High Speed Transaction Recovery

By Craig S. Mullins

AVAILABILITY is the Holy Grail of database administrators. If your data is not available, your applications cannot run, and therefore your company is losing business. Lost business translates into lower profitability and perhaps a lower stock valuation for your company. These are all detrimental to the business so the DBA must do everything in his power to ensure that databases are kept on line and operational. This has been the duty of the DBA since the days of the first DBMS.

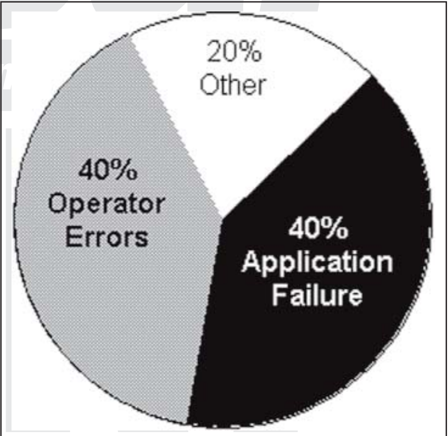
But the need for availability is increasing. The days of the long batch window where databases can be offline for extended periods to perform nightly processing are diminishing. Exacerbating this trend is the drive toward e-business. The Internet dramatically alters the way we do business, creating expectations for businesses to be more connected, more flexible, and more available.

When you integrate the Web with database management, DBAs must keep the databases up and running for longer periods of time. When your business is on the World Wide Web, it never closes. Remember, the Web is *worldwide*. People expect full functionality on sites they visit regardless of the time of day. It may be 3:00 AM in New York, but it is always prime time somewhere in the world. An e-business must be available and operational 24 hours a day, 7 days a week, 366 days a year (don't forget leap year).

These demands for higher availability make traditional forms of database recovery inadequate. Today's DBA must understand Transaction Recovery techniques to be able to prepare an optimal approach for every recovery situation.

This article will discuss Transaction Recovery from a DB2 for OS/390 point-of-view. Herein we will learn how Transaction Recovery provides the speed and ease of a point-in-time recovery with the selective capability provided by custom programming. Our discussion of Transaction Recovery will cover the types of Transaction Recovery available, the steps required to perform each type, and the features required for a comprehensive Transaction Recovery solution.

FIGURE 1: CAUSES OF UNPLANNED APPLICATION DOWNTIME



Source: Gartner Group

TYPES OF RECOVERY

When DBAs hear the word "recovery" the first thing that usually comes to mind is handling some sort of disaster. This disaster could be anything from a simple media failure to a natural disaster destroying your data center. Applications are completely unavailable until the recovery is complete.

Another traditional type of recovery is a Point-in-Time (PIT) recovery. PIT recovery usually is performed to deal with an application level problem. Conventional techniques to perform a PIT recovery will remove the effects of *all* transactions since that specified point in time. This can cause problems if there were any valid transactions during that timeframe that needed to be kept.

Transaction Recovery is a third type of recovery that addresses the shortcomings of the traditional types of recovery: downtime and loss of good data. Transaction Recovery is an application recovery whereby the effects of specific transactions during a specified timeframe are removed from the database.

BAD TRANSACTIONS HAPPEN TO GOOD DATABASES

Historically, recovery was performed primarily to overcome disasters and hardware failures. However, this is simply not the case any more. Application failures, not hardware failures, are the predominant drivers of recovery needs. Industry analysts at the Gartner Group estimate that as much as 80% of application errors are due to software failures and human error (see Figure 1). Although hardware and operating system failures were common several years ago, modern hardware and operating systems are more reliable, with a high mean time between failure.

In reality, except for disaster recovery tests, very few DBAs need to perform true disaster recovery. While media does fail, it's actually quite rare in this day and age. User errors and application failures are the most common causes of problems requiring recovery, and therefore, the primary cause for system unavailability. Any number of problems can occur at the application level. Consider:

- Edit checks aren't always 100% reliable. If there is any way garbage data can creep in, it will. If you don't believe me, do a spot check of your production databases.
- A disgruntled employee tampered with data before being discharged.
- Somebody changed job schedules or didn't check completion codes and processes were run out of sequence.
- No matter how much testing is done there is always the possibility that bugs will be found once the code hits production.
- There are times, particularly in test environments, where you may want to run a test and then roll the results back and try it again (and again and again….)

As databases grow in size and complexity, so, too, do the chances that bad transactions will corrupt the data on which your business depends.

TRANSACTION RECOVERY DEFINED

Transaction Recovery is the process of removing the undesired effects of specific transactions from the database. This statement, while simple on the surface, hides a bevy of complicated details.

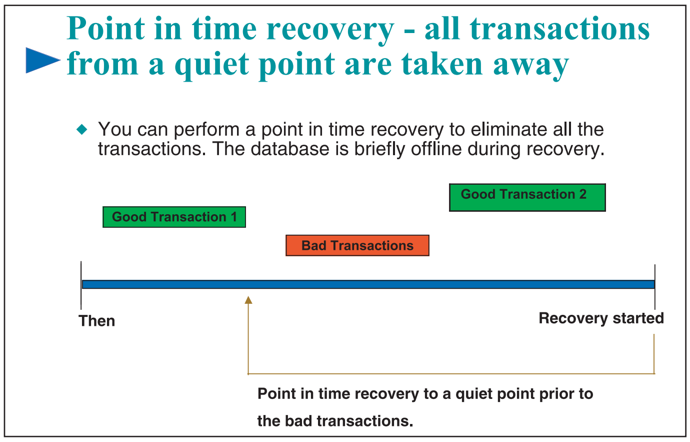
Transaction Recovery is the process of removing the undesired effects of specific transactions from the database.

Traditional recovery is at the database object level: for example, at the tablespace or index level. When performing a traditional recovery, a specific database object is chosen, a backup copy of that object is applied, and then log entries are applied for changes that occurred after the image copy was taken. This approach is used to recover the database object to a specific, desired point-in-time. If multiple objects must be recovered, this approach is repeated for each object impacted.

Transaction Recovery allows a user to recover a specific portion of the tablespace based on user-defined criteria. So only a portion of the data is affected. Any associated indexes are automatically recovered as the transaction is recovered.

The transaction may impact data in multiple tablespaces, too. A transaction is a set of related operations that, when grouped together, define a logical unit of work within an application. Transactions are defined by the user's view of the process, for example, the set of panels comprising a new hire operation or perhaps the set of jobs that post to the General Ledger.

FIGURE 2: POINT-IN-TIME (PIT) RECOVERY


Point in time recovery - all transactions from a quiet point are taken away
You can perform a point in time recovery to eliminate all the transactions. The database is briefly offline during recovery.
Good Transaction 1
Bad Transactions
Good Transaction 2
Then
Recovery started
Point in time recovery to a quiet point prior to the bad transactions.

Using Transaction Recovery, application problems can be addressed quicker, thereby maintaining a higher level of data availability. The database does not always need to be taken offline while Transaction Recovery occurs (it depends on the type of Transaction Recovery being performed).

So, how exactly is Transaction Recovery performed? Well, there are three types of Transaction Recovery: Point-In-Time, UNDO, and REDO. Let's examine each of these possibilities in more detail.

POINT-IN-TIME (PIT) RECOVERY

Point-in-time recovery is the simplest strategy. It also is the only one supported by native DB2 utilities. Refer to Figure 2. With PIT recovery you remove all transactions since a given point in time and then manually reenter the valid work. The desired result is to maintain "Good Transaction 1" and "Good Transaction 2," while removing the "Bad Transactions" from the system.

You must be able to determine a common recovery point for a set of tablespaces. A DB2 QUIESCE works fine, but if that is not available, you will have to determine a point of consistency to be used for recovery.

After the point-in-time recovery, good transactions are missing from the database. If the information is still available, the user could rerun or re-enter "Good Transaction 2". Regardless of the type of recovery to be performed, if the error that caused the recovery to be performed is caught too late, subsequent processing could have occurred using the "bad data". How to deal with these types of problems depends on the nature of the data and the type of updates applied, and needs to be handled on an application by application basis.

UNDO TRANSACTION RECOVERY

The second possibility is to deploy UNDO Transaction Recovery (refer to Figure 3). This is the simplest type of SQL-based Transaction Recovery. It involves generating UNDO SQL statements to reverse the effect of the transactions in error. To generate UNDO SQL, the DB2 log is read to find the data modifications that were applied during a given timeframe and:

- INSERTs are turned into DELETEs
- DELETEs are turned into INSERTs
- UPDATEs are reversed to UPDATE to the old value

To accomplish this transformation a solution is required that understands the DB2 log format and can create the SQL needed to undo the data modifications.

Note that in the case of UNDO Transaction Recovery, the portion of the database that does not need to be recovered remains undisturbed. When undoing erroneous transactions, recovery can be done online without suffering an application or database outage. But, the potential for anomalies causing failures in the UNDO is certainly a consideration. We will discuss this later.

REDO TRANSACTION RECOVERY

The REDO Transaction Recovery strategy is a combination of the first two recovery techniques we have discussed—but with a twist (refer to Figure 4).

Instead of generating SQL for the bad transaction that we want to eliminate, we generate the SQL for the transactions we want to save. Then we do a standard point in time recovery eliminating all the transactions since the recovery point. Finally we re-apply the good transactions captured in the first step.

Unlike the UNDO process, which creates SQL statements that are designed to back out all of the problem transactions, the REDO process creates SQL statements designed to reapply only the valid transactions from a consistent point of recovery to the current time. Since the REDO process does not generate SQL for the problem transactions, performing a recovery and then executing the REDO SQL can restore the tablespace to a state that does not include the problem transactions.

To generate the REDO SQL statements, you will need a solution that can read the DB2 log and create the necessary SQL to redo the data modifications.

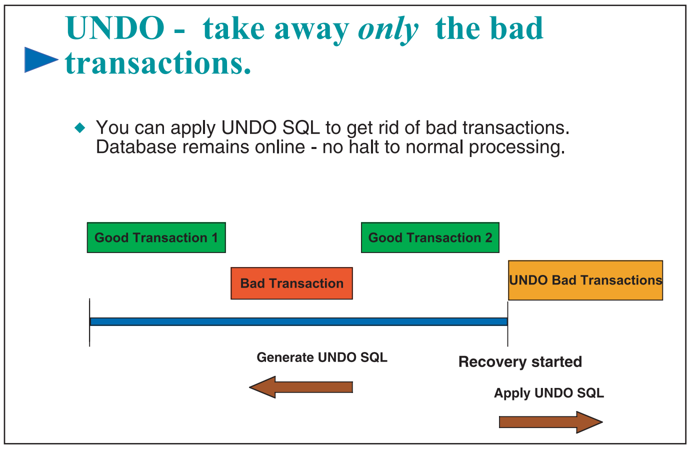
When redoing transactions in an environment where availability is crucial, a PIT recovery can be done and then the application and database can be brought online. The subsequent redoing of the valid transactions to complete the recovery can be accomplished with the data online, thereby reducing application downtime.

CHOOSING THE OPTIMUM RECOVERY STRATEGY

While Transaction Recovery may seem like the answer to all your recovery problems, there are a number of cases where it may be neither possible nor advisable. Consider the following questions when determining if Transaction Recovery is appropriate:

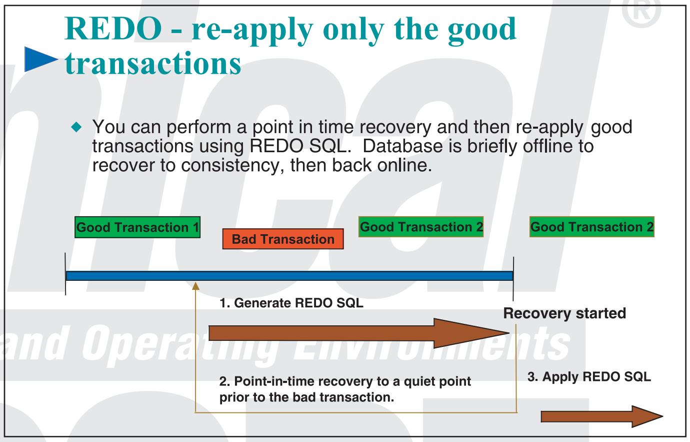
1. **Transaction Identification.** Can all problem transactions be identified? You must be able to actually identify the transactions that will be removed from the database. Can all the work that was originally done be located and redone?
2. **Data Integrity.** Has anyone else updated the rows since the problem occurred? If they have, can you still proceed? Is all required data still available? Recovering after a REORG, LOAD, or mass DELETE may require image copies. Will any other data be lost? If so, can the lost data be identified somehow?

FIGURE 3: UNDO TRANSACTION RECOVERY


UNDO - take away *only* the bad transactions.

- You can apply UNDO SQL to get rid of bad transactions. Database remains online - no halt to normal processing.

FIGURE 4: REDO TRANSACTION RECOVERY


REDO - re-apply only the good transactions

- You can perform a point in time recovery and then re-apply good transactions using REDO SQL. Database is briefly offline to recover to consistency, then back online.

3. **Availability.** How fast can the application become available again? Can you afford to go offline?

These questions boil down to a matter of cost. What is the cost of rework and is it actually possible to determine what would need to be redone? This cost needs to be balanced against the cost of long scans of log data sets to isolate data to redo or undo, and the cost of applying that data using SQL.

The ultimate Transaction Recovery solution should analyze your environment and the transaction(s) needing to be recovered, and recommend which type of Transaction Recovery to perform.

TRANSACTION RECOVERY PLANNING

When planning for Transaction Recovery, you should review your image copy frequency. This frequency will help determine how far back you have to go to get a consistent state for a point-in-time recovery. Also, review your policy on Data Change Capture (DCC). DCC is an option of the DB2 CREATE TABLE statement that causes logged data changes to be written in an expanded format. To implement DCC, alter your tables to include the clause DATA CAPTURE CHANGES. With DCC enabled, generating UNDO and REDO SQL statements will be faster because the SQL generation process need not perform expensive row completion operations. However, your Transaction Recovery solution should be able to function with, or without, DCC enabled.

To effectively plan for, and perform a Transaction Recovery, follow these steps:

1. Identify the problem
2. Identify the transactions causing the problem
3. Identify available recovery points
4. Identify the relative cost of UNDO versus PIT + REDO (Strategy Selection)
5. Identify the cost of performing additional operations to resolve any integrity issues caused by UNDO.
6. Choose the optimal recovery strategy: PIT, PIT+REDO, or UNDO
7. Generate the recovery job(s)
8. Execute the recovery job(s)

TRANSACTION RECOVERY SOLUTIONS

The Transaction Recovery solution should address all of the problems associated with quickly analyzing problem transactions, determining the scope of the recovery, getting information to determine the best recovery method, and managing recovery resources that result in optimal recovery performance. Such a solution should provide:

- A single point-of-control over each stage of the recovery process. A single interface for the entire Transaction Recovery solution minimizes the difficulty of analyzing, generating, and executing the recovery. Switching from one interface to another to accomplish the recovery would introduce needless complexity to an already volatile situation.
- Sophisticated filters to identify the objects that are affected by the transaction. It should be possible to identify the transaction by many different criteria, for example, by correlation ID (batch job), correlation type (batch, CICS, IMS), plan name, authorization ID, tablespace name, table name, object identifier (OBID), column name, update type, or other such qualifying criteria.
- The ability to read and analyze log data to: provide detailed diagnostic information about the selected transactions, diagnose the problem and identify where recovery should begin, find all bad transactions, and generate UNDO and/or REDO SQL.
- Assistance in choosing the optimal recovery method. The solution must understand the tradeoffs in terms of the scope of the transaction, the potential impact on the application, and the time required for each type of recovery to determine whether to perform PIT, UNDO, or REDO Transaction Recovery.
- Automated job generation to eliminate programming errors.
- Ability to restart the process should the recovery fail anywhere along the way.
- A high performance 'capture agent' to quickly gather the required elements of the Transaction Recovery.
- A high-speed capability for applying the SQL statements during the UNDO and REDO processing.

RECOVERY ANALYSIS

Your solution should provide reports on image copy frequency and Data Change Capture impact to help determine your readiness for Transaction Recovery. During an actual Transaction Recovery, your solution must be able to analyze the log for a quiet point. Quiet points are often used as a beginning point of recovery. You still must determine if the quiet point found is the correct starting point for the recovery that you need to perform.

A very important feature of the Transaction Recovery solution is the ability to assist you in choosing between the various recovery methods. Once the appropriate transaction definition and recovery information is collected, a relative work estimate should be prepared for each type of transaction recovery.

BACKOUT INTEGRITY

Maintaining backout integrity is critical. It must be possible to determine the feasibility and impact of performing an UNDO or REDO process. Data may have been changed after the offending application process was run. Data changes of this nature can have an impact on the integrity of the data after a proposed Transaction Recovery.

Maintaining backout integrity is critical. It must be possible to determine the feasibility and impact of performing an UNDO or REDO process.

Analysis of the subsequent activity on the database is critical to determining the feasibility of the recovery. Generally the activity will conform to known application patterns. It is the responsibility of the DBA and the application group together to determine the impact of the recovery on the subsequent work (and vice versa). It may be possible that through judicious selection of the rows to be processed and the columns to include that only the data in error can be processed. For example, if the process in error just updated one or more columns that have no subsequent activity even if other columns on the same rows have been updated it should be possible to limit the generated SQL to just the "bad" columns leaving the rest of the changes alone.

As a rule of thumb, anomaly analysis will yield one of two outcomes; either there is little subsequent activity and a transaction recovery is feasible or there is a significant amount of activity and transaction recovery may be difficult or impossible.

Let's take a look at several examples of anomalies. Consider the following scenario:

- 08:00 Monday—row is added for a new customer.
- 12:00 Monday—new customer enters a large order.
- 18:00 Monday—new customer provides a new shipping address.

Now, think about the havoc that would be caused if the same transaction that updates the shipping address at 18:00 also accidentally clears the total orders column for the customer. Simply backing out the transaction would remove the incorrect data (the total orders column) but it would also cause you to lose valid data (the new shipping address).

The key to this issue is an understanding of the error. Once it is understood that the only column in error is the total orders field, the Transaction Recovery can be made sensitive to that column only.

Let's look at a more complicated case. Consider the following scenario:

- 08:15 Tuesday—row is added for a new customer
- 09:30 Tuesday—new customer fails a credit check and is deleted
- 09:00 Wednesday—new customer reapplies with updated information and passes the credit check

In this scenario, what would happen if the job that added the new customer at 8:15 on Tuesday runs out of sequence, causing numerous other problems? In this scenario, knowledge of the error alone is insufficient to understand the implications of backing it out.

The desired result actually is to leave the new customer alone since the final record in the database is actually correct. If you just rollback the bad job, the insert of the new customer's first record will be turned into a delete that will remove the good record. The only alternative is to scan the log for any other activity on the new customer's record and report it. Once reported, the DBA can analyze the anomalies and hopefully make some reasonable choices on how to proceed. The bottom line is that the Transaction Recovery process needs to take this type of scenario into account, or good data may be lost.

Let's look at another case:

- 13:00 Monday—employee receives a pay increase.
- 23:00 Tuesday—employee's increase is updated in the PAYROLL table.
- 12:00 Thursday—employee gets the promotion associated with the pay increase and it is updated into the EMPLOYEE table.

What if the payroll clerk mis-entered some data on Tuesday night when the employee's increase was entered into the PAYROLL table and the error is not discovered until Thursday afternoon?

The timeliness of problem recognition is crucial. The sooner you recognize a problem, the more feasible Transaction Recovery becomes. Had the problem with the clerk been recognized on Wednesday prior to the next payroll run, the problem would have been manageable. By waiting until Thursday, the possibility of accurately backing out the bad clerk's data has decreased significantly. Even if it turns out to be possible, a great deal of manual effort most likely will be required to select the set of data to rollback.

If you simply rollback the clerk's work, both the employee's pay and promotion will be lost. You may have planned on re-entering the transactions from Tuesday at 23:00, but did you plan on re-entering all the other subsequent transactions (such as those from Thursday at 12:00)?

Let's look at a fourth and final case:

- 14:00 Tuesday—customer submits a loan application
- 23:00 Wednesday—clerk enters the customer's loan application into the system
- 09:00 Thursday—loan officer uses the information to grant the customer's loan

Consider the impact of an error when the loan application is entered into the system on Wednesday. If the information is mis-entered, users who subsequently rely upon the data could make bad decisions. For example, approving a loan for someone who only qualifies because of incorrect data.

This example illustrates that regardless of the amount of information that is obtained automatically, the human element can introduce complications. This does not invalidate the usage of Transaction Recovery to correct the problem, because indeed, any type of simple data recovery cannot resolve a problem with a human. That is, if the customer already has spent the loan money, no type of recovery can get it back!

APPLYING THE SQL FOR TRANSACTION RECOVERY

Speed is critical for Transaction Recovery. Your Transaction Recovery solution should use a multi-tasked SQL apply process that efficiently distributes work among multiple streams for parallel processing. The apply process must be restartable should it fail. Workload balancing by table and partition with respect for referential integrity constraints and table group should be supported. The high performance apply should give you the ability to convert from dynamic SQL to static SQL 'on the fly' to enhance performance. Other desirable features include the ability to process very large volumes of transactions and the option to apply changes online or in batch.

The apply process should have the capability to react to anomalies found during the processing of the SQL. A minimum set of capabilities would include:

- Retry on deadlock situations.
- The ability to log and defer SQL statements with problems (e.g. inserts where the key is already in the table, updates where the column values do not match, etc.).
- The ability to ignore any errors and just continue processing.

SUMMARY

Applications are prone to all types of problems, bugs, and errors. Therefore, Transaction Recovery has become a critical need in any complete recovery toolbox, especially for e-businesses. The Transaction Recovery solution chosen must provide:

- Powerful diagnostic features for problem identification
- Automated assistance in choosing optimal recovery method
- Integrated Transaction Recovery analysis, generation, and apply features
- Features that provide speed, manageability, and accuracy

And with a Transaction Recovery solution in your arsenal, you just might be able to deliver the availability required to help transform your company into an e-business.

Craig S. Mullins is Director of Technology Planning for BMC Software. Craig has close to two decades of experience in all facets of database systems development having worked with DB2 since Version 1. He also has experience using Oracle, Sybase, and Microsoft SQL Server. Craig is the author of the book DB2 Developer's Guide, which contains over 1200 pages of in-depth technical information on Version 8 of DB2 for z/OS. Craig can be reached via his web site at http://www.craigsmullins.com.