

April 16, 1999

## Strategies for Updating DB2 Databases With Cactus/Maintain

This technical memo describes programming techniques for updating a DB2 database using the Cactus/Maintain language. It applies to the Maintain facility on all platforms as well as the Maintain language component of Cactus applications. The DB2 Interface is used throughout this tech memo for purposes of illustration; however, you can apply the same concepts to other relational Interfaces with modifications to account for RDBMS-specific behavior.

When updating a relational database, it is very important to make sure that another application does not modify a row after your application first retrieves the row and before it actually updates the row. If the row was modified by another application after it was first retrieved by Maintain and before it was updated by the same Maintain application, the update must not take place. A set of database transactions that must occur together or not at all is called a logical transaction or Logical Unit of Work (LUW).

The following two strategies are available for ensuring transaction integrity:

- **Depend on DB2's row locking mechanism.** With this technique, DB2 locks each row as it is read and retains the lock until the logical transaction is released by a COMMIT or ROLLBACK command. This technique is simple to use. All it requires is that you use an isolation level of Repeatable Read for all database I/O. However, the set of rows read can remain locked indefinitely, waiting for the user to respond to the terminal display and complete the edits to the data. During this time, other users may be denied access to the set of rows.
- **Change Verify Protocol (CVP).** This technique locks each row while it is being retrieved, releases the lock, and then relocks the row when it is ready for update, after making sure that the row was not changed by another user in the interim. The DB2 Interface using the MODIFY facility supports Change Verify Protocol and handles it automatically. Since Maintain allows updates of sets of rows, the same CVP facility is not applicable to Maintain procedures. Therefore, the Maintain programmer will need to develop a coding strategy for implementing a Change Verify function.

This technical memo describes how to implement these two strategies for updating relational databases with Maintain. Each strategy has advantages and disadvantages. The strategy you choose depends on your site's needs and DBMS programming standards. Your strategy will depend on the following factors:

- How Maintain implements logical units of work (LUW).
- RDBMS row locking requirements.
- Application change verify protocol coding.
- DB2 Interface isolation levels.

## Overview: Logical Units of Work

A logical unit of work is the span of time that starts when a transaction connects to a database and ends when a COMMIT or ROLLBACK command is issued to the database. The COMMIT or ROLLBACK command releases the locks held by the update or retrieval process.

Correctly acquiring and releasing locks is crucial in Maintain applications that update a database. Since the COMMIT and ROLLBACK commands release locks, you need to understand how and when a Maintain application communicates the COMMIT command to the Interface. The relational database Interface issues automatic commits to the RDBMS differently for Maintain language requests than for FOCUS or EDA requests. Note that a ROLLBACK command is never issued automatically by Maintain.

Maintain automatically (implicitly) issues a COMMIT at the end of every procedure *unless you specify a GOTO END KEEP in the called procedure*. The Maintain command GOTO END KEEP is the only way to sustain one logical unit of work during the execution of multiple Maintain procedures. In addition, you can explicitly issue a COMMIT or ROLLBACK command in any Maintain procedure.

## Automatic Commits in Maintain

Maintain does not support the FOCUS/EDA SQL DB2 SET AUTOCOMMIT environmental command that can be invoked in MODIFY or FOCUS TABLE requests to control automatic commits. Maintain automatically issues a COMMIT *at the end of every Maintain procedure*. To keep changes from taking place from one Maintain procedure to the next, the called procedure *must contain a GOTO END KEEP command*. With this command, locks that are acquired in the calling procedure and in called procedures remain in effect until the main calling procedure ends.

Given these factors, you have two strategies for writing Maintain procedures that update a DB2 database:

- Allow DB2 to control all row locking.

Using this strategy, you depend on the RDBMS to keep another user from modifying a row that Maintain is going to update. However, locks will be retained on all retrieved rows until the end of the top level Maintain procedure.

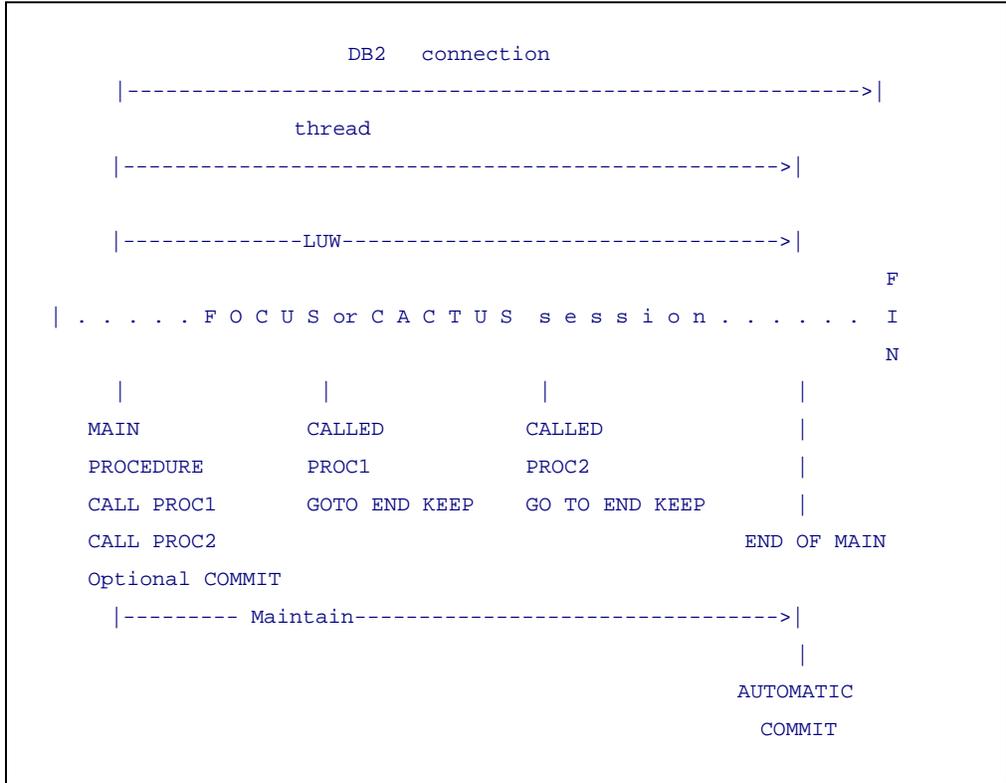
- Design the Maintain application to include its own Change Verify function.

With this strategy, locks will not be acquired until the application is ready to update the database; however, additional Maintain coding is needed to implement the Change Verify function.

Using Strategy 1: Allow DB2 to Control Row Locking

## Using Strategy 1: Allow DB2 to Control Row Locking

The following illustration shows the duration of connections, threads and Logical Units of Work (LUWs) using this strategy:



The requirements for this option are as follows.

1. The DB2 Interface plan must be bound with an isolation level of Repeatable Read.

This is a DB2 Interface installation BIND PLAN parameter. An isolation level of Repeatable Read retains row locks on all rows that Maintain retrieves into a stack. The locks are not released until a COMMIT command is issued.

2. A Maintain procedure can call other Maintain procedures as long as each called Maintain procedure ends its execution with the GOTO END KEEP command.

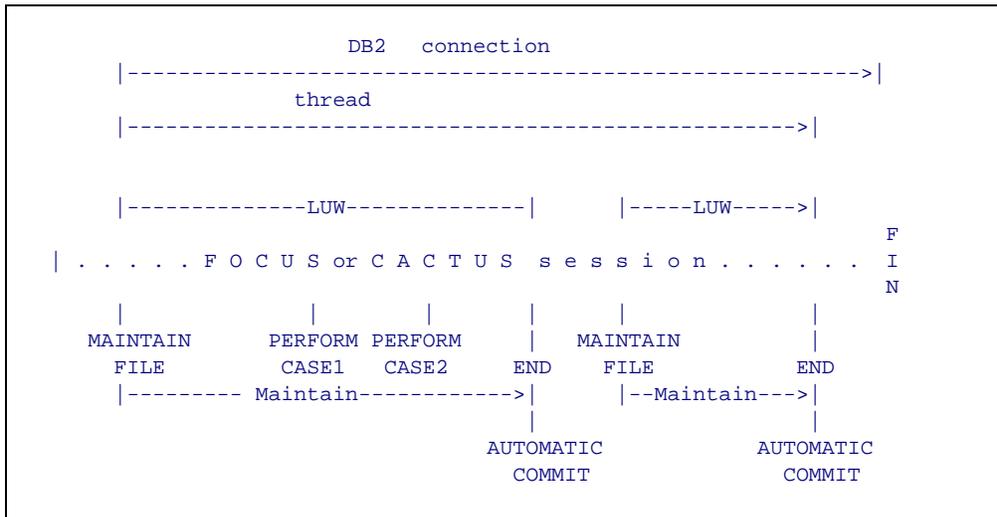
Any COMMIT or ROLLBACK command within any called procedure will release all locks, thus leaving the application vulnerable to a possible update breach by another user. A called procedure that ends without the GOTO END KEEP command will also release all locks.

3. Issue updates, inserts or deletes within the main or called Maintain procedures.
4. Do not issue a Maintain COMMIT or ROLLBACK command before all updates have taken place.

## Using Strategy 1: Allow DB2 to Control Row Locking

The key to this strategy is the use of the GOTO END KEEP command to exit from each called procedure. If you call a procedure that does not exit with this syntax, *you will compromise the integrity of the LUW*. The only way to avoid using the GOTO END KEEP command is to code the entire transaction within a single Maintain procedure, using cases instead of separate procedures for the separate parts of the transaction.

The following illustration shows the duration of connections, threads and Logical Units of Work (LUWs) using this technique:



The requirements for this option are as follows.

1. The DB2 Interface plan must be bound with an isolation level of Repeatable Read.

This is a DB2 Interface installation BIND PLAN parameter. An isolation level of Repeatable Read retains row locks on all rows that Maintain retrieves into a stack. The locks are not released until a COMMIT command is issued.

2. Do not issue any CALL procedure commands within a single Maintain.

The end of a called Maintain procedure would issue an automatic commit, releasing all locks and leaving the application vulnerable to a possible update breach by another user.

3. Issue all updates, inserts or deletes within one main Maintain procedure.

Instead of writing several Maintain procedures, write individual cases within one Maintain procedure, and execute these cases using the Maintain statements Perform CASE1, Perform CASE2, Perform CASE3. In this way, all processing takes place within one pair of MAINTAIN FILE ... END commands and, therefore, within one LUW.

4. Do not issue a Maintain COMMIT or ROLLBACK command before all updates have taken place.

## Using Strategy 2: Implement a Change Verify Function in the Maintain Application

### Implications of Strategy 1

The advantage of this strategy is that letting DB2 handle row locking allows the Maintain code to be much simpler.

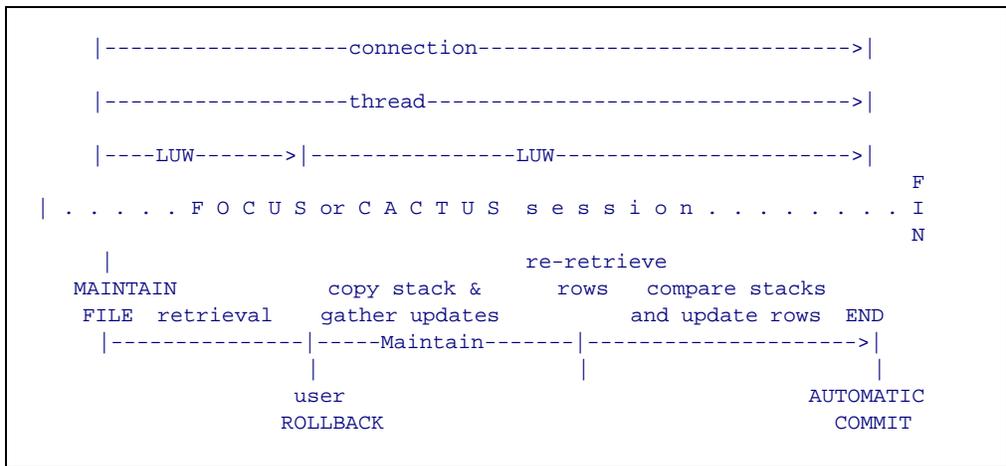
Possible drawbacks to this strategy include the following:

- Since the DB2 Interface plan has an isolation level of RR, there is a chance that other users may experience time outs due to row locking. DB2 will return either a -911 or -904 SQL code.
- Using this method, row locks are in effect for the greatest amount of time. These locks remain in effect for the database on which the updates take place until the terminal user completes the updates, which may interfere with ad-hoc queries that access the maintained database. Therefore, a separate DB2 database for Cactus applications or other activity scheduling may be needed.

This approach is the suggested method unless concurrency issues make it prohibitive.

### Using Strategy 2: Implement a Change Verify Function in the Maintain Application

The following illustration shows the duration of connections, threads and Logical Units of Work (LUWs) using this strategy:



The Maintain procedure should do the following:

1. Retrieve the rows from DB2 using a plan with an isolation level of Repeatable Read.
2. Execute a ROLLBACK command after reading either:
  - A set of rows with the FOR {n|ALL} NEXT command.
  - Each row in a repeat loop, until all of the rows are retrieved.

The ROLLBACK releases the lock on the rows held by the repeatable read isolation level.

## Data Concurrency

3. Make a copy of the stack.
4. Apply the updates to the first stack.
5. Before making the updates to the database, retrieve each row that is to be updated once again. Compare the newly retrieved row to the row in the saved stack. If there are no differences, issue the update on that row; otherwise return an error message indicating that the update is not allowed.

You can do all of this within one procedure and, therefore, take advantage of the automatic commit after the end of the procedure. Executing a `CALL` to another procedure will also work.

### Implications of Strategy 2

Advantages of this strategy include the following:

- This method retrieves rows and releases locks as soon as possible.
- The stack can be manipulated for as long as needed without any impact on the rest of the DB2 database.
- The database retrieval and update logic can be placed in separate Maintain procedures since locks are not needed until the actual update takes place.

However, the Maintain coding is more complicated.

## Data Concurrency

Data concurrency can be described as the total effect of multiple users on a database. The effect of multiple update and retrieval processes on a single database raises issues of data availability. The more rows that are locked or in use, the greater the chance that the application will hang for long periods waiting for the row or rows to be freed. A popular application technique is for the program to check for a locked condition upon reading a row. If the read request returns a row lock, the program can try to read it in a loop until the row is freed. The row is read for a user a specified number of times; after exceeding the number of attempts, a screen is displayed to the user indicating that the row is in use by another and to try again later. This technique can be used to mitigate the effects of row locks.

This technique is recommended for strategy 1 since the likelihood and number of locks held will be greater than with strategy 2. Since row locks will be held for longer periods of time, the handling of lock situations by the Maintain procedure will be helpful.

### Maintain Variables

As described previously, a Maintain procedure can help to ease the effects of row locks. There are new features planned for the Maintain language that will help you take into account relational database return codes. These return codes will be available in new Maintain variables to allow a Maintain procedure to query the contents. Depending on the SQLCODEs returned, the Maintain procedure can be aware of a row lock situation and take the appropriate steps.