DB2 for z/OS: Utilities and Application Development

ABIS Training & Consulting
www.abis.be
training@abis.be

Document number: 0092_03b.fm
11 January 2006

Address comments concerning the contents of this publication to:
ABIS Training & Consulting, P.O. Box 220, B-3000 Leuven, Belgium
Tel.: (+32)-16-245610, Fax: (+32)-16-245691

TABLE OF CONTENTS

Utilities & Application Development

Objectives :

- **to introduce the main difference between SQL processing and the execution of utilities**
- **to give an overview of utility functions that can replace SQL processing**
- **to show with simple and more complex examples how and when you can take advantage from this functionality**

Utilities and application-related functionality 1.1

UNLOAD:

- for READ actions

LOAD:

- for WRITE actions

REORG:

- for REMOVE actions

REORG PAUSE or UNLOAD - (RE)LOAD:

- for REWRITE or COMPLEX actions

Utility processing versus SQL processing

SQL modifications:

- **a lot of <u>overhead</u>:**
  - **logging**
  - **locking**
  - **on the fly:**
    - ***<u>index maintenance</u>***
    - referential integrity control (and/or actions: cascade/nullifies)
- **In addition, if large amount of modifications:**

  ---> often, a REORGANISATION needed

For simple actions on 1 table:

---> UTILITIES are much <u>cheaper</u>

- most actions directly on VSAM files

Usability

In general:

- **for simple actions on high volumes**
  - stand-alone tables and activities
- **if application activity is always (or often) combined with utility processing**
  - e.g. batch job followed by a REORG
- **if asynchronous execution is allowed**
  - e.g action is not a part of a larger transaction

Alternative: dropping indexes

If SQL is needed for the modifications

- more tables involved in the process
- SQL functions needed
- ...

using utilities is no option any more

Alternative:

- **DROP the INDEXES that cause overhead, before processing**
  - indexes that are maintained 'in flight'
  - and not needed for the process itself
- **CREATE them again after processing**

BE prepared for INVALID application plans and packages

- **Ask a REBIND for these plans/packages:**
  - check the related access paths (EXPLAIN)

UNLOAD utility for READ activities 2.1

- **Example of a “Select of many rows”**

```
UNLOAD
    TABLESPACE TBD7971.TBSSTMOD
    FROM TABLE STOCKMODIFICATIONS
            (SMID, SM_PRNO, SMTIMESTAMP, SMQUANTITY)
        WHEN (SMTIMESTAMP BETWEEN ‘01.01.2004’ AND ‘31.12.2004’)
    SHRLEVEL CHANGE ISOLATION UR
```

  - **one scan of the input data set**
    (no indexes are used)
  - **selected rows are written as records in an output file**
  - **base table stays intact and available (see SHRLEVEL)**
- **Remark:**
  - UNLOAD utility = COPY of a formatted SUBSET
  - COPY utility = exact replication (backup)

UNLOAD utility for READ activities (I)

- **Additional possibilities:**

```
UNLOAD
    TABLESPACE TBD7971.TBSSTMOD
    FROMCOPYDDN DB2COPY1
    FROM TABLE STOCKMODIFICATIONS
        SAMPLE 5 LIMIT 2500
            (SMID,
             SMTIMESTAMP TIMESTAMP EXTERNAL,
             SMCOMMENT VARCHAR STRIP TRAILING)
        WHEN (SMTIMESTAMP BETWEEN '01.01.2004' AND '31.12.2004')
    PUNCHDDN DB2PUNCH
    DELIMITED COLDEL ','
    SHRLEVEL CHANGE ISOLATION UR
```

UNLOAD utility for READ activities (II)

- **FROMCOPYDDN**
  - to use an IMAGE COPY as input
- **PUNCHDDN**
  - to create corresponding LOAD utility statements
- **SAMPLE - LIMIT**
  - to limit the number of output records
- **TIMESTAMP EXTERNAL - VARCHAR STRIP TRAILING**
  - to format output fields
- **DELIMITED COLDEL ‘,’**
  - to format output records

LOAD utility for WRITE activities

- **Example of an <u>"Insert of many rows"</u>**

```
LOAD DATA
    RESUME YES
    SHRLEVEL NONE
    INTO TABLE STOCKMODIFICATIONS
    FORMAT UNLOAD or <FIELD-specifications>
```

- **input records are added as rows to the table(space)**
- **new index entries are sorted**
- **they are <u>merged with the existing entries</u>**
- **corrections are done for:**
  - duplicate keys
  - R.I. violations

LOAD utility for WRITE activities (II)

- **Additional possibilities:**

```
LOAD DATA
    LOG NO NOCOPYPEND
    ENFORCE NO
    RESUME YES
    SHRLEVEL CHANGE
    INTO TABLE STOCKMODIFICATIONS
    FORMAT UNLOAD or <FIELD-specifications>
```

- **LOG NO NOCOPYPEND - ENFORCE NO**
  - less overhead - faster processing
  - only for controlled data
- **SHRLEVEL CHANGE:**
  - no solution for performance
    (= mass insert with in flight maintenance of indexes)

REORG utility for REMOVE activities

- **Example of a "Delete of many rows"**

```
REORG TABLESPACE TBD7971.TBSSTMOD
    LOG NO SORTKEYS COPYDDN(SYSCOPY)
    STATISTICS TABLE ALL INDEX ALL
    SHRLEVEL REFERENCE
    DISCARD FROM TABLE STOCKMODIFICATIONS
        WHEN (SMTIMESTAMP < CURRENT DATE - 3 YEAR)
```

- **one scan of the input data set (tablespace)**
  (clustering index may be used)
- **non-discarded rows are written as records in an output file**
- **they are rewritten in clustering sequence into the table(space)**
- **remaining index entries are sorted and used to rebuild indexes**

REORG utility for REWRITE activities

- **Example of an "Update of many rows"**

```
REORG TABLESPACE TBD7971.TBSSTMOD
    UNLOAD PAUSE
```

```
Do while
    READ from <unload-file>
    modify record
    WRITE to <reload-file>
End
```

```
RESTART REORG TABLESPACE ...
```

- **unload-file has no indexes**
- **indexes are REBUILT after restart**
- **remarks:**
  - constraints (check and R.I.) are not checked
  - there will be no CHKP-status

Examples of simple scenarios

3

Adding many rows

3.1

```
Do while
    READ input-record
    ...
    INSERT INTO StockModifications
        VALUES(:WS-StockModifications:WS-SMInd)
End
```

- **Problem:**
  - **large input-files and 4 indexes**

    --> long execution time
  - **locking:**
    - from exclusive table lock and low ‘commit’ frequency to page locks and high ‘commit’ frequency:

      --> more concurrency

      --> +/- doubled elapsed time

Adding many rows (II)

- **Alternative**

```
LOAD DATA
    RESUME YES
    SHRLEVEL NONE
    INTO TABLE STOCKMODIFICATIONS
    <FIELD-specifications>
```

- **Plus:**
  - less execution time (between 25 and 50%)
  - no programming effort
- **Minus:**
  - FIELD-specifications
  - SHRLEVEL NONE
    (SHRLEVEL CHANGE = INSERT)
  - only KEY and R.I. control
  - no TRIGGER support

Copying many rows


```
DECLARE CopyStM CURSOR FOR
    SELECT  *
        FROM StockModifications
        WHERE SMTimestamp = CURRENT DATE - 1 DAY

LOCK TABLE StockModifications IN SHARE MODE
OPEN CopyStM
Do while
    FETCH CopyStM INTO :WS-StockModifications:WS-SMInd
    ...
    WRITE uotput-record
end
```

- **Problem:**
  - **time that the table is unavailable (exclusively locked) is too long**
  - **lock is needed: concurrent updates can give a wrong result**

Copying many rows (II)

- **Alternative 1:**

```
UNLOAD DATA
    FROM TABLE STOCKMODIFICATIONS
        WHEN (SMTIMESTAMP = CURRENT DATE - 1 DAY)
    SHRLEVEL REFERENCE
```

  - **Plus:**
    - faster than SQL (typical value: 3 times faster)
    - no programming effort
    - format of output records is controllable
  - **Minus:**
    - no complex conditions

Copying many rows (III)

- **Alternative 2:**

```
COPY
    TABLESPACE TBD7971.TBSSTMOD
    SHRLEVEL REFERENCE
```

```
UNLOAD
    TABLESPACE TBD7971.TBSSTMOD
    FROMCOPYDDN ...
    FROM TABLE STOCKMODIFICATIONS
        WHEN (SMTIMESTAMP = CURRENT DATE - 1 DAY)
```

- **Plus:**
  - COPY +/- 2 times faster than UNLOAD (of alternative 1)
- **Minus:**
  - second job (UNLOAD)
    (but without impact on the base table)

Deleting many rows


```
DELETE
FROM StockModifications
WHERE SMTimestamp < CURRENT DATE - 2 DAYS
AND SMStatus = '7'
```

- **Problem:**
  - **+/- 50% of the rows are deleted**
  - **table has 4 indexes**

    --> very long execution time (60 minutes)
  - **space management**

    --> REORG needed after execution
  - **remarks:**
    - locking: exclusive table lock

      --> low CPU-usage, many time-outs
    - R.I. is no problem (dependent table)

Deleting many rows (II)

- **Alternative**

```
REORG TABLESPACE TBD7971.TBSSTMOD
    LOG NO SORTKEYS COPYDDN(SYSCOPY)
    STATISTICS TABLE ALL INDEX ALL
    SHRLEVEL REFERENCE
    DISCARD FROM TABLE STOCKMODIFICATIONS
        WHEN (SMTIMESTAMP = CURRENT DATE - 2 DAYS
                    AND SMSTATUS = '7')
```

  - **Plus:**
    - much faster than SQL (few minutes)
    - no programming effort
    - REORG and DISCARD: one job
  - **Minus:**
    - no complex conditions
    - if parent table: CHKP for dependent tables

Examples of mixed scenarios

4

Deleting many parent rows

4.1

- **Process:**
  - **parent rows with a given status must be deleted (+/- 75%)**
  - **delete rule is ‘cascade’**
  - **on average 500 dependents for a parent row**
- **SQL solution**

```
DELETE
FROM LastOrders
WHERE LOStatus = ‘OK’
```

  - **Problem:**
    - index maintenance on LastStockModifications (dependent table)
    - (index maintenance on LastOrders)

Deleting many parent rows (II)

- **Alternative**

```
UPDATE LastStockModifications
    SET LSMStatus = 'OK'
    WHERE EXISTS (SELECT 1 FROM LastOrders
                  WHERE LOStatus = 'OK'
                  AND LO_PrNo = LSM_PrNo)
```

· efficient UPDATE process if no indexes on LSMStatus

```
REORG TABLESPACE TBD7971.TBSLSTSM
    DISCARD FROM TABLE LASTSTOCKMODIFICATIONS
        WHEN (LSMSTATUS = 'OK')
```

```
REORG TABLESPACE TBD7971.TBSLSTOR
    DISCARD FROM TABLE LASTORDERS
        WHEN (LOSTATUS = 'OK')
```

· DISCARD of parent rows --> dependent table: CHKP

Use of INCURSOR

- **Process:**
  - **information must be copied from a source table to a target table**
  - **availability of the target table is critical**
  - **complex selection criteria for the source table**
- **SQL solution**

```
INSERT INTO  StockModifications
    SELECT * FROM LastStockModifications T
        WHERE NOT EXISTS (SELECT 1 FROM Products
                                  WHERE PrNo = T.LSM_PrNo
                                      AND PrStatus = '7')
```

  - **Problem:**
    - index maintenance on StockModifications

Use of INCURSOR (II)

- **Alternative 1:**

```
EXEC SQL
    DECLARE GETSTMOD CURSOR FOR
        SELECT * FROM LASTSTOCKMODIFICATIONS T
            WHERE NOT EXISTS (SELECT 1 FROM PRODUCTS
                                WHERE PRNO = T.LSM_PRNO
                                    AND PRSTATUS = '7')
            ORDER BY LSMTIMESTAMP
ENDEXEC
LOAD DATA
    INCURSOR GETSTMOD
    RESUME YES
    SHRLEVEL NONE
    INTO TABLE STOCKMODIFICATIONS
```

Use of INCURSOR (III)

- **Alternative 2:**

```
INSERT INTO TempStockModifications
SELECT * FROM LastStockModifications T
        WHERE NOT EXISTS (SELECT 1 FROM Products
                          WHERE PrNo = T.LSM_PrNo
                            AND PRStatus = '7')
```

· efficient SQL if no indexes on TempStockModifications

```
UNLOAD DATA
    SHRLEVEL NONE
    FROM TABLE TEMPSTOCKMODIFICATIONS
```

```
LOAD DATA
    RESUME YES SHRLEVEL NONE
    INTO TABLE STOCKMODIFICATIONS
```

INSERT after complex control

Process:

- uncontrolled information (but correct format) must be copied from an input-file into a target table
- availability of the target table is critical
- **complex control criteria** for the input information

- SQL solution

```
Do while
    READ input-record
    <control activities>
    If OK
        INSERT INTO StockModifications
            VALUES(:WS-StockModifications:WS-SMInd)
End
```

- **Problem:**
  - again index maintenance on StockModifications

INSERT after complex control (II)

- **Alternative 1:**

```
LOAD DATA
    RESUME YES
    SHRLEVEL NONE
    INTO TABLE STOCKMODIFICATIONS
    <FIELD>-specifications
```

· by default: SMStatus = NULL (i.e. not yet available)
--> additional VIEW (SMStatus IS NOT NULL)?

```
DECLARE NewStMod CURSOR FOR
SELECT * FROM StockModifications
WHERE SMStatus IS NULL
Do while
    FETCH NewStMod INTO :WS-StockModifications:WS-SMInd
    <control activities>
    If NOT-OK
        <correction>
End
```

INSERT after complex control (III)

- **Alternative 2:**

```
LOAD DATA
    RESUME YES
    SHRLEVEL NONE
    INTO TABLE NEWSTOCKMODIFICATIONS
    <FIELD>-specifications
```

· no impact on the base table

```
DECLARE NewStMod CURSOR FOR
SELECT * FROM NewStockModifications
Do while
    FETCH NewStMod INTO :WS-StockModifications:WS-SMInd
    <control activities>
    If NOT-OK
        <correction>
End
```

· no impact on the base table

INSERT after complex control (IV)

- **Alternative 2 (cont.):**

```
EXEC SQL
    DECLARE GETSTMOD CURSOR FOR
        SELECT * FROM NEWSTOCKMODIFICATIONS
            WHERE NSMSTATUS IS NOT NULL
            ORDER BY NSMTIMESTAMP
ENDEXEC
LOAD DATA
    INCURSOR GETSTMOD
    RESUME YES
    SHRLEVEL NONE
    INTO TABLE STOCKMODIFICATIONS
```

· efficient index manipulation of StockModifications

Utilities and remote access

Process:

- refresh of a replicated table inside DB2
- at runtime initiated by a **‘remote Java application’**

Standard solution using SQL:

```
DELETE FROM LastStockModifications

INSERT INTO LastStockModifications
    SELECT *
        FROM STOCKMODIFICATIONS
        WHERE SMTIMESTAMP = CURRENT DATE
```

Problem:

- in flight index maintenance
- how to use utilities as alternative?

Stored Procedure: DSNUTILS

- **St Proc to execute utilities:**
  - **allocates dynamically data sets**
  - **creates the utility statements**
  - **invokes DB2 utilities using program DSNUTILB**
  - **inserts the result (SYSPRINT) into a temporary table (SYSIBM.SYSPRINT)**
  - **declares a cursor:**

```
DECLARE Sysprint
    CURSOR WITH RETURN FOR
    SELECT seqno, text
    FROM sysibm.sysprint
    ORDER BY seqno
```

  - **OPENs the cursor and RETURNs**

Stored Procedure: DSNUTILS (II)

- **Calling DSNUTILS:**

```
CALL DSNUTILS
        (<utility-id>,
         <restart-indicator>,
         <utility-statements>,
         <return-code>,
         <ANY / <utility-name>,
         <dataset-name1>,
         .....)
```

- **Utility statements**

```
UNLOAD DATA
    FROM TABLE STOCKMODIFICATIONS
        WHEN (SMTIMESTAMP = CURRENT DATE)
```

```
LOAD DATA REPLACE
    INTO TABLE LASTSTOCKMODIFICATIONS
```

Stored Procedure: DSNUTILS (III)

- **Checking the execution**
  - **SQLCODE (in calling program):**
    - 466: PROCEDURE ... RETURNED ... QUERY RESULTS SETS
  - **return-code (out parameter from St Proc)**
    - utility RETURN CODE (0, 4, 8 ...)
  - **utility report:**
    - can be fetched from SYSIBM.SYSPRINT