Unit=8

Introduction to Transaction Processing Concepts and Theory:

⊛. Single-user versus multiuser system:

Characteristics	Single-user system	Multi-user system
Definition	i) A single-user system is a system in which only one user can access the computer system at a time.	i) A multi-user system is a system that allows more than one user to access a computer system at one time.
Super User	ii) A super user gets all the powers of maintaining the system and making changes to ensure the system runs smoothly.	ii) Super user does not exist when it comes to a multi-user system as each entity has control over their working.
Complexity	iii) Single-user system is simple.	iii) Multi-user system is complex.
Performance	iv) Only one task at a time gets performed.	iv) Shedules different tasks for performance at any rate.
Example	v) Windows, Apple Mac OS.	v) UNIX, LINUX

⊛. Transactions, Database Items, Read and Write Operations & 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 via a high-level query language such as SQL.

If the database operations in a transaction do not update the database but only retrive data, the transaction is called a read-only transaction; otherwise it is known as read-write transaction.

Database Item (Data item):

A database is basically represented as a collection of named data items. The size of a data item is called its granularity. A data item can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database.

Each data item has a unique name, but this name is not typically used by the programmer; rather, it is just a means to uniquely identify each data item.

Read and Write Operations:

The basic datatabse access operations that a transaction can include ~~as~~ are as follows;

1) read_item(X) → Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also ~~in~~ named X.

2) write_item(X) → Writes the value of program variable X into the database item named X.

Executing the read_item(X) command includes following steps:

→ Find the address of disk block that contains item X.
→ Copy the disk block into a buffer in main memory (if that disk block is not already in some main memory buffer). The size of the buffer is the same as the disk block size.
→ Copy item X from the buffer to the program variable named X.

Executing a write_item(X) command includes following steps:

→ Find the address of the disk block that contains item X.
→ Copy the disk block into a buffer in main memory.
→ Copy item X from the program variable named X into its correct location in the buffer.
→ Store the updated disk block from the buffer back to disk (either immediately or at some later point in time).

DBMS Buffers:

A database buffer is a temporary storage area in the main memory. It allows storing the data temporarily when moving from one place to another. A database buffer stores a copy of disk blocks. But, the version of block copies on the disk may be older than the version in the buffer.

The DBMS will maintain in the database cache a number of data buffers in main memory. Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed. When these buffers are all occupied, and additional database disk blocks must be copied into memory, some buffer replacement policy is used to choose which of the current occupied buffers is to be replaced. Some commonly used buffer replacement polices are LRU (least recently used).

⊛. Why do we need concurrency control?

⇒ Concurrency Control is the management procedure that is required for controlling existing execution of the operations that take place on a database. It is a procedure of managing simultaneous operations without conflicting with eachother. We need concurrency control method in DBMS for following reasons:

→ To apply isolation through mutual exclusion between conflicting transactions.

→ To resolve read-write and write-write conflict issues.

→ To preserve database consistency through constantly preserving execution obstructions.

→ The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.

→ Concurrency control helps to ensure serializability.

⊛. Why do we need recovery?

⇒ Database systems, like any other computer system, are subject to failures but the data stored in it must be available when required. When a database fails it must

passess the facilities for fast recovery. It must also have atomicity i.e, either transactions are completed successfully and committed or the transaction should have no effect on the database. So, to prevent data loss recovery techniques based on immediate update or backing up data can be used. The techniques used to recover the lost data due to system crash, transaction errors, viruses, incorrect command execution etc. are database recovery techniques.

Ⓐ Transaction States and Operations:

A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates and aborts. Therefore the recovery manager of the DBMS needs to keep track of the following operations: ← (i.e, transaction 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 the READ and WRITE transaction operations have ended and marks the end of transaction execution.

COMMIT_TRANSACTION → This signals a successful end of the transaction so that any 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.

Recovery techniques use the following operators:

UNDO → Similar to rollback except that it applies to a single operation rather than to a whole transaction.

REDO → This specifies that certain transaction operations must be redone to ensure that all the operations of a commited transaction have been applied successfully to the database.

Transaction States:-

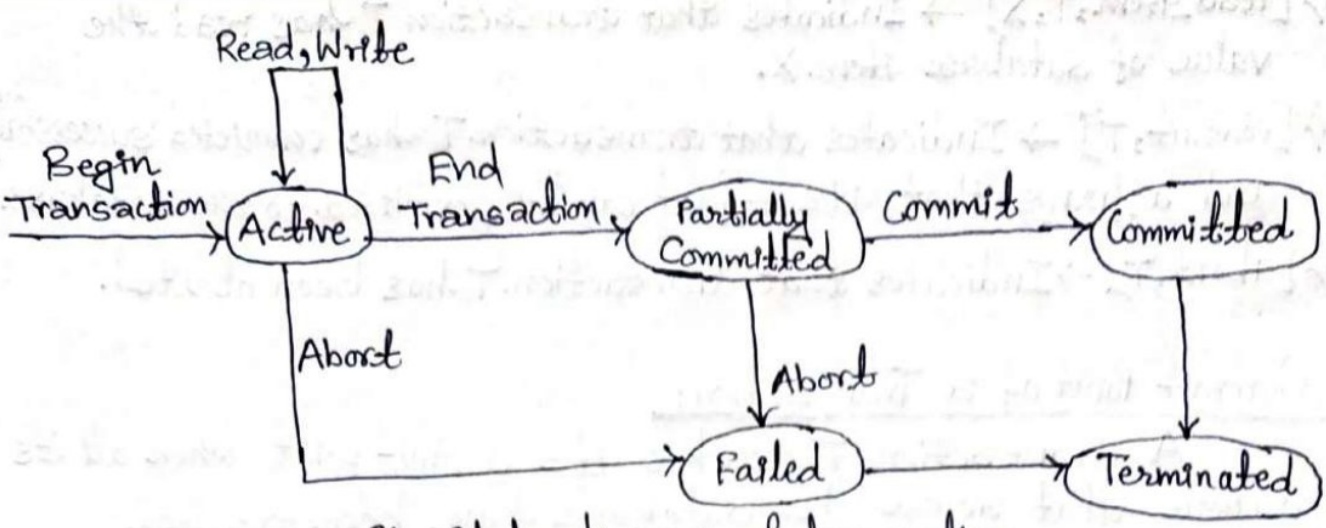

Fig. State diagram of transaction.

The above diagram illustrates that how a transaction moves through its ~~execution~~ execution states. A transaction goes into an active state immediately after it starts execution, where it can execute READ and WRITE operations. When transaction ends, it moves to the partially commited state. Once some recovery protocols are ensured, the transaction is said to have reached its commit point and enters the committed state. Finally it has concluded its execution successfully and all it's changes must be recorded permanently in the database, even if a system failure occurs.

⊛. The System log:

The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure. One or more main memory buffers, called the log buffers, hold the last part of the log file. The following are the types of entries – called log records – that are written to the log file and the corresponding action for each log record. In these entries, T refers to a unique transaction-id that is generated automatically by the system for each transaction and that is used to identify each transaction:

1) [start_transaction, T]. → Indicates that transaction T has started execution.

2) [write_item, T, X, odd_value, new_value]. → Indicates that transaction T has changed the value of database item X from odd_value to new_value.

iii) [read_item, T, X] → Indicates that transaction T has read the value of database item X.

iv) [commit, T] → Indicates that transaction T has completed successfully and affirms that its effect can be committed to the database.

v) [abort, T] → Indicates that transaction T has been aborted.

⊛. Commit Point of a Transaction:

A transaction T reaches it's 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 has 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 an entry [commit, T] into the log.

If a system failure occurs we can 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 database can be redone from the log records. Notice that the log file must be kept on disk because at the time of crash, the contents of main memory may be lost.

⊛. Buffer replacement polices:

If all the buffers in the DBMS cache are occupied and new disk pages are required to be loaded into main memory from disk, a page replacement policy is needed to select the particular buffers to be replaced. Some page replacement polices that have been developed specifically for database systems are as follows:-

* Domain Seperation (DS) Method → In this method, the DBMS cache is divided into seperate domains (set of buffers). Each domain handles one type of disk pages, and page replacements within each

domain are handled via the basic LRU (least recently used) page replacement. This achieves better performance on average but it does not adopt to dynamically changing loads because the number of available buffers for each domain is predetermined.

ii) Hot Set Method → This algorithm is useful in queries that have to scan a set of pages repeatedly, such as when a join operation is performed using the ~~tee~~ nested-loop method. This algorithm does not replace disk pages until their processing is completed.

iii) The DBMIN Method → This policy uses a model known as QLSM (query locality set model), which predetermines the pattern of page refrences for each algorithm for a particular type of database operation.

⊛. Desirable properties of transactions:-

Transactions should possess several properties, often called the ACID properties. The following are the ACID properties;

i) Atomicity → A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

ii) Consistency preservation → A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

iii) Isolation → A transaction should not make its updates visible to other transactions until it is committed. That is, the execution of a transaction should not be interfered by any other transactions executing concurrently.

iv) 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.

⊛ Schedules (Histories) of Transactions:-

A schedule (or history) S of n transactions $T_1, T_2, \ldots, T_n$ is an ordering of the transactions. The transactions executing

concurrently in an interleaved fashion, from various transactions forms is known as transaction schedule. For each transaction T_i that participates in the schedule S, the operations of T_i in S must appear in the same order in which they occur in T_i. The order of operations in S is considered to be a total ordering, meaning that for any two operations in the schedule, one must occur before the other.

⊗ Characterizing schedules based on recoverability:

For some schedules it is easy to recover from transaction and system failures, whereas for other schedules the recovery process can be quite involved. In some cases, it is even not possible to recover correctly after a failure. Hence, it is important to characterize the types of schedules for which recovery is possible, as well as for which recovery is relatively simple. These characterizations do not actually provide the recovery algorithm; they only attempt theoretically characterize the different types of schedules.

i) Recoverable schedule → One where no transaction needs to be rolled back. A schedule S is recoverable if no transaction T in S commits until all transactions T that have written an item that T reads have committed.

ii) Cascadeless schedule → One where every transaction reads only the items that are written by committed transactions.

iii) Schedules requiring cascaded rollback → A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.

iv) Strict Schedules → A schedule in which a transaction can neither read nor write an item X until the last transaction that wrote X has committed.

#Characterizing Schedules Based on Serializability:-

The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules. This section defines serializability and discuss how it may be used in practice. A serializable schedule is of two types: Conflict Serializable and view serializable.

i) Conflict Serializable → A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:

- → They belong to different transactions
- → They operate on the same data item
- → At least one of them is a write program.

ii) View Serializable → A schedule is called view serializable if it's view equals to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

⊛. Testing for conflict serializability of a schedule:

Algorithm for testing conflict serializability of a schedule S:

1. Looks at only read_Item(X) and write_Item(X) operations.
2. Constructs a precedence graph - a graph with directed edges.
3. An edge is created from T_i to T_j if one of the operation in T_i appears before a conflicting operation in T_j.
4. The schedule is serializable if and only if the precedence graph has no cycles.

⊛. How serializability is used for concurrency control:

Ans:- A concurrency control is said to be serializable if the final result of that schedule is totally same as the final result given by the serial schedule. So in executing transaction it will be allowed to access multiple transaction with the DBMS in orders to enhance the efficiency and the maximum usability of the resources. But it should give the same result for a particular transaction, otherwise it is useless. So through serializability it will lead to the same final outcome.