Overview of Transaction Management
Unit 2 and 4

Topics:

Transaction- Transaction Concepts, States, ACID properties, Concurrent executions, Serializability, Transaction Processing, Transaction and System Concepts, Properties of Transactions Locking Techniques for Concurrency Control, Time-stamp based schedules, Database Recovery Techniques

Overview

- Transaction processing systems are systems with large databases and hundreds of concurrent users that are executing database transactions
- Examples:
  - Reservation systems
  - Banking systems
- They require high availability and fast response time for hundreds of concurrent users.
- A transaction is an execution of a user program, seen by the DBMS as a series or *list*, of actions i.e. read and write operations.
- For performance reasons, a DBMS has to interleave (mix) the instructions of several transactions.
- The interleaving is done carefully to ensure that the result of a concurrent execution of transactions should be equivalent (in its effect upon the database) to some serial, or one-at-a-time, execution of the same set of transactions.
- Transactions submitted by the various users may execute concurrently and may access and update the same database items.
- If this concurrent execution is uncontrolled, it may lead to problem, such as inconsistent database.
- DBMS handles concurrent executions and it is an important aspect of transaction management and is the subject of *concurrency control*.
- DBMS handles partial transactions, or transactions that are interrupted before they run to normal completion.
- The DBMS ensures that the changes made by such partial transactions are not seen by other transactions and it is the subject of *crash recovery*.
- Thus, two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes: Crash Recovery
  - Concurrent execution of multiple transactions: *concurrency control*

The ACID Properties

- There are four important properties of transactions that a DBMS must ensure to maintain data in case of concurrent access and system failures and these properties are known as ACID properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity

- Either all operations in transactions are carried out or none.
- The system should ensure that updates of a partially executed transaction are not reflected in the database
- Transactions can be incomplete for three kinds of reasons:
  - A transaction can be aborted, or terminated unsuccessfully by DBMS
  - The system may crash
  - A transaction may encounter an unexpected situation and decide to abort
- Thus a DBMS must find a way to remove the effects of partial transactions from the database, that is, it must ensure transaction atomicity: either all of a transaction's actions are carried out, or none are.

Consistency

- After the successful execution of transaction the database should be in consistent state.
- Erroneous transaction logic can lead to inconsistency

Consistency (Example)

- Transaction to transfer Rs. 50 from account A to account B: Assume A=100, B=200
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B)*
- Consistency requirement:
  - The sum of A and B is unchanged by the execution of the transaction.

Isolation

- A transaction should appear as though it is being executed in isolation from other transactions i.e. the execution of a transaction should not be interfered by any other transactions executing concurrently. This property is referred to as **isolation**: Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.
- For example, if two transactions *T*1 and *T*2 are executed concurrently, the net effect is guaranteed to be equivalent to executing (all) *T*1 followed by executing *T*2 or executing *T*2 followed by executing *T*1.

Durability

- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

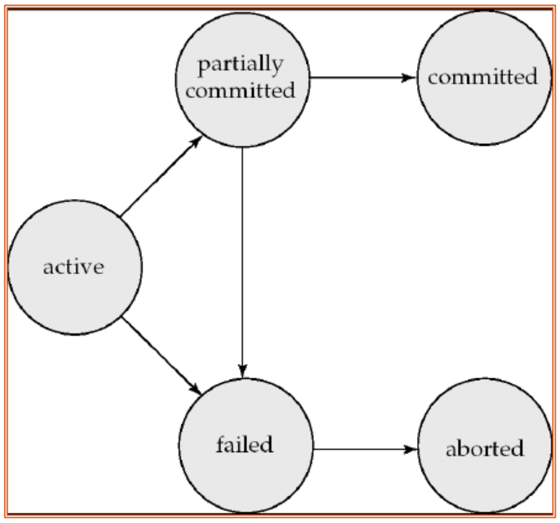
- DBMS maintains a record, called the *log*, of all writes to the database.
- The log is used to ensure durability: If the system crashes before the changes made by a completed transaction are written to disk, the log is used to remember and restore these changes when the system restarts.

Transaction States

A transaction may be in any of the following state during its lifetime.

- **Active**: The initial state when the transaction has just started execution.
- **Partially Committed:** When the transaction is executing its final instruction.
- **Committed:** If no failure occurs then the transaction reaches the COMMIT POINT. All the temporary values are written to the stable storage and the transaction is said to have been committed.
- **Failed:** If the transaction fails for some reason. The temporary values are no longer required, and the transaction is set to ROLLBACK. It means that any change made to the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then the ROLLBACK operation should add Rs 100/- to account A.
- **Aborted:** after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction can be done only if no internal logical error
  - kill the transaction.

Transaction states



Transactions and Schedules

- A transaction is seen by the DBMS as a series, or *list*, of actions.

- The actions that can be executed by a transaction include **reads** and **writes** of *database objects*.

Two important assumptions can be made:

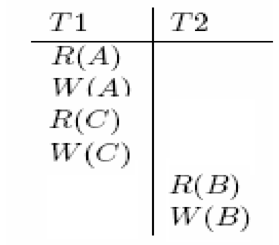
- Transactions interact with each other only via database read and write operations, and they are not allowed to exchange messages.
- The action of a transaction *T* reading an object *O is denoted* as $R_T(O)$; and writing as $W_T(O)$.
- In addition to reading and writing, each transaction *must* specify as its final action either **commit** (i.e., complete successfully) or **abort** (i.e., terminate and undo all the actions carried out so far).
- $Abort_T$ denotes the action of *T* aborting, and $Commit_T$ denotes *T* committing.

- A **schedule** is a list of actions (reading, writing, aborting, or committing) from a set of transactions. It is description of order of execution of operations of transactions.

- There are two types of schedules:

1. Serial schedule 2. Non-Serial schedule

- If the actions of different transactions are not interleaved i.e. order of execution of transactions is sequential.
- Basic Assumption is each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.



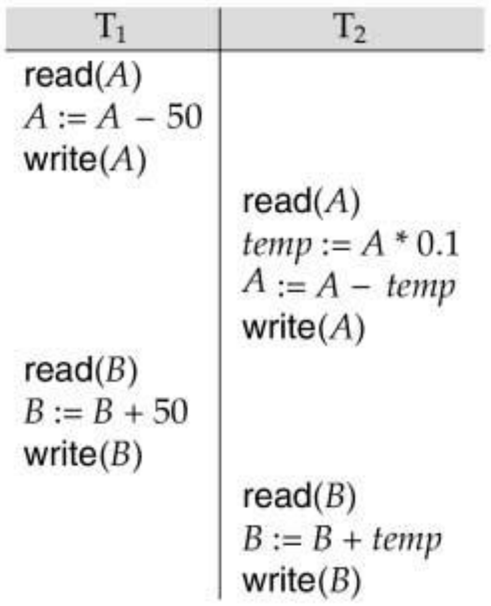
$T1$	$T2$
$R(A)$	
$W(A)$	
$R(C)$	
$W(C)$	
	$R(B)$
	$W(B)$

Serial Schedule Involving Two Transactions

Non-Serial Schedule:

Operations are interleaved from multiple transactions.

Ex:



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Concurrent Execution of Transactions

- The DBMS interleaves the actions of different transactions to improve performance, in terms of increased throughput or improved response times for short transactions, but not all interleaving should be allowed.
- Ensuring transaction isolation while permitting such concurrent execution is difficult, but is necessary for performance reasons.
- While one transaction is waiting for a page to be read in from disk, the CPU can process another transaction.
- Because I/O activity can be done in parallel with CPU activity in a computer.
- Overlapping I/O and CPU activity reduces the amount of time disks and processors are idle, and increases **system throughput** (the average number of transactions completed in a given time).
- Interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly.
- In serial execution, a short transaction could get stuck behind a long transaction leading to unpredictable delays in **response time**, or average time taken to complete a transaction.

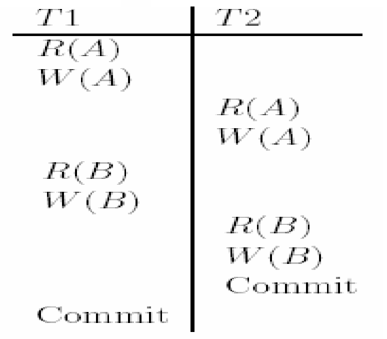
Serializability

- A **serializable schedule** over a set *S* of committed transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over *S*.

or

- The database instance that results from executing the given schedule is identical to the database instance that results from executing the transactions in *some* serial order.
- Thus a (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule.

- Even though the actions of T1 and T2 are interleaved, the result of this schedule is equivalent to running T1 and running T2.
- T1's read and write of B is not influenced by T2's actions on A, and the net effect is the same if these actions are 'swapped' to obtain the serial schedule T1:T2.

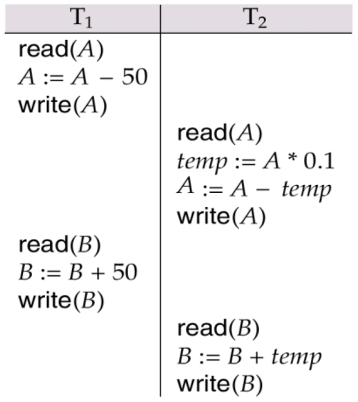


$T1$	$T2$
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
$R(B)$	
$W(B)$	
	$R(B)$
	$W(B)$
	Commit
Commit	

A Serializable Schedule equivalent to T1;T2

Serializability (Example)

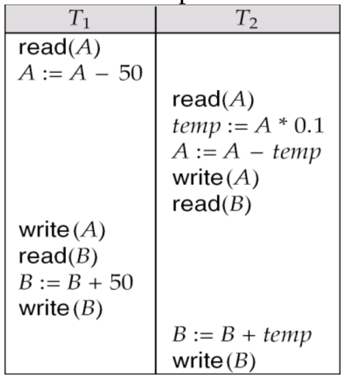
This schedule is not a serial schedule, but it is *equivalent* to Schedule T1;T2.



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Serializability (Example)

This concurrent schedule does not preserve the value of (*A* + *B*).



T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Types of Serializibility:

1. **Conflict Serilizibility**

2. **View Serilizibility**

Concurrency Control:
Why Concurrency Control is required?

- To Guarantee Serializability in concurrent transactions
- To ensure isolation property of concurrent transactions

Techniques are classified as:

- Lock Based: locking data items and thus preventing concurrent transaction from accessing.
- Time Stamp Based: Execute transaction based on time based ordering.

Locks

- Lock is a variable to control access to a data item
- A transaction *must get a lock before operating on the data*
- Two types of locks:
- *Shared (S) locks (also called read locks)*
  - Obtained if we want to only read an item
  - Other transaction are allowed to read item
- *Exclusive (X) locks (also called write locks)*
  - Obtained for updating a data item
  - Single transaction exclusively holds the item

Lock Instructions:

- Lock_item(X) has three states
  - read_lock(X)- Shared Lock
  - Write_lock(X) –Exclusive Lock
  - UnLock(X)

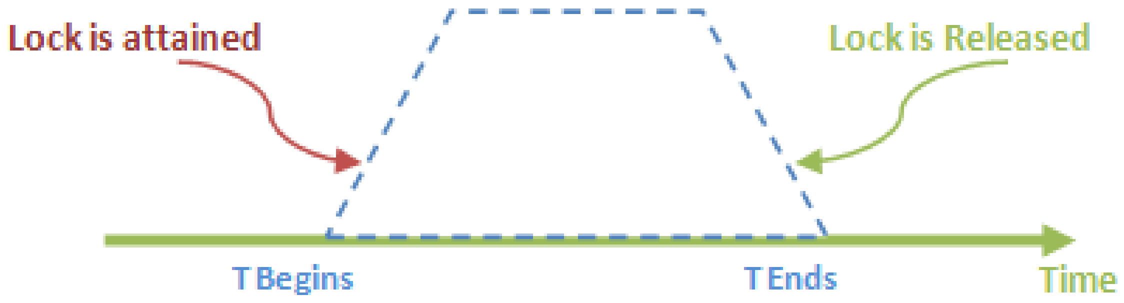
Read_item(X)- read the value of item
Write_item(X)- update the value of item
Note: Transaction must have issued read_lock(X)/write_lock(X) before using the item for read or write and Unlock(X) after all read/write operations

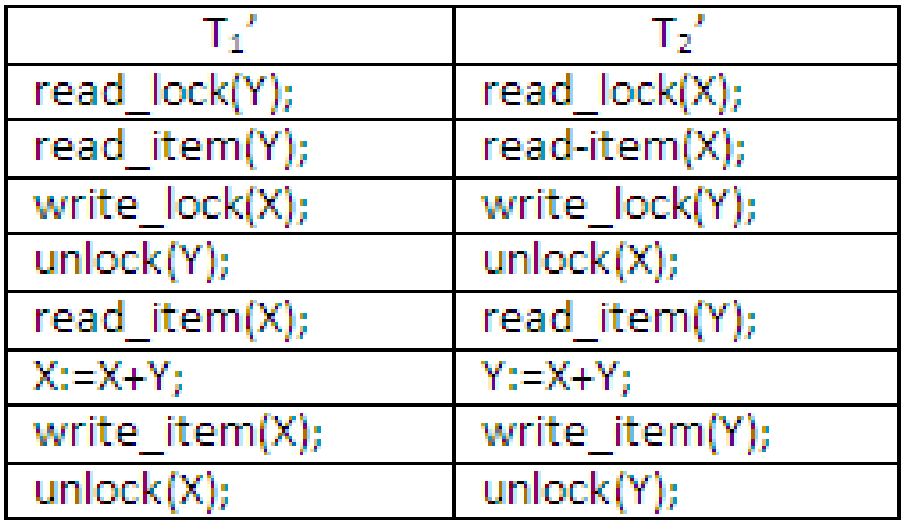
Two Phase Locking Protocol

- Defines rules to lock and unlock data items while operating the data item
- Ensures serializibility
- Contains 2 Phases
  1. Growing Phase
     - transaction may obtain locks
     - transaction may not release locks
     - When all the possible locks have been acquired Transaction reaches LOCK POINT
  2. Shrinking Phase

- transaction may release locks after LOCK POINT
- transaction may not obtain locks

Upgrading of lock(Read to Write) is not possible in Shrinking Phase but possible in Growing Phase





T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read-item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Drawback of Locking Protocols

- 2PL May lead to Deadlock
  - deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks

Time Stamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 time and

transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.

No Deadlock in this protocol

Read and Write Time Stamp

Data item will be having two variables to keep track of the time of access and write.

- Max-rts(x): max time stamp of a transaction that read x .
- Max-wts(x): max time stamp of a transaction that wrote x.

Protocol for Read Operation

Check the following condition whenever a transaction Ti issues a Read (X) operation

```
• Read (x)
If ts(Ti ) < max-wts(x) then
    Abort Ti
Else
    • Perform Read (x)
    • Max-rts(x) = ts(Ti )
```

Note: consider R->W conflict

Protocol for Write Operation

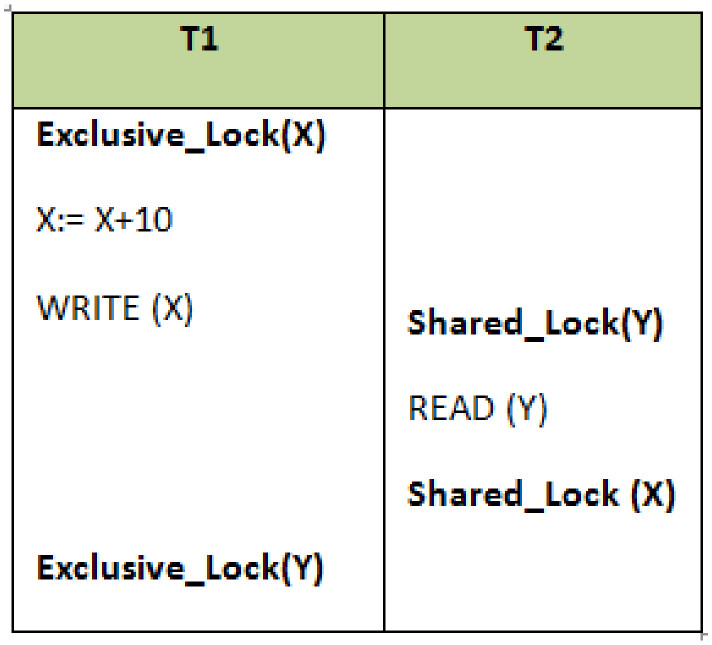
Check the following condition whenever a transaction Ti issues a Write(X) operation

```
• Write(x)
If ts(Ti )<max-rts(x) or ts(Ti )<max-wts(x) then
    Abort Ti
Else
    Perform Write (x)
    Max-wts(x) = ts(Ti )
```

Note: consider W->W&R conflict

Dead Locks in Concurrent Transaction:

- Suppose transaction T1 is updating the value X. At the same time transaction T2 is tries to read X by locking it. In addition T1 will try to lock Y which is held by lock of T2. It can be represented more clearly like below :



T1	T2
Exclusive_Lock(X)	
X:= X+10	
WRITE (X)	
	Shared_Lock(Y)
	READ (Y)
	Shared_Lock (X)
Exclusive_Lock(Y)	

Here T2 is waiting for T1 to release lock on X, while T1 is waiting for T2 to release lock on Y.

Classification of failure

1) Transaction Failure: A Transaction can fail to execute further due to either

- **Logical errors:** Transaction fails due to logical errors in the code.
- **System errors:** Transaction fails due to the errors Computer System.

2) System Crash: System crash occurs when there is a hardware or software failure or external factors like a power failure.

3) Disk Failure: Disk Failure occurs when there are issues with hard disks like formation of bad sectors, disk head crash, unavailability of disk etc.

Database Recovery Techniques

Database systems, like any other computer system, are subject to failures but the data stored in it must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.

There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.

Some of the backup techniques are as follows :

- **Full database backup –** In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series.
- **Differential backup –** It stores only the data changes that have occurred since last full database backup. When same data has changed many times since last full database backup, a differential backup stores the most recent version of changed data. For this first, we need to restore a full database backup.
- **Transaction log backup –** In this, all events that have occurred in the database, like a record of every single statement executed is backed up. It is the backup of transaction log entries and contains all transaction that had happened to the database. Through this, the database can be recovered to a specific point in time. It is even possible to perform a backup from a transaction log if the data files are destroyed and not even a single committed transaction is lost.