Transaction Recovery – Save Points

Database Systems like any other computer system, are subject to failures but the data stored in them 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.

Types of Recovery Techniques Database recovery techniques are used in database management systems (DBMS) to restore a database to a consistent state after a failure or error has occurred. The main goal of recovery techniques is to ensure data integrity and consistency and prevent data loss.

There are mainly two types of recovery techniques used in DBMS

- Rollback/Undo Recovery Technique
- Commit/Redo Recovery Technique
- CheckPoint Recovery Technique

Database recovery techniques ensure data integrity in case of system failures. Understanding how these techniques work is crucial for managing databases effectively.

Rollback/Undo Recovery Technique

The rollback/undo recovery technique is based on the principle of backing out or undoing the effects of a transaction that has not been completed successfully due to a system failure or error. This technique is accomplished by undoing the changes made by the transaction using the log records stored in the transaction log. The transaction log contains a record of all the transactions that have been performed on the database. The system uses the log records to undo the changes made by the failed transaction and restore the database to its previous state.

Commit/Redo Recovery Technique

The commit/redo recovery technique is based on the principle of reapplying the changes made by a transaction that has been completed successfully to the database. This technique is accomplished by using the log records stored in the transaction log to redo the changes made by the transaction that was in progress at the time of the failure or error. The system uses the log records to reapply the changes made by the transaction and restore the database to its most recent consistent state.

Checkpoint Recovery Technique

Checkpoint Recovery is a technique used to improve data integrity and system stability, especially in databases and distributed systems. It entails preserving the system's state at

regular intervals, known as checkpoints, at which all ongoing transactions are either completed or not initiated. This saved state, which includes memory and CPU registers, is kept in stable, non-volatile storage so that it can withstand system crashes. In the event of a breakdown, the system can be restored to the most recent checkpoint, which reduces data loss and downtime. The frequency of checkpoint formation is carefully regulated to decrease system overhead while ensuring that recent data may be restored quickly.

Overall, recovery techniques are essential to ensure data consistency and availability in Database Management System, and each technique has its own advantages and limitations that must be considered in the design of a recovery system.

Database Systems

There are both automatic and non-automatic ways for both, backing up data and recovery from any failure situations. The techniques used to recover lost data due to system crashes, transaction errors, viruses, catastrophic failure, incorrect command execution, etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred updates and immediate updates 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 during 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:** A 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 a 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, the 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 the 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 the original are updated to refer new replacement page.
- **Backward Recovery:** The term ” Rollback ” and ” UNDO ” can also refer to backward recovery. When a backup of the data is not available and previous modifications need to be undone, this technique can be helpful. With the backward recovery method, unused modifications are removed and the database is returned to its prior condition. All adjustments made during the previous traction are reversed during the backward recovery. In other words, it reprocesses valid transactions and undoes the erroneous database updates.
- **Forward Recovery: “** Roll forward “and ” REDO ” refers to forwarding recovery. When a database needs to be updated with all changes verified, this forward recovery technique is helpful. Some failed transactions in this database are applied to the database to roll those modifications forward. In other words, the database is restored using preserved data and valid transactions counted by their past saves. Backup Techniques There are different types of Backup Techniques. Some of them are listed below.

- **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 the last full database backup. When some data has changed many times since the last full database backup, a differential backup stores the most recent version of the 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 transactions 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.

Savepoint

A SAVEPOINT is used to create a checkpoint within a transaction. We can roll back to a specific SAVEPOINT instead of rolling back the entire transaction. This allows us to **undo part** of the transaction **rather than** the entire transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

Example

SAVEPOINT SP1;

//Savepoint created.

DELETE FROM Student WHERE AGE = 20;

//deleted

SAVEPOINT SP2;

//Savepoint created.

Output

Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

output

Explanation:

From the above example **Sample table1**, Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database by keeping Savepoints. Here SP1 is first SAVEPOINT created before deletion. In this example one deletion have taken place. After deletion again SAVEPOINT SP2 is created.

ROLLBACK TO SAVEPOINT

The **ROLLBACK TO SAVEPOINT** command allows us to roll back the transaction to a specific savepoint, effectively undoing changes made after that point.

Syntax:

ROLLBACK TO SAVEPOINT SAVEPOINT_NAME;

Example

Deletion have been taken place, let us assume that we have changed our mind and decided to ROLLBACK to the SAVEPOINT that we identified as SP1 which is before deletion. So, In this case the DELETE operation is undone, and the transaction is returned to the state it was in at the SP1 savepoint.

ROLLBACK TO SP1;

//Rollback completed

Output

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

output

RELEASE SAVEPOINT Command

This command is used to remove a **SAVEPOINT** that we have created. Once a SAVEPOINT has been released, we can no longer use the **ROLLBACK command** to undo transactions performed since the last SAVEPOINT. It is used to initiate a **database transaction** and used to specify characteristics of the transaction that follows.

Syntax:

1. RELEASE SAVEPOINT SAVEPOINT_NAME

   Example

   - Once the savepoint SP2 is released, we can no longer roll back to it.
2. RELEASE SAVEPOINT SP2; -- Release the second savepoint.

Why Use Transactions in Banking?

In this case, without a transaction, you risk scenarios where money is deducted from one account but not added to the other, leaving the system in an inconsistent state. Transactions ensure that such issues are avoided by guaranteeing that both operations succeed or fail together.