UNIT - 7: TRANSACTION PROCESSING CONCEPTS

UNIT STRUCTURE

7.1 Learning Objectives
7.2 Introduction
7.3 Introduction to Transaction Processing
7.3.1 Single–User V/S Multiuser Systems
7.3.2 Transactions, R/W Operations and DBMS Buffers
7.3.3 Need of Concurrency Control
7.3.4 Need of Recovery
7.4 Transaction and System Concepts
7.4.1 Transaction States and Additional Operations
7.4.2 The System Log
7.4.3 Commit Point of a Transaction
7.5 Desirable Properties of Transactions
7.6 Characterizing Schedules Based on Recoverability
7.7 Characterizing Schedules Based on Serializability
7.8 Let Us Sum Up
7.9 Answers To Check Your Progress
7.10 Further Readings
7.11 Model Questions

7.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of transaction processing
- differentiate between single-user and multiuser systems
- know some basic concepts of DBMS transactions
- know the concepts of concurrency and recovery
- learn the different elements and operations of a transaction
- learn about characterizing schedules

7.2 INTRODUCTION

In the previous units we came to know about several basic concepts of a Database Management System. We got to know the different database models, their different architectures, the relation between elements etc. We were also introduced to the relational model that uses the advantage of the Structured Query Language to perform several database operations. Normalization, which is a primary operation to be performed in a database, was also discussed along with its different forms.

In this unit we discuss the concept of transaction processing systems. We also define the concept of a transaction, which is used to represent a logical unit of database processing that must be completed in its entirety to ensure correctness. We also discuss the concurrency control problem, which occurs when multiple transactions submitted by various users interfere with one another in a way that produces incorrect results.

7.3 INTRODUCTION TO TRANSACTION PROCESSING

Transaction Processing Systems are the systems with large databases and hundreds of concurrent users executing database transactions. The concept of transaction provides a mechanism for describing logical units of database processing. Examples of Transaction Processing Systems include systems for reservations, banking, credit card processing, stock markets, supermarket checkouts etc. Such systems require high availability and fast response time for hundreds of concurrent users.

7.3.1 SINGLE USER V/S MULTIUSER SYSTEMS

In this section we compare single-user and multiuser database systems and demonstrate how concurrent execution of transactions can take place in multiuser systems.

One criterion for classifying a database system is according to the number of users who can use the system concurrently-that is, at the same time. A DBMS is single-user if at most one user at a time can use the system, and it is multiuser if many users can use the system-and hence access the database-concurrently. Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser. For example, an airline reservations system is used by hundreds of travel agents and reservation clerks concurrently.

Multiple users can access databases-and use computer systems-simultaneously because of the concept of multiprogramming, which allows the computer to execute multiple programs-or processes-at the same time. If only a single central processing unit (CPU) exists, it can actually execute at most one process at a time. However, multiprogramming operating systems execute some commands from one process, then suspend that

process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved. **Figure 7.1** shows two processes A and B executing concurrently in an interleaved fashion. Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation, such as reading a block from disk. The CPU is switched to execute another process rather than remaining idle during I/O time. Interleaving also prevents a long process from delaying other processes.

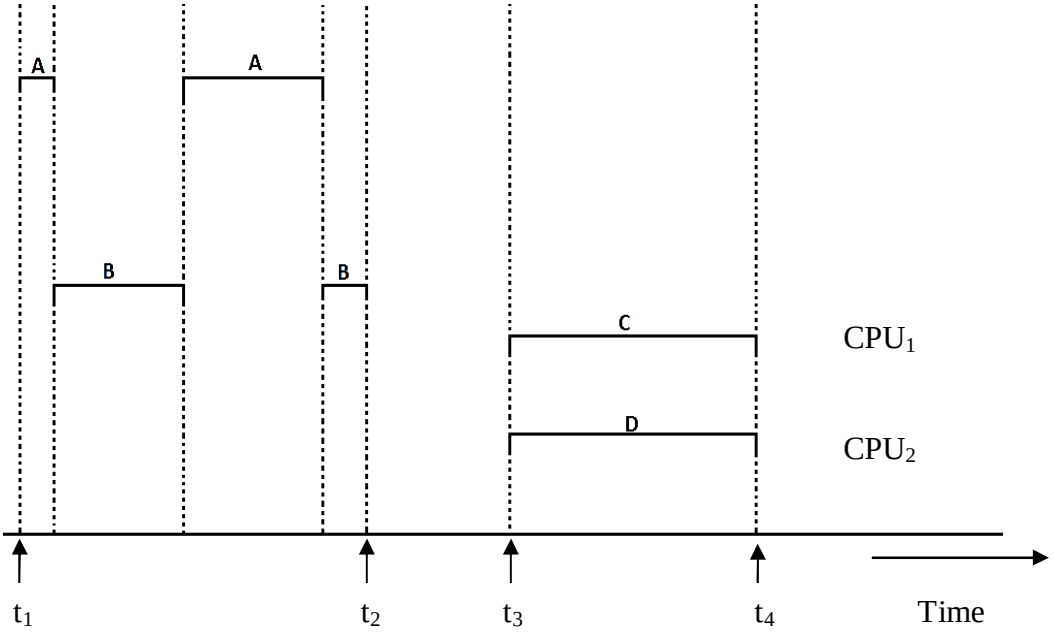


Fig 5.1: Interleaved processing versus parallel processing of concurrent transactions

If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible, as illustrated by processes C and D in **Figure 7.1**

7.3.2 TRANSACTIONS, R/W OPERATIONS AND DBMS BUFFERS

A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations-these can include insertion, deletion, modification, or retrieval operations. The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program; in this case, all database

access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction.

The basic database access operations that a transaction can include are as follows:

- read_item(X): Reads a database item named X into a program variable.
- write_item(X): Writes the value of program variable X into the database item named X.

Executing a read_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Executing a write_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

The DBMS will generally maintain a number of buffers in main memory that hold database disk blocks containing the database items being processed. When these buffers are all occupied, and additional database blocks must be copied into memory, some buffer replacement policy is used to choose which of the current buffers is to be replaced. If the chosen buffer has been modified, it must be written back to disk before it is reused.

A transaction includes **read_item** and **write_item** operations to access and update the database. **Figure 7.2** shows examples of two very simple transactions. The **read-set** of a transaction is the set of all items that the transaction reads, and the **write-set** is the set of all items that the transaction writes. For example, the read-set of T1 in **Figure 7.2** is {X, Y} and its write-set is also {X, Y}.

7.3.3 NEED OF CONCURRENCY CONTROL

Several problems can occur when concurrent transactions execute in an uncontrolled manner. **Figure 7.2(a)** shows a transaction T_1 that transfers N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y. **Figure 7.2(b)** shows a simpler transaction T_2 that just reserves M seats on the first flight (X) referenced in transaction T_1.

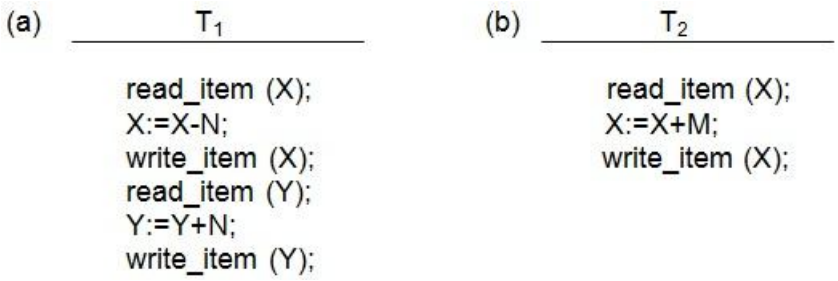
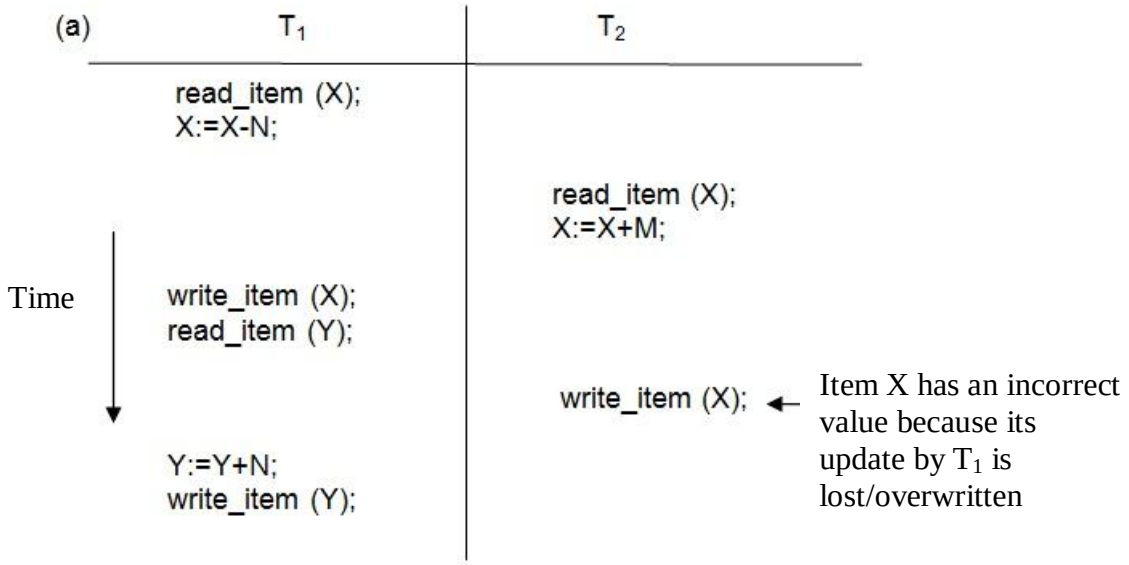


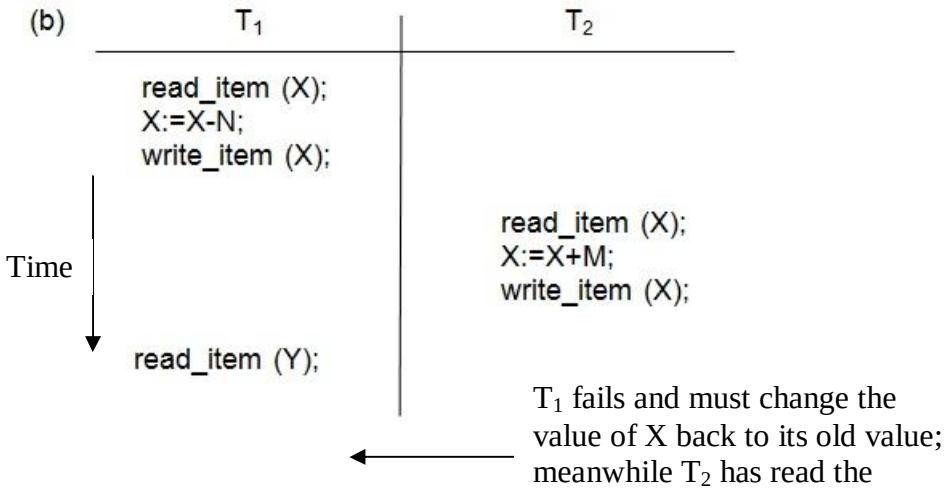
Fig 7.2: Two sample transactions. (a) Transaction T_1 (b) Transaction T_2

The types of problems that may be encountered with these two transactions if they are run concurrently are as follows.

The Lost Update Problem –

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect. Suppose that transactions T_1 and T_2 are submitted at approximately the same time, and suppose that their operations are interleaved as shown in **Figure 7.3 (a)**; then the final value of item X is incorrect, because T_2 reads the value of X before T_1 changes it in the database, and hence the updated value resulting from T_1 is lost.





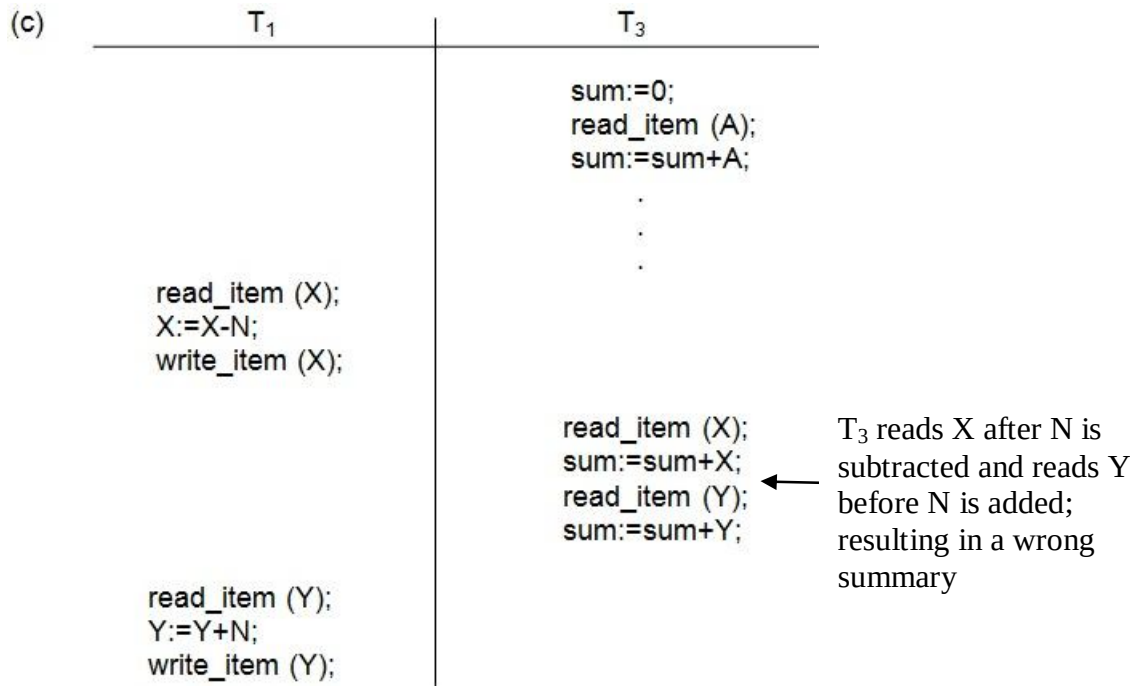


Fig 7.3: (a) The lost update problem (b) The temporary update problem (c) The incorrect summary problem

The Temporary Update (or Dirty Read) Problem –

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value. **Figure 7.3 (b)** shows an example where T_1 updates item X and then fails before completion, so the system must change X back to its original value. Before it can do so, however, transaction T_2 reads the "**temporary**" value of X, which will not be recorded permanently in the database because of the failure of T_1. The value of item X that is read by T_2 is called dirty data, because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the dirty read problem.

The Incorrect Summary Problem –

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated. For example, suppose that a transaction T_3 is calculating the total number of reservations on all the flights; meanwhile, transaction T_1 is executing. If the interleaving of operations shown in **Figure 7.3 (c)** occurs, the result of T_3 will be off by an amount N because T_3 reads the value of X after N seats have been subtracted from it but reads the value of Y before those N seats have been added to it.

Another problem that may occur is called unrepeatable read, where a transaction T reads an item twice and the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item.

7.3.4 NEED OF RECOVERY

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either (1) all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or (2) the transaction has no effect whatsoever on the database or on any other transactions. The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not. This may happen if a transaction fails after executing some of its operations but before executing all of them.

Types of Failures –

Failures are generally classified as Transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

- **A computer failure (system crash)** – A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures-for example, main memory failure.

- **A transaction or system error –** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

- **Local errors or exception conditions detected by the transaction –** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception should be programmed in the transaction itself, and hence would not be considered a failure.

- **Concurrency control enforcement –** The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

- **Disk failure –** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

- **Physical problems and catastrophes –** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

CHECK YOUR PROGRESS

1. Fill in the blanks:

(a) Transaction Processing Systems are the systems with large _____ and hundreds of concurrent users executing database _____.

(b) Multiple users can access databases and use computer systems simultaneously because of the concept of _____.

(c) _____ reads a database item named X into a program variable.

(d) _____ writes the value of program variable X into the database item named X.

(e) A _____ includes read_item and write_item operations to _____ and _____ the database.

(f) The _____ of a transaction is the set of all items that the transaction reads, and the _____ is the set of all items that the transaction writes.

(g) The _____ problem occurs when one transaction _____ a database item and then the transaction fails for some reason

(h) During _____ a transaction T reads an item twice and the item is changed by another transaction T' between the two reads

7.4 TRANSACTION AND SYSTEM CONCEPTS

While discussing about Transaction Processing Systems there are some other relevant concepts that form the basis of effective and efficient transaction processing. Such issues include some discussion on the relevant operations like the different states that a transaction goes through while it is being executed during transaction processing. The system log, which keeps information needed for recovery, is the key player in case the need of a system recovery arises.

7.4.1 Transaction States and Additional Operations

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts. Hence, the recovery manager keeps track of the following operations:

- **BEGIN_TRANSACTION**: This marks the beginning of transaction execution.
- **READ OR WRITE**: These specify read or write operations on the database items that are executed as part of a transaction.
- **END_TRANSACTION**: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason.
- **COMMIT_TRANSACTION**: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- **ROLLBACK (OR ABORT)**: This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Figure 7.4 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an **active state** immediately after it starts execution, where it can issue **READ** and **WRITE** operations. When the transaction ends, it moves to the **partially committed state**. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently. Once this check is successful, the transaction is said to have reached its commit point and enters the committed state.

Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its **WRITE** operations on the database. The **terminated state** corresponds to the transaction leaving the system.

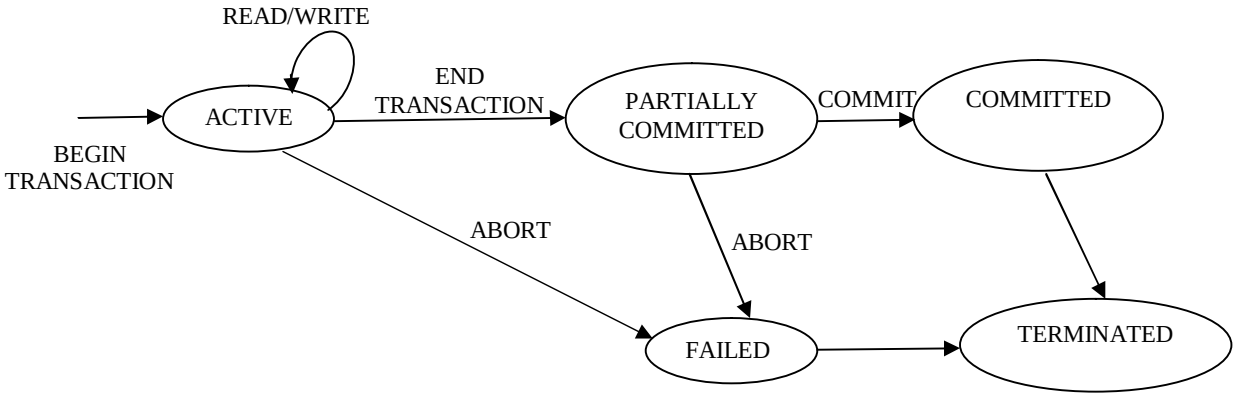


Fig 7.4: State transition diagram of states for transaction execution

7.4.2 THE SYSTEM LOG

The system maintains a log to keep track of all transaction operations that affect the values of database items in order to be able to recover from failures that affect transactions. This information may be needed to permit recovery from failures. We now list the types of entries – called log records – that are written to the log and the action each performs. In these entries, T refers to a unique transaction-id that is generated automatically by the system and is used to identify each transaction:

- [start-transaction, T]: Indicates that transaction T has started execution.
- [write_item, T, X, old_value, new_value]: Indicates that transaction T has changed the value of database item X from old_value to new_value.
- [read_item, T, X]: Indicates that transaction T has read the value of database item X.

- [commit, T]: Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
- [abort, T]: Indicates that transaction T has been aborted.

7.4.3 COMMIT POINT OF A TRANSACTION

A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log. Beyond the commit point, the transaction is said to be **committed**, and its effect is assumed to be permanently recorded in the database. The transaction then writes a commit record [commit, T] into the log. If a system failure occurs, we search back in the log for all transactions T that have written a [start_transaction, T] record into the log but have not written their [commit, T] record yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process. Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, so their effect on the database can be redone from the log records.

It is important to note that the log file must be kept on disk. Updating a disk file involves copying the appropriate block of the file from disk to a buffer in main memory, updating the buffer in main memory, and copying the buffer to disk. It is also common to keep one or more blocks of the log file in main memory buffers until they are filled with log entries and then to write them back to disk only once, rather than writing to disk every time a log entry is added. This saves the overhead of multiple disk writes of the same log file block. At the time of a system crash, only the log entries that have been written back to disk are considered in the recovery process because the contents of main memory may be lost. Hence, before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called **force-writing** the log file before committing a transaction.

7.5 DESIRABLE PROPERTIES OF TRANSACTION

Transactions should possess several properties. These are often called the **ACID** properties, and they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:

1. **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

2. **Consistency preservation**: A transaction is consistency preserving if its complete execution takes the database from one consistent state to another.
3. **Isolation**: A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **Durability or permanency**: The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

The atomicity property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity. If transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effect of the transaction on the database.

The preservation of consistency is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints. A database state is a collection of all the stored data items in the database at a given point in time. A consistent state of the database satisfies the constraints specified in the schema as well as any other constraints that should hold on the database. A database program should be written in a way that guarantees that, if the database is in a consistent state before executing the transaction, it will be in a consistent state after the complete execution of the transaction, assuming that no interference with other transaction occurs.

Isolation is enforced by the concurrency control system of the DBMS. If every transaction does not make its updates visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks. There have been attempts to define the level of isolation of a transaction. A transaction is said to have Level 0 isolation if it does not overwrite the dirty reads of higher level transaction. Level 1 isolation has no lost of updates; and Level 2 isolation has no lost updates and no dirty reads. Finally, Level 3 isolation has in addition to degree 2 properties, repeatable reads.

Finally, durability property is the responsibility of the recovery subsystem of the DBMS.

CHECK YOUR PROGRESS

2. Fill in the blanks:

(a) A transaction is an _____ unit of work that is either completed in its entirety or not done at all

(b) _____ marks the beginning of transaction execution.

(c) _____ specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution.

(d) _____ signals a successful end of the transaction.

(e) _____ signals that the transaction has ended unsuccessfully.

(f) A transaction is _____ if its complete execution takes the database from one consistent state to another.

(g) A _____ is a collection of all the stored data items in the database at a given point in time.

(h) A transaction is said to have _____ isolation if it does not overwrite the dirty reads of higher level transaction.