Module 5: Transaction Processing and Concurrency Control

Transaction:

It refers to execution of any one user program in dbms.

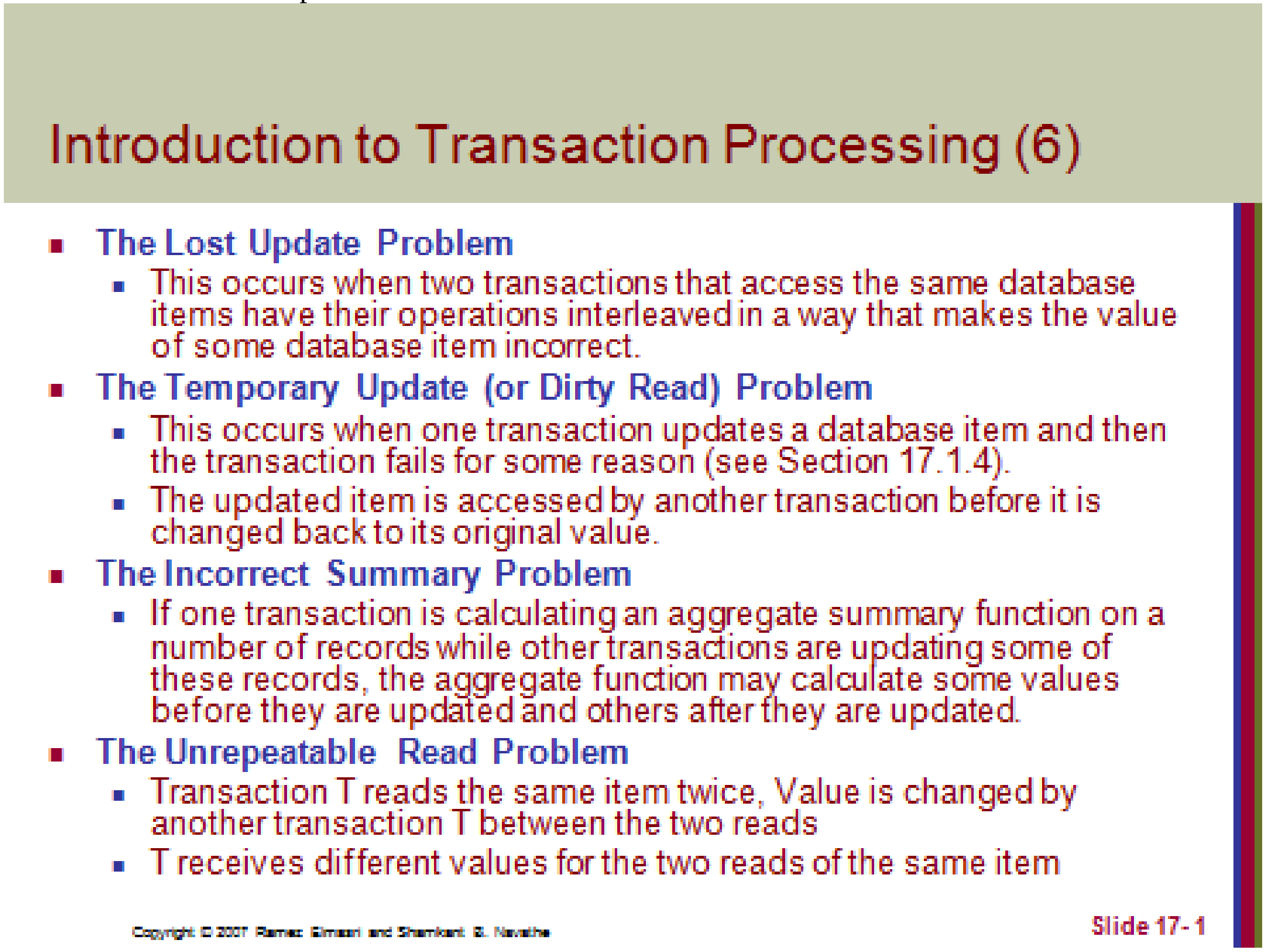
(Or)

It can be defined as group of tasks being executed.

(Or)

It also referred to as an event that which occur on a database with read/write operation.

- **Types of problems we may encounter with transactions running concurrently without control:**
  - The Lost Update Problem.
  - The Temporary Update (or Dirty Read) Problem
  - The Incorrect Summary Problem
  - The Unrepeatable Read Problem



Introduction to Transaction Processing (6)

- The Lost Update Problem
  - This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
- The Temporary Update (or Dirty Read) Problem
  - This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 17.1.4).
  - The updated item is accessed by another transaction before it is changed back to its original value.
- 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.
- The Unrepeatable Read Problem
  - Transaction T reads the same item twice, Value is changed by another transaction T between the two reads
  - T receives different values for the two reads of the same item

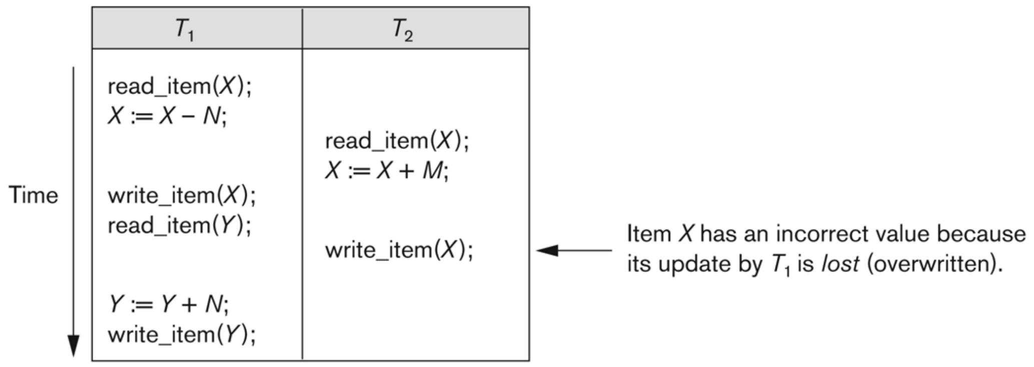
Copyright © 2007 Ramez Elmasri and Shamkant B. Navathe

Slide 17- 1

Concurrent execution is uncontrolled: (a) The lost update problem.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(a)



T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;

write_item(X);
read_item(Y);	

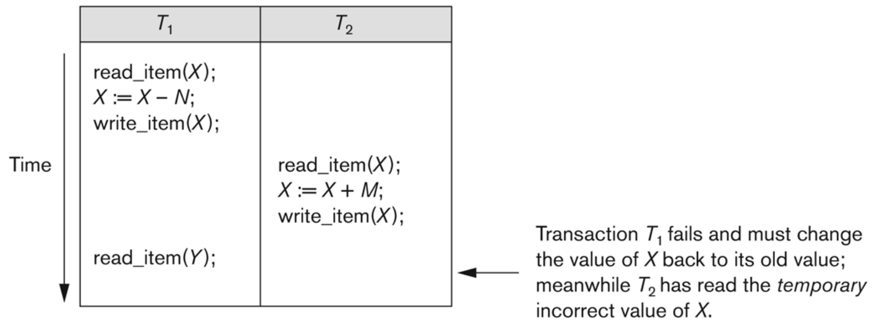
	write_item(X);
$Y := Y + N$; write_item(Y);	

Time
Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

Concurrent execution is uncontrolled: (b) The temporary update problem.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(b)



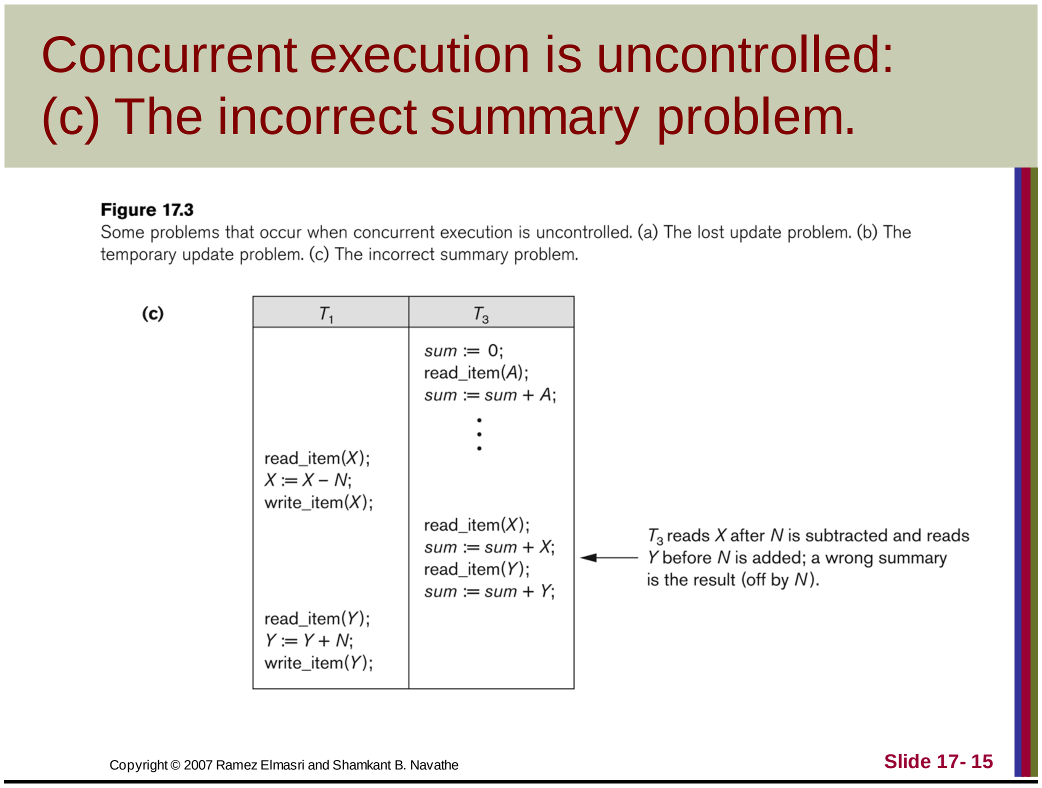
T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);	

Time
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.



Concurrent execution is uncontrolled: (c) The incorrect summary problem.

Figure 17.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(c)

T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	⋮

read_item(*X*);
X := *X* − *N*;
write_item(*X*);	

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).

Types of transaction failures

1. A computer failure (system crash):

A hardware or software error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.

2. 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.

3. Local errors or exception conditions detected by the transaction:

Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal from that account, to be canceled. A programmed abort in the transaction causes it to fail.

4. 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.

5. 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.

6. 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.

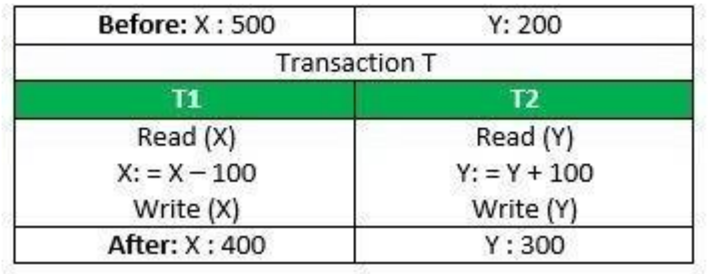
PROPERTIES OF TRANSACTION(ACID PROPERTIES):

- To ensure consistency, completeness of the database in scenario of concurrent access, system failure ,the following **ACID** properties can be enforced on to database.
  1. **Atomicity,**
  2. **Consistency,**
  3. **Isolation and**
  4. **Durabilit**

yAtomicity:

- This property states that all of the instructions with in a transaction must be executed or noneof them should be executed.
- This property states that all transactions execution must be atomic i.e. all actions should be carried out or none of the actions should be executed.

  - It involves following two operations.
    —**Abort**: If a transaction aborts, changes made to database are not visible.
    —**Commit**: If a transaction commits, changes made are
    visible. Atomicity is also known as the 'All or nothing rule'.
    Example:
  - Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 fromaccount **X** to account **Y**.

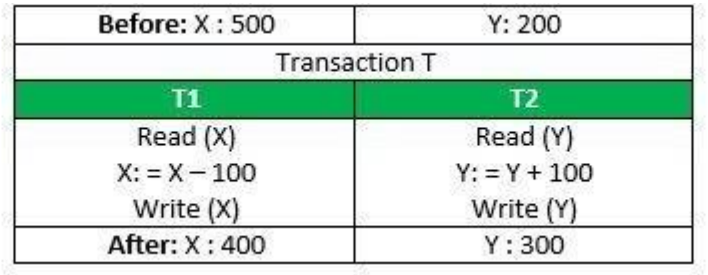


Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

  - If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

Consistency:

- The database must remain in consistence state even after performing any kind of transaction ensuring correctness of the database.
- If we execute a particular transaction in isolation (or) together with other transaction in multiprogramming environment ,the transaction should give same result in any case.
- Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called **consistency** and the DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.
  example:



Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

- Referring to the example above,
  The total amount before and after the transaction must be maintained.
  Total **before T** occurs = **500 + 200 = 700**.
  Total **after T occurs** = **400 + 300 = 700**.
  Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

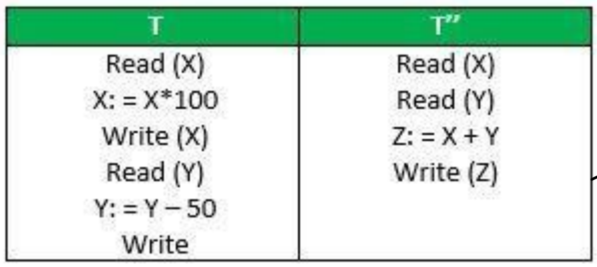
Isolation:

- When executing multiple transactions concurrently & trying to access shared resources the system should create an order such that the only one transaction can access the shared resource at the same time & release it after completion of it's execution for other transaction.
- This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

Note: To achieve isolation you should use locking mechanism among shared resources.

example:

Let **X**= 500, **Y** = 500.
Consider two transactions **T** and **T".**



T	T"
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y − 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T"** starts. As a result , interleaving of operations takes place due to which **T"** reads correct value of **X** but incorrect value of **Y** and sum computed by
T": (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of transaction:
T: (X+Y = 50, 000 + 450 = 50, 450).
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after a they have been made to the main memory.

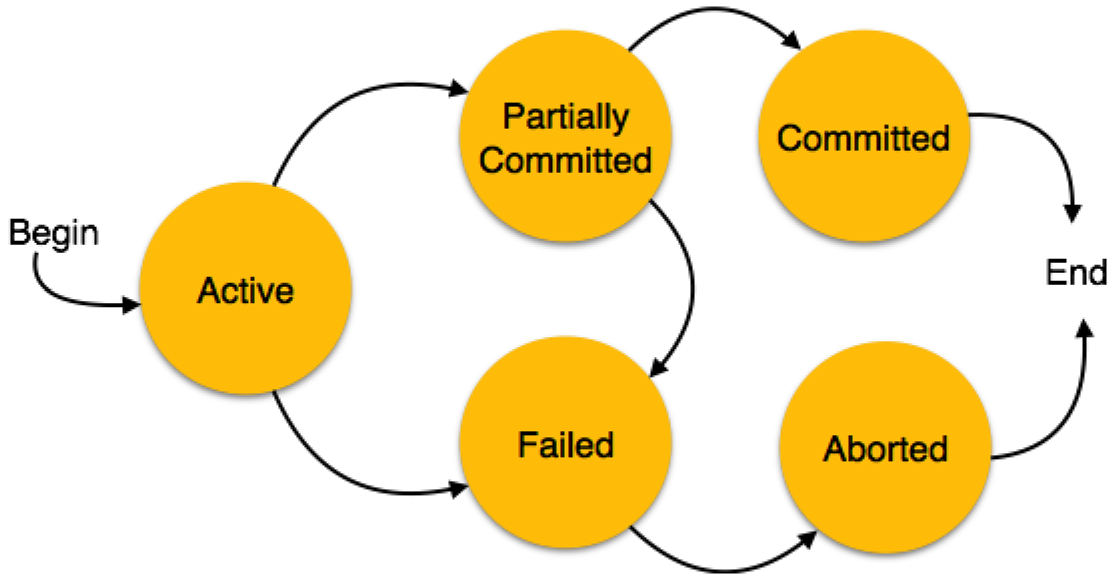
Durability:

- This property states that once after the transaction is completed the changes that made shouldbe permanent & should be recoverable even after system crash/power failure.
- This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even is system failure occurs. These updates now become permanent and are stored in a non-volatile memory.

Transaction states:

Every transaction undergoes several states in its execution.
A transaction can be in any one of the following states:

1. start
2. partially committed
3. committed
4. failed
5. aborted or terminate



Transaction state diagram

- **Active -** This is the first state of transaction and here the transaction is being executed. For example, updating or inserting or deleting a record is done here. But it is still not saved to the database. When we say transaction it will have set of small steps, and those steps will be executed here.
- **Partially Committed -** This is also an execution phase where last step in the transaction is executed. But data is still not saved to the database. In example of calculating total marks, final display the total marks step is executed in this state.
- **Committed -** In this state, all the transactions are permanently saved to the database. This step is the last step of a transaction, if it executes without fail.
- **Failed -** If a transaction cannot proceed to the execution state because of the failure of the system or database, then the transaction is said to be in failed state. In the total mark calculation example, if the database is not able fire a query to fetch the marks, i.e.; very first step of transaction, then the transaction will fail to execute.
- **Aborted -** If a transaction is failed to execute, then the database recovery system will make sure that the database is in its previous consistent state. If not, it brings the database to consistent state by aborting or rolling back the transaction. If the transaction fails in the middle of the transaction, all the executed transactions are rolled back to it consistent state before executing the transaction. Once the transaction is aborted it is either restarted to execute again or fully killed by the DBMS.

CONCURRENT EXECUTION:

Executing a set of transactions simultaneously in a pre emptive and time shared method.

In DBMS concurrent execution of transaction can be implemented with interleaved execution.

TRANSACTION SCHEDULES:

Schedule:

- It refers to the list of actions to be executed by transaction.
- A **schedule** is a process of grouping the transactions into one and executing them in a predefined order.
- Schedule of actions can be classified into 2 types.

1. Serializable schedule/serial schedule.
2. Concurrent schedule.

1. Serial schedule:

In the serial schedule the transactions are allowed to execute one after the other ensuring correctness of data.

A schedule is called serial **schedule**, if the transactions in the schedule are defined to execute one after the other.

2. Concurrent schedule:

Concurrent schedule allows the transaction to be executed in interleaved manner of execution.

Complete schedule:

It is a schedule of transactions where each transaction is committed before terminating. The example is shown below where transactions T1 and T2 terminates after committing the transactions.

Example:

T1	T2
A=1000	
Read(A)	
A=A+100	
Write(A)	Read(A)
	B=A-100
	Write(B)
	Commit
Read(B)	
Write(B)	
Commit	

SERIALIZABILITY:

A transaction is said to be **Serializable** if it is equivalent to serial schedule.

Serializability aspects are:

1. Conflict serializability.
2. View serializability.

1. Conflict serializability:

A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule.

Conflict Equivalent: Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

Conflict Serializable: A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

Conflicting operations: Two operations are said to be conflicting if all below conditions aresatisfied:

- They belong to different transaction
- They operation on same data item
- At Least one of them is a write operation

it refers to two instructions of two different transactions may want to access same data to perform read/write operation.

Rules for conflict serializability:

- If two different transactions are both for read operation, then there is no conflict and can allowed to execute any order.

- If one instruction performing read operation and other instruction performing write operation there will be conflict hence instruction ordering is important.
- If both transactions performing write operation then there will be in conflict so ordering the transaction can be done.

2. View serializability:

This is another type of serializability that can be derived by creating another schedule out of an existing Schedule.

A schedule is **view serializable** if it is view equivalent to some serial schedule. Everyconflict serializable schedule is view serializable, although the converse is not true.

Two schedules *S*1 and *S*2 over the same set of transactions --any transaction that appears in either *S*1 or *S*2 must also appear in the other are **view equivalent** under these conditions:

1. If *Ti* reads the initial value of object *A* in *S*1, it must also read the initial value of *A* in *S*2.
2. If *Ti* reads a value of *A* written by *Tj* in *S*1, it must also read the value of *A* written by *Tj* in *S*2.
3. For each data object *A*, the transaction (if any) that performs the final write on *A* in *S*1 must also perform the final write on *A* in *S*2.

- The above two schedules are view serializable or view equivalence, if the transactions in both schedules performs the actions in similar manner.
- The above two schedules satisfy result view equivalence if the two schedule produces the same Result after execution.
  Ex:
  s1:R1(A),W1(A),R2(A),W2(A),R1(B),W1(B),R2(B),W2
  (B)

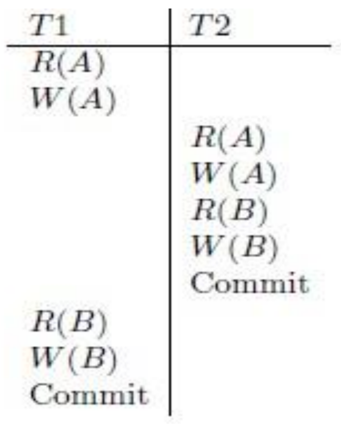
<u>Anomalies due to interleave execution of transaction:</u>

Due to interleaved execution of transaction the following anomalies can occur

1. reading uncommitted values(WR conflicts)
2. un repeatable reading data operation(RW conflicts)
3. Overwriting uncommitted data(WW)

1. reading uncommitted values(WR conflicts):

- If you try to the read the value which is not written on to the data base(not committed) will leads to write-read conflict which is called **dirty read operation.**



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

Figure 18.2 Reading Uncommitted Data

In above example, T1 write operation on data item A is not committed but it is being read by T2. So reading an uncommitted data will leads to inconsistency in database which is called dirtyread operation.

2. UN repeatable reading data operation(RW conflicts):

Reading the same object twice before committing the transaction might yield an inconsistency

–Read-then-Write (RW) Conflicts (Write-After-Read)

Unrepeatable problem means we get different values in different reads. For example in S1 say T2 read initially x=5 then T1 updated x=1 so now T2 will read x=1 here T2 has read two different values during consecutive reads This shouldn't have been allowed as T1 has not committed

3. Overwriting uncommitted data(WW conflicts)

WW conflicts if one transaction could over write the value of an object A which has been already modified by other transaction while first transaction still in progress .this kind of conflict refer to **blind write conflict.**

Implementation of Isolation:

- When more than one instruction of several transaction are being executed concurrently by using some sharable resources , the execution of instruction of one transaction should not interrupted the execution of instruction of anther transaction.
  1. **Access to sharable resources should be order by using some locking mechanism:** Where one transaction locks the sharable resource before starting it's execution &release the lock to other transaction after completion of it's execution.
  2. **Locking protocols:**
     Locking mechanism can be implemented by using locking protocols which definedset of standard rule based on which transaction access, sharable resources.

Transaction control commands supported with SQL:

1. Commit.
2. Save point.
3. Roll back.

explain about usage of above 3 commands with syntaxes.

<u>Precedence graph in serializability:</u>

Precedence graph or serializability graph is used commonly to test conflict serializability of a schedule.

- It is a directed graph which consist of nodes G(V,E) where nodes(v) represents set of transaction &E represents set of edges {E1,E2,….En}.
- The graph contains one node for each transaction Ti. Each edge Ei is of the form Tj□Tk Where Tj is starting node of edge j&Tk is ending node of edge k.
- An edge is constructed between nodes if one of the operation in transaction Tj appear in the schedule before some conflicting operation in transaction Tk.

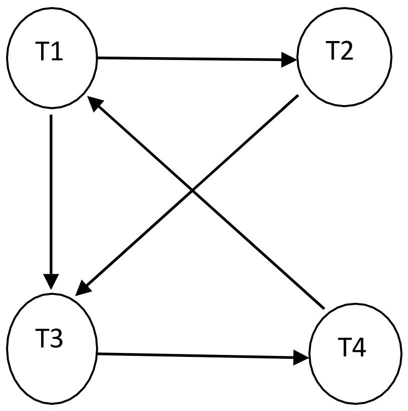
Algorithm:

1. Create a node T n the graph for each participating transaction in the schedule.
2. Draw edges from one transaction to anther transaction when satisfy anyone of the following condition.
   Condition 1:
   - If T1 execute write operation i.e. write(x) followed by T2 execute read operation i.e. read(x).
     Condition 2:
   - When T1 executes read(x) followed by T2 execute write(x).Condition 3:
   - When T1 execute write(x) followed by T2 execute write(x).
3. The given schedule is serializable if there are no cycles in the precedence graph.

Example for precedence graph:

draw precedence graph for below transaction schedule.

T1	T2	T3	T4
Read(x)			
	Read(x)		
Write(x)			
		Read(y)	
	Read(y)		
	Write(x)		
		Read(w)	
		Write(y)	
			Read(w)
			Read(z)
			Write(w)
Read(z)			
Write(z)			



As precedence graph is having cycles or closed loops, the givenschedule is not serializable.

Example of conflict serializability:

S2:R1(X), R2(X), R2(Y), W2(Y), R1(Y), W1(X)

Sol:

S21:R2(X), R1(X),R2(Y),W2(Y),R1(Y),W1(Y)

S22:R2(X),R2(Y),R1(X),W2(Y),R1(Y),W1(Y)

S23:R2(X),R2(Y),W2(Y),R1(X),R1(Y),W1(Y)

The schedule S2 derives 3 more schedules (s21,s22,s23) which is called **conflict equivalence**

Concurrency Control:

In case of concurrent instruction executions to preserve atomicity, isolation and serializability, we use 'lock-based' protocol like .

Types of Locks:

1. Binary locks
2. Shared /exclusive locks

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared(S)/exclusive(X)** − This type of locking mechanism differentiates the locks basedon their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Lock Compatibility Matrix –

- Lock Compatibility Matrix controls whether multiple transactions can acquire locks on the same resource at the same time.

	Shared	**Exclusive**
Shared	True	False
Exclusive	False	False

- If a resource is already locked by another transaction, then a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on item, no other transaction may hold any lock on the item.
- compatible locks held by other transactions have been released. Then the lock is

granted.

Locking protocols:

1. Simple lock based protocol

2. Conservative (or) pre-claim locking protocol.3.2-phase locking protocol
4. Strict 2 phase locking protocol

5. Rigorous 2 phase locking protocol

Simple lock based 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.

problems with simple locking are:

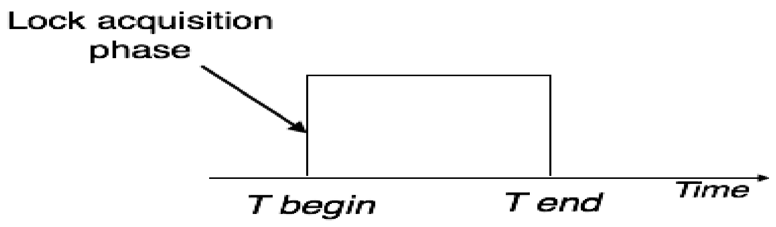
1. deadlocks

2. starvation

Conservative (or) pre-claim locking 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 locksit 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.

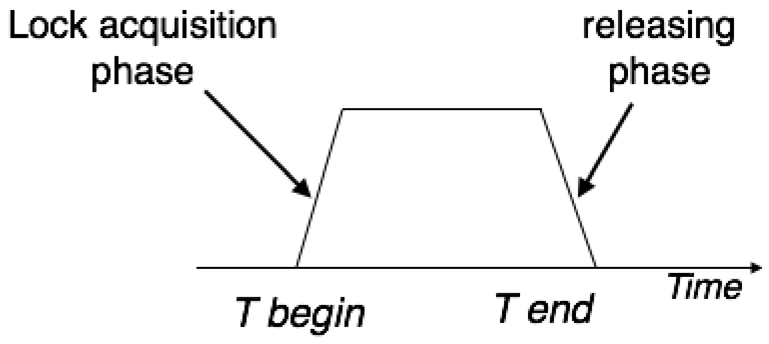


2-phase locking protocol:

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 third phase, the transaction cannot demand any new locks; it only releases the acquired locks.

This protocol can be divided into two phases,
1. In Growing Phase, a transaction obtains locks, but may not release any lock.
2. In Shrinking Phase, a transaction may release locks, but may not obtain any lock.



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.

Types of Two – Phase Locking Protocol

Following are the types of two – phase locking protocol:

1. Strict Two – Phase Locking Protocol
2. Rigorous Two – Phase Locking Protocol
3. Conservative Two – Phase Locking Protocol

Strict Two-Phase Locking:

1. If a transaction want to read any value it can refers to a shared lock
2. If a transaction to write any particular value it can refers to an exclusive locks
3. A shared lock acquire by multiple transaction at same time.
4. An exclusive lock can be requested by only one transaction at a time on any data item.
5. Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
6. It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.

phases in strict 2 phase locking:

phase 1: The first phase of Strict-2PL is same as 2PL i.e. when the transaction starts executing, it seeks permission for the locks it requires.

phase 2: After acquiring all the locks in the first phase, the transaction continues to execute normally.

phase 3: But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holdsall the locks until the commit point and releases all the locks at a time.

Note: It releases only all exclusive locks but not shared locks after a transaction is committed .
This protocol is not free from deadlocks

DEAD LOCKS:

Consider two transaction t1 and t2.if t1 holds lock on data item x and t2 holds lock on data item y now t1 refers lock over y & t2 request lock over x then **deadlock situation** occur when none of the transaction are ready to release locks on x ,y.

- The following two techniques can be used for deadlock handling(prevention):
    1. wait-die
    2. wait-wound

1. wait-die:

- In wait die technique the older transaction waited in queue &younger will die.
    - The older transaction waits for the younger if the younger has accessed the granule first.
    - The younger transaction is aborted (dies) and restarted if it tries to access a granule after an older concurrent transaction.
- The wait-die based on time stamp of the transaction request for conflicting resources.
  1)Ts(t1)<ts(t2):t1 will wait in a queue&t2 will die/abort.
  2)ts(t1)>ts(t2):t2 will be waiting in queue & t1 will abort/die

For example:

Suppose that transaction T_{22}, T_{23}, T_{24} have time-stamps 5, 10 and 15 respectively. If T_{22}requests a data item held by T_{23} then T_{22} will wait. If T_{24} requests a data item held by T_{23}, then T_{24} will berolled back.

2. wait wound technique:

- It based on time stamp of transaction request
    - It is a preemptive technique for deadlock prevention. It is a counterpart to the wait-die scheme. When Transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j, otherwise T_j is rolled back (T_j is wounded by T_i)

    For example:

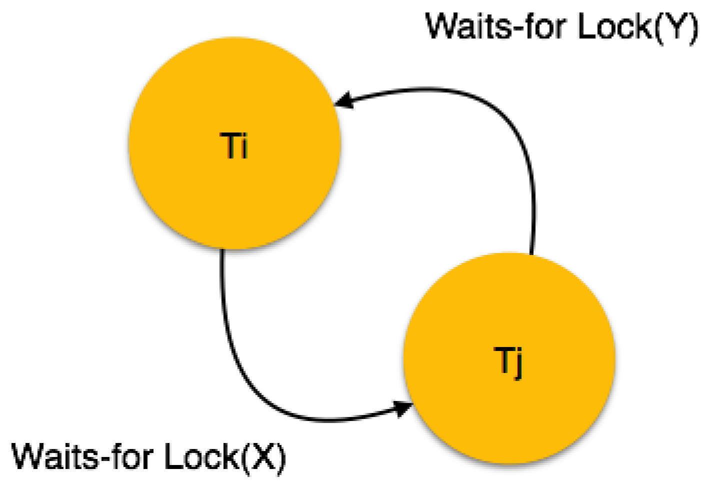
    - Suppose that Transactions T_{22}, $T_{23,}$ T_{24} have time-stamps 5, 10 and 15 respectively . If T_{22}requests a data item held by T_{23}, then data item will be preempted from T_{23} and T_{23} will be rolled back. If T_{24} requests a data item held by T_{23}, then T_{24} will wait.
- Here the younger transactions are made to wait in queue& older transaction going to abort.
  1) Ts (t1) <ts (t2): t2 will be in waiting state &t1 in abort.
  2) Ts (t1)>ts (t2):t1 will be in waiting & t2 in abort.

DEAD LOCK AVOIDANCE:

Wait for graph:

- We use this technique for dead lock avoidance.

- This is a simple method available to track if any deadlock situation may arise.
- For each transaction entering into the system, a node is created.
- When a transaction T_i requests for a lock on an item, say X, which is held by some other transaction T_j, a directed edge is created from T_i to T_j. If T_j releases item X, the edge between them is dropped and T_i locks the data item.
- The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches −

- First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the **victim** and the process is known as **victim selection.**