HPE Shadowbase Data Recovery Software
Shadowbase REDO and Shadowbase UNDO

A Gravic, Inc. White Paper

Executive Summary

The HPE Shadowbase product suite distributes real-time critical data and information to target databases and to application environments throughout the enterprise. Uses for HPE Shadowbase products include achieving high or continuous availability by synchronizing the databases of active or passive redundant systems, integrating applications, feeding data warehouses and real-time business intelligence facilities, and driving extract, transform, and load (ETL) utilities.

In today's online economy, a company's data is, for the most part, stored in databases in its IT datacenters. The information stored in these databases is fundamental to the operation of the IT applications that drive the decisions that management makes and the services that the company provides to its stakeholders – employees, customers, vendors, regulators, and shareholders.

Erroneous information in a company's databases can wreak havoc on the recipients of application services, and to the operations of the company (significant costs may be incurred, audit compliance may be compromised, embarrassment to the organization, etc.). Hence, if a database becomes corrupted, then it is important to be able to restore it quickly and completely to a consistent (and correct) state to minimize the impact of erroneous data and to restore correct business operations. This capability is provided by the ***HPE Shadowbase Data Recovery Software*** products, comprised of ***HPE Shadowbase REDO*** and ***HPE Shadowbase UNDO***.

Shadowbase REDO maintains a REDO Queue of all changes that were made to a database. If a new version of an application will be run, if a new database version will be deployed, or if some other system change will be made, then before executing the system modification a copy of the database is created and saved. If problems with the system changes result in database corruption, the saved database copy can be quickly restored and the database changes selectively re-applied using Shadowbase REDO. Shadowbase REDO applies only the valid database changes from its REDO Queue to the saved copy in order to roll it forward to a current and correct state. A major benefit of Shadowbase REDO is that if a lot of database corruption has occurred, then it is very fast to recover the database to a known consistent and correct state.

Shadowbase UNDO approaches the repair of a corrupted database from the opposite direction of Shadowbase REDO. After corruption occurs, rather than rolling forward an earlier database copy to a current correct state by applying valid data updates made since the copy was taken, Shadowbase UNDO starts with the current online database, and undoes the corrupting database changes while retaining correct database changes. The database is thus selectively restored (rolled back) to a known, consistent, and current state. A major benefit of Shadowbase UNDO is that rollback of corrupted data can be accomplished while the application continues its processing functions and the database remains online (i.e., no application service outage occurs).

With HPE Shadowbase Data Recovery Software products, companies can safely restore corrupted databases with a minimum of downtime and a maximum of confidence, eliminating the potential costs and risks associated with corrupted data. Hewlett Packard Enterprise globally sells and supports Shadowbase solutions under the name *HPE Shadowbase*. For more information, please contact your local HPE Shadowbase representative or visit our website.

Table of Contents

Executive Summary **2**

Removing Database Corruption with HPE Shadowbase Data Recovery Software **4**

The HPE Shadowbase Data Replication Engine **4**
The Basic HPE Shadowbase Data Replication Engine 5
The HPE Shadowbase Data Replication Engine with Queuing 5

HPE Shadowbase REDO **6**
System Upgrades Can Cause Database Corruption 6
Current Techniques for Roll Forward 7
Backups *7*
Split Mirrors *7*
HPE Shadowbase REDO Roll Forward of a Corrupted Database 8
HPE Shadowbase REDO Roll Forward of a Corrupted Target Database 9

HPE Shadowbase UNDO **9**
HPE Shadowbase UNDO Rollback of a Corrupted Database 10
HPE Shadowbase UNDO Reverse Operations 11
An HPE Shadowbase UNDO Example 11
Preserving Subsequent Changes 12
Undoing DDL Changes 13
HPE Shadowbase UNDO Rollback of a Corrupted Target Database 13

Summary **14**

International Partner Information **16**

Gravic, Inc. Contact Information **16**

Table of Figures

Figure 1 – Repairing Database Corruption with HPE Shadowbase REDO and HPE Shadowbase UNDO 4
Figure 2 – Basic HPE Shadowbase Data Replication Engine 5
Figure 3 – HPE Shadowbase Data Replication Engine with Queuing 6
Figure 4 – HPE Shadowbase REDO Roll Forward 8
Figure 5 – HPE Shadowbase UNDO Rollback 10
Figure 6 – A Database Restore Using HPE Shadowbase UNDO 11
Figure 7 – HPE Shadowbase UNDO Protecting Subsequent Changes 13
Figure 8 – Undoing Replicated Corruption 14

HPE Shadowbase Data Recovery Software
HPE Shadowbase REDO and HPE Shadowbase UNDO

A company's data is its lifeblood. Statistics show that 93% of companies that have lost their data file for bankruptcy within a year. In today's online economy, a company's data is, for the most part, stored in databases in its IT datacenters. The information stored in these databases is fundamental to the operation of the IT applications that drive the decisions that management makes and the services that the company provides to its stakeholders – its employees, its customers, its vendors, its regulators, and its shareholders.

Erroneous information in a company's databases can wreak havoc on the recipients of application services, and to the operations of the company (significant costs may be incurred, audit compliance may be compromised, corporate reputation can be damaged, etc.). Credit cards cancelled in error prevent cardholders from making purchases. Incorrect banking account balances can cause unintended overdrafts or credit denials. Incorrect stock trades may lose thousands or even millions of dollars. Erroneous medical records can cause improper treatment of patients and, in extreme cases, even death. Degradation of services range from severe disruptions (such as these), to the mundane, such as providing a theatergoer with the wrong times for a movie.

Removing Database Corruption with HPE Shadowbase Data Recovery Software

Errors can be introduced into a database by a faulty application, by user error, by the action of a system administrator, employee malfeasance, or by some other entity. It is imperative that such erroneous changes be removed quickly and efficiently from a corrupted database so that the database can be restored to a correct and consistent state, thereby restoring accurate business operations. This is the role of the *HPE Shadowbase Data Recovery Software* products, comprising *HPE Shadowbase REDO* and *HPE Shadowbase UNDO*.

As shown in Figure 1a, Shadowbase REDO *rolls forward* only changes known to be correct, as it brings an earlier saved (and correct) version of a database to a current correct version. In so doing, it eliminates the effects of erroneous changes that were made since the earlier saved version was taken, and that have corrupted the current version of the database.

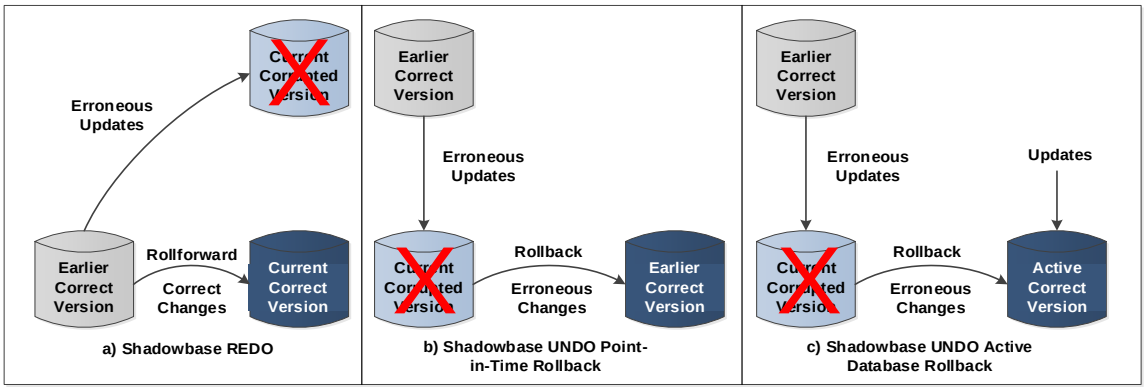

Figure 1 – Repairing Database Corruption with HPE Shadowbase REDO and HPE Shadowbase UNDO

Shadowbase UNDO *rolls back* all incorrect changes to a corrupted database, in order to recreate a correct version of it at some previous point in time, known as "Point-in-Time Rollback"
(Figure 1b). Alternatively, Shadowbase UNDO can rollback all incorrect changes while the database remains online and continues to be updated by applications, to restore the database to its current and correct state, known as "Active Database Rollback" (Figure 1c). A description of the architecture and use of Shadowbase REDO and Shadowbase UNDO follows.

The HPE Shadowbase Data Replication Engine

Both Shadowbase REDO and Shadowbase UNDO rely upon the Shadowbase data replication engine as their underlying foundation. The purpose of this engine is to keep a target database in close synchronization with a source database. The engine sends source database updates as they occur in real-time to the target system and applies them immediately to the target database. Shadowbase solutions are heterogeneous and support a broad range of source and target platforms, operating systems, and databases.

The Basic HPE Shadowbase Data Replication Engine

The basic Shadowbase data replication engine is shown in Figure 2. It comprises a Collector that runs on the source system and a Consumer that runs on the target system. The Collector follows changes being made to the source database via a type of change log that records every source database change as it is made. The change log is often the transaction log maintained by a transaction manager for purposes of reconstructing a database if it becomes corrupted or lost. Examples of transaction logs are the TMF Audit Trail in HPE NonStop systems and the Redo log in Oracle databases.

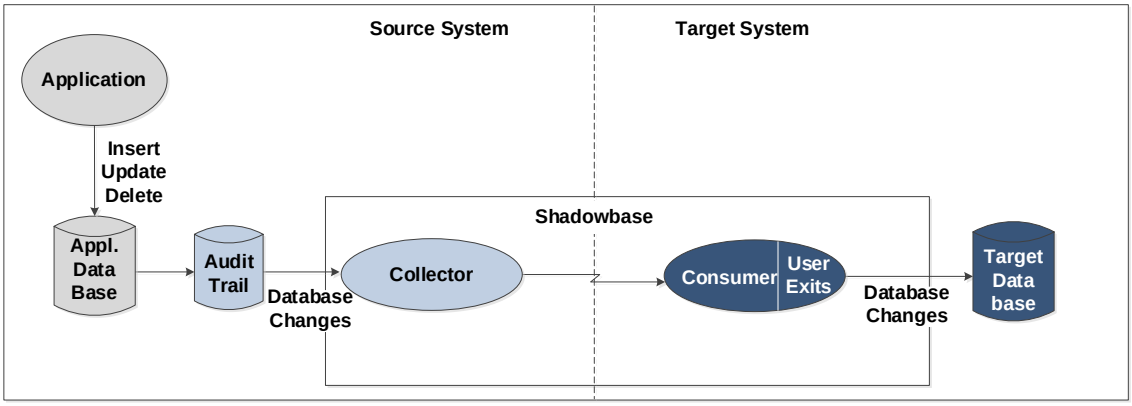

Figure 2 – Basic HPE Shadowbase Data Replication Engine

If a transaction log is not available, the Shadowbase Collector can be driven by an application-created change log or by database triggers. Regardless, as soon as the Collector reads a source database change, it sends the change through a communication channel to the Consumer on the target system. The Consumer then applies the change to the target database. The Consumer and Collector support scripts and embedded user exits that allow database changes to be filtered, reformatted, and enhanced as necessary to meet the needs of the target database.

In this basic configuration, replication of an update does not complete until the Consumer has applied the database update to the target database. Applying random updates to a database is a lengthy process. To achieve low latency and high replication-processing throughput, the Shadowbase engine can be multithreaded by configuring one or more Consumers driven by one or more Collectors. Further, an interposed *Queue Manager* (QMGR) can be used. (The QMGR is discussed further in the next section.)

The HPE Shadowbase Data Replication Engine with Queuing

An optional configuration of Shadowbase replication can dramatically improve the delivery of data to the target environment. Rather than delivering database updates directly to the Consumer for writing into the target database, instead they are delivered to a queue resident on the target system, as shown in Figure 3. By inserting a queue into the replication path, the delivery of data to the target system is divorced from the replaying of the data to the target database. The intermediate queue accepts the data changes and holds them until the Consumer can apply them to the target database. It is much faster to write the data updates into a sequential queue file, than it is to do random writes into the target database; therefore, the use of the intermediate queue file dramatically improves data replication throughput and further reduces latency.

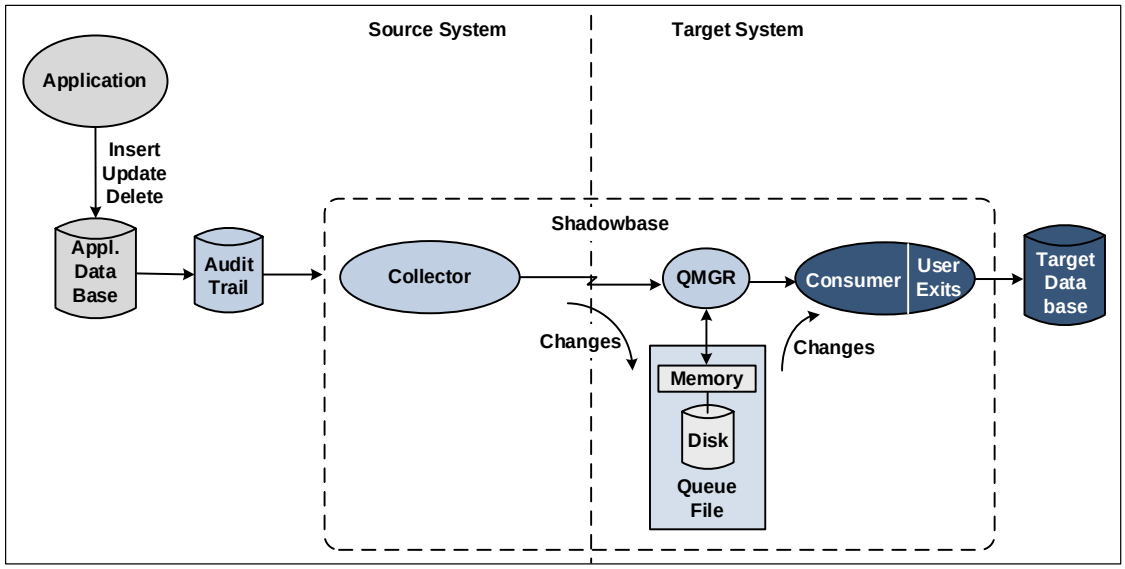

Figure 3 – HPE Shadowbase Data Replication Engine with Queuing

The Shadowbase queuing mechanism works as follows. With reference to Figure 2, the application makes changes to the application database via inserts, updates, and deletes. These changes are also recorded in the application's change log (for instance, the HPE NonStop TMF Audit Trail).

The Shadowbase Collector on the source system follows the Audit Trail and sends changes to the QMGR on the target system. The QMGR writes the changes into a Queue File on the target system. The Queue File also comprises a memory buffer into which replicated changes are stored while they await processing by the Consumer.

If the Consumer(s) falls behind the replication stream, rather than delaying replication and increasing the possibility of lost data if the source system fails, the changes are held in the Queue File until the Consumers have time to read and apply them to the target database. Thus, replication is delayed only by the time to write changes sequentially to the Queue File, rather than the time required applying changes randomly to the target database. The result is less data left in the replication pipeline should a failure occur because the amount of database updates that could be lost is greatly reduced.

The use of the Queue File has a number of significant benefits. The Queue File is also the fundamental architectural construct that enables the functionality of Shadowbase REDO and Shadowbase UNDO.

HPE Shadowbase REDO

System Upgrades Can Cause Database Corruption

It is often necessary to make upgrades to the functionality or the capacity of an existing computer system. Upgrades may include changing application versions, adding processors or disks for expanded capacity, changing platforms, or modifying the database structure to accommodate new functionality. When such a major change to a system's infrastructure will be made, there is always the concern that problems will be encountered after the upgrade occurs and that the system will have to be returned to its initial state (a procedure often referred to as *failback* or *fallback*). One such potential problem is data corruption. A remedy for this problem is to fallback to a known good copy of the database, and then reapply only those transactions that did not contribute to the corruption, thereby bringing the database into a current and correct state before resuming the application. This remedy is called *roll forward* and is the basis for the REDO approach.

Current Techniques for Roll Forward

Backups

Several ways are being used today to restore a corrupted database using a roll forward approach. One way is to make a copy of the database on magnetic tape or disk prior to initiating processing with the upgraded system. If problems occur, the database is loaded with the database copy to restore it to its state prior to the upgrade, and a change log is used to roll forward the changes that were made during processing by the upgraded system.

This technique presents some significant challenges. One challenge is the amount of data that needs to be processed to perform the restore operation. Typically, the entire database (or perhaps just a set of files or tables) has to be restored to the prior backup point even if only a small amount of the database was subsequently corrupted. This restoration can involve the recovery of massive amounts of data from the backup medium and can require access to hundreds or thousands of tapes or disk packs.

Another challenge is the time it takes to copy and to restore a large database from magnetic tape or disk. Many application databases today are massive, often measuring several terabytes (trillions of bytes) in size. Even backing up and restoring from high-speed disk rather than from magnetic tape can take hours if not days. During this recovery time, the application is typically down and normal business operations cannot proceed if the data required to perform those services is corrupted.

Another problem is that the roll forward capability provided by many transaction managers does not allow selective roll forward. All of the changes that are in the change log are applied to the database to bring it to a current state. This capability can result in the transaction manager reapplying the corrupting changes, causing the newly restored database to also become corrupted, and defeating the whole purpose. It is also possible that the roll forward will be unsuccessful, as some of the database changes made during the subsequent period may have depended upon a new data schema and hence cannot be applied to the database copy. If this is the case, the roll forward operation is aborted by the transaction manager, and cannot be completed.

Split Mirrors

The time taken to restore the database copy that is needed for the recovery operation can be improved via another recovery method. In fault-tolerant systems such as HPE NonStop servers, the disks are mirrored. Every logical disk actually comprises two physical disks, and all updates are simultaneously applied to the two disks (the mirrored pair). In this instance, prior to running the upgraded system, the mirrors are split. One mirror is used by the upgraded application, and the other mirror is saved as the pre-upgrade copy as a “fast-recovery” version of the database. Splitting a mirror and attaching the application to one of the mirrors is a relatively fast operation. If desired, a second mirror can be attached to the new database and revived so that the application is once again running against a mirrored database during the upgrade trial. This method helps protect the upgraded system from subsequent media failure that may cause a system outage.

If problems are created by the upgraded system, the recovery process involves reverting to the system infrastructure prior to the upgrade and mounting the saved mirror of the database. Thus, the time to back up the initial version of the database and then to restore it in the event of a problem is very small. The database can be re-mirrored (revived) while the application is running so that the database copy and restoration activity do not significantly impact application availability.

However, the saved mirror typically cannot be rolled forward because the data change queue is on the mirror that has been abandoned and is not generally accessible by the transaction manager. Even if the transaction manager could access the change queue, the same issues as discussed above still apply. Since the new database schema may be different from the restored database schema, the transaction manager typically could not selectively roll forward the change queue updates. Thus, the application database cannot be rolled forward to its current, correct state using this method. All of these challenges are resolved with Shadowbase REDO.

HPE Shadowbase REDO Roll Forward of a Corrupted Database

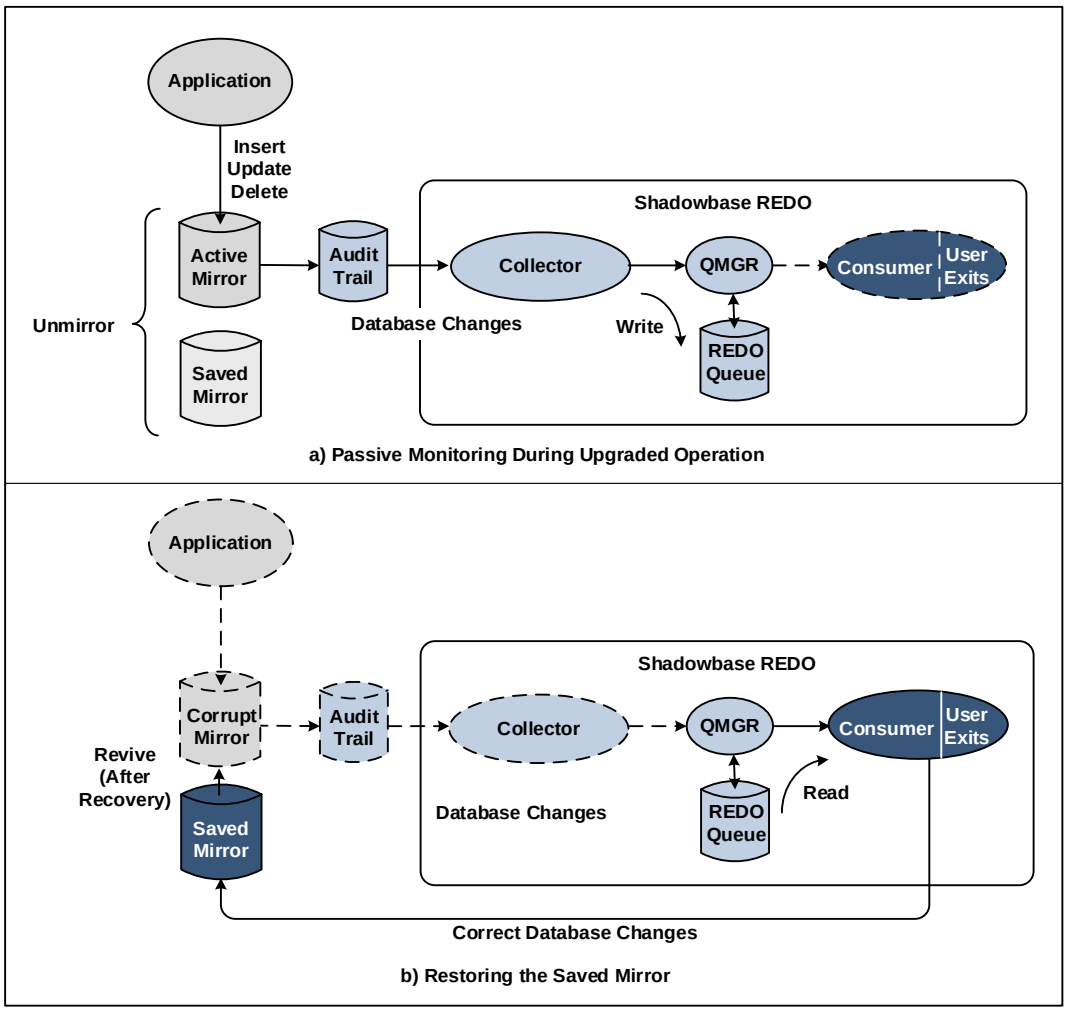

Figure 4 – HPE Shadowbase REDO Roll Forward

The Shadowbase REDO engine is shown in Figure 4. It has a similar architecture to that for the Shadowbase engine with queuing, shown in Figure 3. Note that the Queue File is used as the change data source, and is shown as the "REDO Queue" in the figure.

When the database of a system is undergoing an upgrade under protection with Shadowbase REDO, the first step is to save a fast recovery copy of the application database, for example by un-mirroring the application database, as discussed above, and shown in Figure 4a. Using the database mirrors, one side of the mirror (the saved mirror) is saved for potential database recovery in the event of a problem, the other half (the active mirror) serves as the active database for the upgraded application. A new mirror can be added to re-mirror the active database to provide disk media fault tolerance.

The next step is to perform the required system upgrade. The active mirror now reflects the upgraded database. The newly upgraded environment is then started. Shadowbase REDO is configured and running when the upgraded system starts processing.

As the upgraded system runs, updates made to the application database (that is, to the active mirror) are saved in the Audit Trail. The Shadowbase REDO Collector follows the Audit Trail and captures the REDO data by sending each change to the QMGR, which writes the change into its REDO Queue. If the user determines that the updated system is running successfully and is deemed correct and that no recovery operation will be needed, then the activities of Shadowbase REDO can be terminated and the contents of its REDO Queue can be deleted. However, if a problem should arise, the database is rolled back by mounting the saved mirror copy, as shown in Figure 4b. In this case, update activity for the application is stopped, though reads of the database may still be allowed.

The QMGR in Shadowbase REDO reads the changes from its REDO Queue in forward-time order. Any invalid or undesirable changes, such as to nonexistent tables, can be filtered out by the Consumer, and valid changes are written to the saved mirror to bring it into a current (minus the corrupted changes) and correct state. For many changes, Shadowbase replication can be configured to handle the change through configuration parameters. For some particularly complex changes Shadowbase replication supports adding additional business logic (called User Exits) into the Consumer processing logic path to direct the Shadowbase engine on the actions to take during recovery. If desired, Shadowbase REDO can set aside invalid changes for later processing, such as adding them to the active database via a SQL JOIN operation after the appropriate schema changes are made to the database.

Upon emptying the REDO Queue and rolling back any other necessary changes (for example, a bad application), normal processing can resume from the point of the last-known good transaction. At this time, the corrupted mirror can be revived by copying the now-current database from the saved mirror to the corrupted mirror. Typically, reviving can be accomplished while the application is running.

Thus, Shadowbase REDO solves many of the problems of existing methods for correcting a corrupted database using roll forward:

- A backup copy of the database can be rapidly created by un-mirroring a mirrored pair.
- The backup copy can be restored rapidly by using the mirrored copy.
- Most importantly, data updates are filtered to post only correct changes to the restored saved database copy.
- The roll forward capabilities of Shadowbase REDO do not depend upon a copy of the transaction log, which may be lost on the abandoned old active copy.
- Shadowbase REDO will not terminate if the database schema does not support an update. The update is simply ignored and can be processed later through other means.

HPE Shadowbase REDO Roll Forward of a Corrupted Target Database

The use of Shadowbase REDO to restore a corrupted application database, as described above, can be extended to the restoration of a corrupted target database in a replication environment. In this instance, the application is typically running on a source system and is updating a source database. Changes made to the source database are replicated in real-time by Shadowbase replication to a target database running on a target system, as shown in Figure 4. All changes are replicated, including corrupt changes. Therefore, if the source database becomes corrupted, that corruption is passed to the target database via the replication process.

In order to restore a corrupted target database with Shadowbase REDO, a copy of the target database must have been saved prior to running the modified or upgraded environment, usually by splitting the mirrored database for source system REDO. To roll forward the target database, the saved target mirror is made active. Thereafter, rolling forward changes on the source database will cause those changes to be replicated to the target database, thus restoring it to the same consistent state as the source database as the REDO process progresses.

Note that the REDO approach is a procedural approach to restoring a corrupted database to a known-consistent initial state, then applying selected transactions against it, to “roll it forward” and preserve the newly created data. As discussed above, it requires an outage of the application/database environment while the REDO operation occurs, and for this reason, it is inferior to the next utility discussed for data recovery, Shadowbase UNDO.

HPE Shadowbase UNDO

Shadowbase UNDO approaches the repair of a corrupted database from the opposite direction of Shadowbase REDO. Rather than rolling forward an earlier database copy to a current correct state by applying valid data updates made since the copy was taken, Shadowbase UNDO starts with the current online database, and undoes the corrupt database changes (by rolling them back) while retaining correct database changes. The database is thus restored to a known, consistent, and current state.

A unique feature of Shadowbase UNDO is that online business services can continue to actively perform their processing functions and update the database while the UNDO process is in progress. Corrupted data objects

will be returned to their correct state prior to the corruption. However, if a data object was properly updated following its corruption, the new correct state can be retained.

Shadowbase UNDO runs the Shadowbase replication engine with queuing on the system whose database needs to be protected or restored, as shown in Figure 5. This configuration is essentially the same one as shown in Figure 4 except that the QMGR, now known as the *UNDO Queue*, is modified as described below to support the UNDO function.

HPE Shadowbase UNDO Rollback of a Corrupted Database

Shadowbase UNDO can be configured permanently on the application system, or it may be configured only when it is needed to repair a corrupted database. If it is configured permanently, the Shadowbase replication engine is installed (Figure 5) and monitors via its Collector the changes that are added to the Audit Trail. The Shadowbase engine sends the changes to the QMGR, which writes them into its UNDO Queue. The changes in the UNDO Queue are available in the event that Shadowbase UNDO needs to be invoked to correct a corrupted database. However, in this case, Shadowbase UNDO is essentially passive during normal application processing.

If instead, Shadowbase UNDO was configured after database corruption occurred, then its Collector will read the Audit Trail from some point prior to the corruption up until some point after the corruption and will send these database changes to the QMGR, which will write the changes to the UNDO Queue.

In either event, when database corruption occurs, Shadowbase UNDO can prepare a report based on the data in its UNDO Queue to provide guidance as to when the corruption began, when it ended, what caused the corruption, and what part of the database is affected. Armed with this information, the system administrator can specify to Shadowbase UNDO the begin and end times of the corruption, or alternatively, the beginning and ending transactions, or change log positions over which the corruption took place. Knowing the parts of the database that were affected or the time range over which the corruption occurred, and optionally knowing the source of the corruption, the system administrator can specify the transactions that should be undone according to a variety of identifying information that is known to the change log (e.g., event time, file name, table name, and more).

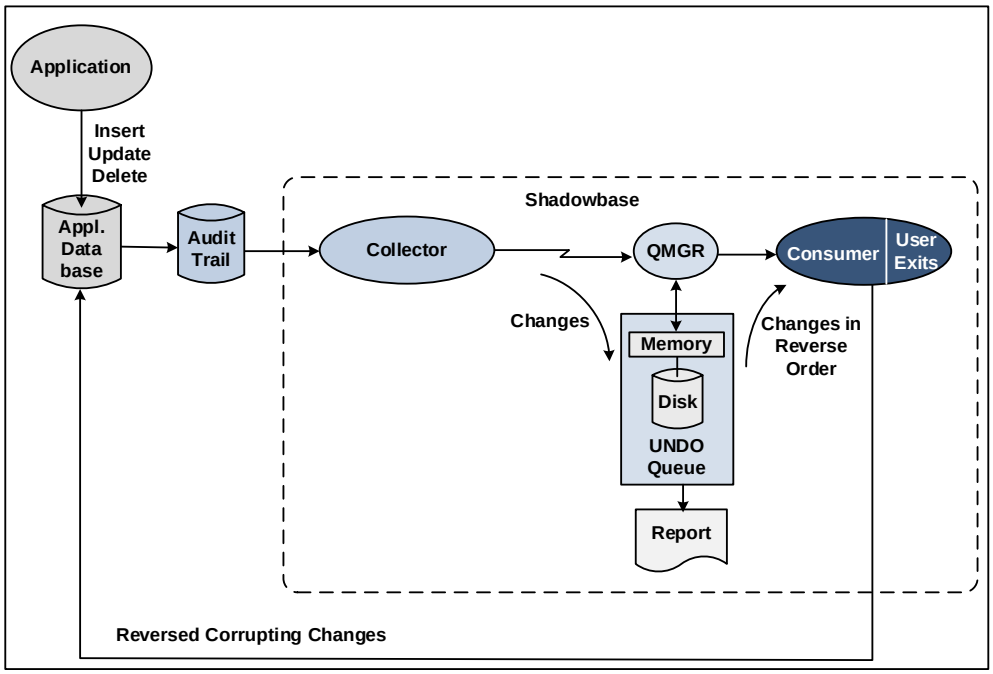

Figure 5 – HPE Shadowbase UNDO Rollback

Once given the required parameters (a “restore specification”), Shadowbase UNDO begins its recovery processing. The QMGR reads database change events from its UNDO Queue in reverse time order (that is, the most recent changes are read first). It filters out the valid events that are of no interest according to the restore specification. It also filters out change events that are within the scope of aborted transactions.

As it reads each change to be rolled back, the QMGR converts the change into a reverse operation to undo its effect on the database (as described in Figure 5). It sends these undo events to the Consumer, which applies them to the application database in order to return the database to its correct state before the corruption occurred.

HPE Shadowbase UNDO Reverse Operations

As the QMGR obtains the pertinent database changes that must be reversed from the UNDO Queue, it translates them into the database operations that will undo the effect of the original change. These undo operations are as follows:

Original Change (Original Order)	UNDO Change (Replay Order)	Effect
Begin transaction (1)	Commit transaction (5)	Commit an UNDO transaction
Insert Row A (2)	Delete Row A (4)	An erroneously inserted row is deleted
Update Row B from x to y (3)	Update Row B from y back to x (3)	An erroneous update is reversed
Delete Row C (4)	Insert Row C (2)	An erroneously deleted row is inserted
Commit transaction (5)	Begin transaction (1)	Begin an UNDO transaction

Table 1 – Undo Changes to Corrupting Updates

Each event in the UNDO Queue contains the before and after images of the row that was changed. The Shadowbase UNDO operations are accomplished by using the before images:

- The before image of an insert is a null row (that is, the row does not exist).
- The before image of an update is the value of the row before the update was made.
- The before image of a delete is the value of the row before the delete was made.

The Shadowbase UNDO changes are composed into a transaction in the reverse order of the corrupting transaction for applying to the application database. The Commit directive of the corrupting transaction becomes the Begin directive for the Shadowbase UNDO transaction. Likewise, the Begin directive of the corrupting transaction becomes the Commit directive for the Shadowbase UNDO transaction.

An HPE Shadowbase UNDO Example

Consider the sequence of events shown in the left half of Figure 6, which shows a corrupted sequence of operations involving Row A in the database. Row A was initially set to the value “w.” The first operation of the corrupted sequence deleted Row A. The next operation inserted Row A with a value of “x.” The last operation changed the contents of Row A from “x” to “y.”

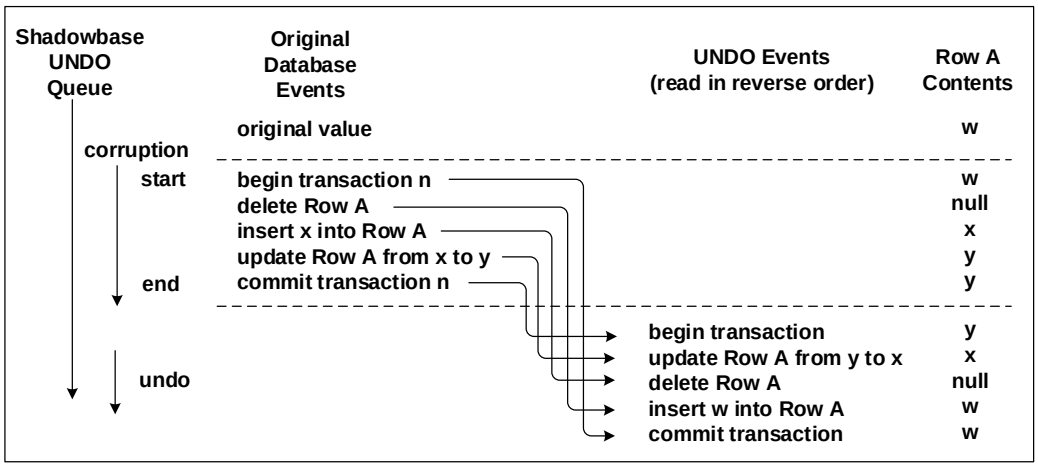

Figure 6 – A Database Restore Using HPE Shadowbase UNDO

This sequence of operations was identified as erroneous by the system administrator, who created a restore specification for Shadowbase UNDO to back out the sequence.

Shadowbase UNDO reads the change operations in the corrupted sequence from the UNDO Queue in reverse order. In this example:

- The first entry that Shadowbase UNDO reads is the last operation of the corrupted sequence, the commit directive. Shadowbase UNDO converts it to a begin-transaction directive.
- The next entry is the "update Row A from x to y" operation. The before image of this entry is a value of "x" for Row A, so Shadowbase UNDO updates Row A to "x."
- The next entry is the insert operation. Therefore, Shadowbase UNDO deletes Row A.
- The fourth entry is the "delete Row A" operation, which before image is "w," so Shadowbase UNDO inserts Row A with a value of "w."
- Finally, the begin transaction event is read and is converted to a commit transaction to complete the Shadowbase UNDO transaction that is applied to the application database.

The Shadowbase UNDO sequence has thus returned the database to its state before the corruption took place – a Row A value of "w."

Preserving Subsequent Changes

A significant feature of Shadowbase UNDO is that it can reverse corruption of a database while the application remains online and continues to update the database. This feature is particularly useful if the corruption is isolated to a subset of the database or, for example, a particular set of users, and it would be unacceptable to take the database offline in order to recover as that would cause a service outage.

Shadowbase UNDO can detect changes that occurred to a corrupted data object following the corruption. If a corrupted data object was subsequently modified, its value can optionally be considered as valid, and left intact. Otherwise, its value is reversed to reflect its value prior to the corruption.

For instance, consider the example in Figure 6, under the condition that the application continues to run following the corruptive events. If Row A has been further modified following the corrupted sequence, the corrupted value may be overwritten by a valid value. The procedure described above will overwrite the valid value with the original value prior to the corruption. The new, valid value will be lost. However, Shadowbase UNDO provides the means to both restore the database to a previous uncorrupted state, and to preserve new values that have been entered into the database since the corruption occurred.

Consider the sequence of events shown in the left half of Figure 6, which shows a corrupted sequence of operations involving Row A in the database. Row A was initially set to the value "w." The first operation of the corrupted sequence deleted Row A. The next operation inserted Row A with a value of "x." The last operation changed the contents of Row A from "x" to "y."

Figure 6 reflects both the problem and the solution. The example in Figure 7 is extended, and a second, valid, transaction is executed that updates Row A from its corrupted value "y" to a correct value "z." The result of the Shadowbase UNDO operation, as described above, will result in a value for Row A of "w." This value was correct before the corruption occurred, but the new, correct value of "z" has been lost.

A subsequent valid update can be detected during the Shadowbase UNDO operation by comparing the after image of the last corrupting database operation to the current value of that row in the database. If they are different, the row has been subsequently modified and should not be changed.

With reference to Figure 7, the last operation in the corrupted transaction is "update Row A from x to y." Shadowbase UNDO desires to undo this operation with "update Row A from y to x." If there has been no subsequent operation on Row A, its database value will still be "y," which is the value of the after image of the corrupted operation. Therefore, the row has been corrupted and can be corrected. However, Row A has been subsequently modified to a valid value of "z" following the corruption. This value is different from the after image ("y") of the corrupted update. Therefore, a subsequent operation has been performed on Row A, and no Shadowbase UNDO operations should be performed if the intent is to protect subsequent operations. The result is that Row A is left with its ultimate correct value of "z" rather than its original pre-corruption value of "w."

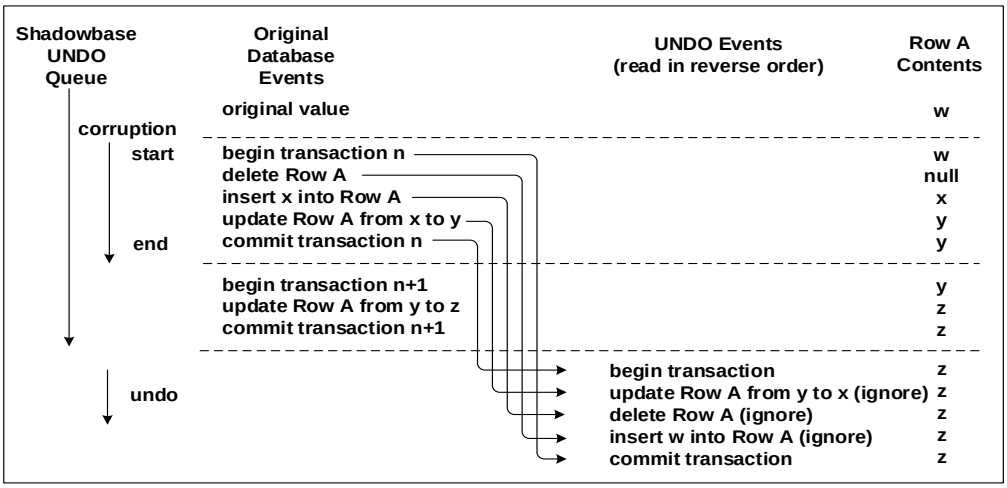

Figure 7 – HPE Shadowbase UNDO Protecting Subsequent Changes

One issue occurs with this algorithm if a subsequent valid change sets the value of the data object to the same value that it had at the end of the corruption event as the now-correct value of the data object will be mistaken for the ending corrupted value. In this case, the data object will be reset to its value prior to the corruption, thus losing the valid value. This problem can be resolved by using version information in the rows. The version information can be, for instance, a timestamp or a row version number. When beginning the Shadowbase UNDO process on a data object, if the current version of the row is later than the version of the reversing operation as contained in the UNDO Queue, the row has been subsequently updated and should not be rolled back.

Another issue with this approach can occur if the subsequent update ("z" in the example above) was determined *relative* to the corrupted value ("y" in the example above) instead of in an absolute sense. For example, if "y" was computed by adding +1 to "x," and then if "z" was computed by adding +1 to "y," then leaving "z" in place as "x+2" will itself be incorrect as it should instead be set to "x+1." In this case, a Shadowbase User Exit can be added to the UNDO logic processing in the Consumer to track the correct value to use to properly reverse the corruption during the UNDO sequence.

Undoing DDL Changes

A limitation of the Shadowbase UNDO method is that not all database changes are capable of being rolled back. For instance, in HPE NonStop systems, if a column is added to a table, the file management system does not allow this column to be deleted. Shadowbase UNDO can rollback all data manipulation language (DML) changes and some data description language (DDL) changes, but may not be able to rollback all DDL changes. In this instance, the operations can be reported and skipped, or a more complex User Exit can be developed to handle the processing. Alternatively, this limitation is resolved by Shadowbase REDO, as described earlier.

HPE Shadowbase UNDO Rollback of a Corrupted Target Database

The use of Shadowbase UNDO to restore a corrupted application database, as described above, can be extended to the restoration of a corrupted target database in a replication environment. In this case, the application is running on a source system and is updating a source database. Changes made to the source database are replicated in real-time by Shadowbase replication to a target database running on a target system, as shown in Figure 8. All changes are replicated, including corrupting changes. Therefore, if the source database becomes corrupted, that corruption is passed to the target database.

By the same token, changes applied to the source database by Shadowbase UNDO to correct the source database corruption also will be replicated to the target database, thus correcting the target database's corruption.

To simultaneously correct both a source and a target database, two instances of Shadowbase replication must be deployed, as shown in Figure 8. One instance is the normal Shadowbase replication engine used for production to keep the target database synchronized with the source database. The other is Shadowbase UNDO running on the source system.

To correct database corruption, the corrupted range of database updates are loaded from the Audit Trail into the UNDO Queue of Shadowbase UNDO by its Collector and the QMGR. Once this has been completed, the Consumer in Shadowbase UNDO reads the corrupting changes starting with the latest one, reverses their operations as shown in Figure 7, and applies them to the source database to reverse the corrupted change events.

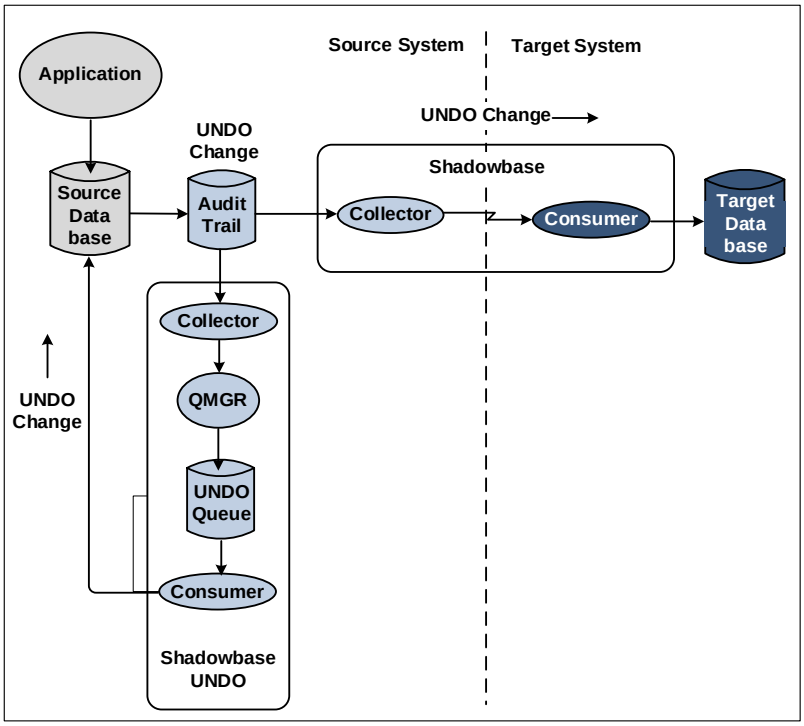

Figure 8 – Undoing Replicated Corruption

As each reverse event is posted to the source database, that event is written to the Audit Trail and is replicated by the Shadowbase replication engine to the target database. Consequently, the target database is corrected and is synchronized with the source database (albeit after a short time delay). Ultimately, both databases are returned to an earlier point in time that is known to be correct, except that valid changes made to the source database following the corrupting events may optionally be preserved at both the source and target databases, as described in the section, “Preserving Subsequent Changes.”

Summary

Database corruption can be caused by programming errors, user errors, system faults, employee malfeasance, and other actions. Erroneous information in a company’s databases can wreak havoc on the recipients of application services, and to the operations of the company (significant costs may be incurred, audit compliance may be compromised, corporate reputation can be damaged, etc.). Hence, if a database becomes corrupted, then it is important to be able to restore it quickly and completely to a consistent state to minimize the impact of erroneous data and to restore correct business operations. This capability is provided by the *HPE Shadowbase Data Recovery Software* products, comprising *HPE Shadowbase REDO* and *HPE Shadowbase UNDO*.

If a new version of an application is run, if a new database version is deployed, or if some other system change is made, a copy of the database before executing the system modification may be created and saved. Shadowbase REDO maintains a REDO Queue of all changes that are subsequently made to the database. So, if problems result in database corruption, Shadowbase REDO reads selected valid database changes from its REDO Queue and applies them to the saved copy in order to quickly roll the database forward to a known current and correct state.

Conversely, by maintaining the UNDO Queue of changes that were made to a database, Shadowbase UNDO follows the UNDO Queue in reverse time order to the initial point of corruption and reverses any corrupting changes that were made, in order to roll the database back to a known current and correct state. The rollback

of corrupted data is accomplished by Shadowbase UNDO while the application remains online, providing business services to users. Correct updates made after the period of corruption are preserved.

Helpful reports are generated from either the UNDO Queue or REDO Queue to aid in determining the period of corruption and the contributing sources. The scope of corrective activity is specified by a user or a system administrator to include a time or transaction range, a list of affected files and tables, and a list of sources of corrupted transactions.

HPE Shadowbase Data Recovery Software products provide the tools needed to quickly and easily recover from the serious effects of corrupted data. The software is flexible and able to meet specific needs, including allowing extension via its powerful User Exit capability.

Depending on the nature of the corruption:

- HPE Shadowbase REDO may be used when significant data corruption has occurred, affecting a large part of the database. However, an application/database outage is required to complete the operation.
- HPE Shadowbase UNDO may be used if the amount of data corruption is small, localized, or service availability must be maintained while recovery is in process.

Whatever the circumstances, with HPE Shadowbase Data Recovery Software products, companies can safely restore corrupted databases with a minimum of downtime and a maximum of confidence, eliminating the potential costs and risks associated with corrupted data.

International Partner Information

Global

Hewlett Packard Enterprise
6280 America Center Drive
San Jose, CA 95002
USA
Tel: +1.800.607.3567
www.hpe.com

Japan

High Availability Systems Co. Ltd
MS Shibaura Bldg.
4-13-23 Shibaura
Minato-ku, Tokyo 108-0023
Japan
Tel: +81 3 5730 8870
Fax: +81 3 5730 8629
www.ha-sys.co.jp

Gravic, Inc. Contact Information

17 General Warren Blvd.
Malvern, PA 19355-1245
USA
Tel: +1.610.647.6250
Fax: +1.610.647.7958
www.shadowbasesoftware.com
Email Sales: shadowbase@gravic.com
Email Support: sbsupport@gravic.com

Hewlett Packard Enterprise Business Partner Information

Hewlett Packard Enterprise directly sells and supports Shadowbase Solutions under the name ***HPE Shadowbase***. For more information, please contact your local HPE account team or visit our website.