INFORMATION MANAGEMENT SERVICE (IMS)

Checkpoint / Restart Guidelines

POLICY/GUIDELINE NUMBER: 00286-01 **Revised Date:** September 27, 2016

SUBJECT: Issuing commits in DB2 for Z/OS.

Introduction: Issuing Commits in an application program, during a 'unit of work', has three important functions. First, if the program should fail, all the table modifications are backed out to the last COMMIT. If there are no commits, the backout process could take 2 – 3 times longer to put the data back to its original state before processing began. Second, if you restart the program, after abending, you are redoing work that our clients have already paid for. Third, if no Commits are coded, the locks are being held, thereby negating the use of concurrent access to data. Of course, being able to restart the program, after an abend, from the last commit is also important and takes skill in testing and implementing.

Purpose/Scope: This guideline has been established to ensure all 'batch' processing commits 'units of work' within reasonable time frames or after reasonable record counts to free resources and prevent unnecessary backout times. Section I refers to Inquiry type processing. Section II refer to modification (i.e. Insert, Update, and Delete) types of processing. Section III discusses additional performance enhancements.

Section I. INQUIRY (i.e. Select) processing.

The business needs or criteria for checkpoints should be the controlling factor in the number or timeframe for issuing commits. Each application is different, so these guidelines should be viewed as helping and not hindering you in your commit logic.

Batch processing.

If you are using batch access and commits are taken during timeframes and/or record counts, these are the recommendations:

For those wishing to use timeframes for commit processing use these guidelines:

- During Prime Time (i.e. when CICS are fully operational) hours, a commit should be every 30 seconds.
- For Non Prime Time hours, a commit should be every 5 to 10 minutes.
  Note: if you use the high-end time there should be no concurrent processing.

For those wishing to use record/row counts these are the policies:

- During Prime Time hours, a commit should be every 500 rows processed.
- During Non Prime Time hours. A commit should occur every 500 – 3000 rows processed.
  Note: if you use the high-end number there should be no concurrent processing.

For those wishing to commit work at regular intervals, a combination of both timeframes and record counts should be incorporated into your commit logic.

Section II. MODIFICATION (i.e. Insert, Update, and Delete) processing.

The business needs or criteria for checkpoints should be the controlling factor in the number or timeframe for issuing commits. Each application is different, so these guidelines should be viewed as helping and not hindering you in your commit logic.

Batch processing.

If you are using batch access (very important in Data Sharing environment!) and commits are taken during timeframes and/or record counts, these are the recommendations:

For those wishing to use timeframes for commit processing use these guidelines.

- During Prime Time (i.e. when CICS are fully operational) hours, a commit should be every 30 seconds.

- For Non Prime Time hours.

  A commit should be every 5 minutes for concurrency processing.

  A commit should be every 10 minutes for no concurrency processing.

For those wishing to use record/row counts these are the guidelines.

- During Prime Time hours. A commit should be every 500 rows processed.

- During Non Prime Time hours. A commit should occur every 500 – 1000 rows processed. Note: if you use the high-end number there should be no concurrent processing.

For those wishing to commit work at regular intervals, a combination of both timeframes and record counts should be incorporated into your commit logic.

Section III. Additional Performance enhancements.

Most application will tolerate these 'suggested' bind options; but take into account your checkpointing needs to release resources.

- Bind with **CURRENTDATA(NO)** for read-only processing.

- Bind with **RELEASE(DEALLOCATE)** to reduce tablespace locking activity. There is system overhead when release(commit) is used (i.e. allocating resources, un-allocating, and reallocating, etc.). This also prevents concurrency.

- Consider **ISOLATION(UR)** for virtually no locking. Good candidate for decision support applications where read-only processing.

- An alternative to **BIND**ing with **ISOLATION(UR)**, incorporate **WITH(UR)** clause in SELECT statement within the program.

- Coding **COMMIT**s in all DB2 programs is sound coding practice. If there is the slightest possibility the program could run longer, in the future, as data is added to the database. The program then can handle this change.

- Code **LOCK TABLE** statements when scanning the entire table with update intent (i.e. a delete program) or if the table is large. **Lock Table** in **Exclusive Mode** is recommended along with the **release(deallocate**) bind parameter. The lock table should not be used if there is concurrency processing.