DATABASE RECOVERY MANAGEMENT

Database recovery is the process of restoring the database to a correct state in the event of a failure.

Causes of failure in database processes are:

- System crashes due to hardware or software errors, resulting in loss of main memory;
- Media failures, such as head crashes or unreadable media, resulting in the loss of parts of secondary storage;
- Application software errors, such as logical errors in the program that is accessing the database, which cause one or more transactions to fail;
- Natural physical disasters, such as fires, floods, earthquakes, or power failures;
- Carelessness or unintentional destruction of data or facilities by operators or users;
- Sabotage, or intentional corruption or destruction of data, hardware, or software facilities.

Recovery Facilities

A DBMS should provide the following facilities to assist with recovery:

- A backup mechanism, which makes periodic backup copies of the database;
- Logging facilities, which keep track of the current state of transactions and database changes;
- A checkpoint facility, which enables updates to the database that are in progress to be made permanent;
- A recovery manager, which allows the system to restore the database to a consistent state following a failure.

Levels of Backups

1. A full back up of the database of dump of the database.

2. Reference backup of the database in which only the last modifications done on the database are copied.

3. A back-up of the transaction log only this level backup all the transaction log operations that are not reflected in the previous back-up copy of the database. Log contains information about all updates to the database.

The database backup is stored in a secure place usually in a different building and protected against dangers such as fire, theft flood and other potential calamities. Back-up existence guarantees recovery system (hardware/software) failures.

Recovery Protocols

1. Restart Procedures - It assumes that no transactions are accepted until the database has been repaired and includes:

(i) Emergency restart this follows when a system fails without warning e.g. due to power failure.

(ii) Cold restart - The system is restarted from archive when the log and restart file has been corrupted.

(iii) Warm restart - It allows controlled shut down of the system

2. Archiving - Creation of periodic back-ups.

3. Mirroring - 2 complete copies of the database are maintained online in different stable storage devices. It is used in environments with non-stop fault tolerant operations

4. Undo/Redo - It is undoing and re-doing a transaction after failure. The transaction manager keeps an active list and abort list and a commit list. These comprise of transactions that have begun, abort and committed transactions respectively.

5. 2-Phase Commit - It is used in a database system and has 2 phases:

(i) Voting phase - where participants are asked whether or not they are prepared to commit the transaction.

(i) Decision phase where the transaction is committed.

Transaction Management

A transaction is a series of actions carried out by a single user or application program, which must be treated as a single logical unit of work.

It results from the execution of a user program delimited by statements (or function calls) of the form begin transactions and end transactions.

Properties of Transactions

These properties are usually referred to as ACID properties.

(i) Atomicity

It is the all or nothing property.

Either all the operations of the transactions are reflected in the database properly or none are. This means that a transaction is an indivisible unit.

(ii) Consistency

A transaction must transform the database from one consistent state to another consistent state. The DBMS can ensure consistency by enforcing all the constraints that have been specified on the database schema, such as integrity and enterprise constraints.

(iii) Isolation

Transactions execute independently of one another. In other words, the partial effects of incomplete transactions should not be visible to other transactions.

(iv) Durability

The effects of a successfully completed (committed) transaction are permanently recorded in the database and must not be lost because of a subsequent failure. It is the responsibility of the recovery subsystem to ensure durability.

Concurrency Control

Concurrency control is the process of managing simultaneous operations on the database without having them interfere with one another.

There are two main concurrency control techniques that allow transactions to execute safely in parallel subject to certain constraints: locking and timestamp methods.

1. Locking Methods

Locking is a procedure used to control concurrent access to data. A lock guarantees exclusive use of data item to a current transaction. Transaction T1 does not have access to a data item that is currently used by transaction T2. A transaction acquires a lock prior to data access. The lock is released (Unlock) when the transaction is completed so that another transaction can lock the data item for its exclusive use.

The basic rules for locking are:

a) Shared lock If a transaction has a shared lock on a data item, it can read the item but not update it.

b) Exclusive lock If a transaction has an exclusive lock on a data item, it can both read and update the item.

Locks are used in the following way:

- Any transaction that needs to access a data item must first lock the item, requesting a shared lock for read only access or an exclusive lock for both read and write access.
- If the item is not already locked by another transaction, the lock will be granted.
- If the item is currently locked, the DBMS determines whether the request is compatible with the existing lock. If a shared lock is requested on an item that already has a shared lock on it, the request will be granted; otherwise, the transaction must wait until the existing lock is released.
- A transaction continues to hold a lock until it explicitly releases it either during execution or when it terminates (aborts or commits). It is only when the exclusive lock has been released that the effects of the write operation will be made visible to other transactions.

The protocol used in locking method is the strict two-phase locking protocol.

Strict two-phase locking uses shared and exclusive locks to protect data. A transaction must hold all the required locks before executing, and does not release any lock until the transaction has completely finished.

Deadlock

It occurs when 2 transactions T1 and T2 exist in the following modes:

T1 = access data items X and Y

T2 = access data item Y and X

If T1 has not unlocked Y then T2 cannot begin.

If T2 has not unlocked data item X, T1 cannot continue.

Consequently, T1 and T2 wait indefinitely each waiting for the other to unlock the required data item. Such a deadlock is known as deadly embrace.

Techniques to Control Deadlocks

a) Deadlock Prevention

A transaction requesting a new lock is aborted if there is a possibility that a dead lock can occur. If the transaction is aborted, all the changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlock.

b) Deadlock Detection

The DBMS periodically tests the database for deadlocks. If the deadlock is found one of the transactions (the "victim") is aborted (rolled back and restarted) and the other transaction continues.

c) Deadlock Avoidance

The transaction must obtain all the locks it needs before it can be executed. This technique avoids rolled up of conflicting transactions by requiring that locks be obtained in successions, but the serial lock assignment increase action response times.

Conclusion:

The best deadlock control method depends on the database environment, if the probability is low, deadlock detection is recommended, if probability is high, deadlock prevention is recommended and if response time is not high on the system priority list

2. Time Stamping Method

Timestamp is a unique identifier created by the DBMS that indicates the relative starting time of a transaction.

Time stamping is a concurrency control protocol that orders transactions in such a way that older transactions, transactions with smaller timestamps, get priority in the event of conflict.

In the time stamping approach, to schedule concurrent transactions assign a global unique time stamp to each transaction. The time stamp value uses an explicit order in which transactions are submitted to the DBMS. The stamps must have 2 properties;

i. Uniqueness - which assures that no equal time stamp values can exist.

ii. Monotonicity - which assures that time stamp values always increase.

All database operations that read and write within the same transaction must have the same time stamp. The DBMS executes conflicting operations in the time stamp order thereby ensuring serialisability of the transactions.

If 2 transactions conflict, one is often stopped, re-scheduled and assigned a new time stamp value. The main drawback of time stamping approach is that each value stored in the database requires 2 additional time- stamp fields, one for the last time the field was read and one for the last update. Time stamping thus increases the memory needs in the databases.