To: Distribution

From: O. D. Friesen

Date: April 6, 1978

Subject: MDBM Recovery and Concurrency Control

Introduction

This MTB discusses the recovery and concurrency control features to be made available to users of the Multics Data Base Manager (MDBM) for MR7.0. Information about the MDBM as it is currently implemented may be gleaned from the Multics Relational Data Store (MRDS) Reference Manual (Order No. AW53), the Multics Integrated Data Store (MIDS) Reference Manual (Draft), and the LINUS Reference Manual (Draft). Please mail any comments or suggestions to Friesen.Multics on System M or call (602) 249-7245 or HVN 8*341-7245.

Proposal

With the transaction processing facilities to be available as a part of MR7.0 (December 1978) it will be possible for MDBM users to access a data base concurrently for update purposes without being required to reserve relations on an exclusive basis. (Which is currently the case in MRDS when using set_scope and dl_scope and in MIDS when using keepx and freex.) Instead users will be able to interact with the checkpoint and rollback features and will be able to lock out portions of the data base (or data bases) on an 'as needed' basis. Thus, in the event that processing is interrupted for some reason, users will be able to restore the data base to its state of existence when the last checkpoint was executed. Furthermore the level of granularity for lockout will be at the block level, rather than the relation level. (A block consists of Multics pages, the number of which is determined at data base creation time. For more detail concerning the proposed changes to the data base architecture, see MTB-359, "Enhancements to the Multics Data Base Manager.")

Concurrent users will also be assured that data accessed between the execution of checkpoints is consistent and repeatable. That is, data retrieved from a tuple at time t1 will contain identical values when the tuple is retrieved again at time t2, if no checkpoint has been executed between times t1 and t2. Alternatively, users will have the option of ignoring the repeatability feature, if they are only retrieving data. Thus, data retrieved from a tuple at time t1 may contain different values when the tuple is retrieved again at time t2.

Repeatability can be ignored only if the user is retrieving and not updating data. Repeatability is always enforced for update users.

Data bases created after the MR7.0 release will be assigned a new version number and will be recoverable using rollback and checkpoint. The scope mechanism (including keepx and freex in MIDS) used by current version data bases will still be operational for current version data bases after MR7.0. In order to use the rollback and checkpoint entry points, it will be necessary to redefine and reload the data base as a new version (recoverable) data base. A recoverable data base will not recognize the scope requests.

An administrative opening mode will be provided for use by the Data Base Administrator(DBA). With this opening mode it will be possible to quiesce selected files within a data base for dumping or restructuring. Exclusive openings of recoverable MRDS data bases will no longer be supported, and a null mode must be specified when opening a data base.

MIDS will continue to be usable with current version data bases as well as new version data bases. Currently, MIDS does not recognize the concept of area as an entity separate and distinct from the concept of data base. That is, MIDS areas are regarded as synonymous with MIDS data bases. This will continue to be the case for MR7.0.

For new version data bases the mode parameter supplied in the call to ready an MIDS data base will be interpreted to conform to the notion of repeatability. Thus, for new version MIDS data bases the concepts of concurrent retrieval/update and protected retrieval/update have no meaning. The concept of an area as a subset of a data base will be supported with the introduction of MIDS-II, hopefully prior to the release of MR 8.0.

Recovery of a data base by 'bringing it forward' using after images is made possible by the -after option to be provided by vfile_. This allows after images of each updated block to be collected on a sequential file at checkpoint time. This sequential file may reside on disk or tape.

MRDS User Interface

It is proposed that the dsl_$open entry be modified and that six new entries be provided for dsl_. They are dsl_$set_ctl_file, dsl_$reset_ctl_file, dsl_$ready_file, dsl_$finish_file, dsl_$checkpoint and dsl_$rollback.

Entry: dsl_$open

This entry causes the specified data bases to be opened for processing in the designated modes. For each opened data base, an index that is to be used to specify that data base in future MRDS calls is returned. If two or more data bases are to be concurrently open to the same process they must be opened in the same call to dsl_$open. If one or more of the data bases specified cannot be opened for any reason, none of the others will be opened.

The maximum number of data bases permitted to be open to any process is 64.

Usage

```
declare dsl_$open entry options (variable);

call dsl_$open (data_submodel_path1, data_base_index1, mode1,
... ,
     data_submodel_pathn, data_base_indexn, moden, code);
```

where:

1. data_submodel_pathi (Input) (char(*))
   is a character string containing the absolute or relative path name of the data submodel (or the data base) defining the relevant portion of the data base. If the path of the data base itself is specified, the data model is used in place of the data submodel.

2. data_base_indexi (Output) (fixed bin(35))
   is an integer that is to be used in subsequent MRDS calls to specify the corresponding data base designated in this opening.

3. modei (Input) (fixed bin(35))
   is an integer (0,1,2,3,or 4) indicating the usage mode for which the data base is to be opened.

   0 null mode is used when opening new version data bases. This is the only mode in which a new version data base may be opened.

   1 retrieval with concurrent access to the data base (both for update and retrieval) by other processes allowed. This is only applicable for old version data bases.

   2 update with concurrent access to the data base (both for update and retrieval) by other processes allowed. This is only applicable for old version data bases.

3 exclusive retrieval prohibiting the concurrent access to the data base by other processes for update purposes. This is only applicable for old version data bases.

4 exclusive update prohibiting the concurrent access to the data base by other processes for either update or retrieval. This is only applicable for old version data bases.

4. code (Output) (fixed bin(35))
is a standard status code.

Entry: dsl_$set_ctl_file

To insure recoverability and to guard against conflicts in a concurrent environment it is necessary for the MRDS user to associate the data base (or data bases) with a control file. The control file is used by vfile_ to coordinate transactions made against the data base and to record the state of completion of each transaction for the purposes of checkpoint and rollback.

Usage

```
dcl dsl_$set_ctl_file entry options (variable);

call  dsl_$set_ctl_file (control_sw, dbi1, ..., dbin, code);
```

where:

1. control_sw (Input) (char(32))
   Is the switch_name of the control file used by vfile_ to record the state of each transaction.

2. dbii (Input) (fixed bin(35))
   are the data base indexes of the data submodels or data bases against which accesses are to be coordinated.

3. code (Output) (fixed bin(35))
   is a standard status code.

Notes

Only one control file per process is allowed to be set at any one time for a given data base for each user of the data base.

This call must be executed after the data base has been opened.

If the control_sw is not already attached, then an attach description is created and the attachment is performed. If the control_sw is not open, then it is opened for keyed sequential update.

Entry: dsl_$reset_ctl_file

To terminate the use of a control file the dsl_$reset_ctl_file entry is called.

Usage

```
dcl dsl_$reset_ctl_file entry (char (32), fixed bin(35));

call dsl_$reset_ctl_file (control_sw, code);
```

where:

1. control_sw (Input)
   is the switch_name of the control file used by vfile_ to record the state of each transaction.

3. code (Output)
   is a standard status code.

Note

This call should be made only after all file accesses have been completed, else an error will be returned upon an attempt to access the file. Normally this call would be made between the finishing of one set of files and the readying of another set of files.

Entry: dsl_$ready_file

After a new version (or recoverable) data base has been opened, before any data can be accessed it is necessary to call dsl_$ready_file. A call to ready_file, after a data base has been opened, causes the specified files to be attached and opened. If the ready mode desired for this file equals 1 (= retrieve), then the -transact option is not included in the vfile_ attach description. If the ready mode is 2 (= monitor_retrieve) or 3 (= update), the attach description includes the -transact <switch_name> option, where <switch_name> is the control_sw entered in the call to set_ctl_file. The attach options -stationary and -shared are also included in the attach description.

All files to be readied concurrently must be readied within the same call.

Usage

```
dcl dsl_$ready_file entry options (variable);

call dsl_$ready_file (dbi1, file_name1, rdy_mode1,
file_name2, rdy_mode2, ..., dbin, file_namem, rdy_modem, code);
```

where:

1. dbii (Input) (fixed bin(35))
   are the indexes to the data bases or data submodels of which the files to be readied are a part.

2. file_namei (Input) (char (30))
   are the file names representing each file to be readied in each of the open data bases.

3. rdy_modei (Input) (fixed bin)
   is the mode in which each of the files are to be readied.
   1 => retrieve (repeatability not assured)
   2 => monitor_retrieve (repeatability assured)
   3 => update (repeatability assured)

4. code (Output) (fixed bin(35))
   is a standard status code.

Notes

All files readied must be readied in the same call statement. Before another ready_file call can be made it is necessary to call finish_file for all readied files or to close all data bases which contain a readied file.

If the data base is being referenced through a data submodel,

then the file may in reality reference a subset of those relations appearing in the file.

When files are readied in the monitor_retrieve or update mode the user is assured that repeated accesses of the same tuple, uninterrupted by checkpoints, will either yield identical data or will return an error code of mrds_error_$asynch_change. When this error code is returned the user should call dsl_$rollback. The data base is then returned to its state when the most recent call to dsl_$checkpoint was executed by this user. (Only changes made by this user are rolled back, and changes made by other users remain unaffected.)

When a file is readied in retrieve mode the checkpoint and rollback facilities need not be used. In this mode the user is assured only that the retrieved data is "clean" or valid data. It is entirely possible that other users may update data being queried by this user. Hence, repeatable retrievals cannot be guaranteed.

Entry: dsl_$finish_file

After a file has been readied it is necessary that the file be finished. This is accomplished implicitly whenever the containing data base is closed. Files may also be finished explicitly by calling dsl_$finish_file.

Usage

```
dcl dsl_$finish_file entry options (variable);

call dsl_$finish_file (dbi1, file_namem, file_namem, ...,
dbin, file_namem, file_namel, ..., code);
```

where:

1. dbii (Input) (fixed bin(35))
   are the indexes to the data submodels or data bases of which the files to be finished are a part.

2. file_namei (Input) (char (30))
   are the file names representing each file to be readied in each of the open data bases.

3. code (Output) (fixed bin(35))
   is a standard status code.

Entry: dsl_$checkpoint

In order to mark a series of updates to a data base as complete and final the user calls dsl_$checkpoint. This entry is also called to release the references to retrieved blocks when a file is open in monitor_retrieve mode.

Usage

```
dcl  dsl_$checkpoint entry (fixed bin (35), fixed bin (35));

call dsl_$checkpoint (trans_id, code);
```

where:

1. trans_id (Output)
   is an identifier for the transaction being checkpointed.

2. code (Output)
   is a standard status code.

Entry: dsl_$rollback

This entry allows a user to effectively erase changes made to the data base since the last call to dsl_$checkpoint.

Usage

```
dcl dsl_$rollback entry (fixed bin(35), fixed bin(35));

call dsl_$rollback (trans_id, code);
```

where:

1. trans_id (Output)
   is an identifier for the transaction being rolled back.

2. code (Output)
   is a standard status code.

Example

```
call iox_$attach_name (control_sw, control_iocb_ptr, atd,
ref_ptr, code);
```

where:

1. control_sw (Input) (char(*))
   is the switch name to be assigned to the control file.

2. control_iocb_ptr (Output) (pointer)
   is a pointer to the control block of the control file.

3. atd (Input) (char(*))
   is the attach description for the control file.

4. ref_ptr (Input) (pointer)
   is a pointer to the referencing procedure.

5. code (Output) (fixed bin(35))
   is a standard status code.

```
call iox_$open (control_iocb_ptr, mode, "0"b, code);
```

where:

1. control_iocb_ptr (See above)

2. mode (Input) (fixed bin)
   equals 10 for keyed sequential update.

```
call dsl_$open (db_path, dbi, open_mode, code);
```

where:

1. db_path (Input) (char(*))
   is the path name of the data submodel or data base to be opened.

2. dbi (Output) (fixed bin(35))
   is the data base index.

3. open_mode (Input) (fixed bin(35))
   equals 0 (null).

4. code (Output) (fixed bin(4.))
   is a standard status code.

```
call dsl_$set_ctl_file (control_sw, dbi, code);

call dsl_$ready_file (dbi, file_name, rdy_mode, code);
```

where:

1. dbi (See above)

2. file_name (Input) (char(30))
   is the name of a file to be readied.

3. rdy_mode (Input) (fixed bin)
   is equal to 3 for update.

This gives update permissions to all relations in the specified file.

```
call dsl_$modify (dbi, selection_expression, new_values, ..., code);
```

To checkpoint a transaction after having established a control file and having readied the relevant files, the MRDS user calls dsl_$checkpoint.

```
call dsl_$checkpoint (trans_id, code);
```

If the code after returning from the call to dsl_$modify was non-zero, then it is possible that the user would wish to rollback the data base to the state of its existence at the last checkpoint (which in this example would be equivalent to its state at open time). Then the user would call rollback.

```
call dsl_$rollback (trans_id, code);

call dsl_$finish_file (dbi, file_name, code);

call dsl_$close (dbi, code);
```

MIDS User Interface

It is proposed that dml_$ready be modified and that four new entries be provided for dml_. When readying new version data bases, the mode parameter will be mapped onto the retrieve, monitor_retrieve and update modes recognized by new version data bases. The new entries are dml_$set_control, dml_$reset_control, dml_$checkpoint and dml_$rollback.

Entry: dml_$ready

This entry causes the specified data bases to be readied for processing in the designated modes. For each readied data base, an index which is to be used to specify that data base in future MIDS calls is returned. If two or more data bases are to be concurrently ready to the same process, they must be readied in the same call to dml_$ready. If one or more of the data bases specified cannot be readied for any reason, none of the others will be readied.

Usage

```
declare dml_$ready entry options (variable);

call dml_$ready (sub_schema_path, data_base_index, mode,
    ..., sub_schema_path, data_base_index, mode, code);
```

1. sub_schema_path (Input) (char(*))
   is the absolute or relative path name of the sub-schema defining the relevant portion of the data base.

2. data_base_index (Output) (fixed bin(35))
   is an index which is to be used in future MIDS calls to specify the data base designated by the corresponding sub_schema_path.

3. mode (Input) (fixed bin(35))
   is an integer indicating the usage mode for which the data base is to be opened:

   1 => concurrent retrieval (for old version data bases only) with concurrent access to the data base (both for update and retrieval) by other processes allowed. For new version data bases this mode will be mapped to the retrieve ready_mode.

   2 => concurrent update (for old version data bases only) with concurrent access to the data base (both for update and retrieval) by other processes allowed. For new version data bases this mode will be mapped to the update ready_mode.

   3 => protected retrieval (for old version data bases only) prohibiting the concurrent access to the data base by other process for update purposes. For new version data bases this mode will be mapped to the monitor_retrieve ready_mode.

   4 => protected update (for old version data bases

only) prohibiting the concurrent access to the data base by other processes for either update or retrieval. For new version data bases this mode will be mapped to the update ready_mode.

4. code (Output) (fixed bin(35))
is a standard system return code.

Entry: dml_$set_ctl_file

To insure recoverability and to guard against conflicts in a concurrent environment it is necessary for the MIDS user to associate the data base (or data bases) with a control file. The control file is used by vfile_ to coordinate transactions made against the data base and to record the state of completion of each transaction for the purposes of checkpoint and rollback.

Usage

```
dcl dml_$set_ctl_file entry options (variable);

call dml_$set_ctl_file (control_sw, dbi1, ..., dbin, code);
```

where:

1. control_sw (Input) (char(32))
   is the switch_name of the control file used by vfile_ to record the state of each transaction.

2. dbii (Input) (fixed bin(35))
   are the data base indexes of the data submodels or data bases against which accesses are to be coordinated.

3. code (Output) (fixed bin(35))
   is a standard status code.

Notes

Only one control file per process is allowed to be set at any one time for a given data base for each user of the data base.

This call must be executed before the data base has been readied. If this call is not made prior to the call to ready a data base, then the MIDS creates a control file as a default.

If the control_sw is not already attached, then an attach description is created and the attachment is performed. If the control_sw is not open, then it is opened for keyed sequential update.

Entry: dml_$reset_ctl_file

To terminate the use of a control file the dml_$reset_ctl_file entry is called.

Usage

```
dcl dml_$reset_ctl_file entry (char (32), fixed bin(35));

call dml_$reset_ctl_file (control_sw, code);
```

where:

1. control_sw (Input)
   is the switch_name of the control file used by vfile_ to record the state of each transaction.

3. code (Output)
   is a standard status code.

Note

This call should be made only after all data base accesses have been completed, else an error will be returned upon an attempt to access the data base. Normally this call would be made between the finishing of one set of data bases and the readying of another set of data bases.

Entry: dml_$checkpoint

In order to mark a series of updates to a data base as complete and final the user calls dml_$checkpoint. This entry is also called to release the references to retrieved blocks when a file is open in monitor_retrieve mode.

Usage

```
dcl dml_$checkpoint entry (fixed bin (35), fixed bin  (35));

call dml_$checkpoint (trans_id, code);
```

where:

1. trans_id (Output)
   is an identifier for the transaction being checkpointed.

2. code (Output)
   is a standard status code.

Entry: dml_$rollback

This entry allows a user to effectively erase changes made to the data base since the last call to dml_$checkpoint.

Usage

```
dcl dml_$rollback entry (fixed bin(35), fixed bin(35));

call dml_$rollback (trans_id, code);
```

where:

1. trans_id (Output)
   is an identifier for the transaction being rolled back.

2. code (Output)
   is a standard status code.

LINUS Implications

The LINUS user may also rollback and checkpoint recoverable data bases as well as ready and finish files. When a LINUS user opens a data base the mode no longer needs to be specified. This is done when a file, or files, is readied. If the LINUS user opens an old version (nonrecoverable) data base without specifying a mode, a message is printed informing the user that the data base requires a mode to be specified. The alternative choices are then printed, and the user is requested to enter the open request again.

When the request to open a data base is received by LINUS, a default attach description is generated for a transaction control file and the control file is attached and opened for keyed sequential update.

When a LINUS user readies a file, an appropriate attach description is generated which includes the -share and -transaction <control_sw> options, where the <control_sw> references the control file opened when the data base is opened. After the file to be readied is attached, it is opened for keyed sequential input or output, depending on the user's request.

When a file is readied in update or monitor_retrieve mode a LINUS macro called <unique_name>.recovery.linus is created in the user's process directory. This macro grows with the addition of each LINUS request entered by the user between checkpoint commands. When a checkpoint command is entered the macro is truncated and made ready for receiving the next set of LINUS commands. If the MRDS returns a mrds_error_$asynch_change error, then the data base is automatically rolled back to its state at the time of the most recent checkpoint, after which one of the

following two paths are followed, depending on whether the process is absentee or interactive.

If the process is an absentee process, or if there has been no user interaction with LINUS via the terminal since the execution of the most recent checkpoint, then the <unique_name>.recovery.linus macro is automatically invoked, processed and executed. A message is printed on the user terminal saying "Consistency verification being performed." Any output generated by the <unique_name>.recovery.linus macro is then generated. If the MRDS error is again encountered at the same point, rollback is performed up to a total of 10 times. Then if the process is an absentee process the data base is closed and the LINUS session is terminated. If the process is interactive, then the message "File <file_name> is busy." is printed on the user terminal, and the user is free to take whatever action is deemed advisable.

If the process is interactive and if the user has interacted with LINUS via the terminal since the execution of the most recent checkpoint, then the "Consistency verification being performed." message is printed. It is followed by the first command in the <unique_name>.recovery.linus macro, which is printed at the user terminal. The user is then asked whether the command is to be executed or skipped. Each command in the macro is stepped through in this manner. Thus, the user may choose to ignore those commands which may not be relevant to the internal data base structure.

If a user desires to use a LINUS program in conjunction with another program which accesses different data bases or files and if the user desires to maintain consistency across all files, then the transact command can be used to reference an exec_com which might use LINUS and other programs. An example of such a usage would be:

```
transaction control_sw "ec user.ec"
```

where control_sw is the switch name of a control_file and user.ec is a user written exec_com containing references to LINUS and any other programs the user wishes to execute. This control_sw references a different control file than is referenced by the LINUS macro invoked within the exec_com.

Request: ready, rdy

This request allows a user to ready a file or files in one of three modes. Before any accesses can be made to a data base, it is necessary to ready the appropriate files.

After a file has been readied it is not possible to do another ready until all readied files have been finished. (See the finish request.)

Usage

ready file_name1 rdy_mode1 ... file_namen rdy_moden

where:

1. file_namei
   are the names of the files to be readied. These are the files containing the tables which the user wishes to access.

2. rdy_modei
   are the modes in which the respective files are to be readied. The allowable modes are:
   update, u
   retrieve, r
   monitor_retrieve, mr

Request: finish

When a user no longer desires to access a file, the finish request can be issued. All readied files are implicitly finished when the data base is closed by the user.

Usage

finish file_name1 ... file_namen

where:

1. file_namei
   are the names of the files with which the user is finished.

Request: checkpoint, chp

This request causes all updates made since the most recent checkpoint request to be marked as complete, if the files are open in update mode. If the files are open in monitor_retrieve mode then this request causes the monitor to be reset, so that the data repeatability controls do not take into account any previous retrievals.

Usage

checkpoint

Request: rollback, rlb

If a user realizes that erroneous updates to a data base have been made, they may be effectively erased by issuing the rollback request.

Usage

rollback

(END)