**Transaction**

An action, or series of actions, carried out by a single user or application program, which reads or updates the contents of the  
database.

Read only ( Database failure causes

PARTIALLY COMMITTED, which occurs after the final statement has been executed.  
At this point, it may be found that the transaction has violated serializability (see Section 20.2.2) or has violated an integrity constraint and the transaction has to be aborted.  
Alternatively, the system may fail and any data updated by the transaction may not have  
been safely recorded on secondary storage. In such cases, the transaction would go into  
the FAILED state and would have to be aborted. If the transaction has been successful,  
any updates can be safely recorded and the transaction can go to the COMMITTED state.  
n FAILED, which occurs if the transaction cannot be committed or the transaction is aborted  
while in the ACTIVE state, perhaps due to the user aborting the transaction or as a result  
of the concurrency control protocol aborting the transaction to ensure serializability.  
)

**Transaction properties:**

There are properties that all transactions should possess.   
The four basic, or so-called ACID properties

***Atomicity*** The ‘all or nothing’ property. A transaction is an indivisible unit that is  
either performed in its entirety or is not performed at all. It is the responsibility of the  
recovery subsystem of the DBMS to ensure atomicity.

***Consistency*** A transaction must transform the database from one consistent state to  
another consistent state. It is the responsibility of both the DBMS and the application  
developers to ensure consistency.

***Isolation*** Transactions execute independently of one another. In other words, the  
partial effects of incomplete transactions should not be visible to other transactions. It  
is the responsibility of the concurrency control subsystem to ensure isolation.

***Durability*** The effects of a successfully completed (committed) transaction are permanently recorded in the database and must not be lost because of a subsequent failure.  
It is the responsibility of the recovery subsystem to ensure durability.

|  |  |
| --- | --- |
| **Concurrency**  **control** | The process of managing simultaneous operations on the database without having them interfere with one another. |

**Repeated from paper answers:**

When two or more users are accessing the database simultaneously and at least one is updating data, then it is called concurrency control problem.

Examples: lost update problem, the uncommitted dependency problem, and the inconsistent analysis problem.

1. The lost update problem: when an apparently successfully completed update operation by one user can be overridden by another user.
2. The uncommitted dependency problem occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed.
3. The problem of inconsistent analysis occurs when a transaction reads several values from the database but a second transaction updates some of them during the execution of the first.

**Scheduler:**

Special DBMS Program to establish the order of operations in which concurrent transactions are executed Interleaves the execution of database operations to ensure:

1. Serializability

Isolation of Transactions

The Scheduler

Bases its actions on Concurrency Control Algorithms

(Locking / Time Stamping)

Ensures the CPU is used efficiently (Scheduling Methods)

Facilitates Data Isolation 🡪 Ensure that 2 transactions do not update the same data at the same time

Recoverability:

|  |  |
| --- | --- |
| **Recoverable**  **schedule** | A schedule where, for each pair of transactions T*i* and T*j*, if T*j* reads a data item previously written by T*i*, then the commit operation of T*i,* precedes the commit operation of Tj. |

Concurrency Control Algo/Techniques:

1. Locking

A Transaction “locks” a database object to prevent another object from modifying that object.

1. Time-Stamping

Assign a global unique time stamp to each transaction.

1. Optimistic

Assumption that most database operations do not conflict.

Locking Types:

**Binary**

Binary Locks – Lock with 2 States.

Locked – No other transaction can use that object.

Unlocked – Any transaction can lock and use object.

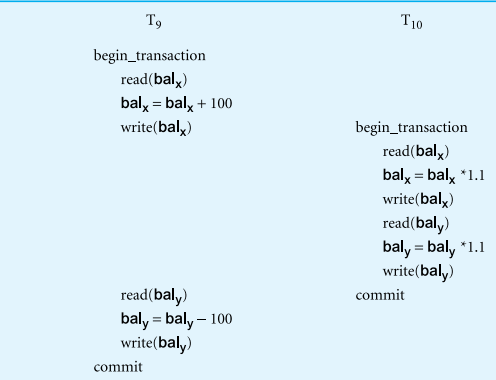
All Transactions require a Lock and Unlock Operation for Each Object Accessed (Handled by DBMS).

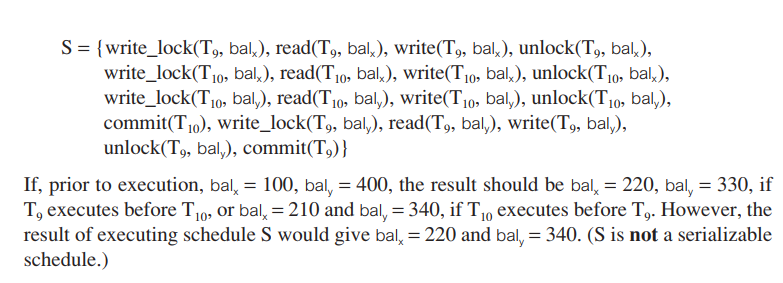
Eliminates Lost Updates.

Too Restrictive to Yield Optimal Concurrency Conditions.

**Shared lock**: If a transaction has a shared lock on a data item, it can read the item but not update it. If a transaction has an exclusive lock on a data item, it can both read and update the item

**Exclusive lock**: If a transaction has an exclusive lock on a data item, it can both read and update the item.



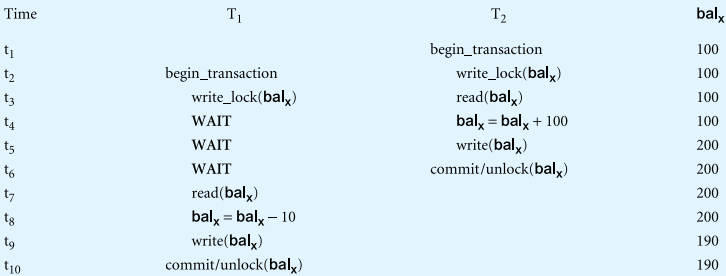


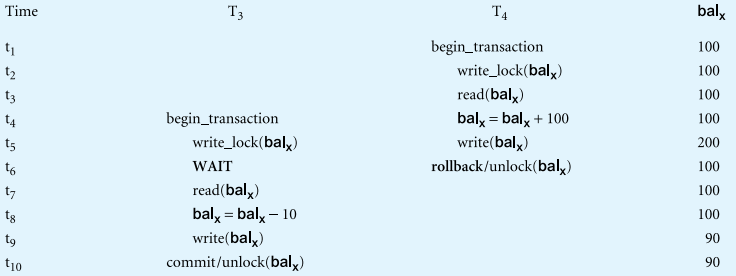
**Two Phased Locking:**

A transaction follows the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.

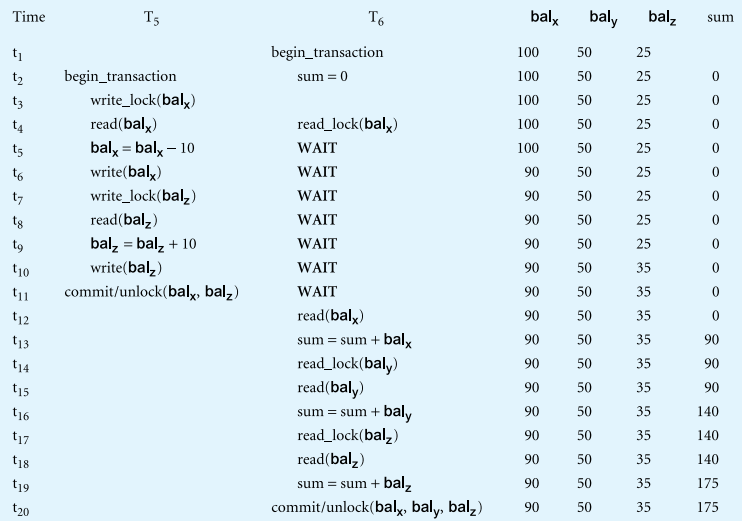
According to the rules of this protocol, every transaction can be divided into two phases:  
first a **growing phase**, in which it acquires all the locks needed but cannot release any  
locks, and then a **shrinking phase**, in which it releases its locks but cannot acquire any  
new locks.

**PREVENTING LOST UPDATE PROBLEM:**

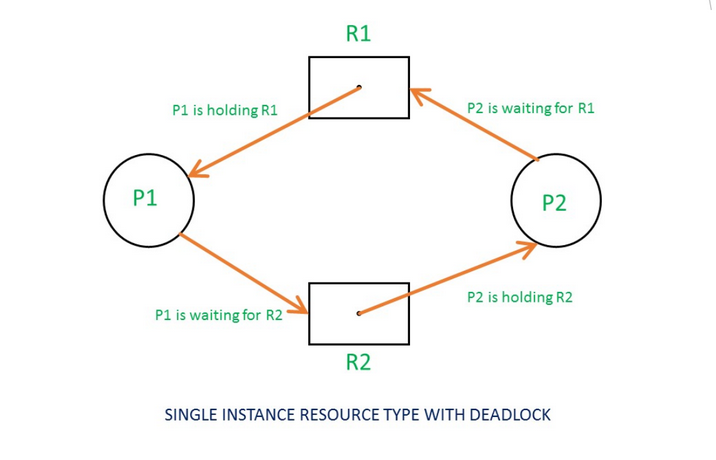


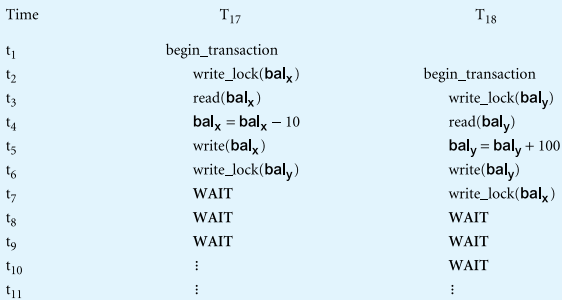
**Preventing the uncommitted dependency problem using 2PL:**   


**Preventing the inconsistent analysis problem using 2PL:**



Deadlock: An impasse that may result when two (or more) transactions are each waiting for locks to be released that are held by the other.





There are three general techniques for handling deadlock: **timeouts, deadlock prevention, and deadlock detection and recovery.**

**N.I: Schedule** A sequence of the operations by a set of concurrent transactions  
that preserves the order of the operations in each of the individual  
transactions.