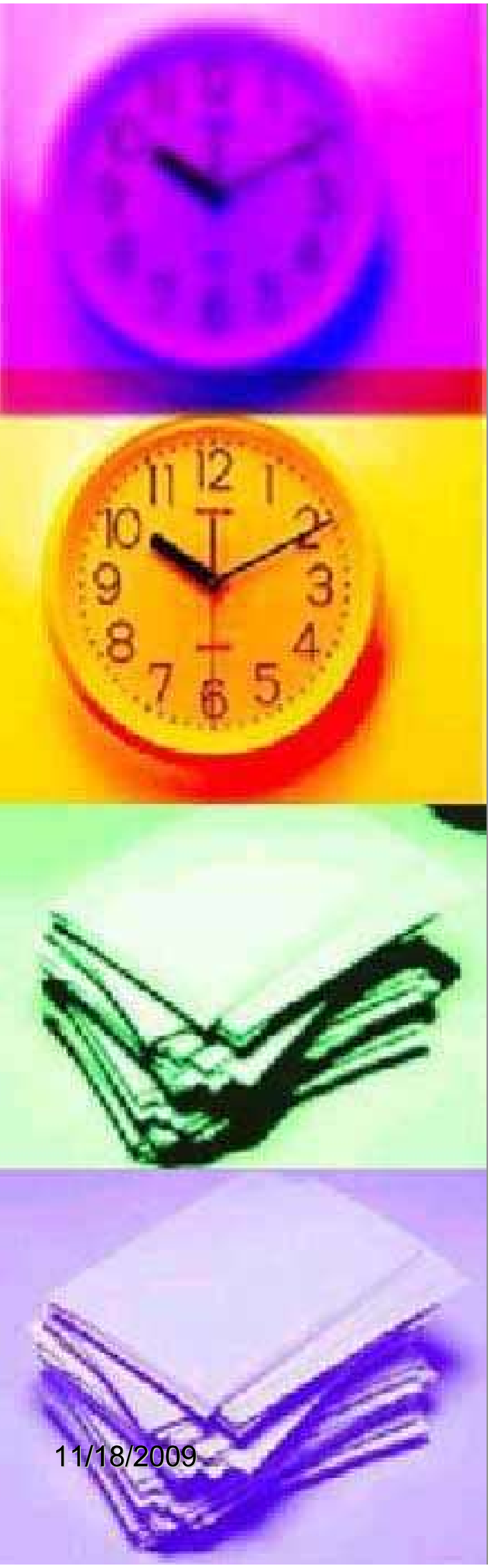
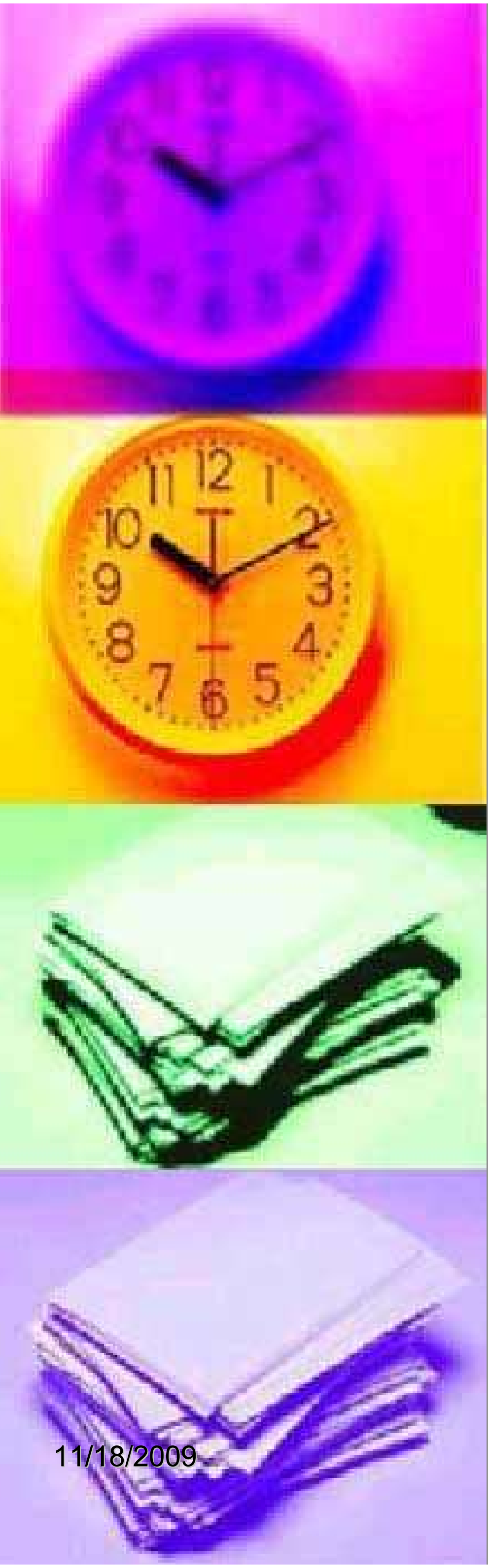


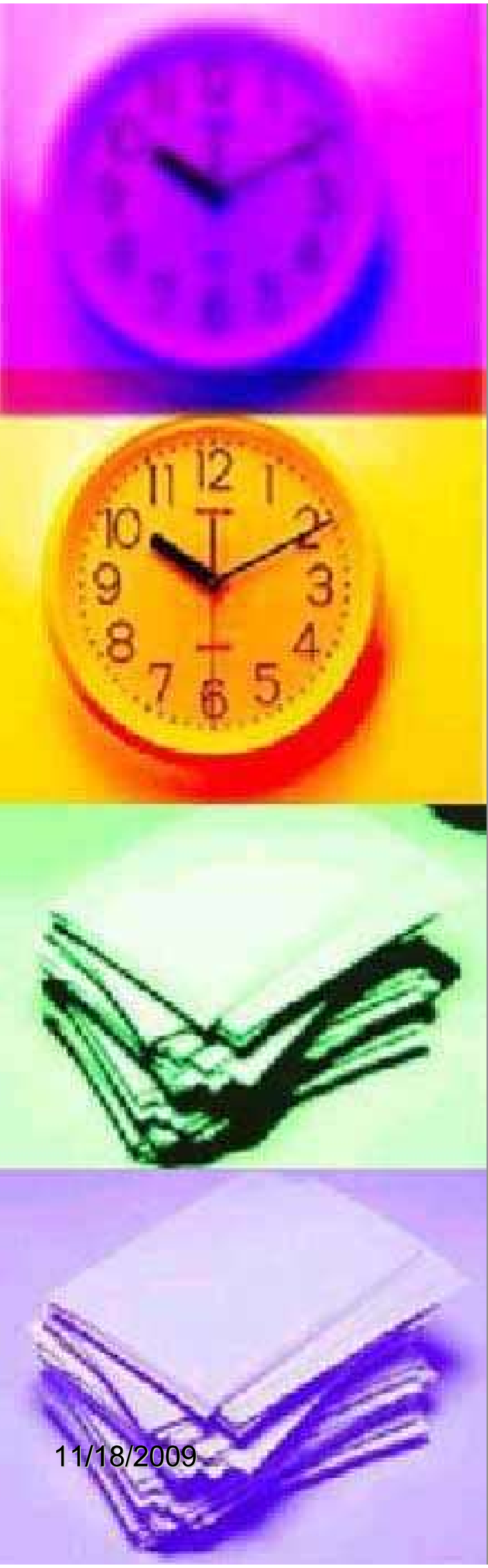
Contents

- **Transaction Concept**
- **Transaction State**
- Implementation of Atomicity and Durability
- Concurrent Executions
- Serializability
- Recoverability



Transaction Concept

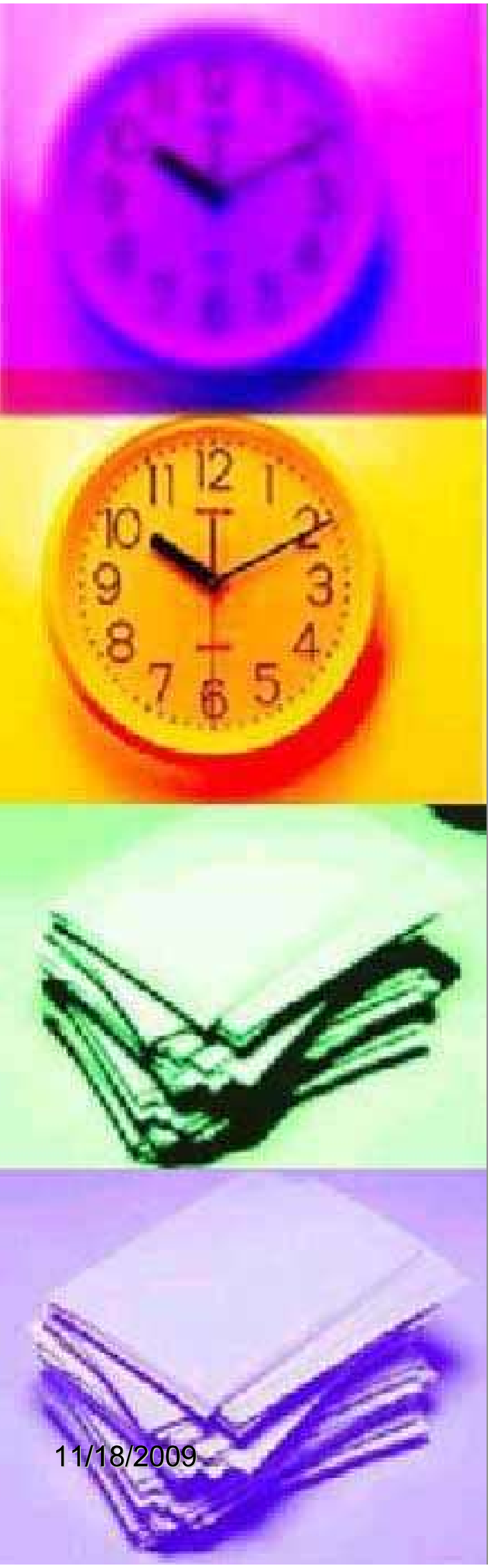
- A ***transaction*** is a *unit* of program execution that accesses and possibly updates various data items.
- A transaction must see a consistent database.
- During transaction execution the database may be inconsistent.
- When the transaction is committed, the database must be consistent.
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions



ACID Properties

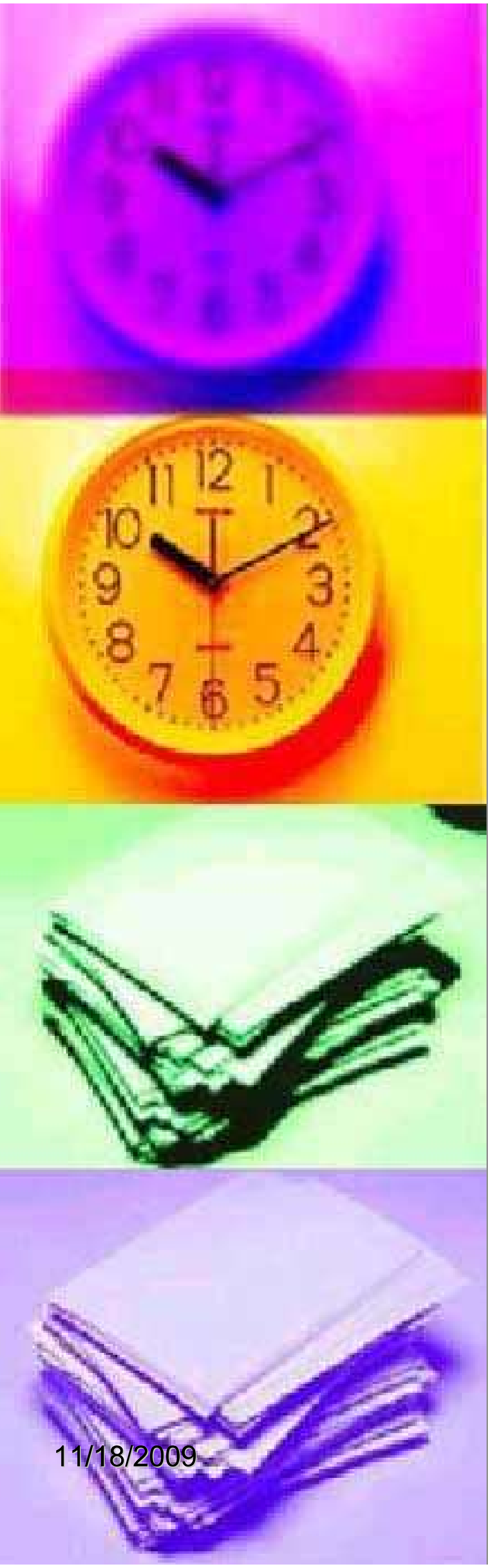
To preserve integrity of data, the database system must ensu

- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are
- **Consistency.** Execution of a transaction in isolation preserves th consistency of the database
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions
  - That is, for every pair of transactions T_i and T_j, it appears to T that either T_j, finished execution before T_i started, or T_j startec execution after T_i finished.



ACID Properties

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



Example of Fund Transfer

- Transaction to transfer $50 from account *A* to account *B*:

A 7000

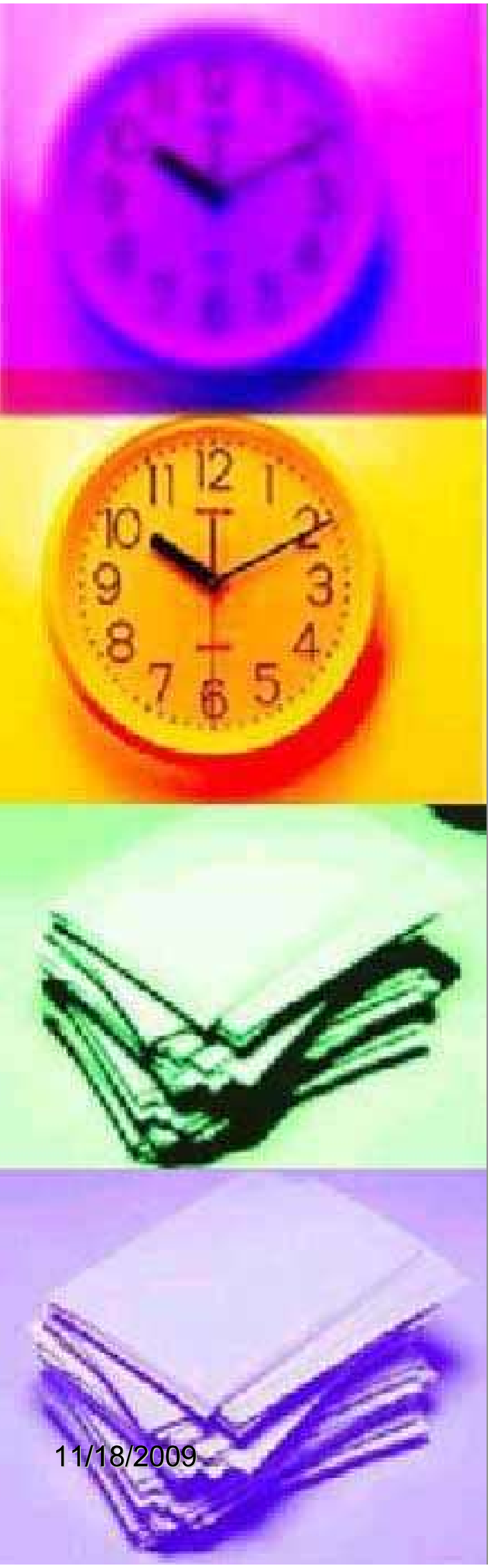
1. **read**(*A*) 3000
2. *A* := *A* – 50 2950
3. **write**(*A*) F
4. **read**(*B*) 4000
5. *B* := *B* + 50 4050 7000
6. **write**(*B*)

commit

2950
4000
6950

- Consistency requirement – the sum of *A* and *B* is unchanged by the execution of the transaction.

- Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.

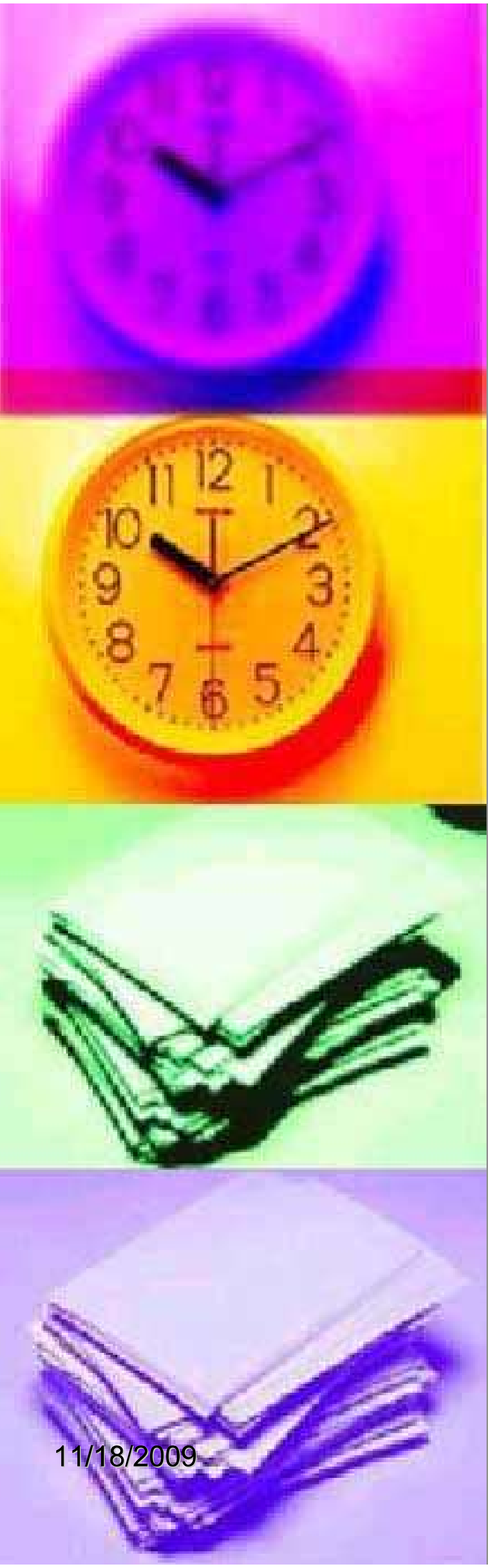


Example of Fund Transfer (Cont.)

- Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist despite failures

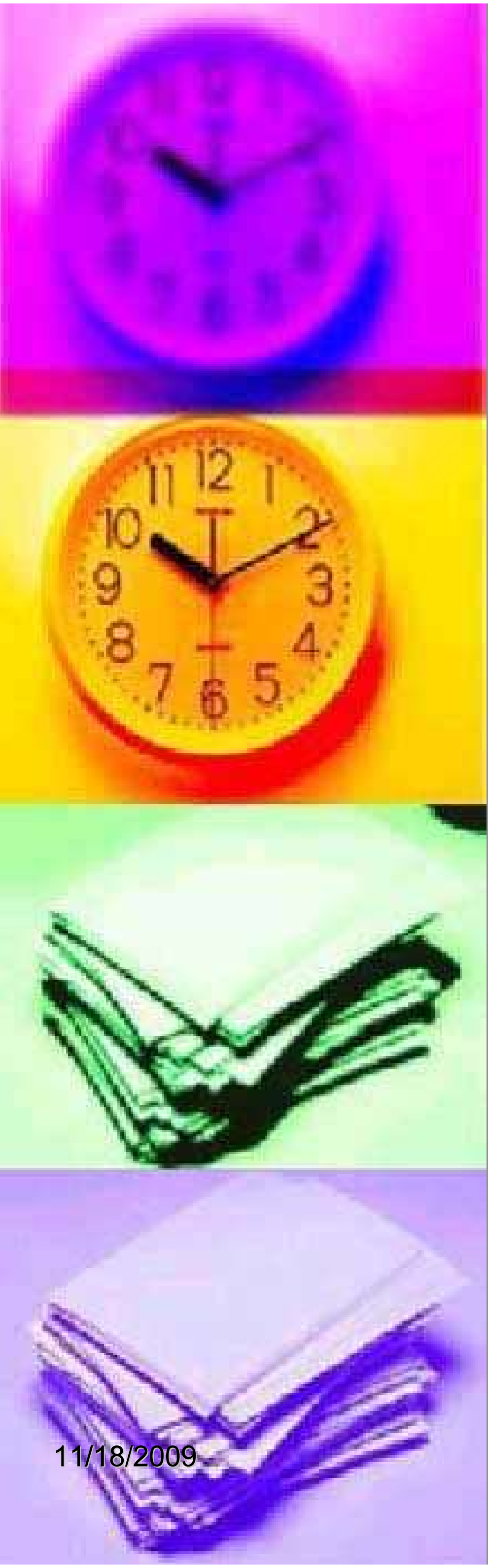
- Isolation requirement — if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database
(the sum $A + B$ will be less than it should be).

- Can be ensured trivially by running transactions ***serially***, that is one after the other. However, executing multiple transactions concurrently has significant benefits, as we will see.



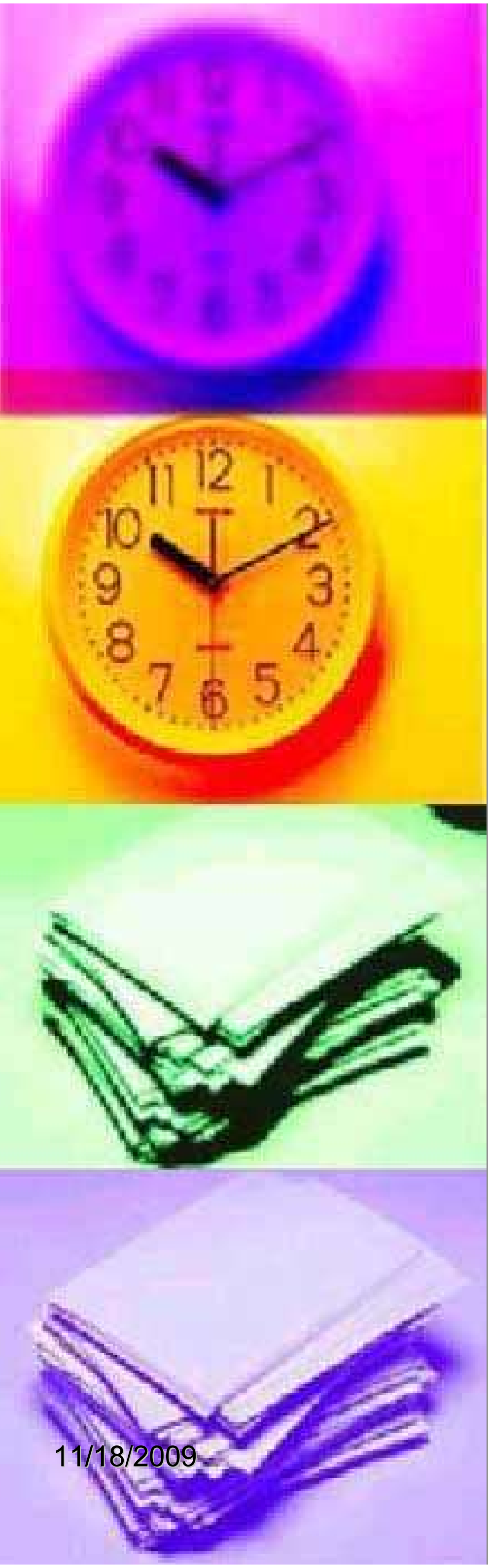
Why Concurrency Control is needed ?

- To overcome The Isolation Problems



Dirty Read or Temporary Update or Write Read Conflict

- Occurs when one transaction updates a database item and then the transaction fails for some reason.
- The update item is accessed by another transaction before it is changed back to the original value.



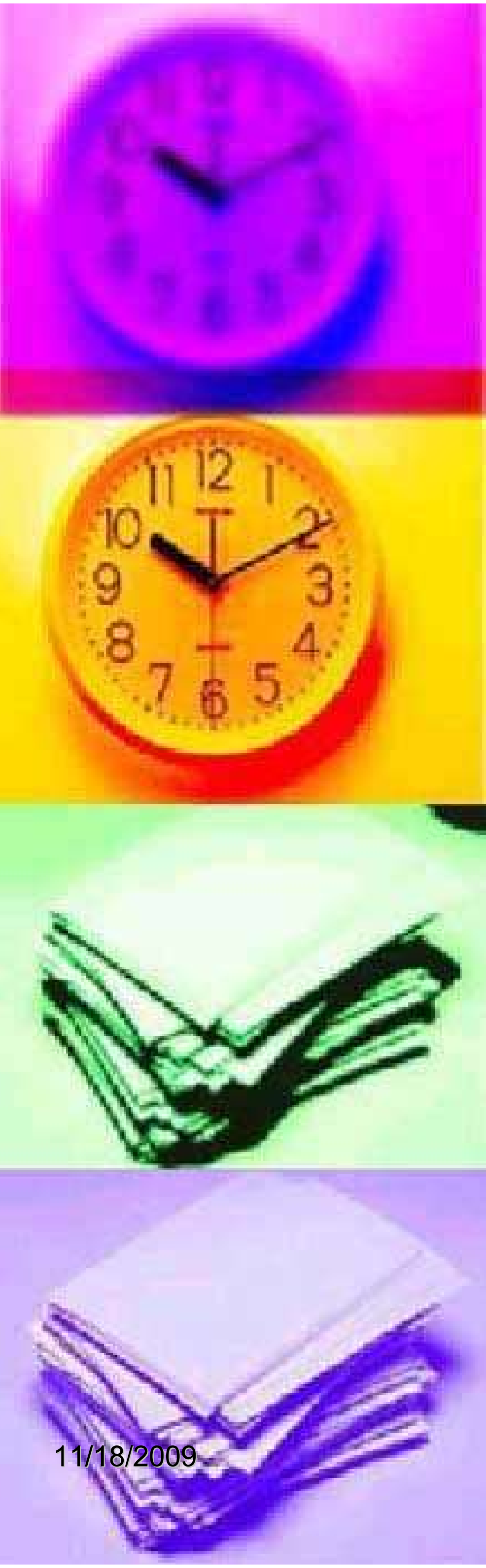
Example:

Name	Sal	Dept
Marry	300	CSE
Scott	500	IT
John	1000	ECE

10 am : Start TX1
10.10 am: set mary's sal to 500
10.20 am : commit

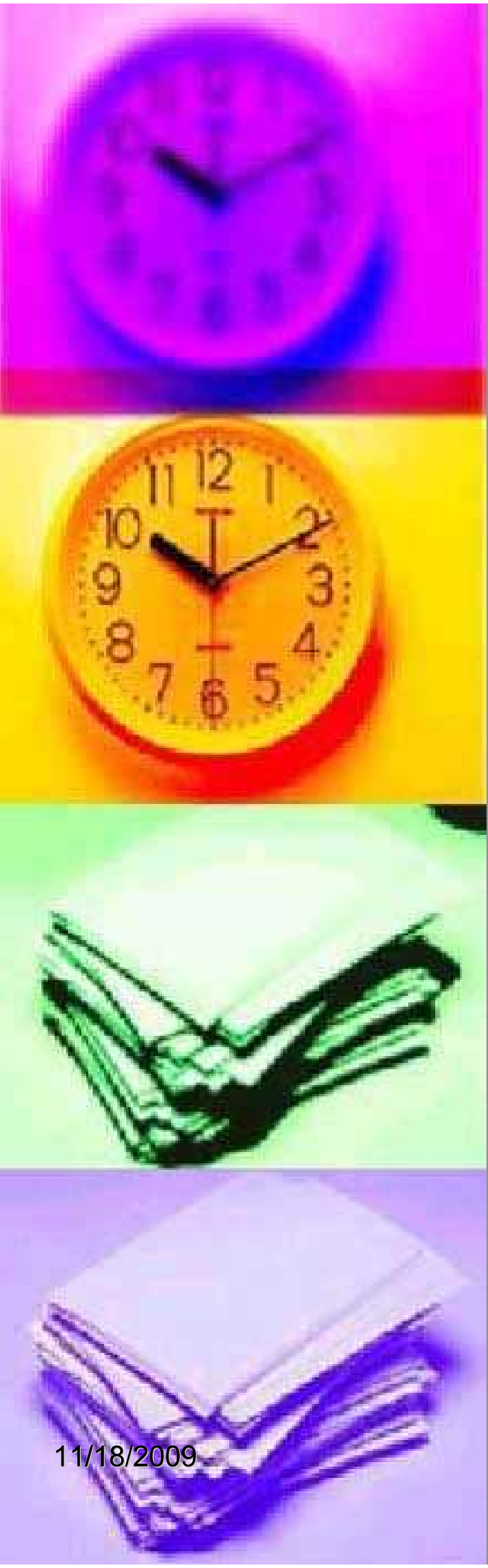
10.15 am : Start TX2
10.16 am: read all sal

T1	T2
Write (X)	
	Read(X)
Commit	



Incorrect Summary Problem

- If one Tx is calculating an aggregate summary function on a no of records while other Tx are updating some of these records,
- the aggregate function may calculate some values before they are updated



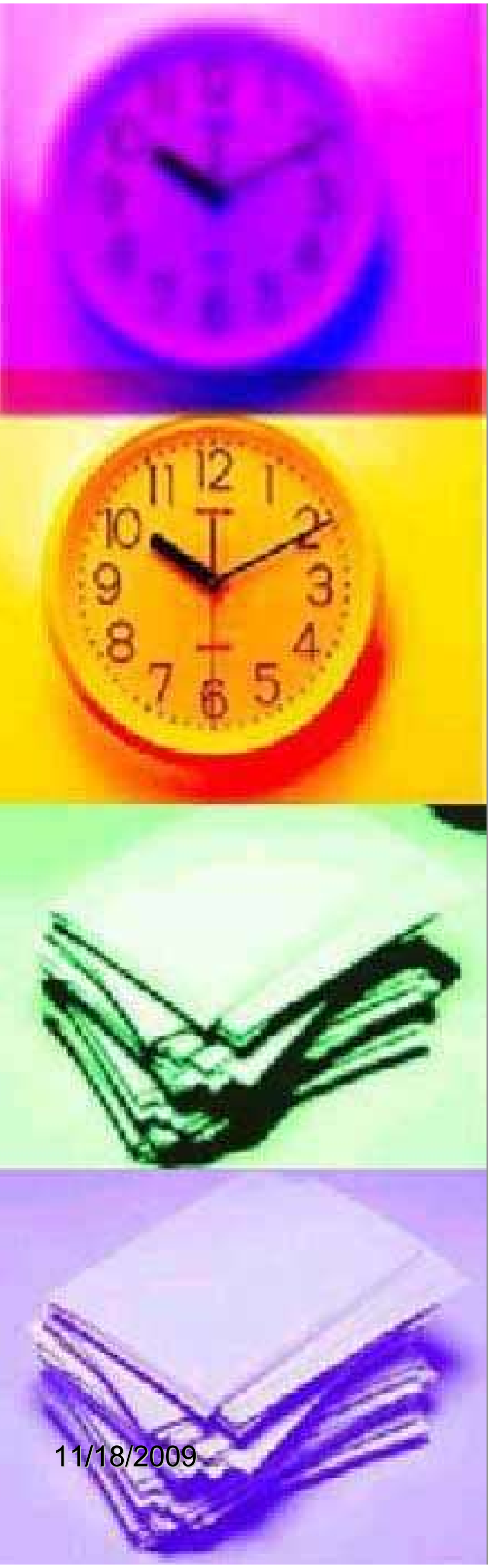
Example:

Name	Sal	Dept
Marry	300	CSE
Scott	500	IT
John	1000	ECE

10 am : Start TX1
10.05 am: set mary's sal to 500
10.08 am : set scott's sal to 1000

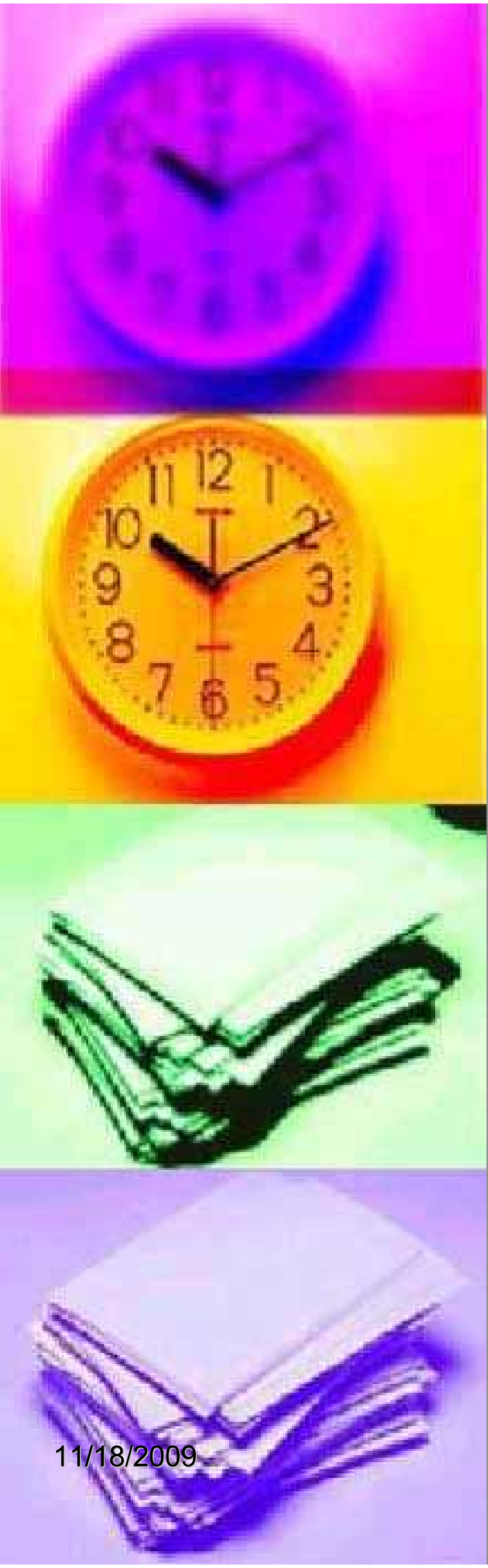
T1	T2
Write1(x)	
	Write2(x)
Write1(x)	Write2(x)

10.06 am : Start TX2
10.07 am: sum all sal (500+500+1000)
10.08 am: sum all sal (500+1000+1000)



Unrepeatable Read

- A Tx reads an item twice and the item is changed by another Tx1 between the two reads



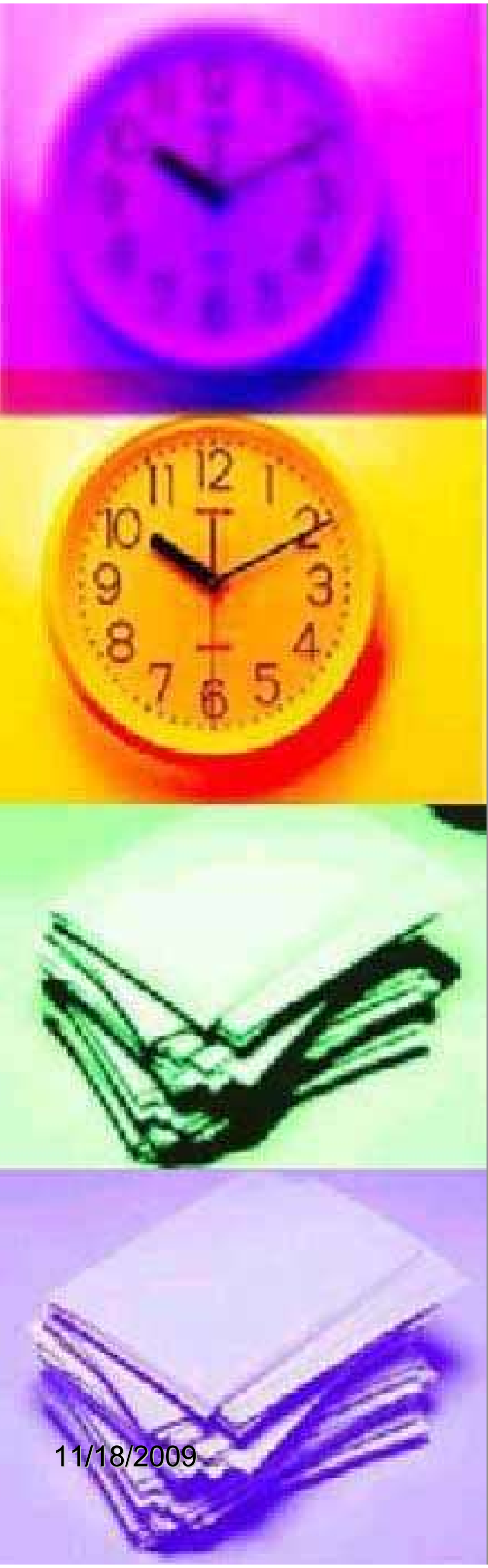
Example:

Name	Sal	Dept
Marry	300	CSE
Scott	500	IT
John	1000	ECE

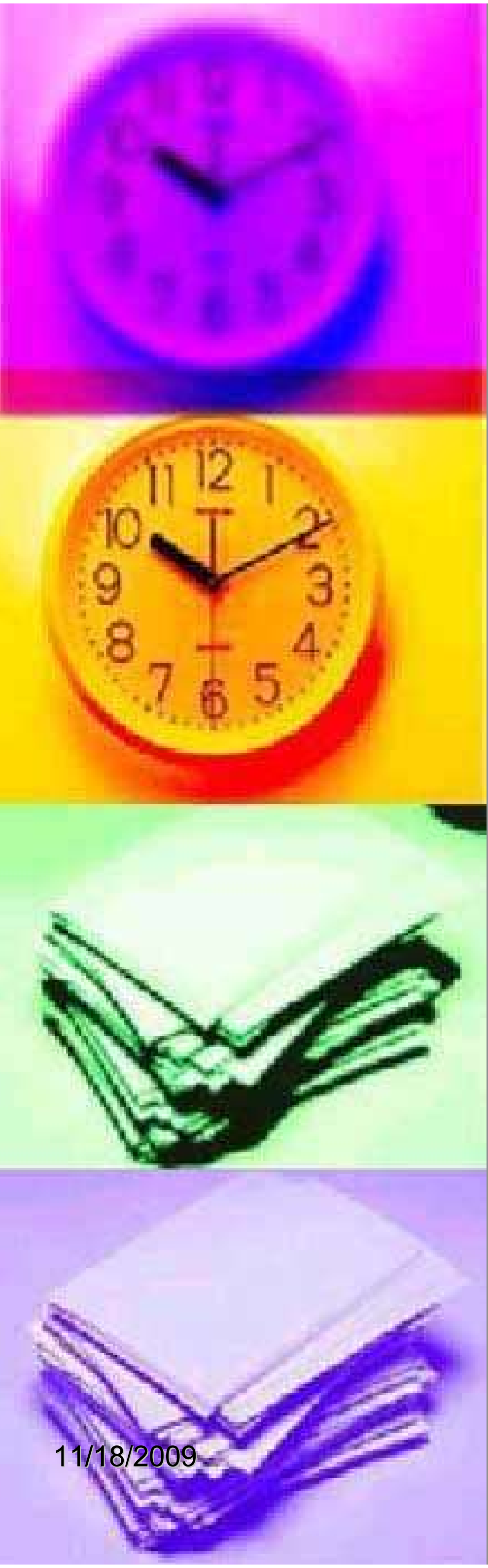
10 am : Start TX1
10.05 am:read all sal
10.15 am :read all sal

T1	T2
read1 (x)	
	write2(x)
	commit
read1(x)	

10.06 am : Start TX2
10.07 am:set marys
Sal to 700
10.08 am : commit



Repeatable Read (RW conflict)



Example:

Name	Sal	Dept
Marry	300	CSE
Scott	500	IT
John	1000	ECE

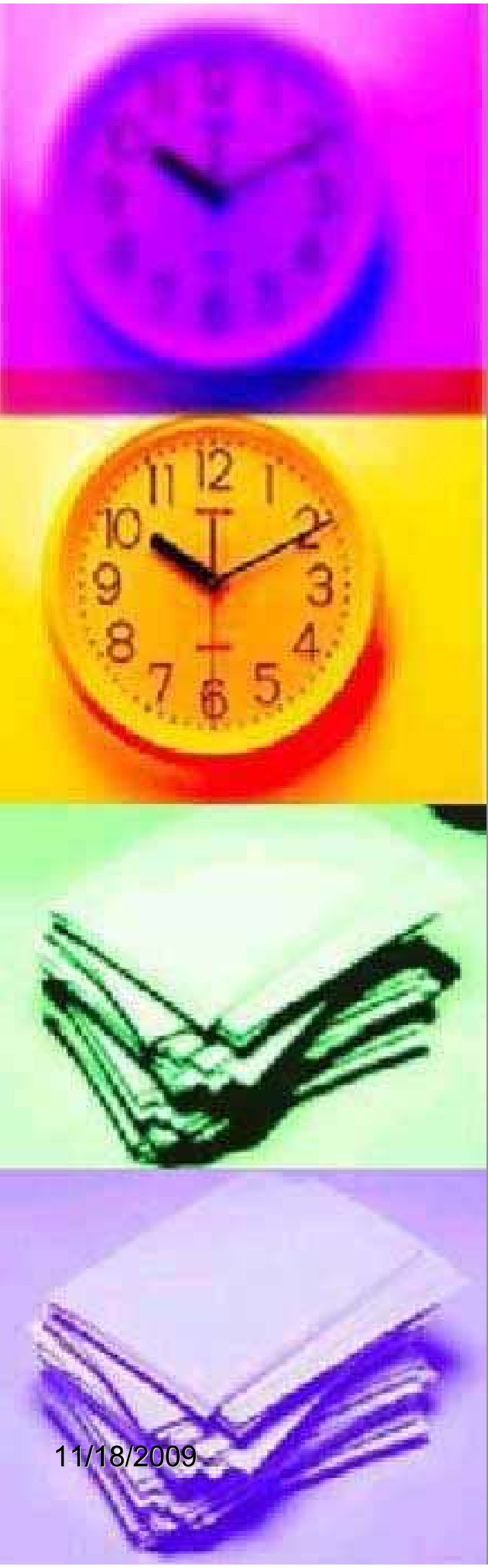
10 am : Start TX1
10.05 am:read all sal
10.15 am :read all sal

10.06 am : Start TX2
10.07 am:
insert (‘mira’,’600’,’HR’)

10.08 am : commit

T1	T2
read1 (x)	
	write2(x)
	commit
read1(x)	

Unstable Cursor :



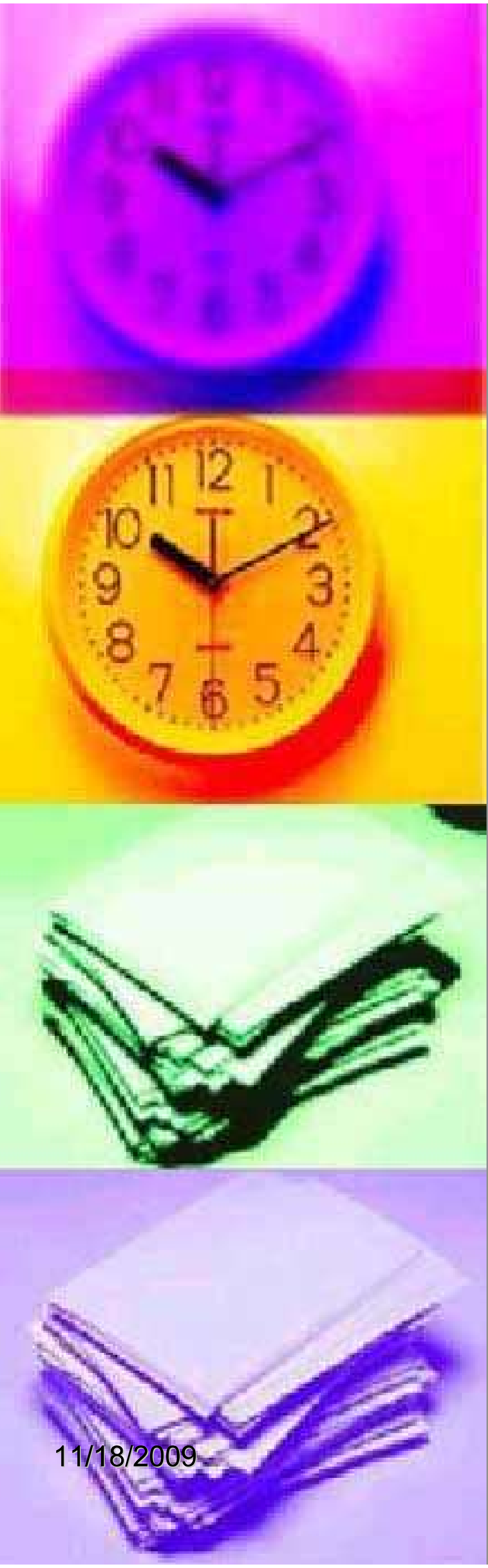
Example:

Name	Sal	Dept
Marry	300	CSE
Scott	500	IT
John	1000	ECE

10 am : Start TX1
10.05 am:
set mary’s sal to 500
10.15 am :read all sal

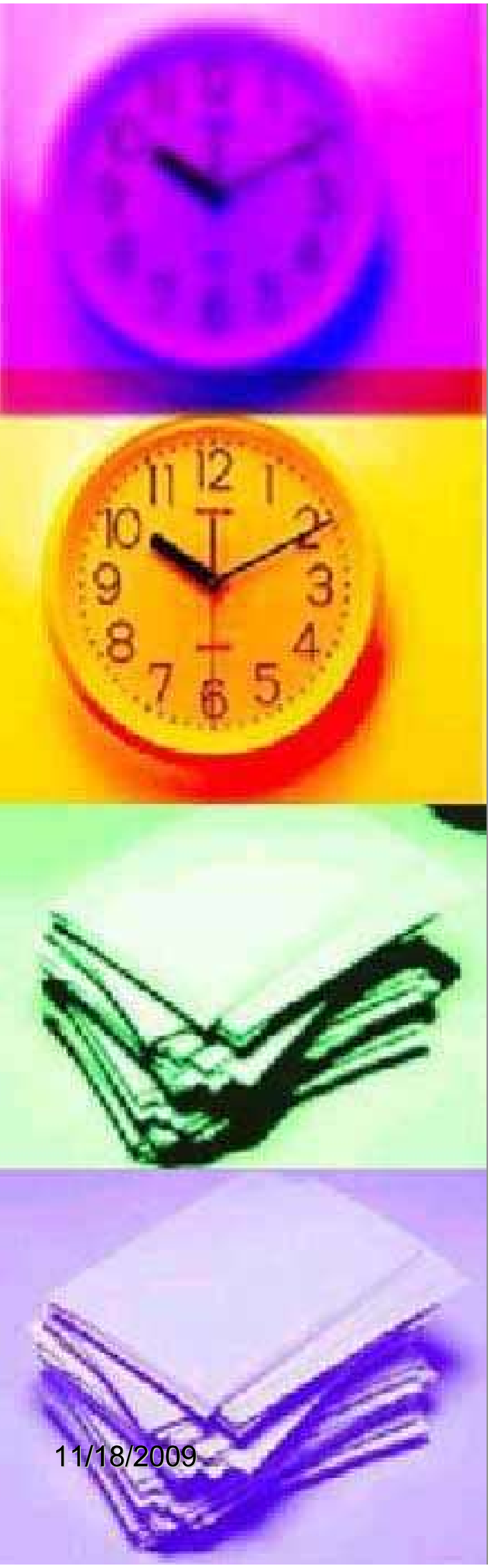
T1	T2
write1(x)	

10.10 am : Start TX2
10.11 am: drop sal column
10.12 am : commit



Lost Update Problem

- Occurs when two Tx's access the same database items have their operations interleaved in a way that makes the value of some database items incorrect



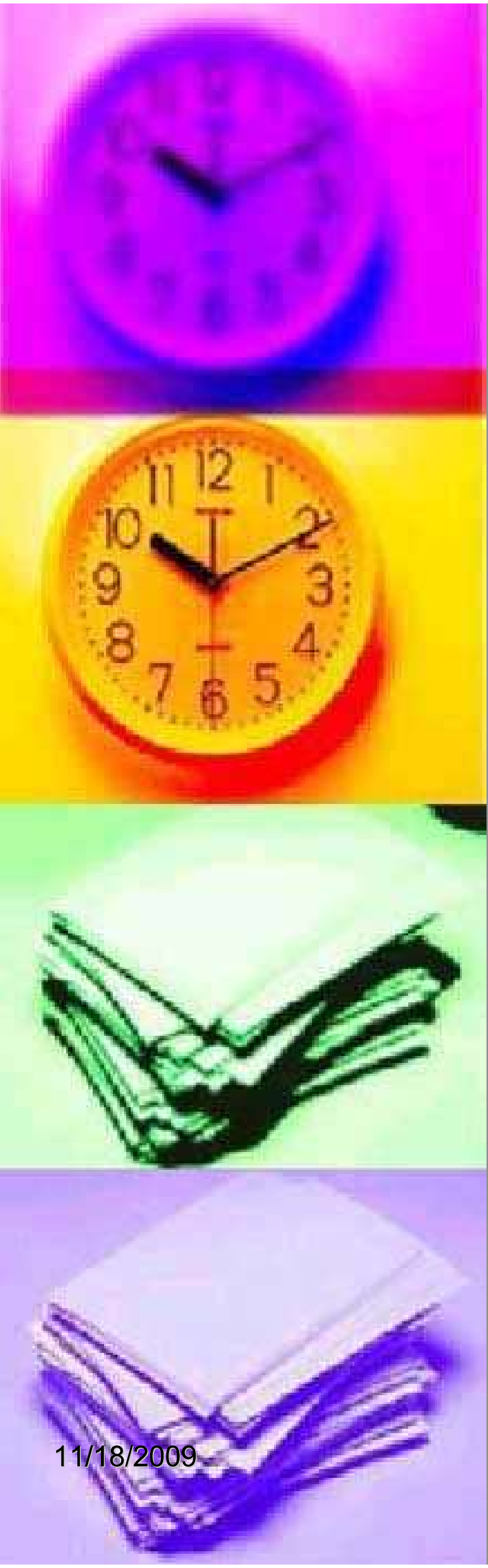
Example:

Name	Sal	Dept
Marry	300	CSE
Scott	500	IT
John	1000	ECE

10 am : Start TX1
10.10 am:
set mary's sal to 500
10.11 am :commit

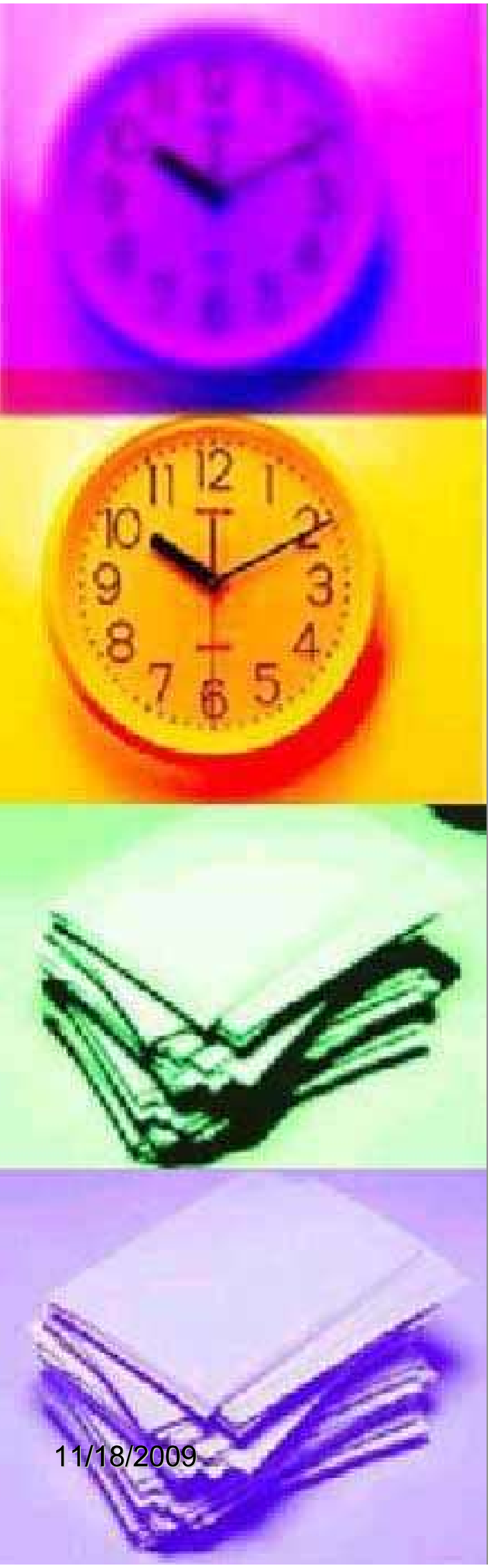
T1	T2
	Write2(x) commit2
Write1(x) commit1	

10.05 am : Start TX2
10.06 am: set mary's sal to 600
10.07 am : commit



Transaction State

- **Active,** the initial state; the transaction stays in this state while it is executing
- **Partially committed,** after the final statement has been executed.
- **Failed,** after the discovery that normal execution can no longer proceed.
- **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 – only if no internal logical error
  - kill the transaction
- **Committed,** after *successful completion*.



Transaction Operations

- BEGIN Transaction
- READ or WRITE
- END Transaction
- COMMIT Transaction
- ROLLBACK (or ABORT)

Transaction State (Cont.)