UNIT – 4 Transaction Management in DBMS

Transactions are a set of operations that are used to perform some logical set of work. A transaction is made to change data in a database which can be done by inserting new data, updating the existing data, or by deleting the data that is no longer required.
A transaction usually means that the data in the database has changed. One of the major uses of DBMS is to protect the user's data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash. The transaction is any one execution of the user program in a DBMS. One of the important properties of the transaction is that it contains a finite number of steps.

Facts about Database Transactions

- A transaction is a program unit whose execution may or may not change the contents of a database.
- The transaction concept in DBMS is executed as a single unit.
- If the database operations do not update the database but only retrieve data, this type of transaction is called a read-only transaction.
- A successful transaction can change the database from one CONSISTENT STATE to another
- DBMS transactions must be atomic, consistent, isolated and durable
- If the database were in an inconsistent state before a transaction, it would remain in the inconsistent state after the transaction.

Example: Consider the following example of transaction operations to be performed to withdraw cash from an ATM vestibule.

Steps for ATM Transaction

1. Transaction Start.
2. Insert your ATM card.
3. Select a language for your transaction.
4. Select the Savings Account option.
5. Enter the amount you want to withdraw.
6. Enter your secret pin.
7. Wait for some time for processing.
8. Collect your Cash.
9. Transaction Completed.

A transaction can include the following basic database access operation.

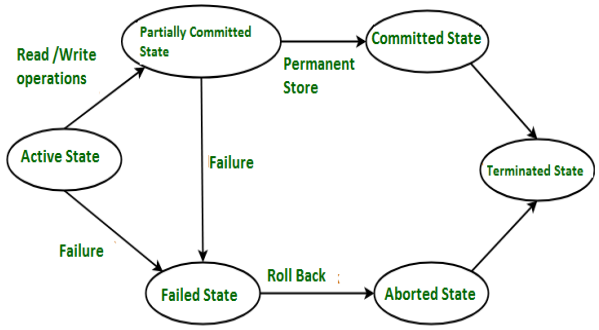
- **Read/Access data (R):** Accessing the database item from disk (where the database stored data) to memory variable.
- **Write/Change data (W): Write** the data item from the memory variable to the disk.
- **Commit:** Commit is a transaction control language that is used to permanently save the changes done in a transaction

Example: Transfer of 50₹ from Account A to Account B. Initially A= 500₹, B= 800₹. This data is brought to RAM from Hard Disk.

```
R(A) -- 500       // Accessed from RAM.
A = A-50          // Deducting 50₹ from A.
W(A)--450         // Updated in RAM.
R(B) -- 800       // Accessed from RAM.
B=B+50            // 50₹ is added to B's Account.
W(B) --850        // Updated in RAM.
commit            // The data in RAM is taken back to Hard Disk.
```

✓ Transaction States in DBMS

During the lifetime of a transaction, there are a lot of states to go through. These states update the operating system about the current state of the transaction and also tell the user about how to plan further processing of the transaction. These states decide the regulations which decide the fate of a transaction whether it will commit or abort.



Transaction States in DBMS

Following are the different types of transaction States :

Active State: When the operations of a transaction are running then the transaction is said to be active state. If all the read and write operations are performed without any error then it progresses to the partially committed state, if somehow any operation fails, then it goes to a state known as failed state.

Partially Committed: After all the read and write operations are completed, the changes which were previously made in the main memory are now made permanent in the database, after which the state will progress to committed state but in case of a failure it will go to the failed state.

Failed State: If any operation during the transaction fails due to some software or hardware issues, then it goes to the failed state . The occurrence of a failure during a transaction makes a permanent change to data in the database. The changes made into the local memory data are rolled back to the previous consistent state.

Aborted State: If the transaction fails during its execution, it goes from failed state to aborted state and because in the previous states all the changes were only made in the main memory, these uncommitted changes are either deleted or rolled back. The transaction at this point can restart and start afresh from the active state.

Committed State: If the transaction completes all sets of operations successfully, all the changes made during the partially committed state are permanently stored and the transaction is stated to be completed, thus the transaction can progress to finally get terminated in the terminated state.

Terminated State: If the transaction gets aborted after roll-back or the transaction comes from the committed state, then the database comes to a consistent state and is ready for further new transactions since the previous transaction is now terminated.

✓ Properties of Transaction in DBMS

There are four major properties that are vital for a transaction to be successful. These are used to maintain state consistency in the database, both before and after the transaction. These are called ACID properties. These are used to maintain consistency in a database, before and after the transaction.
Property of Transaction:
1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity:

t states that all operations of the transaction take place at once if not, the transactions aborted.
There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.
Atomicity involves the following two operations:
Abort: If a transaction aborts, then all the changes made are not visible.
Commit: If a transaction commits then all the changes made are visible.

Consistency:

The integrity constraints are maintained so that the database is consistent before and after the transaction.
The execution of a transaction will leave a database in either its prior stable state or anew stable state.
The consistent property of database states that every transaction sees a consistent database instance.
The transaction is used to transform the database from one consistent state to another consistent state.

Isolation:

It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1ends.
The concurrency control subsystem of the DBMS enforced the isolation property

Durability:

The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
The recovery subsystem of the DBMS has the responsibility of Durability property.

✓ IMPLEMENTATION OF ATOMICITY AND DURABILITY:

The recovery-management component of a database system can support atomicity and durability by a variety of schemes.
E.g. the shadow-database scheme:

Shadow copy:

In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database.
All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
This scheme is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time.
The scheme also assumes that the database is simply a file on disk. A pointer called db pointer is maintained on disk; it points to the current copy of the database.

✓ TRANSACTION ISOLATION LEVELS IN DBMS:

Some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.
The SQL standard defines four isolation levels :
1. Read Uncommitted – Read Uncommitted is the lowest isolation level. In this level,one transaction may read not yet committed changes made by other transaction, therebyallowing dirty reads. In this level, transactions are not isolated from each other.
2. Read Committed – This isolation level guarantees that any data read is committed atthe moment it is read. Thus it does not allows dirty read. The transaction holds a read orwrite lock on the current row, and thus prevent other transactions from reading,updating or deleting it.
3. Repeatable Read – This is the most restrictive isolation level. The transaction holdsead locks on all rows it references and writes locks on all rows it inserts, updates.deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.
4. Serializable – This is the Highest isolation level. A serializable execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

✓ FAILURE CLASSIFICATION:

To find that where the problem has occurred, we generalize a failure into the following categories:
1. Transaction failure
2. System crash
3. Disk failure

1. Transaction failure:
The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transactions or process is hurt, then this is called as transaction failure.
Reasons for a transaction failure could be –
1. Logical errors: If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.

2. Syntax error: It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. For example, The system aborts an active transaction, in case of deadlock or resource unavailability.

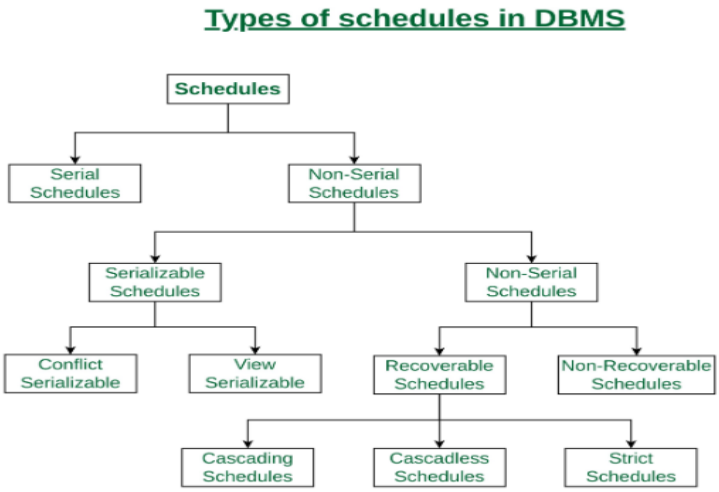
2. System Crash:
System failure can occur due to power failure or other hardware or software failure. Example: Operating system error.
Fail-stop assumption: In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure:
It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

✓ Schedules

Schedule, as the name suggests is a process of lining the transactions and executing them one by one. When there are multiple transactions that are running in a concurrent manner and the order of operation is

needed to be set so that the operations do not overlap each other, Scheduling is brought into play and the transactions are timed accordingly.



<u>Note:</u> Check/Verify Class notes for Schedule and different types of schedules with examples.

✓ <u>DBMS Concurrency Control:</u>

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other. It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.

Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

✓ <u>Locking Methods of Concurrency Control</u>

What are Locks in DBMS?

In database management systems (DBMS), locks are mechanisms used to control concurrent access to shared resources, such as database tables, rows, or other objects. Locks help ensure data integrity and prevent conflicts that can arise when multiple transactions try to access or modify the same data simultaneously.

What are Lock-Based Protocol in DBMS?

Lock-based protocols are concurrency control mechanisms used in database management systems (DBMS) to coordinate the access and modification of shared resources, such as database objects or records, among concurrent transactions. These protocols use locks to enforce serializability and ensure data integrity in a multi-user environment.

Locks are divided into three fields:

1. Lock Granularity
2. Lock Types
3. Deadlocks

Lock Granularity-: The size of the data item chosen as the unit of protection by a concurrency control program is called GRANULARITY and it also indicates the level of lock use.
Locking can take place at following table.
Ø Database level.
Ø Table level.
Ø Page level.
Ø Row (Tuple) level.

Ø Attributes (fields) level.
Database level-: In this entire database is locked
Table level-: Entire table is locked.
Page level-: Entire disk page is locked.
Row (Tuple) level-: Allow concurrent transaction to access different rows of same table. Even if rows are located on the same page.
Attributes (fields) level-: Allow concurrent transaction to access same rows. As long as they require use of different field (attributes) within that row.
Note : Check/Verify Class notes for lock types of lock granularity

✓ Types of Locks in DBMS

In DBMS Lock based Protocols, there are two modes for locking and unlocking data items Shared Lock (lock-S) and Exclusive Lock (lock-X).

Shared Locks

Shared Locks, which are often denoted as lock-S(), is defined as locks that provide Read-Only access to the information associated with them. Whenever a shared lock is used on a database, it can be read by several users, but these users who are reading the information or the data items will not have permission to edit it or make any changes to the data items.
To put it another way, we can say that shared locks don't provide access to write. Because numerous users can read the data items simultaneously, multiple shared locks can be installed on them at the same time, but the data item must not have any other locks connected with it.
A shared lock, also known as a read lock, is solely used to read data objects. Read integrity is supported via shared locks.

Shared locks can also be used to prevent records from being updated.
S-lock is requested via the Lock-S instruction.
Example of Shared Locks: Consider the situation where the value of variable X equals 50 and there are a total of 2 transactions reading X. If one transaction wants to change the value of A, another transaction that tries to read the value will read the incorrect value of the variable X. However, until it is done with reading, the Shared lock stops it from updating.

When the lock-based protocol in DBMS is applied to the transaction (let's say T1) discussed above, all the processes listed below occur.
T1 will gain exclusive access to the data item X.
Find out what the current value of data item A is.
The data item will be accessible once the transaction is finished.

Exclusive Lock:

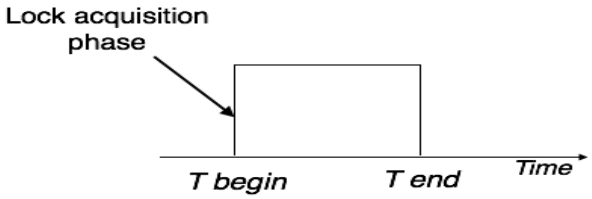
Exclusive Lock allows the data item to be read as well as written. This is a one-time use mode that can't be utilized on the exact data item twice. To obtain X-lock, the user needs to make use of the lock-x instruction. After finishing the 'write' step, transactions can unlock the data item.
By imposing an X lock on a transaction that needs to update a person's account balance, for example, you can allow it to proceed. As a result of the exclusive lock, the second transaction is unable to read or write.
The other name for an exclusive lock is write lock.
At any given time, the exclusive locks can only be owned by one transaction.
Example of exclusive locks: Consider the instance where the value of a data item X is equal to 50 and a transaction requires a deduction of 20 from the data item X. By putting a Y lock on this particular transaction, we can make it possible. As a result, the exclusive lock prevents any other transaction from reading or writing.
There are four types of lock protocols available −

Simplistic Lock Protocol

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

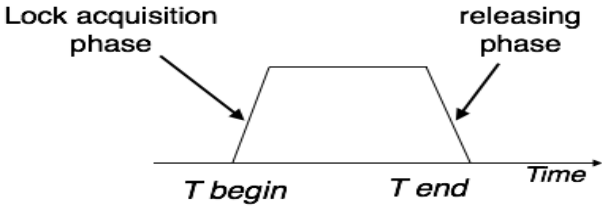
Pre-claiming Lock Protocol

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



Two-Phase Locking 2PL

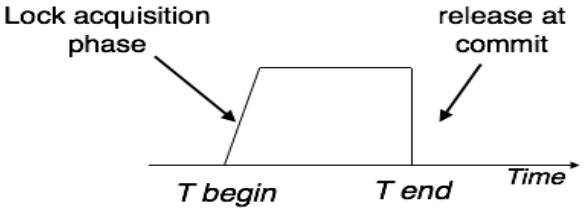
This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.
Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.



Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

Strict Two Phase Locking



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

The timestamp of transaction Ti is denoted as TS(Ti).

Read time-stamp of data-item X is denoted by R-timestamp(X).

Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows −

If a transaction Ti issues a read(X) operation −

If TS(Ti) < W-timestamp(X)

Operation rejected.

If TS(Ti) >= W-timestamp(X)

Operation executed.

All data-item timestamps updated.
If a transaction Ti issues a write(X) operation −
If TS(Ti) < R-timestamp(X)
Operation rejected.
If TS(Ti) < W-timestamp(X)
Operation rejected and Ti rolled back.
Otherwise, operation executed.
Thomas' Write Rule
This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back.
Time-stamp ordering rules can be modified to make the schedule view serializable.
Instead of making Ti rolled back, the 'write' operation itself is ignored.

✓ Deadlocks in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.
For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

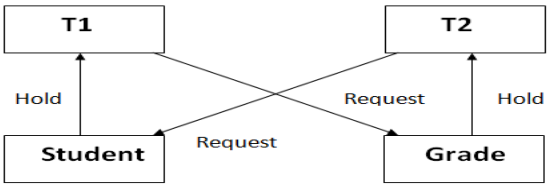


Figure: Deadlock in DBMS

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

Deadlock Avoidance

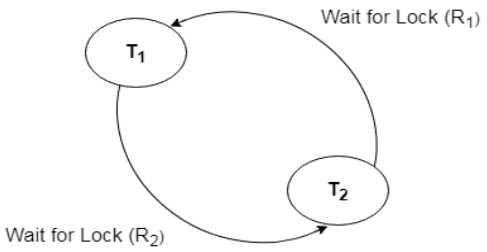
When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.



The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

Deadlock Prevention

Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.

The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.

Check if TS(Ti) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.

If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

✓ The Transaction Log

A DBMS uses a transaction log to keep track of all transactions that update the database. The information stored in this log is used by the DBMS for a recovery requirement triggered by a ROLLBACK statement, a program's abnormal termination, or a system failure such as a network discrepancy or a disk crash. Some RDBMSs use the transaction log to recover a database

forward to a currently consistent state. After a server failure, for example, Oracle automatically rolls back uncommitted transactions and rolls forward transactions that were committed but not yet written to the physical database.
The transaction log is an integral part of SQL Server. Every database has a transaction log that is stored within the log file that is separate from the data file. A transaction log basically records all database modifications. When a user issues an INSERT, for example, it is logged in the transaction log. This enables the database to roll back or restore the transaction if a failure were to occur and prevents data corruption. For example, let's say Sue is using an application and inserts 2000 rows of data. While SQL Server is processing this data let's say someone pulls the plug on the server. Because the INSERT statement was writing to the transaction log and it knows a failure occurred it will roll back the statement.
Log entries are sequential in nature. The transaction log is split up into small chunks called virtual log files. When a virtual log file is full, transactions automatically move to the next virtual log file. As long as the log records at the beginning of the transaction log have been truncated when logging reaches the end of the log, it will circle back around to the start and will overwrite what was there before:
While the DBMS executes transactions that modify the database, it also automatically updates the transaction log.

The transaction log stores:

- A record for the beginning of the transaction.
- For each transaction component (SQL statement):
- The type of operation being performed (update, delete, insert).
- The names of the objects affected by the transaction (the name of the table).
- The "before" and "after" values for the fields being updated.
- Pointers to the previous and next transaction log entries for the same transaction.
- The ending (COMMIT) of the transaction.

✓ 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, the 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 and log records – The log is a sequence of log records, recording all the update activities in the database. In 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 the database, an update log record is created to reflect that modification. An update log record represented as: <Ti, Xj, V1, V2> has these fields:
Transaction identifier: Unique Identifier of the transaction that performed the write operation.
Data item: Unique identifier of the data item written.
Old value: Value of data item prior to write.
New value: Value of data item after write operation.

Other types of log records are:
<Ti start>: It contains information about when a transaction Ti starts.
<Ti commit>: It contains information about when a transaction Ti commits.
<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 the modification of the data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:
Undo: using a log record sets the data item specified in log record to old value.
Redo: using a log record sets the data item specified in log record to new value.
The database can be modified using two approaches –
Deferred Modification Technique: If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.
Immediate Modification Technique: If database modification occur while the 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.
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>.
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:
Log-based recovery is a technique used in database management systems (DBMS) to recover a database to a consistent state in the event of a failure or crash. It involves the use of transaction logs, which are records of all the transactions performed on the database.
In log-based recovery, the DBMS uses the transaction log to reconstruct the database to a consistent state. The transaction log contains records of all the changes made to the database, including updates, inserts, and deletes. It also records information about each transaction, such as its start and end times.
When a failure occurs, the DBMS uses the transaction log to determine which transactions were incomplete at the time of the failure. It then performs a series of operations to undo the incomplete transactions and redo the completed ones. This process is called the redo/undo recovery algorithm.
The redo operation involves reapplying the changes made by completed transactions that were not yet saved to the database at the time of the failure. This ensures that all changes are applied to the database.
The undo operation involves undoing the changes made by incomplete transactions that were saved to the database at the time of the failure. This restores the database to a consistent state by reversing the effects of the incomplete transactions.
Once the redo and undo operations are completed, the DBMS can bring the database back online and resume normal operations.
Log-based recovery is an essential feature of modern DBMSs and provides a reliable mechanism for recovering from failures and ensuring the consistency of the database.