Data recovery in IBM Database 2

by R. A. Crus

This paper presents the various forms of data recovery provided by IBM Database 2 (DB2). It describes the DB2 recovery log, introduces the notion of a unit of recovery, and discusses the two-phase commit protocol used by DB2. Furthermore, it describes what type of information is logged, the DB2 checkpoint process, what a compensation log record is, and how DB2 handles undo/redo processing, media recovery, restart after abnormal system termination, and data unavailability.

Today, more and more organizations rely on data processing systems to manage data critical to the day-to-day operation of the organization. In some cases, if data is unavailable, it is impossible to proceed with normal business operations. If data is inconsistent or contains errors, such discrepancies can also seriously interfere with normal business operations. If Murphy's law holds true ("Anything that can go wrong will go wrong!"), and it seems from experience that it does, data will become unavailable and errors are going to be introduced into the data at one time or another.

If IBM's relational data base management system, IBM Database 2 (DB2), terminates abnormally prior to completing all work, the data that it manages becomes unavailable until DB2 can be restarted, which will return the data to a consistent state. A system failure can be caused by hardware problems, certain software problems, and even by power outages. It is important that DB2 be able to restart quickly after a failure; it is even more important that the data be made consistent after a system failure.

Data errors can be divided into two types, logical errors and physical errors. Logical errors include updates that should have been made to the data base but were not, updates that were made but should not have been, and updates that were made incorrectly. Errors of this type can be caused by system failure, errors in the data base manager, errors in various critical system components, and even errors in the application programs that use the Data Base Management System. Physical errors are caused by media or hardware malfunctions; e.g., data stored on disk cannot be read from the disk or written to it. What DB2 does to recover from logical and physical errors is addressed in the remainder of this paper.

The rest of the paper is divided into three parts in addition to some conclusions that the author has reached concerning data recovery in DB2. The first part is an overview of recovery. The second part describes DB2 logging, including what information is recorded in the log, when it is recorded, and for what purpose. The third part discusses the various DB2 recovery processes.

Overview

As noted in the introduction, data may be damaged in a variety of ways. The basic approach to recovery

used by DB2 is to record all modifications made to data in the DB2 recovery log. When data is updated, both the way it looked before the modification and the way it looked after the modification are recorded in the log. The contents of the log can thus be used either to undo an update or to redo it. In this section, the DB2 recovery log and DB2 commit processing are described, and the notion of a "unit of recovery" is introduced.

The recovery log records both system status information and information describing changes made to data.

DB2 recovery log. The DB2 recovery log is used to record both system status information and information describing changes that are made to data. The DB2 recovery log consists of an active portion, an archive portion, and a special data set called the *Bootstrap Data Set* (BSDS). The active log resides in a set of log buffers in main storage and in a set of preallocated Virtual Storage Access Method (VSAM) entry-sequenced data sets. When an active log data set becomes full, an archive log data set is dynamically allocated, and the active log data set is off-loaded. The archive log is written via the Queued Sequential Access Method (QSAM) to standard label tapes, a direct access storage device (DASD), or a Mass Storage System. After the archive operation is complete, the VSAM data set is available for use as part of the active log again.

When log records are required for recovery, DB2 must determine which active or archive log data sets contain the records. This information is kept in the BSDS along with information used to help restart the system after it terminates either normally or abnormally. The BSDS itself is off-loaded during an archive operation along with the active log data set being archived.

Each portion of the log (i.e., the active log, the archive log, and the BSDS) may optionally be dual-copied. It is highly recommended that an installation maintain dual copies in all production environments, since one of the basic assumptions of recovery is that the integrity of the log can be depended upon.

Each log record written by DB2 has associated with it a Relative Byte Address (RBA), which is six bytes in length. The DB2 log is a linear address space of 2^{48} bytes, and each log record written begins at some offset from the beginning of the log. This offset is called the RBA of the log record. Note that an RBA can be likened to a point in time; that is, the event described by a log record whose RBA is x occurred prior to an event described by a log record whose RBA is $x + n$. Another basic assumption concerning the log and its use is that an RBA will never repeat. For all practical purposes this assumption is true, since if the DB2 log were to grow at a rate of 2^{34} bytes (16 gigabytes) per day, it would fill in 44 years.

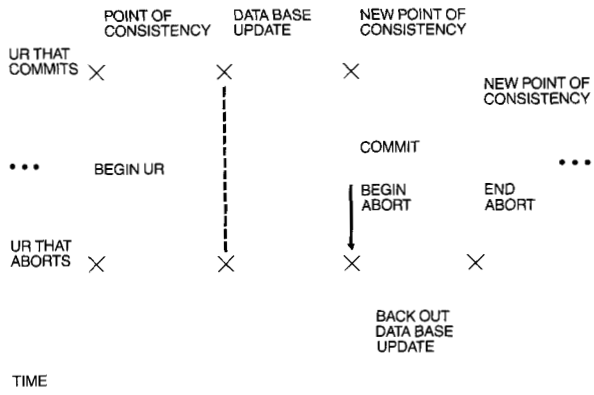
Normally when a log record is written, it is simply moved into a log buffer. Log buffers are written to disk as soon as one of the following occurs:

- The log is explicitly forced (i.e., all log data up to and including a specified RBA is written to disk).
- The installation-specified maximum number of log buffers are full.

When the log is forced and why it is forced are discussed in subsequent sections of the paper. The amount of storage devoted to the active log is under the control of the installation. By providing sufficient space on DASD for the active log, it is possible (almost) to guarantee that for most situations requiring rapid recovery, only information stored in the active log need be referenced.

Unit of recovery. A "unit of recovery" (UR) is the mechanism used by DB2 to track the progress of an application as it updates data stored in a data base. When an application program first attempts an update operation, DB2 establishes a UR for the application. Application in this context includes both user-written programs and the Query Management Facility[1] (QMF—a separate product that uses DB2) or DB2I[2] (an interactive interface to DB2). The UR lasts until a point of consistency is reached. (Note that an implicit point of consistency exists when the application first starts.) A point of consistency is that point at which, according to the application program logic, all data accessed and modified by the UR is in a consistent state. For example, a bank transaction might transfer funds from account A to account B.

Figure 1 Points of consistency



The transaction would first subtract the amount of the transfer from account A and then add the same amount to account B. After the second operation has been completed, the data in the two accounts is consistent. At this point the application can signal DB2 that the data is consistent and the updates are to be committed. When this happens, the UR for the transaction ends. If the transaction continues to run after this point, a new unit of recovery will be established when it next attempts to update the data base.

During the period after the update to A and before the update to B, the data in the two accounts is inconsistent and uncommitted. DB2 does not allow other applications to access uncommitted data stored in DB2 data bases. After the point of consistency has been reached (and the updates have been committed), the two accounts can then be accessed by other applications.

All updates made by a unit of recovery either are completed (and committed) or are aborted, which means the updates are backed out and appear to other applications as though they were never made (Figure 1).

When a UR is initiated, a begin-UR log record is written. All subsequent log records that are generated on behalf of the UR contain the RBA value of the previous UR-related record in the log, thus linking together all log records for a single UR (see Figure 2). When it is determined that a UR is to be committed or aborted, a log record is written (and forced to DASD) indicating the beginning of the commit or abort process. The various phases of the commit process are logged, as is the end of the commit or abort process (which is also forced to DASD). These records are used during restart to determine the state of the UR at the time of system termination.

Commit processing. A single application may involve DB2 and either TSO (the Time Sharing Option of MVS), Information Management System/Virtual Storage (IMS/VS), Customer Information Control System/Operating System/Virtual Storage (CICS/OS/VS), or batch programs. Because DB2 may operate in conjunction with another subsystem (IMS/VS and CICS/OS/VS) that also manages recoverable resources and has a recovery log (separate from the DB2 recovery log), it is necessary to coordinate the recovery activity between DB2 and the other subsystem. For data in the two subsystems to remain consistent, it is necessary that corresponding updates be made in both subsystems or in neither.

During commit processing, one subsystem coordinates the process, which consists of two phases. This subsystem is always IMS/VS or CICS/OS/VS. Neither TSO nor batch programs participate in commit; rather DB2 unilaterally controls the commit process.

The first phase of commit processing is known as the "prepare" phase, in which each subsystem determines that it can continue the commit process. Each subsystem logs this event and forces the log to DASD. The second phase, known as the "commit" phase, is when each subsystem records in the log the fact that the guarantee to commit was made.

Figure 3 shows the two-phase commit process. At point 1, IMS or CICS begins processing a transaction. At 2, a Structured Query Language (SQL) call is issued which attempts to update data, causing 3, a begin-UR event, to be logged in DB2. At point 4, the requested update operation occurs. At 5, IMS/CICS initiates the commit process and notifies DB2, which logs this event, performs its phase 1 processing, logs its completion, and then notifies IMS/CICS (at 7 and 8) that it has completed phase 1. At 9 the irrevocable decision to commit is made by IMS/CICS, and this event is recorded in the IMS/CICS log (on stable storage). At 10, IMS/CICS notifies DB2 that the decision is to commit, and DB2 logs this (11), completes its phase 2 processing, logs the event, and notifies IMS/CICS of the completion (12 and 13). IMS/CICS then completes its phase 2 processing (14).

Note that if either IMS/CICS or DB2 fails before point 9, when the decision to commit (or abort) is recorded in the IMS/CICS recovery log, the UR will be aborted.

Figure 2 UR-related log records

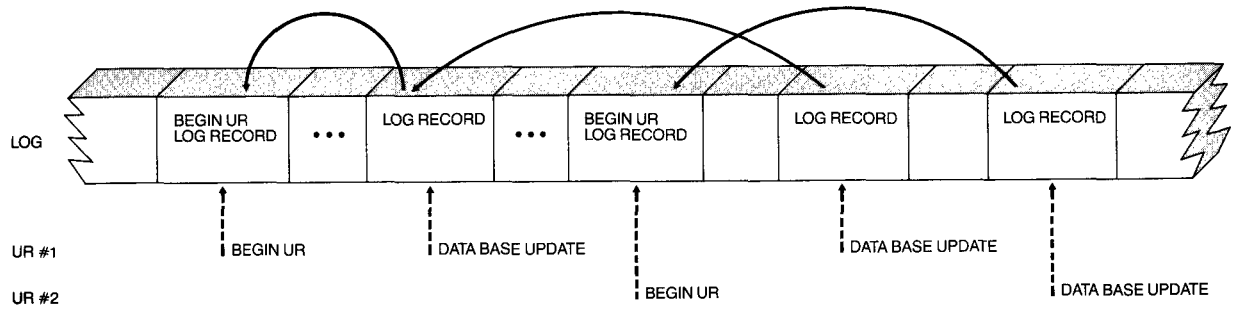
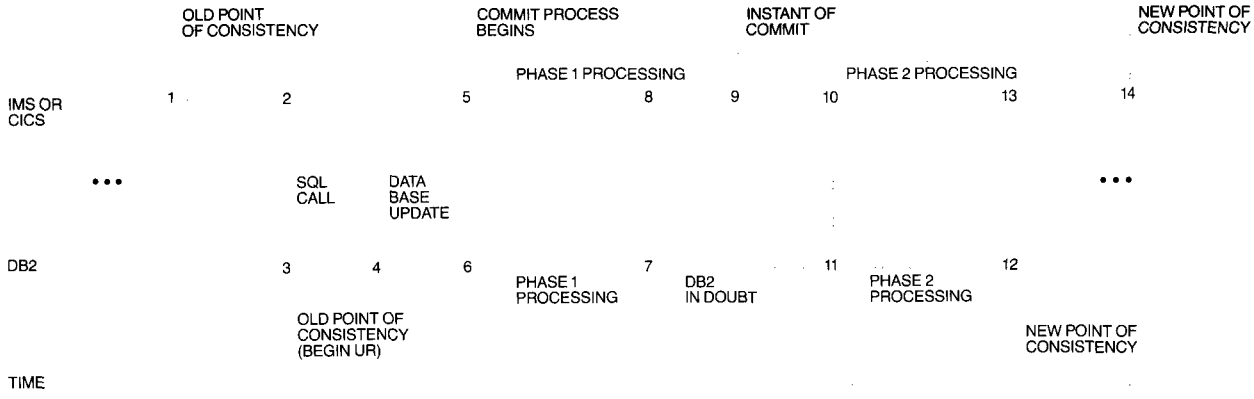


Figure 3 Two-phase commit



A failure of either system after 9 will result in the UR being committed. If DB2 fails between 7 and 11, the status of the UR is in doubt since DB2 has no knowledge of the decision at point 9. The status of an in-doubt UR is resolved after DB2 restarts and re-establishes communication with IMS/CICS.

DB2 logging

In addition to the UR control log records just described, other information is recorded in the log for recovery purposes. Some of this information is related to a UR (e.g., log records describing the updates performed by the UR), and some is not. Both types play an important part in the recovery process. Understanding what is logged, when it is logged, and why it is logged is necessary to understand data recovery in DB2.

Open/close. Whenever a tablespace (a DB2 object in which one or more tables are stored) or an index-space (a DB2 object in which an index defined on a table is stored) is opened or closed (the VSAM open or close of the data set(s) that contain the DB2 object), this event is logged. Note that log records of this type are not related to a single UR since DB2 supports multiple users accessing the same data. Logging this information provides a set of delimiters for updates to the data object; i.e., there may be information of interest in the log about an object only for the period between the open log record and the close log record. Included in the open log record is sufficient information to enable the open operation to be performed again. The open log record is used at restart time to perform the open operation if it is necessary. Since all pending I/O activity is completed prior to writing a close log record, this record is used during the restart process to filter out records relating to the closed object.

Checkpoint. At periodic intervals, depending on the amount of log activity and an installation-specified variable, DB2 initiates what is called a checkpoint operation, during which system status information

is recorded on the log. This process occurs asynchronously with normal data base activity and does not prevent access to any DB2-managed data. The checkpoint process consists of

- Writing a log record that denotes the beginning of the checkpoint.
- Writing a list of active URs in the log. These log records describe the current state of each active UR (in commit processing, in abort, or in neither). This information, along with other UR-related log records, is used at restart time to determine whether or not any recovery actions are required on behalf of active URs and, if so, what type of action.
- Writing information about each open tablespace and indexspace in the log. Included in the log records is sufficient information to perform the open operation at restart time as well as an indication of the earliest point in the log at which an outstanding update operation exists for the DB2 object. An update operation starts when the decision is made to perform the update and completes when the page containing the update has been recorded on DASD.
- Writing a log record that denotes the end of the checkpoint.
- Updating the active log data set information and the RBA of the checkpoint log record in the BSDS.

DB2 always begins the restart process at the last complete checkpoint.

DB2 always begins the restart process at the last complete checkpoint. The frequency at which checkpoints are taken thus has an effect on the amount of time required to perform restart since it tends to limit the quantity of log data that must be processed. Since the frequency of checkpoints is determined by the amount of information written in the log, the more update activity on a system, the more often checkpoints will occur. The checkpoint process itself writes information in the log (the amount of information is somewhat dependent on the system load) and therefore adds overhead and decreases performance. Yet, if the checkpoint frequency is too large, the restart process may have to access the archive log.

Writing pages to disk. DB2 maintains data base information as rows stored in pages on DASD. DB2 also maintains indexes in pages which are also stored on DASD. When a particular row or index entry is to be accessed, the page containing the row or index entry is read into a buffer in memory. DB2 maintains a pool of these buffers in memory. Since SQL is a set-oriented language, it is possible to update many rows contained in many pages in a data base with a single statement. Also, since DB2 allows more than one index to be defined on a table, it is possible for an update to a single row to cause updates to more than one index. It would not be possible or desirable from a performance point of view to hold all of the updated pages in memory until a commit point is reached; therefore, DB2 will write uncommitted changes back to the data base in order to reclaim buffer pool space. This design then makes it necessary for DB2 to be able to undo uncommitted updates after a system failure.

If a required page is already in the buffer pool, no read operation need be performed. If the page is not in the pool, a synchronous read operation is performed since the process cannot continue until the page can be accessed. Write operations, however, are usually performed asynchronously to the process that performed the modification. Once a page modification is complete, the page becomes a candidate for writing back to DASD.

During the modification process, DB2 writes one or more log records that describe the update operation. All data base modifications are logged as UR-related records. These records contain two types of information (where possible in a single log record):

- UNDO—sufficient information to undo an uncommitted update (e.g., the old value of an updated column)
- REDO—sufficient information to be able to redo an operation (e.g., the contents of a newly inserted row)

These log records also include information that identifies the DB2 object and the page number to which the log record applies.

The RBA of the log record written describing the most recent update is stored in a special location in the page. When it becomes necessary to write a page back to DASD, DB2 checks to see that the log record whose RBA is stored in the page has been written to DASD and, if not, forces the log up to that point before writing the page to DASD. Since the log is always written to DASD prior to the page being written, it is possible for DB2 to back out or undo uncommitted updates using information contained in the log.

Compensation log records describe the redo phase of an undo operation.

Compensation log records. Compensation log records (CLRs) describe the redo phase of an undo operation. Since DB2 write operations are not performed synchronously, it is possible that the undo operation of an uncommitted update which was originally externalized, might itself be lost if DB2 fails before the result of the undo is written to DASD. Consider the following sequence without CLRs:

- A UR updates a row, changing a value in a column from A to B. DB2 writes an undo/redo log record which indicates that the new column value is B and the old column value is A.
- The log is written to DASD.
- The page containing the row is written to DASD.
- The UR aborts.
- The log record describing the update is read, the page containing the row is read, and the update is backed out; i.e., the value in the column is changed from B to A.
- The end abort log record for the UR is written, and the log is forced to DASD.
- The system fails. Note that the page containing the back-out uptake has not yet been written to DASD.

If compensation log records are not written, then when the system is restarted, the restart process will read the log record describing the update of the column from A to B. This action will not cause an update since the row on DASD contains a B. The log record indicating the UR completed an abort will be read indicating that it is not necessary to process this UR. After the restart completes, the row contains an uncommitted update and is thus inconsistent.

Compensation log records correct the problem as follows.

- During the backout operation, a CLR is written which has a new column value of A and an old column value of B.
- The end abort log record is written, and the log is forced (including the CLR just written) to DASD.
- The system fails. The row stored on DASD still contains the value B.
- The restart process will read the first log record describing the update of A to B and not perform an update.
- Next it will read the CLR which will cause it to change the column from B back to A, thus completing the backout of the uncommitted update.

Recovery processes

Any recovery process involves applying either undo log records or redo log records or both. Since these functions are common, the rules for performing them are given below prior to the discussion of the recovery processes themselves.

Undo/redo processing. The RBA value stored in the header of a page uniquely identifies the log record that describes the last update operation made to the page. Therefore, it is possible to determine whether the update described by a log record was made to the page by comparing the RBA in the page to the RBA of the log record. If the RBA in the page is less than the RBA of the log record, the change described by the log record was not applied to the page. If the RBA in the page is greater than or equal to the RBA of the log record, the change was applied to the page.

The redo process simply determines whether the update described by the log record has been applied to the page, and if it has not, performs the update and stores the RBA of the log record in the page to indicate that the update has been made.

The undo process is more complicated. If the update operation was performed, a CLR describing the undo of the update is written, the update is undone, and the log RBA of the CLR is stored in the page to indicate

that the undo operation was performed. Even if the update operation was not performed, it is necessary to write a CLR that describes the undo of the update, although the RBA in the page is not modified. This is done so that the update can be undone (after being redone) if media recovery (discussed shortly) is necessary at a later time.

The rules for performing undo operations, including the writing of compensation log records, follow:

- If the log record being processed has only redo information, write an undo-only CLR for the opposite operation.
- If the log record has both undo and redo information, and if the RBA in the page is less than the RBA of the log record, write a redo/undo CLR. The page itself is not modified since the change described by the log record was not applied to the page.
- If the log record has both undo and redo information, and if the RBA in the page is greater than or equal to the RBA of the log record, apply the change to the page, write a redo/undo CLR, and store the RBA of the CLR in the page.
- If the log record has only undo information, and if the RBA in the page is less than the RBA of the log record, write a redo CLR. The page itself is not modified since the change described by the log record was not applied to the page.
- If the log record has only undo information, and if the RBA in the page is greater than or equal to the RBA of the log record, apply the change to the page, write a redo CLR, and move the RBA of the CLR to the page.

These rules assume that the page can always be accessed successfully, which is not always the case (e.g., I/O error); however, compensation log records are always written even if the page cannot be accessed. This is possible because the CLR can be written with only information contained in the log record being processed. It is necessary in order to be able to perform media recovery on the object.

The rules for redo processing are much simpler:

- If the log record contains redo information and the log RBA in the page is less than the RBA of the log record, apply the change to the page and store the RBA of the log record in the page.
- If the log record contains redo information and the log RBA in the page is greater than or equal to the RBA of the log record, do nothing.

Media recovery. Media recovery is generally used to recover from physical errors, i.e., data has been lost due to a media failure (e.g., a bad track, head crash) or from some other error which makes the data unusable. Media recovery consists of starting with a copy of the data (called an image copy) which was made at a known point in the log, and redoing all committed update operations using the appropriate redo information on the log up to the point of failure.

DB2 supports two different types of image copies. The first, called a full image copy, is simply a copy of a tablespace. The second, called an incremental image copy, makes a copy of only the pages that have been updated since the last image copy was made. This is accomplished as follows:

- When a page is first modified, DB2 sets an indicator on corresponding to the data page in another page called a space map page, and sets an indicator on in the data page itself indicating that it has been modified.
- When an incremental image copy is taken, DB2 uses the indicators in the space map to determine which data pages to copy. It resets both the indicator in the data page and the space map indicator when the copy has been made.

An image copy can be made while other users have read-only access to the data or while other users have read/write access to the data. In the case where read-only access is allowed, DB2 saves the log RBA at the point in time when the image copy has completed in the DB2 catalog and saves information about the data set which contains the image copy. In the case where the data may be modified while the copy is being made, the RBA value saved corresponds to one that existed when the image copy process was initiated. The RBA thus saved becomes the starting position in the log for any subsequent media recovery operation that uses the corresponding image copy.

In order to minimize the amount of log that must be processed during media recovery, DB2 remembers the log RBA when the first update is made to a DB2 data object and also remembers the log RBA when the object is closed. This information is stored for each object in a DB2 directory as a list of start/stop RBA values. Media recovery reads the start/stop RBAs associated with the object being recovered and only processes the log within the RBA ranges so defined.

Media recovery involves applying redo log records. These records may be redos of committed updates,

and they may also be redos of uncommitted updates, which are always succeeded by CLRs that are redo records for the undo of the update. If compensation log records were not written, it would be necessary to perform update operations, not knowing whether they were committed or not, and then after determining that the UR was aborted, undo all of the updates that had just been made.

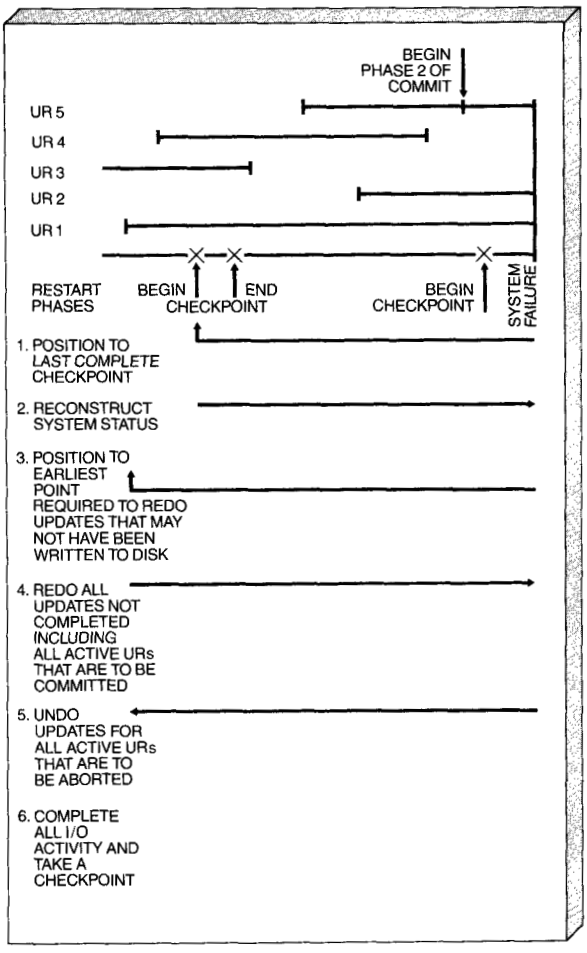
Restart. When a hardware/software failure or a power outage causes an abnormal system termination, the state of the data stored on DASD at that time is almost certainly inconsistent with that of data stored at other times because of the way DB2 performs write operations. If DB2 were to be started after a system failure without knowledge of the events in progress when the failure occurred, the data would be exactly as it was at the time of failure.

When DB2 is restarted after a system termination, it uses information recorded in the log to restore the state of the system (including the data bases) to the point where it was at the time of termination, and in fact requires only information stored in the log (i.e., no system directories, catalogs, etc., need be accessed). This capability is important because DB2 maintains system directories and catalogs in data bases of their own, which may themselves require recovery actions during restart processing. If the termination was not normal, e.g., a power failure or system error, the data bases may be in an inconsistent state in that committed updates may not have been stored externally on DASD, and/or uncommitted updates may have been recorded on DASD. If the termination was via a shutdown command, all activity was made quiescent (all URs were completed, all DB2 objects were closed, and all updates were written to DASD) prior to the termination. Therefore, many of the steps below result in no action being taken even though the same restart process is followed.

Figure 4 presents a summary of the restart process that pertains to the following discussion. Note that the figure shows a checkpoint that started but did not complete before the system termination. Any records associated with the incomplete checkpoint are ignored by DB2. The DB2 restart process is outlined as follows:

a. Read the BSDS and determine the RBA of the last complete checkpoint.
b. Beginning at the start of the last complete checkpoint, (phase 1), read the log in a forward direction and (at phase 2),

Figure 4 Summary of restart processing



- Establish the status of all URs at the time of the termination (i.e., whether the UR was active, in commit, or in abort) by using the UR information recorded in the checkpoint and updating this information with UR-related information recorded in the log (e.g., begin UR, end UR, etc.). In the figure, URs 1 and 2 are to be backed out since they have not committed, URs 3 and 4 have completed commit, and UR 5 will complete commit processing during restart because it entered phase 2 of commit before the system failure.
- Establish the set of all tablespaces and indexspaces that were open at the time of termination by using checkpoint log records for DB2 objects as well as any open/close log records.

- Determine the earliest point in the log that contains records describing updates that may have to be redone because the pages may not have been written to DASD. This is done by determining the minimum value of the RBAS stored in the checkpoint records for DB2 data objects and the RBAS of any open log records that are found.

c. Start reading the log in a forward direction beginning at the RBA determined in the previous step (3), and redo all updates that have not been completed (4). This process reads to the end of the log.
d. Read the log in a backward direction and undo the updates of all URs that were active or in abort at the time of termination (5).
e. Force all updates to be externalized, and wait for all I/O operations to complete and take a checkpoint (6).

The restart process is not complete until the end checkpoint record has been written to the log. If failure occurs during restart prior to this point, DB2 will terminate abnormally and a subsequent restart will be necessary. Therefore, the restart process must provide for this condition, which introduces additional complexity in the process itself. Consider the following: A UR aborts for some reason, and the system terminates abnormally. During the restart process the updates performed by the UR are undone (including the writing of CLRs), and the UR completes abort processing. All log records that describe this process are written to DASD, and then the system fails again, before the pages which were modified during the undo process could be written to DASD. The resulting inconsistency will be corrected during a subsequent restart (which hopefully completes successfully) because, during the forward recovery phase, CLRs describing the backout will be read and processed.

Additional complications may be introduced when DB2 is connected to IMS/VS or CICS/OS/VS. These complications occur when DB2 fails while connected to one of these subsystems and one or more URs have completed the processing of phase 1 of commit (recorded in the log) but have not started phase 2. The status of a UR in this state is termed "in doubt" because only the commit coordinator (IMS or CICS) knows whether the decision was made to commit or abort the UR. During DB2 restart, all update operations are undone for a UR that has not completed phase 1 of commit. Likewise, all updates for a UR that has begun phase 2 of commit are redone. For a UR that is in doubt, DB2 performs redo processing in anticipation of a decision to commit (which is highly probable because the system failure interrupted the commit process) and locks all of the data accessed by the UR to prevent other users from accessing uncommitted modifications. It is the responsibility of the coordinating subsystem to resolve the situations that are in doubt when it and DB2 reconnect.

DB2 attempts to include both redo and undo information in the same log record.

A further complication arises during the restart processing of a UR that is in doubt when individual undo and redo log records are processed. DB2 attempts to include both redo and undo information in the same log record, but this is not always possible. Consider the case where the redo record for an operation precedes the undo record in the log. Remember that during restart, DB2 performs redo for an "in-doubt" UR and that during the redo process, the RBA of the redo log record applied to a page is stored in the page itself. Later, when the status of the UR is resolved, the decision may be made to abort the UR. Then, when the undo log record is read, it appears to the undo procedure that the update operation was not performed because the RBA of the undo record is greater than the RBA of the redo record which was stored in the page. This problem may be solved as follows: If the RBA of the log record is greater than or equal to the RBA of the page during restart when an undo-only log record is processed for an "in-doubt" UR, the RBA of the undo log record is stored in the page even though no other updates are made to the page. In the case where the undo log record precedes the redo log record in the log, the processing of the redo record results in the RBA of the redo record being stored in the page. Thus, if the decision is made later to abort the UR, the undo process will work properly.

On-line recovery. If the data stored in a page becomes logically inconsistent, DB2 must restore the data to a consistent state. One way in which a page can be-

come inconsistent is because of software failures that result in errors being introduced into the page. If this situation is detected by DB2, it marks the page as logically inconsistent and denies further access to the page. This is necessary to prevent access to inconsistent data until the recovery process completes.

A second way that data in a page can become inconsistent is for a process to be aborted at a critical point in time. Updates to pages are in general not atomic (i.e., all or nothing) since they consist of distinct suboperations. For example, inserting a row into a page requires moving the row into free space within the page, writing a log record describing the insert, updating a page directory to point to the newly inserted record, updating the amount of free space remaining in the page, and storing the RBA of the record in the page. This operation is not atomic because a failure part way through will leave the update partially completed. Another example of a nonatomic operation is space reclamation within a page after rows have been deleted. When space reclamation becomes necessary, all the rows in the page are moved toward the beginning of the page, leaving a contiguous free area at the end. For ease of implementation and for performance reasons, it was decided not to log all of the above actions and thus, if the process is aborted at a critical point, the normal undo process cannot recover the page to a consistent state.

When DB2 detects an inconsistent page, it initiates an internal recovery operation.

To detect this condition, DB2 marks the page (in the page header) as being logically inconsistent at the beginning of the update process and marks it logically consistent again after the update completes. In the insert example given above, the page is inconsistent only after the log record has been written and until the RBA is stored in the page header. Although at first it might seem that the simplest solution might be to throw away the copy of the page in memory, this could cause the loss of committed updates that have never been written to DASD.

When DB2 detects an inconsistent page, it initiates an internal recovery operation which uses the log to perform forward recovery of the page using the version stored on DASD as the base. When the recovery operation completes, the page is marked consistent and can be accessed again. During the period when the page is marked inconsistent, it is unavailable for access, and a user attempting to access the page will receive a resource-not-available return code.

Unavailable data. In certain error situations, DB2 will itself make data unavailable until the data can be recovered.

If DB2 is unable to write a page to DASD because of a write I/O error, the page is added to what is called the write error range. Subsequent requests to access the page are denied, and the user will receive a resource-not-available return code. During abort and restart, if DB2 is unable to read a page because of a read I/O error, the page is also added to the write error range. This is necessary since the page cannot be read and the undo/redo operation cannot be performed.

Other situations in which DB2 will make data unavailable are during restart when a DB2 object (i.e., a tablespace or indexspace) cannot be opened, or when an error is detected (e.g., an addressing error is detected by the CPU) while DB2 is attempting to apply a log record to a page that has been read. In both of these cases, DB2 "stops" the DB2 object (the tablespace or indexspace), thus making it unavailable for access. When the object is stopped by DB2 during the restart process, compensation log records will still be written as discussed earlier.

Data made unavailable by DB2 will be made available again only after having been recovered. In the case of an I/O error, media recovery must be performed. In the case where the DB2 object is stopped, data may be recovered by performing a "deferred restart" or via media recovery. Deferred restart is a process that uses the version of the DB2 object that exists on DASD as the base and then performs redo starting at the point on the log established for the DB2 object from its checkpoint log record or open log record. The method chosen to recover the data depends on the reason why the object was made unavailable. If the problem was caused because a disk pack was not ready or was not mounted, or some similar condition, deferred restart would be chosen (since it would probably require the processing of fewer log records). In the case of such problems as a head crash that

damaged a disk pack, or a fallback if deferred restart failed, media recovery can be used to make the DB2 object available again.

Concluding remarks

Both the procedures that write log records and those that perform undo/redo operations based on them are fairly complex. The subjective cost of implementation mentioned by Gray[3] seems about right: "writing a recoverable action is 30 percent harder and requires about 20 percent more code than a nonrecoverable action." The total cost in lines of code for data recovery in DB2 is very close to that of System R, i.e., 11 percent for DB2 versus 10 percent for System R.[4]

Maintaining the active log on DASD allowed us to use the same log to perform undo operations during normal processing, as well as at restart time, and allowed us to use a single log for both undo and redo information.

It is often the case that an operation viewed externally as a single update actually requires that more than one internal update operation be performed. Two approaches can be followed to provide atomicity of an update operation that consists of many lower-level update operations:

- Do things twice—first access and lock all of the data items that have to be modified, and if everything is all right, re-access them and perform the updates.
- Do things once—access, lock, and modify the data, and if anything goes wrong, invoke an internal backout mechanism to undo the updates that have been performed.

The second approach was chosen, not only because it performs better if most operations complete successfully, but also because it was easier to implement because much of the backout mechanism was already required to support abort and restart. This backout mechanism is used internally by DB2 to give the appearance of atomicity to multirow operations; e.g., a multirow update operation either succeeds completely or no rows are updated. It is also used to support updates to multiple indexes. An example of this is a table with three indexes, not all of which allow duplicate key values. The insertion of a row might meet the uniqueness criteria of the first two indexes but fail that of the third, at which time the internal mechanism would be invoked to undo the two successful insertions.

We recognize that improvements can be made in the area of data recovery, and we also recognize the importance of this capability to our users. We expect that as DB2 evolves, additional procedures and techniques will be developed to continue to improve this very important area.

Cited references

1. J. J. Sordi, "The Query Management Facility," *IBM Systems Journal* **23,** No. 2, 126–150 (1984, this issue).
2. K. R. Hammond and M. R. Zimowski, "TSO Attach: A multipurpose communication channel to IBM Database 2," *IBM Systems Journal* **23,** No. 2, 151–164 (1984, this issue).
3. J. Gray, P. McJones, M. Blasgen, R. Lorie, T. Price, F. Putzolu, and I. Traiger, *The Recovery Manager of a Data Management System*, Research Report RJ-2623, IBM Research Division, 5600 Cottle Road, San Jose, CA 95193 (August 15, 1979).
4. M. W. Blasgen et al., "System R: An architectural overview," *IBM Systems Journal* **20,** No. 1, 41–62 (1981).

General references

W. C. McGee, "The information management system IMS/VS," *IBM Systems Journal* **16,** No. 2, 84–168 (1977).

IBM Database 2 General Information, S370-20, IBM Corporation; available through IBM branch offices.

J. P. Strickland, P. P. Uhrowczik, and V. L. Watts, "IMS/VS: An evolving system," *IBM Systems Journal* **21**, No. 4, 490–510 (1982).

Reprint Order No. G321-5217.

Richard A. Crus *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Crus joined IBM in 1965. He has been involved with the architecture and with the design and development of DB2 since its inception and was the technical team leader for the data manager component of DB2. Mr. Crus is currently an advisory programmer in the DB2 Advanced Development Department. He has a B.A. in mathematics from the University of Utah.