**(Introduction to Database Recovery)**

**Database Recovery Management in DBMS** mainly concerns transaction **Recovery System is DBMS** or how to recover the Database in case of system failure. This tutorial will learn and understand the different transaction recovery techniques used in database recovery management in DBMS.

A database is a very huge system with lots of data and transaction. The transaction in the database is executed at each seconds of time and is very critical to the database. If there is any failure or crash while executing the transaction, then it expected that no data is lost. It is necessary to revert the changes of transaction to previously committed point. There are various techniques to recover the data depending on the type of failure or crash.

As we have seen already, each transaction has ACID property. In case of transaction failure or system crash, it should maintain its ACID property. Failing to maintain ACID is the failure of database system. That means, any transaction in the system cannot be left at the stage of its failure. It should either be completed fully or rolled back to the previous consistent state.

Suppose there was a transaction on the Student database to enter the marks of a student in 3 subjects and then to calculate his total. Suppose there is a transaction failure, when 3rd mark has been entered into the table. This transaction cannot be left at this stage because student has marks in two subjects already entered. When the system is recovered and total is calculated, it is calculated based on two subject marks, which is not correct. In this case, either the transaction has to be completed fully to enter the 3rd mark and calculate the total, or remove the marks that have entered already. Either completing the transaction fully or revert the transaction fully brings the database into a consistent state and data will not lead to any miscalculation.

**We can conclude from above example that**

* States of all the executed transaction should be verified. This is necessary to know because if there is any incomplete transaction.
* Any incomplete transaction should either be completed fully or reverted fully to the consistent state of DBMS before the transaction.
* ACID property of transaction should be met in case of failure too.

**We can recover the database in two methods :**

* Log Based Recovery
* Shadow Paging

**Log Based Recovery**

In this method, log of each transaction is maintained in some stable storage, so that in case of any failure, it can be recovered from there to recover the database. But storing the logs should be done before applying the actual transaction on the database.

Every log in this case will have informations like what transaction is being executed, which values have been modified to which value, and state of the transaction. All these log information will be stored in the order of execution.

Suppose there is a transaction to modify the address of a student. Let us see what logs are written for this transaction.

* As soon as transaction is initiated, it writes ‘start’ log.

      <Tn, Start>

* When the transaction modifies the address from ‘Troy’ to ‘Fraser Town’, another log is written to the file.

   <Tn, ADDRESS, ‘Troy’, ‘Fraser Town’>

* When the transaction is completed, it writes another log to indicate end of the transaction.

  <Tn, Commit>

**There are two methods of creating this log files and updating the database**

**Deferred database modification**

In this method, all the logs for the transaction is created and stored into stable storage system first. Once it is stored, the database is updated with changes. In the above example, after all the three log records are created and stored in some storage system, database will be updated with those steps. That means, log is created and stored in stable memory first; then the actual DB changes are done by checking the logs.

**For example, suppose we have transaction T1 which will read the value of X and Y, and update their values as below :**

Let initially values of X and Y be 25 and 30. Then deferred database modification will log all the steps into log file first. It will not update the database as soon as WRITE (X) is logged into log file. It will wait till all the logs for T1 is updated into log file. Once it is done, it will read the log file and update the database. The log file for T1 will look as above.

If there is any crash, then the data is recovered by checking these logs. If there is both <Tn, Start> and <Tn, Commit> for the transaction, then that transaction has to be re-executed. But the crash can happen while updating the log files or may be while updating the database itself.

**Suppose in addition to T1 above we have another transaction T2, which updates the values for Z. Then the transactions and log files will be as below :**

So it is clear that when <T1, Commit> is reached, T1 is updated to DB and T2 is updated to DB, when <T2, Commit> is reached in log file. Suppose the system fails while logging the logs. If it fails at step 3, <T1, Y, 30, 20>, then no need to redo T1. This is because, log file has not updated commit for T1 still. If it fails after logging <T1, Commit>, then T1 has to be re-executed. If it fails after <T2, Commit>, then both T1 and T2 has to be re-executed.

**Immediate database modification**

After creating each log record, database is modified for each step of log entry immediately or it can be updated DB before transaction issues commit. In the above example, database is modified at each step of log entry i.e.; after first log entry, transaction will hit the database to fetch the record, then second log will be entered followed by updating the address, then the third log followed by committing the database changes.

Since here database is updated as soon as log is created, when recovering the DB, two operations – undo and redo has to be performed. An undo operation has to be executed for all the transactions that are not committed during the failure. All uncommitted transaction has to be rolled back. i.e.; if a log file contains <T1, Start>, but no <T1, Commit> then such transactions have to be rolled back or undone. All the committed transactions have to redone during the recovery. That means, these transactions are already executed during the failure and its value has to be retained in the DB. That is if the log file contains <T1, Start> and <T1, Commit> then such transactions have to be re-executed. If undo and redo executed multiple times, effect of undo and redo on the system should be same. It should be result in same result as though it is executed only once. This is because, based on the logs, transactions will be undone or redone. However undo operations are performed first, followed by redo.

Consider the same example of X, Y and Z above. Suppose transaction fails after logging <T1, Y, 30, 20>, then T1 has to be rolled back – undo (T1). i.e.; it should rollback Y to 30 and X to 25. Suppose transaction has failed as soon as <T1, Commit> is executed. Then T1 has to be redone, because log has both start and commit entries. Hence X = 125 and Y = 30. It should not consider old value of X = 125 while re-executing. Old values of X and Y should always remain as 25 and 30 and new values always should be 125 and 20. If the transaction failed after executing <T2, Z, 15, 20>, then T2 has to be undone first to the value Z = 15 and T1 has to be re-executed; X and Y should have new values as 125 and 20 respectively.

**Database systems**, like any other computer system, are subject to failures but the data stored in it must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.

There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.

Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur in the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

* The log is kept on disk start\_transaction(T): This log entry records that transaction T starts the execution.
* read\_item(T, X): This log entry records that transaction T reads the value of database item X.
* write\_item(T, X, old\_value, new\_value): This log entry records that transaction T changes the value of the database item X from old\_value to new\_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
* commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
* abort(T): This records that transaction T has been aborted.
* checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

A transaction T reaches its **commit** point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not **abort** (terminate without completing). Once committed, the transaction is permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, item is searched back in the log for all transactions T that have written a start\_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process

* **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write\_item(T, x, old\_value, new\_value) and setting the value of item x in the database to old-value.There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.
* **Deferred update –** This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm**
* **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.**
* **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.
* **Shadow paging –** It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to original are updated to refer new replacement page.

Some of the backup techniques are as follows :

* **Full database backup –** In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series.
* **Differential backup –** It stores only the data changes that have occurred since last full database backup. When same data has changed many times since last full database backup, a differential backup stores the most recent version of changed data. For this first, we need to restore a full database backup.
* **Transaction log backup –** In this, all events that have occurred in the database, like a record of every single statement executed is backed up. It is the backup of transaction log entries and contains all transaction that had happened to the database. Through this, the database can be recovered to a specific point in time. It is even possible to perform a backup from a transaction log if the data files are destroyed and not even a single committed transaction is lost.

**OTHER REFRENCES**

•      [Data Recovery in DBMS - Data Recovery in SQL (tutorialcup.com)](https://www.tutorialcup.com/dbms/data-recovery.htm#:~:text=Data%20Recovery%20in%20DBMS%20A%20database%20is%20a,then%20it%20expected%20that%20no%20data%20is%20lost.)

•      [Introduction of DBMS (Database Management System) | Set 1 - GeeksforGeeks](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/)

•      [Database Recovery Management in DBMS (computersciencejunction.in)](https://www.computersciencejunction.in/2020/12/21/database-recovery-management-in-dbms/)

. [Database Recovery Techniques in DBMS - GeeksforGeeks](https://www.geeksforgeeks.org/database-recovery-techniques-in-dbms/)

**SUGGESTED BOOK REFERENCES**

* C.J.Date, “An Introduction to DatabaseSystems”, Addison Wesley.
* Thomas M. Connolly, Carolyn & E.Begg,“Database Systems: A Practical Approach to Design, Implementationand Management”, 5/E, University of Paisley, Addison-Wesley.
* Rob,”Database Principal Fundamental Design, Cengage Learning.