Name of Faculty: DR. Virendra Kumar Tiwari
Designation: Professor
Department: LNCT-MCA
Subject: 202-DBMS
Unit: IV
Topic: ***Recovery and atomicity- log based recovery***

DATABASE RECOVERY IN DBMS

There can be any case in database system like any computer system when database failure happens. So data stored in database should be available all the time whenever it is needed. So Database recovery means recovering the data when it get deleted, hacked or damaged accidentally. Atomicity is must whether is transaction is over or not it should reflect in the database permanently or it should not effect the database at all. So database recovery and database recovery techniques are must in DBMS.

What is Data Recovery

It is the method of restoring the database to its correct state in the event of a failure at the time of the transaction or after the end of a process. Earlier, you have been given the concept of database recovery as a service that should be provided by all the DBMS for ensuring that the database is dependable and remains in a consistent state in the presence of failures. In this context, dependability refers to both the flexibility of the DBMS to various kinds of failure and its ability to recover from those failures. In this chapter, you will gather a brief knowledge of how this service can be provided. To gain a better understanding of the possible problems you may encounter in providing a consistent system, you will first learn about the need for recovery and its types of failure, which usually occurs in a database environment.

What is the Need for Recovery of data?

The storage of data usually includes four types of media with an increasing amount of reliability: the main memory, the magnetic disk, the magnetic tape, and the optical disk. Many different forms of failure can affect database processing and/or transaction, and each of them has to be dealt with differently. Some data failures can affect the main memory only, while others involve non-volatile or secondary storage also. Among the sources of failure are:

- Due to hardware or software errors, the system crashes, which ultimately resulting in loss of main memory.
- Failures of media, such as head crashes or unreadable media that results in the loss of portions of secondary storage.
- There can be application software errors, such as logical errors that are accessing the database that can cause one or more transactions to abort or fail.
- Natural physical disasters can also occur, such as fires, floods, earthquakes, or power failures.
- Carelessness or unintentional destruction of data or directories by operators or users.
- Damage or intentional corruption or hampering of data (using malicious software or files) hardware or software facilities.

Whatever the grounds of the failure are, there are two principal things that you have to consider:

- Failure of main memory, including that database buffers.
- Failure of the disk copy of that database.

Recovery Facilities

Every DBMS should offer the following facilities to help out with the recovery mechanism:

- Backup mechanism makes backup copies at a specific interval for the database.
- Logging facilities keep tracing the current state of transactions and any changes made to the database.
- Checkpoint facility allows updates to the database for getting the latest patches to be made permanent and keep secure from vulnerability.

- Recovery manager allows the database system for restoring the database to a reliable and steady-state after any failure occurs.

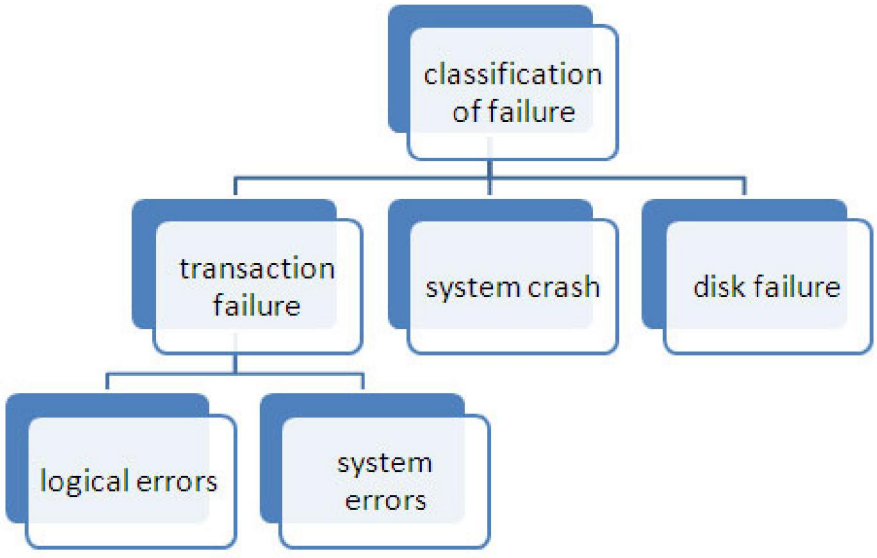
Crash Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification

To see wherever the matter has occurred, we tend to generalize a failure into numerous classes, as follows:

- **Transaction failure**
- **System crash**
- **Disk failure**



To see where the problem has occurred, we generalize a failure into various categories, as follows –

Transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be −

- **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.
- **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Storage Structure

We have already described the storage system. In brief, the storage structure can be divided into two categories −

- **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

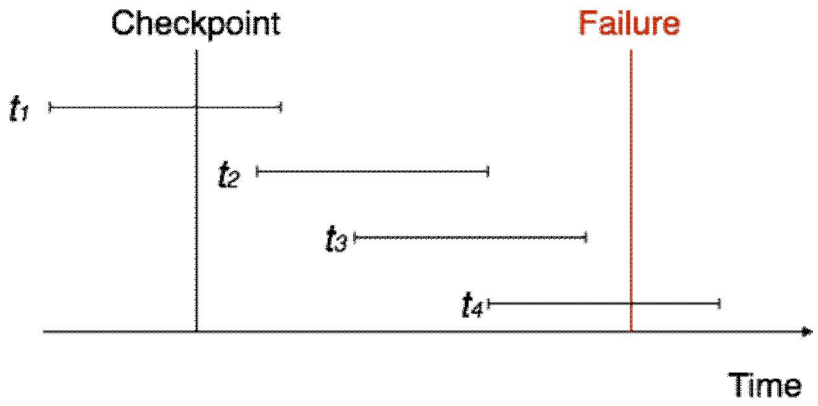
- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner –



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

Log based Recovery in DBMS

Atomicity property of DBMS states that either all the operations of transactions must be performed or none. The modifications done by an aborted transaction should not be visible to database and the modifications done by committed transaction should be visible.

To achieve our goal of atomicity, user must first output to stable storage information describing the modifications, without modifying the database itself. This information can help us ensure that all modifications performed by committed transactions are reflected in the database. This information can also help us ensure that no modifications made by an aborted transaction persist in the database.

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows –

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

Log and log records

The log is a sequence of log records, recording all the update activities in the database. In a stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded is on the log. Prior to performing any modification to database, an update log record is created to reflect that modification.

An update log record represented as: <Ti, Xj, V1, V2> has these fields:

1. **Transaction identifier:** Unique Identifier of the transaction that performed the write operation.
2. **Data item:** Unique identifier of the data item written.
3. **Old value:** Value of data item prior to write.
4. **New value:** Value of data item after write operation.

Other type of log records are:

1) <Ti start>: It contains information about when a transaction Ti starts.
2) <Ti commit>: It contains information about when a transaction Ti commits.
3) <Ti abort>: It contains information about when a transaction Ti aborts.

Undo and Redo Operations –

Because all database modifications must be preceded by creation of log record, the system has available both the old value prior to modification of data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

1. **Undo:** using a log record sets the data item specified in log record to old value.
2. **Redo:** using a log record sets the data item specified in log record to new value.

The database can be modified using two approaches –

1. **Deferred Modification Technique:** If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.
2. **Immediate Modification Technique:** If database modification occur while transaction is still active, it is said to use immediate modification technique.

Recovery using Log records –

After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.
2. Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

Use of Checkpoints –

When a system crash occurs, user must consult the log. In principle, that need to search the entire log to determine this information. There are two major difficulties with this approach:

1. The search process is time-consuming.
2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will cause recovery to take longer.

To reduce these types of overhead, user introduce checkpoints. A log record of the form <checkpoint L> is used to represent a checkpoint in log where L is a list of transactions active at the time of the checkpoint. When a checkpoint log record is added to log all the transactions that have committed before this checkpoint have <Ti commit> log record before the checkpoint record. Any database modifications made by Ti is written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on Ti.

After a system crash has occurred, the system examines the log to find the last <checkpoint L> record. The redo or undo operations need to be applied only to transactions in L, and to all transactions that started execution after the record was written to the log. Let us denote this set of transactions as T. Same rules of undo and redo are applicable on T as mentioned in Recovery using Log records part.

Note that user need to only examine the part of the log starting with the last checkpoint log record to find the set of transactions T, and to find out whether a commit or abort record occurs in the log for each transaction in T. For example, consider the set of transactions {T0, T1, . . ., T100}. Suppose that the most recent checkpoint took place during the execution of transaction T67 and T69, while T68 and all transactions with subscripts lower than 67 completed before the checkpoint. Thus, only transactions T67, T69, . . ., T100 need to be considered during the recovery scheme. Each of them needs to be redone if it has completed (that is, either committed or aborted); otherwise, it was incomplete, and needs to be undone.

Assignment:

Que-1. What is Data Recovery? Why is the Need for Recovery of data?

Que-2. Explain data Recovery Facilities in DBMS.

Que-3. Explain different types of Failure Classification in DBMS.

Que-4. Define log based recovery and atomicity with suitable example