BEGIN
 declare WS_SOURCE_CURSOR_COUNTER INTEGER;
 set WS_SOURCE_CURSOR_COUNTER = 0;
 FOR SOURCE_CURSOR AS SOURCE_CURSOR INSENSITIVE
  CURSOR WITH HOLD FOR
   SELECT PID,QUANTITY,LOCATION FROM FRANK.INVENTORY
    FOR READ ONLY  OPTIMIZE FOR 2 ROWS  WITH UR
 DO
```

- Full generated MERGE loop code in notes to this slide
  - Email me for the full SQL PL program *eponymous@frankmcewen.com*

```
BEGIN
declare WS_SOURCE_CURSOR_COUNTER INTEGER;
declare WS_TARGET_CURSOR_COUNTER INTEGER;
declare WS_DELETED_ROWS INTEGER;
set WS_SOURCE_CURSOR_COUNTER = 0;
FOR SOURCE_CURSOR AS SOURCE_CURSOR INSENSITIVE CURSOR WITH HOLD FOR
SELECT
PID
,QUANTITY
,LOCATION
 FROM FRANK.INVENTORY
 FOR READ ONLY
 OPTIMIZE FOR 2 ROWS
 WITH UR
 DO
 MERGE INTO FRANK.INVENTORY_TWO AS TARGET
 USING (
VALUES(
 SOURCE_CURSOR.PID
, SOURCE_CURSOR.QUANTITY
, SOURCE_CURSOR.LOCATION
)
) as SOURCE(
PID
,QUANTITY
,LOCATION
)
 ON
 SOURCE.PID = TARGET.PID
 WHEN NOT MATCHED THEN
INSERT (
PID
,QUANTITY
,LOCATION
 )
VALUES (
 SOURCE.PID
, SOURCE.QUANTITY
, SOURCE.LOCATION
 )
WHEN MATCHED  AND (
 SOURCE.PID <> TARGET.PID
 OR  SOURCE.QUANTITY <> TARGET.QUANTITY
 OR  SOURCE.LOCATION <> TARGET.LOCATION
 )
THEN
UPDATE SET
PID = SOURCE.PID
,QUANTITY = SOURCE.QUANTITY
,LOCATION = SOURCE.LOCATION
 ELSE IGNORE;
 set WS_SOURCE_CURSOR_COUNTER = WS_SOURCE_CURSOR_COUNTER + 1;
 IF MOD(WS_SOURCE_CURSOR_COUNTER,2)  = 0 THEN
 COMMIT;
 END IF;
 END FOR;
END
```

Synchronising Data Efficiently
Generated MERGE loop code 2

```
MERGE INTO FRANK.INVENTORY_TWO AS TARGET
USING (
      VALUES(SOURCE_CURSOR.PID,SOURCE_CURSOR.QUANTITY
            ,SOURCE_CURSOR.LOCATION)
      ) as SOURCE(PID,QUANTITY,LOCATION)
   ON  SOURCE.PID = TARGET.PID
WHEN NOT MATCHED THEN INSERT (PID,QUANTITY,LOCATION)
      VALUES (SOURCE.PID,SOURCE.QUANTITY,SOURCE.LOCATION)
WHEN MATCHED  AND (SOURCE.PID <> TARGET.PID
                  OR SOURCE.QUANTITY <> TARGET.QUANTITY
                  OR SOURCE.LOCATION <> TARGET.LOCATION)
THEN UPDATE SET PID = SOURCE.PID,QUANTITY =
SOURCE.QUANTITY,LOCATION = SOURCE.LOCATION
ELSE IGNORE;
```

Synchronising Data Efficiently
Generated MERGE loop code 3

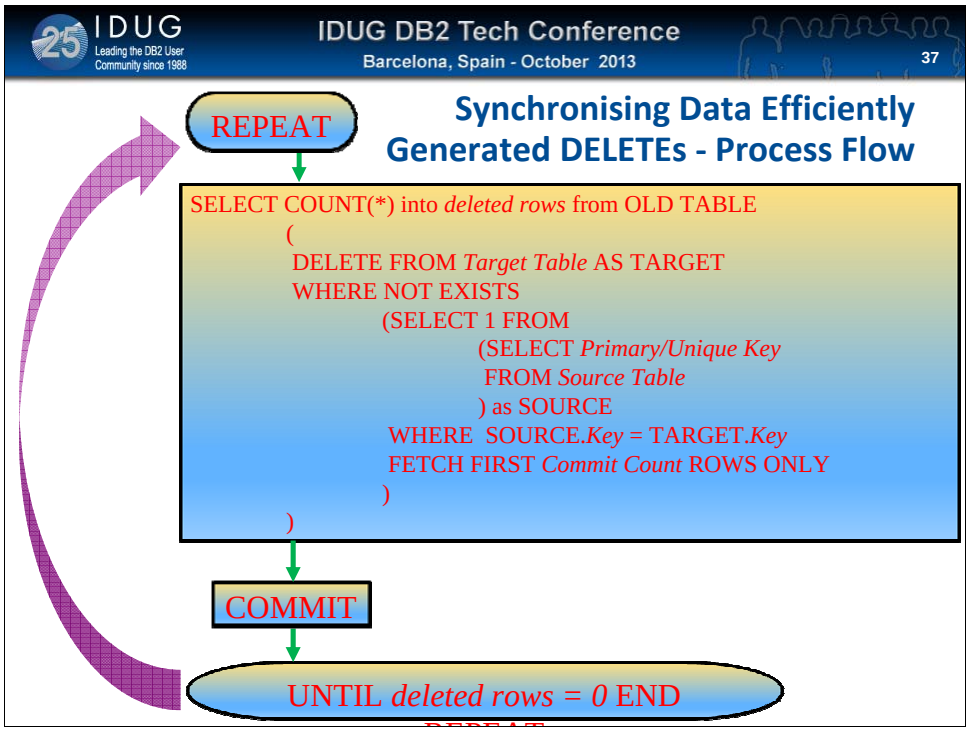
```
  set WS_SOURCE_CURSOR_COUNTER=
      WS_SOURCE_CURSOR_COUNTER+1;
  IF MOD(WS_SOURCE_CURSOR_COUNTER,2)  = 0 THEN
   COMMIT;
  END IF;
 END FOR
END
```

Commit logic

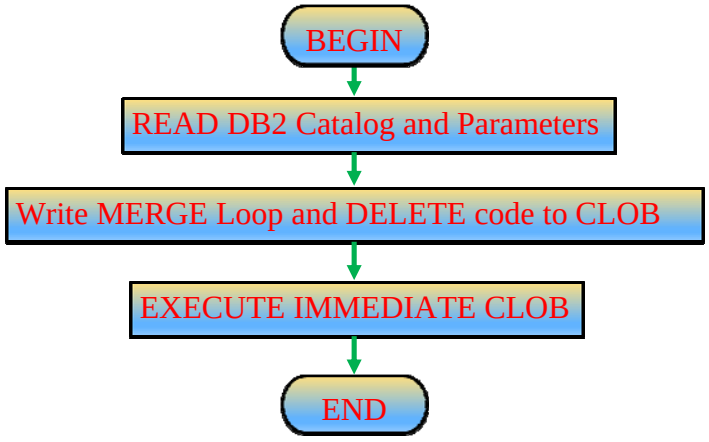
Synchronising Data Efficiently Generated DELETES

- DELETE or SYNCHRONISE option DELETEs rows in target table not in source table
- DELETE is not logically possible in CURSOR/MERGE logic
  - Available with INGEST
- COMMIT according to parameter
- Use REPEAT UNTIL no rows deleted
  - DELETE from target table where not exists in source table
  - FETCH FIRST *commit-rate* ROWS ONLY
  - SELECT COUNT(*) FROM *DELETE* to determine number of rows deleted
  - UNTIL no further rows deleted
- Full Generated DELETE Code is in notes to this slide

```
set WS_DELETED_ROWS = 0;
REPEAT
                SELECT COUNT(*) into WS_DELETED_ROWS
                from OLD TABLE
                                (
                                                DELETE
                                                FROM FRANK.INVENTORY_TWO as TARGET
                                                WHERE NOT EXISTS
                                                                (SELECT 1
                                                                                FROM
                                                                                (

                SELECT

                PID AS PID

                ,QUANTITY AS QUANTITY

                ,LOCATION AS LOCATION

                FROM FRANK.INVENTORY
                                                                                ) as
SOURCE
                                                                 WHERE

SOURCE.PID = TARGET.PID
                                                                FETCH FIRST 2 ROWS ONLY
                                                                )
                                )
                ;
                COMMIT ;
UNTIL WS_DELETED_ROWS = 0 END REPEAT ;
```


Synchronising Data Efficiently
Generated DELETEs - Process Flow
REPEAT
SELECT COUNT(*) into deleted rows from OLD TABLE
(
DELETE FROM Target Table AS TARGET
WHERE NOT EXISTS
(SELECT 1 FROM
(SELECT Primary/Unique Key
FROM Source Table
) as SOURCE
WHERE SOURCE.Key = TARGET.Key
FETCH FIRST Commit Count ROWS ONLY
)
)
COMMIT
UNTIL deleted rows = 0 END

Synchronising Data Efficiently
Generating Dynamic code - Process Flow



Synchronising Data Efficiently
Applications for using SQL PL to Synchronize

- SQL or Q Data Replication
  - Alternative to using asntdiff then asntrep
- External data sources
  - Federate or view
    - Spreadsheets
    - Flat files
    - XML files
    - Other databases
- User maintained MQTs
- HADR
  - have to RESTORE from the primary
    - Can't use anything else if logs can't be applied to the standby
  - SQL PL Proc minimizes logging impact synchronizing table on primary

Synchronising Data Efficiently Limitations and Performance

- Does not like XML columns
  - SQL0789N The data type for parameter or SQL variable *xxxx* is not supported in the routine, compound SQL statement, or parameter list of a cursor value constructor. LINE NUMBER=6. SQLSTATE=429BB
  - I need some time to address this in my code
    - Perhaps use UDF instead of "=" predicate
- If the TARGET is a NICKNAME then MERGE is not allowed
  - Moving target - was allowed in prior releases
  - Instead generate code to emulate MERGE
  - SELECT/UPDATE/INSERT
  - Performs poorly so not recommended for large volumes but OK for small volumes

Synchronising Data Efficiently
Comparison of Methods

Method	Delta only	SYNCHRONISE / DELETE	Concurrency	Externalised	RI Check Required	SQL or other coding required	Performance
asntdiff asntrep	Yes	Yes	None	Yes	Yes	No	Poor
EXPORT to pipe then INGEST	Yes	EXPORT then INGEST with DELETE	Good	Yes	No	Yes – SELECT SQL	Excellent
EXPORT to pipe then IMPORT INSERT_UPDAT	No	No	Good	Yes	No	Yes – SELECT SQL	Poor
EXPORT to pipe then LOAD	No	No	Poor	Yes	Yes	Yes – SELECT SQL	Good
LOAD from CURSOR	Possible in CURSOR SQL	No	None	No	Yes	Yes – SELECT SQL	Excellent
Bespoke Program	Yes	Yes	Depends on Code	No	No	Yes	Depends on Code
Stored Proc	Yes	Yes	Excellent	No	No	No	Excellent

Synchronising Data Efficiently
Deployment: Reference data refresh

- Large reference tables sourced from DB2 for z/OS
- Monthly update
- 1.3 Million rows
- Typically 100 to 3000 new or changed rows in one table
  - Between 0.012% and 0.231% inserted or updated
- Up to 3 LUW copies of z/OS table on DB2 LUW
  - Required for latency
  - Required for 24X7 as z/OS application was not 24X7
  - Table structures differed on DB2 for LUW

Synchronising Data Efficiently
Deployment: Reference data refresh – Old process

- Old process was:
  - All rows read from mainframe DB2 for z/OS tables into central DB2 for LUW database
  - Stored procs then ran to replace all rows in other DB2 LUW databases via NICKNAME tables
  - Multiple stored procs customized per database per table
    - Different logic and layouts
- Logging overhead
- Outage of up to 2 hours on 24X7 web system
  - Also delayed code deployments

Synchronising Data Efficiently
Deployment: Reference data refresh – New process

- NICKNAME and view of mainframe table
- MERGE_APPLY stored proc applies delta to centralized Reference data DB2 for LUW database tables
- Views on centralized Reference data tables
  - Encapsulate logic and layout customization changes required for application databases
- Application databases access reference data via NICKNAMEs on Views
- Centralised single copy of reference data

Synchronising Data Efficiently
Deployment: Reference data refresh – New process

- 24X7 process
  - Commits
  - No outage
  - No performance degradation
  - No HADR impact
  - Can be done during day
  - Reduced delay at deployments

Synchronising Data Efficiently
Deployment: Reference data refresh – New process

Before

Table Unavailable

After

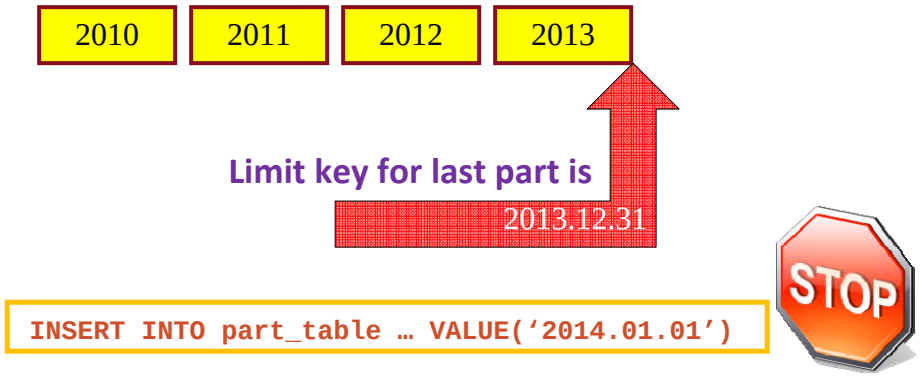
10 Minutes

No Outage

Techniques 2: Looking for Partitioned Table Limits

Looking for Partitioned Table Limits Introduction

Partitioned table




```
INSERT INTO part_table … VALUE('2014.01.01')
```

Looking for Partitioned Table Limits Introduction

- Are any Partitioned tables are going to exhaust partition ranges?
- 200-300 databases and no documentation then you need queries to check
  - Query 1: DB2 catalog to find the limit expression
    - static SQL query
  - Query 2: Based on query 1, query the partition table to find what the highest value is
    - SQL is dynamic as depends on the results of query 1
- Set in place an alert
  - Send a email

Looking for Partitioned Table Limits Query 1

```
select dpe.tabschema,dpe.tabname
      ,dpe.datapartitionkeyseq
      ,dpe.datapartitionexpression
      ,dpe.nullsfirst,dp.highvalue
  from syscat.datapartitionexpression dpe
      ,syscat.datapartitions          dp
  where dpe.tabschema not like 'SYS%'
    and dpe.tabschema = dp.tabschema
    and dpe.tabname   = dp.tabname
    and dp.highvalue <> ''
```

- Full query in notes
  - SELECT maximum value from the above

```
select
            tabschema
           ,tabname
           ,datapartitionkeyseq
           ,datapartitionexpression
           ,max(highvalue) as datapartitionhighvalue
          from
          (
           select
                    dpe.tabschema
                   ,dpe.tabname
                   ,dpe.datapartitionkeyseq
                   ,cast(substr(dpe.datapartitionexpression,1,50) as char(50)) as
datapartitionexpression
                   ,dpe.nullsfirst
                   ,dp.highvalue
               from syscat.datapartitionexpression dpe
                   ,syscat.datapartitions          dp
              where dpe.tabschema not like 'SYS%'
                and dpe.tabschema = dp.tabschema
                and dpe.tabname   = dp.tabname
                and dp.highvalue <> ''
          )
         group by
             tabschema
            ,tabname
            ,datapartitionkeyseq
            ,datapartitionexpression
            ,nullsfirst
         order by tabschema,tabname
        with ur
```

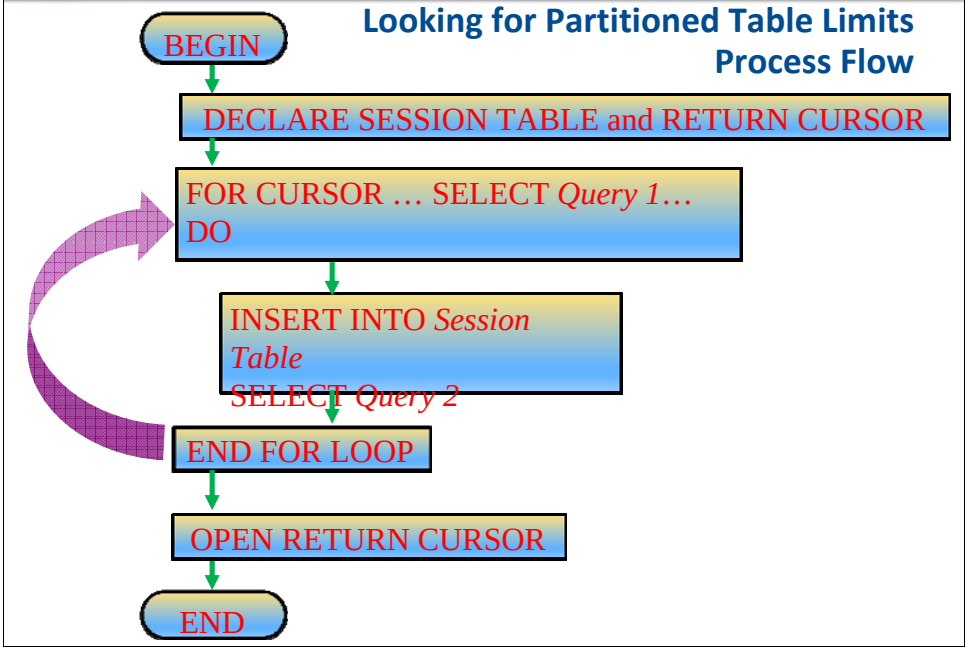
Looking for Partitioned Table Limits Query 2

- This is pretty simple

```
select max(end_date)
  from FRANK.REFERENCE
```

- Manual SQL coding
  - Tedious
  - Prefer a canned query
    - not possible as data type of partition expression can vary
- Write a stored proc

Looking for Partitioned Table Limits Process Flow



Looking for Partitioned Table Limits Stored Proc 1

- Returns a row for any table nearing limit
- 2 parameters
  - Percentage E.G. if within 10% of the maximum for partition expressions that are not date or timestamp
  - Number of days E.G. if within 90 days of the maximum for partition expressions that are date or timestamp

```
call FRANK.PartitionRangeStatus(10,90)
```

Looking for Partitioned Table Limits Stored Proc 2

- Full source of the stored proc in notes
- DATAPARTITIONEXPRESSION column is CLOB but only holds the partitioning Column name
- Must use session table
  - SQL is generated by the SQL PL
  - Cannot directly return results of generated SQL in dynamic cursor

```
declare global temporary table
  session.part_ranges
        (tabschema        … )
      on commit preserve rows
```

```
create or replace procedure FRANK.PartitionRangeStatus
(Alert_percentage smallint
,Alert_days      smallint
)
-------------------------------------------------------------------------------
-- Procedure to return a result set of the partition range keys
-- on partitioned tables so you can see if the ranges are going to run out
--
-- deploy:
--   db2 -tdi@ -vf PartitionRangeStatus_proc.sql
-- call:
--   db2 "call DB2DBA.PartitionRangeStatus"
-------------------------------------------------------------------------------
-- (c) 2013 Frank McEwen
-------------------------------------------------------------------------------
dynamic result sets 1
language sql
begin
  declare DynInsert             CLOB(4000);

  -- can't return a cursor from a dynamically built SQL so have to use a session table
  declare global temporary table session.part_ranges
    (tabschema                  varchar(128) not null
    ,tabname                    varchar(128) not null
    ,datapartitioncolumn        varchar(128) not null
    ,coltype                    varchar(128) not null
    ,datapartitionhighvalue     char(50) not null
    ,tablehighvalue             char(50) not null
    ,range_alert                char(50) not null
    )
    on commit preserve rows
    ;
  begin
    declare part_ranges cursor with return to client for
      select
        tabschema
        ,tabname
        ,datapartitioncolumn
        ,coltype
        ,datapartitionhighvalue
        ,tablehighvalue
        ,range_alert
      from  session.part_ranges
      order by
        tabschema
        ,tabname
      ;
    -- first select the partitioning column
    for part_cursor as
      select
        tabschema
        ,tabname
        ,datapartitionkeyseq
        ,datapartitioncolumn
        ,coltype
        ,nullsfirst
        ,max(highvalue) as datapartitionhighvalue
      from
      (
        select
            dpe.tabschema
          ,dpe.tabname
          ,dpe.datapartitionkeyseq
          ,cast(dpe.datapartitionexpression as varchar(128)) as datapartitioncolumn
          ,dpe.nullsfirst
          ,dp.highvalue
          ,cl.typename as coltype
        from
              syscat.datapartitionexpression dpe
            ,syscat.datapartitions          dp
            ,syscat.columns                 cl
        where dpe.tabschema not like 'SYS%'
          and dpe.tabschema = dp.tabschema
          and dpe.tabname   = dp.tabname
          and dp.highvalue <> ''
          and dpe.tabschema = cl.tabschema
          and dpe.tabname   = cl.tabname
          and dpe.datapartitionexpression   = cl.colname
      )
      group by
        tabschema
        ,tabname
        ,datapartitionkeyseq
        ,datapartitioncolumn
        ,coltype
        ,nullsfirst
      order by tabschema,tabname
      with ur
    do
      -- set default alert thresholds if not passed
      if alert_days is null then
        set alert_days = 90;
      end if;
      if alert_percentage is null then
```

Looking for Partitioned Table Limits Stored Proc 3

- Cursor on session table

```
declare part_ranges cursor
with return to client
for select …
     from  session.part_ranges
     …
```

Looking for Partitioned Table Limits Stored Proc 4

- FOR loop on CURSOR selecting partition tables
  - SELECT as shown on earlier slide
  - FOR cursor minimizes working storage variable declare statements

```
for part_cursor as
        select …..
do
    …
end
```

Looking for Partitioned Table Limits Stored Proc 5

- INSERT alerts into session table
  - Select maximum partition expression value from table
  - Include DB2 catalog high range
  - CAST the expressions as VARCHAR(128) so information for all tables can be stored in one session table regardless of the partition expression type
    - Matches SYSCAT.COLUMNS.COLNAME
    - Could use any other character data type instead
  - Dynamically build INSERT statement
  - EXECUTE IMMEDIATE INSERT statement

Looking for Partitioned Table Limits Stored Proc 6

- Column “RANGE_ALERT” is set to “OK” or “Alert”
  - depending on the values and passed parameters
  - Value of “Alert” in the result set will cause alert to be sent by the calling script
- One row is inserted for each partitioned table in the cursor loop
- Before ending open CURSOR on the session table
- Call to stored proc returns result set

Looking for Partitioned Table Limits Stored Proc 7

- Final result

```
call db2dba.PartitionRangeStatus(10,90)

  Result set 1

TABSCHEMA TABNAME     DATAPARTITIONCOLUMN COLTYPE DATAPARTITIONHIGHVALUE TABLEHIGHVALUE RANGE_ALERT
FRANK     REFERENCE   STMT_END_DATE       DATE    2015-12-31             2013-07-04     OK
FRANK     RFERENCE_BU STMT_END_DATE       DATE    2015-12-31             2013-02-09     OK

  2 record(s) selected.

  Return Status = 0
```

SQL PL on z/OS

- Some SQL PL and SQL statements are not supported on DB2 for z/OS
  - Cannot port all code to DB2 for z/OS
  - Can be recoded
- Array support added in DB2 for z/OS V11
  - Hopefully including CREATE TYPE
- ANCHOR is not supported on DB2 for z/OS
- Usage of VALUES differs from DB2 for LUW
  - Used in MERGE
  - Can use SELECT from SYSIBM.SYSDUMMY1 instead

Conclusion

- SQL PL rich language
  - Getting better as IBM continues to enhance it
- Understand and use the newer SQL PL features
- Build and execute of block SQL PL code dynamically
  - Equivalent of a macro language in SQL PL
- Think about building generic stored procedures
- No need to use external “diff” type processing to resolve differences between tables

Questions?

IDUG Leading the DB2 User Community since 1988

IDUG DB2 Tech Conference
Barcelona, Spain - October 2013

Frank McEwen

Independent Consultant

eponymous@frankmcewen.com

Session D5

Using DB2 for LUW SQL PL to make it faster, better, cheaper and simpler

INTERNATIONAL DB2 USERS GROUP
Independent • Not-for-Profit • User Run
Founded 1988