INFORMATION MANAGEMENT SERVICE (IMS)

Checkpoint / Restart Scenario

HERE'S THE SCENARIO:

Suppose, a batch program that basically reads an input file and posts the updates/inserts/deletes to DB2 tables in the database was abended before the end of the job because of some reasons; Is it possible to tell - How many input records were processed? Were any of the updates committed to the database or can the job be started from the beginning?

Assume that COMMIT logic was not coded for large batch jobs that process millions of records. If an ABEND occurs all database updates will be rolled back and the job can be resubmitted from the beginning. If an ABEND occurs near the end of the process, the rollback of all the updates is performed. Also, DB2 will maintain a large number of locks for a long period of time, reducing concurrency in the system. In fact, the program may ABEND if it tries to acquire more than the installation-defined maximum number of locks.

Program without COMMIT logic causes excessive locking in **BASE SYSPLEX** and **PARALLEL SYSPLEX** causes excessive consumption of memory. This can no longer continue if **DATASHARING** for DB2 is to provide workload balancing. These applications will cause the **COUPLING** facility to be over committed with large number of locks and huge storage requirements.

To avoid the above difficulties **COMMIT-RESTART LOGIC** is recommended for all the batch programs performing updates/inserts/deletes.

This involves setting up a batch-restart control table (CHECKPOINT_RESTART in our case) to store the last input record processed and other control information. The restart control table can also be used as an instrumentation table to control the execution, commit frequency, locking protocol and termination of batch jobs.

One of the problems with restart is synchronizing DB2 tables and output files. DB2 will rollback all work on DB2 tables to the last commit point; but for output files we have to delete all the records up to the last commit point. (One option to do this would be via a global temporary table, **FILE_POSITION_GTT,** See FILE REPOSITIONING section for further details.).

COMMIT Function:

The COMMIT statement ends a unit of recovery and commits the relational database changes that were made in that unit of recovery. If relational databases are the only recoverable resources used by the application process, COMMIT also ends the unit of work. The unit of recovery in which the statement is executed is ended and a new unit of recovery is effectively started for the process. All changes made by **ALTER, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON**, **RENAME, REVOKE** and **UPDATE** statements executed during the unit of recovery are committed.

SQL connections are ended when any of the following apply:

- The connection is in the release pending state
- The connection is not in the release pending state but it is a remote connection and:
  - ⇒ The **DISCONNECT(AUTOMATIC)** bind option is in effect, or
  - ⇒ The **DISCONNECT (CONDITIONAL)** bind option is in effect and an open WITH HOLD cursor is not associated with the connection.

For existing connections,

- All open cursors that were declared without the **WITH HOLD** option are closed.
- All open cursors that were declared with the **WITH HOLD** option are preserved, along with any **SELECT** statements that were prepared for those cursors.
- All other prepared statements are destroyed unless dynamic caching is enabled.
- If dynamic caching is enabled, then all prepared **SELECT, INSERT, UPDATE** and **DELETE** statements that are bound with **KEEPDYNAMIC (YES)** are kept past the commit.

  Prepared statements cannot be kept past a commit if:
  - **SQL RELEASE** has been issued for that site, or
  - Bind option **DISCONNECT(AUTOMATIC)** was used, or
  - Bind option **DISCONNECT (CONDITIONAL)** was used and there are no hold cursors.
- All implicitly acquired locks are released, except for those required for the cursors that were not closed.
- All rows of every global temporary table of the application process are deleted.
- All rows of global temporary tables are not deleted if any program in the application process has open **WITH HOLD** cursor that is dependent on that temporary table.
  - In addition, if RELEASE (COMMIT) is in effect, the logical work files for those temporary tables whose rows are deleted are also deleted.

CHECKPOINT/RESTART LOGIC:

To allow the interrupted program to be restarted from the last unit of recovery ***(COMMIT)*** or at a point other than the beginning of the program we should have a Checkpoint/restart logic. Basically, we need:

- A place to store the details (**CHECKPOINT-COMMIT** record) pertaining to the current execution of the program, like various counts (number of inserts/deletes/updates/selects), number of records processed, processing dates, and other details which are needed in the program after a **RESTART.**
- A reliable FILE RE-POSITIONING logic with minimal changes to the existing PROCJCL.
- Flexibility, to modify the commit frequency without changing the program code.

Where we can store this CHECKPOINT-COMMIT record?

We can store the CHECKPOINT-COMMIT record, COMMIT-FREQUENCY and other relevant information in a DB2 table .

CHECKPOINT_RESTART TABLE DESCRIPTION:

database	Tablename	tablespace	Dclgen
DBMPDBII	CHECKPOINT_RESTART	DBMTS002 (MAXROW=1	DBMDG002

COLUMN NAME	DCLGEN NAME	SIZE	DESCRIPTION
PROGRAM_NAME	PROGRAM-NAME	X(08)	Program name to identify
CALL_TYPE	CALL-TYPE	X(04)	Not used
CHECKPOINT_ID	CHECKPOINT-ID	X(08)	Not used
RESTART_IND	RESTART-IND	X(01)	Indicate that pgm needs to be restarted
RUN_TYPE	RUN-TYPE	X(01)	Prime time or not
COMMIT_FREQ	COMMIT-FREQ	S9(9) COMP	No. of records intervals to commit
COMMIT_SECONDS	COMMIT-SECONDS	S9(9) COMP	No. of seconds intervals to commit
COMMIT_TIME	COMMIT-TIME	X(26)	Update Timestamp
SAVE_AREA	SAVE-AREA-LEN SAVE-AREA-TEXT	S9(4) COMP X(4006)	Length of Commit record Save Area Commit record Save Area

FILE RE-POSITIONING:

At restart, all records written to the output file since the last commit will have to be removed. To accomplish this, FILE_POSITION_GTT global temporary table is used.
SQL statements that use global temporary tables can run faster because:

- DB2 does not log changes to global temporary tables
- Global temporary tables do not experience lock contention
- DB2 creates an instance of the temp table for OPEN/SELECT/INSERT/DELETE stmts. only
- An instance of a temporary table exists at the current server until one of the following actions occur:
  - The remove server connection under which the instance was created terminates
  - The unit of work under which the instance was created completes.
    For ROLLBACK stmt, DB2 deletes the instance of the temporary table.
    For COMMIT stmt, DB2 deletes the instance of the temporary table unless a cursor for accessing the temporary table is defined WITH HOLD and is open.
  - The application process ends.

File re-positioning Logic:

- Open the output file in INPUT mode
- INSERT all records from the output file to FILE_POSITION_GTT global temp table until the last record which was written at the time of last commit
- Close the output file
- Open the output file in OUTPUT mode
- FETCH all rows from the FILE_POSITION_GTT global temp table and write into output file
- In the Next commit, FILE_POSITION_GTT global temp table will be deleted automatically.

FILE_POSITION_GTT Global Temp Table:

Database	tablename	tablespace	Dclgen
DSNDB06	FILE_POSITION_GTT	SYSPKAGE	DSNDG006

COLUMN NAME	DCLGEN NAME	SIZE	DESCRIPTION
RECORD_NUMBER	FPG-RECORD-NUMBER	S9(9) COMP	Record number
RECORD_DETAIL	FPG-RECORD-DETAIL-LEN FPG-RECORD-DETAIL-TEXT	S9(4) COMP X(4000)	Output file length Output file record information

CHECKPOINT/RESTART Implementation:

STEP1: Create the **CHECKPOINT-COMMIT** record in the working storage section, to store the data, which is needed for the next unit of recovery.

STEP2: In the procedure division MAIN para:
First check the restart status flag i.e. RESTART-IND of CHECKPOINT_RESTART table.
If **RESTART-IND = 'N'** then
if any output file exists open output file in OUTPUT mode
start the normal process
end
If **RESTART-IND = 'Y'** then
Move the **SAVE-AREA** information to CHECKPOINT-COMMIT record
if any output file exists
do the FILE REPOSITION:
Open the output file in INPUT mode.
Repeatedly
Read the output record and INSERT it into GLOBAL temp table FILE_POSITION_GTT
Until the last unit of recovery write count.
Close the output file.
Open the output file in OUTPUT mode.
open a cursor for a table **FILE_POSITION_GTT**
repeatedly fetch a cursor and write the record information into the output file
until end of cursor
close a cursor
end
If input for the program is from cursor then skip the rows until **COMMIT-KEY**.
If input for the program is from file then skip the records until **COMMIT-KEY**.
End.
Note: For more than one output files, delete GTT after repositioning each output file.

STEP3: Make a count for each **Insert's/Update's/Deletes** in **RECORDS-PROCESSED-UOR** variable.

STEP4: Go thro' the logic and find out the appropriate place where **COMMIT WORK** can be hosted.
There check the frequency of COMMITS:
IF **RECORDS-PROCESSED-UOR > COMMIT-FREQ**
KEY (input) value of the program TO COMMIT-KEY
MOVE checkpoint-commit record length TO SAVE-AREA-LEN
MOVE checkpoint-commit record TO SAVE-AREA-TEXT
Update the CHECKPOINT_RESTART table with this information

END-COMMIT

STEP5: Before **STOP RUN** statement; reset the RESTART flag of the CHECKPOINT_RESTART table.
i.e. **MOVE 'N' TO RESTART-IND**
Update the CHECKPOINT_RESTART table with the above information.

Sample COBOL code for CHECKPOINT/RESTART Logic:

CHECKPOINT-COMMIT RECORD DEFINITION:

```
*****************************************************************************
*****     GLOBAL TEMPORARY TABLE CURSOR DECLARATION & OPEN  *****
*****************************************************************************
    EXEC SQL
       DECLARE FPG-FPOS CURSOR FOR
         SELECT RECORD_NUMBER
                ,RECORD_DETAIL
           FROM FILE_POSITION_GTT
          ORDER BY RECORD_NUMBER
    END-EXEC.

*******************************************************************************
*****       CHECK-POINT RESTART DATA DEFINITIONS          *****
*******************************************************************************
 01 COMMIT-REC.
     02 FILLER                    PIC X(16) VALUE 'REC. PROCESSED: '.
     02 COMMIT-KEY                PIC 9(06) VALUE 0.
     02 FILLER                    PIC X(14) VALUE 'TOTAL COUNTS: '.
     02 COMMIT-COUNTS.
        03 WS-REC-READ            PIC 9(06) VALUE 0.
        03 WS-REC-REJT            PIC 9(06) VALUE 0.
        03 WS-REC-WRIT            PIC 9(06) VALUE 0.
        03 WS-RECP-READ           PIC 9(06) VALUE 0.
        03 WS-RECP-UPDT           PIC 9(06) VALUE 0.

 01 CHKPRSL-VARS.
     02 RECORDS-PROCESSED-UOR     PIC S9(09) COMP VALUE +0.


**************************************************************
*****                                                    *****
*****       CHECK POINT RESTART LOGIC SECTION            *****
*****                                                    *****
**************************************************************
 RESTART-CHECK.

    MOVE 'XXXXXX  ' TO PROGRAM-NAME.
    PERFORM RESTART-SELECT.
    IF RESTART-IND = 'Y'
      MOVE SAVE-AREA-TEXT TO COMMIT-REC
```

```
      If input is from cursor the skip until the commit-key
      If input is from file then skip the records until the commit-key
    END-IF.


*************************************************
*****            CHECK RESTART STATUS            *****
*************************************************
 RESTART-SELECT.

    MOVE 0 TO RECORD-PROCESSED-UOR.
    EXEC SQL
          SELECT RESTART_IND
                    ,COMMIT_FREQ
                    ,RUN_TYPE
                    ,SAVE_AREA
               INTO :RESTART-IND
                    ,:COMMIT-FREQ
                    ,:RUN-TYPE
                    ,:SAVE-AREA
            FROM CHECKPOINT_RESTART
           WHERE PROGRAM_NAME = :PROGRAM-NAME
    END-EXEC.

    EVALUATE SQLCODE
    WHEN 0
              IF RESTART-IND = 'Y'
                DISPLAY '* * * * * * * * * * * * * * * * * * * * * * * * *'
                DISPLAY '      ***PROGRAM - ' PROGRAM-NAME ' RESTARTED***'
                DISPLAY '* * * * * * * * * * * * * * * * * * * * * * * * *'
              DISPLAY ' '
              END-IF
    WHEN 100
              PERFORM RESTART-INSERT
    WHEN OTHER
              MOVE 'RESTART-SELECT   '          TO  WS-PARA-NAME
              MOVE 'CHECKPOINT_RESTART SELECT ERR'  TO  WS-PARA-MSG
              PERFORM EXCEPTION-ROUTINE
    END-EVALUATE.
/
*************************************************************
*****         INSERT THE NEW RESTART STATUS RECORD        *****
*************************************************************
 RESTART-INSERT.

    MOVE SPACES             TO CALL-TYPE.
    MOVE SPACES             TO CHECKPOINT-ID.
    MOVE 'N'                TO RESTART-IND.
    MOVE 'B'                TO RUN-TYPE.
    MOVE +500               TO COMMIT-FREQ.
    MOVE ZEROES             TO COMMIT-SECONDS.
    MOVE +4006              TO SAVE-AREA-LEN.
    MOVE SPACES             TO SAVE-AREA-TEXT.
    EXEC SQL
       INSERT INTO CHECKPOINT_RESTART
```

```
                ( PROGRAM_NAME
                 ,CALL_TYPE
                 ,CHECKPOINT_ID
                 ,RESTART_IND
                 ,RUN_TYPE
                 ,COMMIT_FREQ
                 ,COMMIT_SECONDS
                 ,COMMIT_TIME
                 ,SAVE_AREA
                )
              VALUES
                ( :PROGRAM-NAME
                 ,:CALL-TYPE
                 ,:CHECKPOINT-ID
                 ,:RESTART-IND
                 ,:RUN-TYPE
                 ,:COMMIT-FREQ
                 ,:COMMIT-SECONDS
                 , CURRENT TIMESTAMP
                 ,:SAVE-AREA
                )
    END-EXEC.

    EVALUATE SQLCODE
    WHEN 0
              CONTINUE
    WHEN OTHER
              MOVE 'RESTART-INSERT   '        TO  WS-PARA-NAME
              MOVE 'CHECKPOINT_RESTART INSERT'  TO  WS-PARA-MSG
              PERFORM EXCEPTION-ROUTINE
    END-EVALUATE.
/
***************************************************************
*****         UPDATE THE CHECKPOINT RECORD              *****
***************************************************************
 RESTART-COMMIT.

    MOVE 'Y'  TO RESTART-IND.
    EXEC SQL

           UPDATE CHECKPOINT_RESTART
                SET RESTART_IND  = :RESTART-IND
                     ,SAVE_AREA    = :SAVE-AREA
                     ,COMMIT_TIME  =  CURRENT TIMESTAMP
            WHERE PROGRAM_NAME = :PROGRAM-NAME
    END-EXEC.
    EVALUATE SQLCODE
    WHEN 0
              EXEC SQL COMMIT WORK END-EXEC
              EVALUATE SQLCODE
              WHEN 0
                        CONTINUE
              WHEN OTHER
                        MOVE 'RESTART-COMMIT' TO  WS-PARA-NAME
                        MOVE 'COMMIT ERROR'   TO  WS-PARA-MSG
```

```
                    PERFORM EXCEPTION-ROUTINE
               END-EVALUATE
               MOVE 0 TO RECORD-PROCESSED-UOR
    WHEN OTHER
               MOVE 'RESTART-COMMIT'              TO  WS-PARA-NAME
               MOVE 'CHECKPOINT_RESTART UPDATE ERR'  TO  WS-PARA-MSG
               PERFORM EXCEPTION-ROUTINE
    END-EVALUATE.

*****************************************************************
*****     RESET THE RESTART FLAG AT THE END OF PROGRAM      *****
*****************************************************************
 RESTART-RESET.

    MOVE  0            TO RECORD-PROCESSED-UOR.
    MOVE 'N'           TO RESTART-IND.
    EXEC SQL
          UPDATE CHECKPOINT_RESTART
               SET RESTART_IND  = :RESTART-IND
                   ,COMMIT_TIME  =  CURRENT TIMESTAMP
          WHERE PROGRAM_NAME = :PROGRAM-NAME
    END-EXEC.
    EVALUATE SQLCODE
    WHEN 0
               EXEC SQL COMMIT WORK END-EXEC
    WHEN OTHER
               MOVE 'RESTART-RESET'               TO  WS-PARA-NAME
               MOVE 'CHECKPOINT_RESTART DELETE ERR'  TO  WS-PARA-MSG
               PERFORM EXCEPTION-ROUTINE
    END-EVALUATE.
/

 ************************************************************
 *****                                                  *****
 *****      OUTPUT FILE REPOSITION LOGIC SECTION        *****
 *****                                                  *****
 ************************************************************
*************************************************************************
*****     GLOBAL TEMPORARY TABLE CURSOR DECLARATION & OPEN  *****
*************************************************************************
 FPG-OPEN.

    EXEC SQL
          OPEN  FPG-FPOS
    END-EXEC.
    EVALUATE SQLCODE
    WHEN 0
               CONTINUE
   WHEN OTHER
               MOVE 'FPG-OPEN'                TO WS-PARA-NAME
               MOVE 'GLOBAL TEMP TABLE OPEN  ERR' TO WS-PARA-MSG
               PERFORM EXCEPTION-ROUTINE
    END-EVALUATE.

 ***************************************************************
```

```
*****     GLOBAL TEMPORARY TABLE CURSOR FETCH            *****
*****************************************************************
 FPG-FETCH.

    EXEC SQL
           FETCH FPG-FPOS
             INTO :FPG-RECORD-NUMBER
                  ,:FPG-RECORD-DETAIL
    END-EXEC.
    EVALUATE SQLCODE
    WHEN 0
                CONTINUE
   WHEN +100
                MOVE  0  TO FPG-RECORD-NUMBER
   WHEN OTHER
                MOVE 'FPG-FETCH '               TO WS-PARA-NAME
                 MOVE 'GLOBAL TEMP TABLE FETCH ERR' TO WS-PARA-MSG
                PERFORM EXCEPTION-ROUTINE
   END-EVALUATE.

*****************************************************************
*****     GLOBAL TEMPORARY TABLE CURSOR CLOSE            *****
*****************************************************************
 FPG-CLOSE.

     EXEC SQL
            CLOSE FPG-FPOS
     END-EXEC.
     EVALUATE SQLCODE
     WHEN 0
                MOVE 0 TO FPG-RECORD-NUMBER
     WHEN OTHER
                MOVE 'FPG-FPOS-CLOSE '          TO WS-PARA-NAME
                MOVE 'GLOBAL TEMP TABLE CLOSE ERR' TO WS-PARA-MSG
                PERFORM EXCEPTION-ROUTINE
     END-EVALUATE.

***************************************************************
*****     GLOBAL TEMPORARY TABLE INSERTS               *****
***************************************************************
 FPG-INSERT.

    ADD  1 TO FPG-RECORD-NUMBER.
    EXEC SQL
       INSERT INTO FILE_POSITION_GTT
       (
         RECORD_NUMBER
        ,RECORD_DETAIL
       )
       VALUES
       (
         :FPG-RECORD-NUMBER
        ,:FPG-RECORD-DETAIL
       )
    END-EXEC.
```

```
    EVALUATE SQLCODE
    WHEN 0
              CONTINUE
    WHEN OTHER
              MOVE 'FPG-INSERT    '            TO WS-PARA-NAME
              MOVE 'GLOBAL TEMP TABL INSERT ERR' TO WS-PARA-MSG
              PERFORM EXCEPTION-ROUTINE
    END-EVALUATE.
/
 RESTART-FILE-REPOSITION.

      OPEN INPUT outputfile-name.
      MOVE LENGTH OF output-record TO FPG-RECORD-DETAIL-LEN.
      READ output-file INTO FPG-RECORD-DETAIL-TEXT.
      PERFORM UNTIL FPG-RECORD-NUMBER >= output record count of last commit
              PERFORM FPG-INSERT
              READ output-file INTO FPG-RECORD-DETAIL-TEXT
      END-PERFORM.
      CLOSE output-filename
      OPEN OUTPUT outputfile-name.
      PERFORM FPG-OPEN.
      PERFORM FPG-FETCH.
      PERFORM UNTIL FPG-RECORD-NUMBER = 0
              WRITE outputfile-record  FROM FPG-RECORD-DETAIL-TEXT
              PERFORM FPG-FETCH
      END-PERFORM.
      PERFORM FPG-CLOSE.

      ---------skip input file until the last commit------------------

      DISPLAY '  *** ALREADY ' COMMIT-KEY ' RECORDS PROCESSED ***'.
      DISPLAY ' '
      DISPLAY ' '.
/
************************************************************
************** E X C E P T I O N  R O U T I N E ***************
************************************************************
EXCEPTION-ROUTINE.

    MOVE     SQLCODE TO WS-SQL-RET-CODE.

    DISPLAY '************************************************'.
    DISPLAY '****  E R R O R   M E S S A G E S  ****'.
    DISPLAY '************************************************'.
    DISPLAY '*  ERROR IN  PARA.....: ' WS-PARA-NAME.
    DISPLAY '*        MESSAGES.....: ' WS-PARA-MSG.
    DISPLAY '*'.
    DISPLAY '*  SQL RETURN   CODE..: ' WS-SQL-RET-CODE.
    DISPLAY '************************************************'.

    CALL CDCABEND USING ABEND-CODE.
```

Output file Disposition in JCL:

- In JCL, disposition must be given as DISP=(NEW,CATLG,CATLG) or DISP=(OLD,KEEP,KEEP)
- Override statement is needed for the output files if job abended:
    1. GDG with DISP=(NEW,CATLG,CATLG)
        Override stmt:
        - Change +1 generation to 0 (current) generation
        - DISP=(OLD,KEEP,KEEP)
    2. GDG with DISP=(OLD,KEEP,KEEP)

    Override stmt:
        - Change +1 generation to 0 (current) generation

Output file with Disposition MOD:

- If output file is already existing, and program is appending records to that, then the File repositioning must be handled in different way according to the requirements.

Internal Sort:

☒ If any Commit-Restart program has Internal Sort, remove it and have an External Sort.

POINTS TO REMEMBER

✍ All the update programs must use COMMIT frequency from the CHECKPOINT_RESTART table only

✍ Avoid – Internal Sorts

- **Avoid – Mass updates (Instead, use cursor with FOR UPDATE clause and update one record at a time)**
- **On-call analyst should back-up all the output files before restart (The procedure should be documented in APCDOC)**
- **Reports to dispatch should be sent to a flat file; send the file to dispatch up on successful completion of the job**
- **Save only the working storage variables that are required for RESTART in the CHECKPOINT_RESTART table**
- **RESET the RESTART_IND flag at the end of the program**
- **If COMMIT-RESTART logic is introduced in an existing program then make relevant changes to the PROCJCL.**

Thought for the day

"**Relationships** and **Relational**