



TRANSACTIONS

Database Management System

SUJAN TAMRAKAR

Transactions

- ▶ A transaction is a small unit of a program and it may contain **several** low-level tasks.
- ▶ It is a **collection of operations** that form a single (atomic) logical unit of work.
- ▶ DB system **must ensure proper execution** of transactions **despite failures**.
- ▶ Transaction consists of all operations executed between the **begin** transaction & **end** transaction.
- ▶ A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as **ACID properties** – in order to ensure accuracy, completeness, and data integrity.

Transactions

- ▶ Various operations:
 - ▶ Begin transaction
 - ▶ Read or Write
 - ▶ End transaction
 - ▶ Commit transaction
 - ▶ successful end of transaction
 - ▶ Rollback or Abort
 - ▶ undoing transaction because of some failure or unsuccessful operation

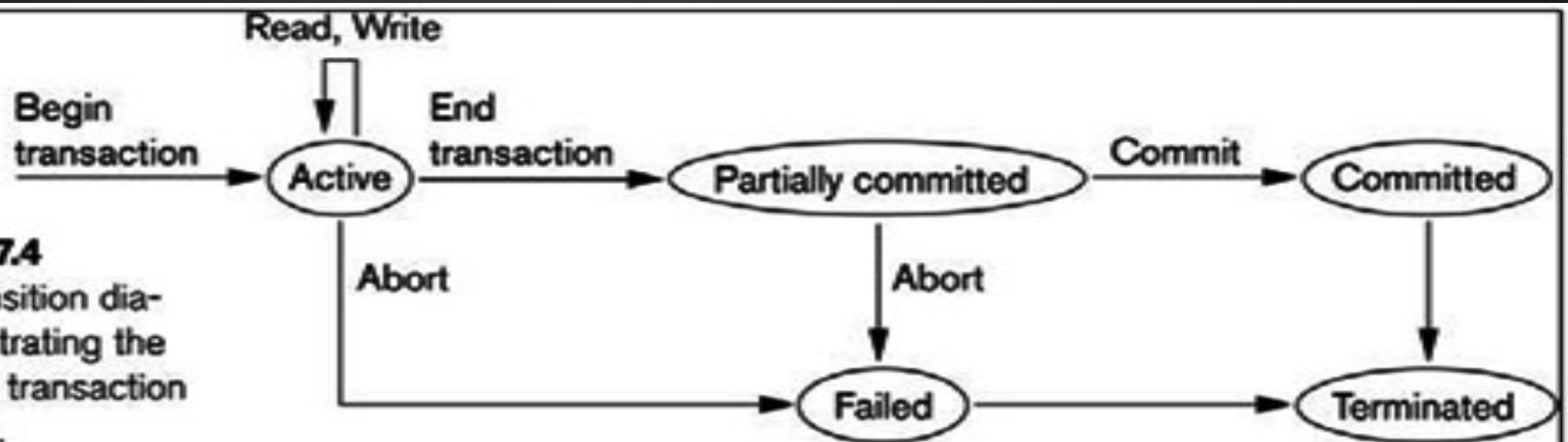


Figure 17.4
State transition diagram illustrating the states for transaction execution.

Transactions

- ▶ To ensure **integrity** of data, following properties of transactions should be maintained;
 1. **A**tomicity
 2. **C**onsistency
 3. **I**solation
 4. **D**urability

Atomicity: This property states that a transaction must be treated as an atomic unit, that is, **either all of its operations are executed or none**. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

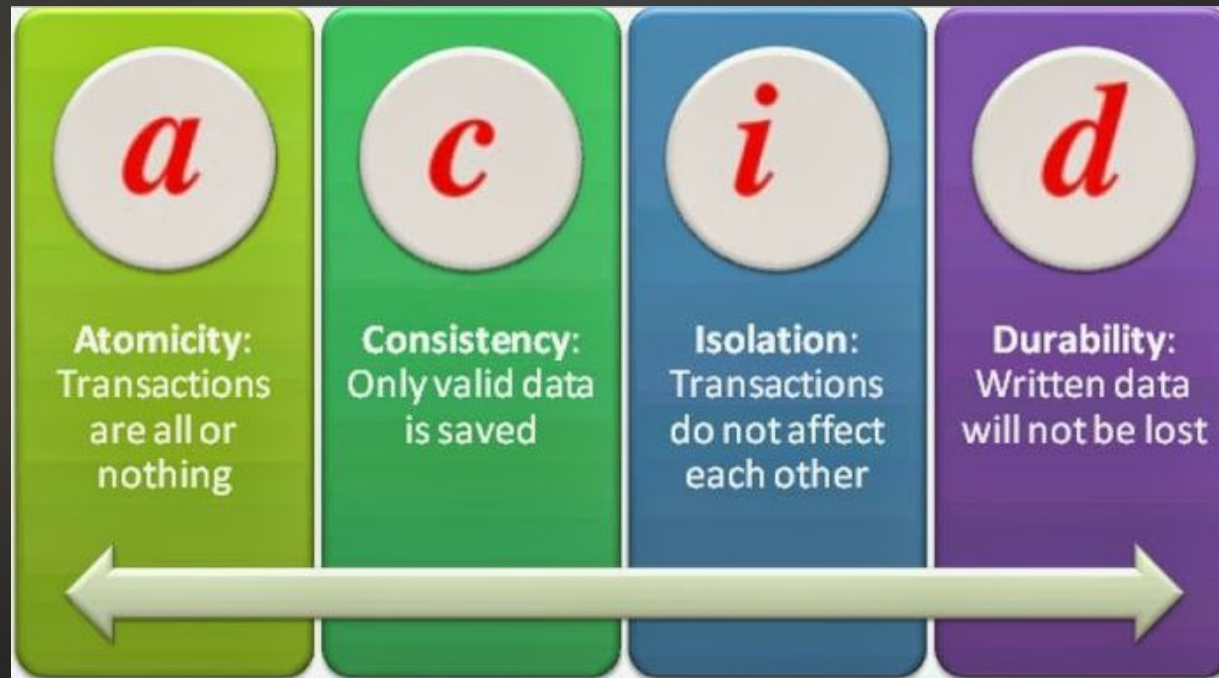
Transactions

Consistency: The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Isolation: In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Transactions

Durability: The database should be **durable** enough to hold all its latest updates **even if the system fails or restarts** . If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.



Schedules & Serializability

- When multiple transactions are being executed by the operating system in a multiprogramming environment, there are **possibilities** that **instructions of one transactions are interleaved with some other transaction**.
- **Schedules** represent the **chronological order** in which instructions are executed in the system.
- A schedule can have **many transactions in it**, each comprising of a number of instructions/tasks.
- A schedule for a set of transactions must consist of all those transactions and **must preserve the order** in which the instructions appear in each individual transaction.

Schedules & Serializability

- Example: write(A) must appear before read(B)

T1	T2
Read(A)	
A:= A-50	
Write(A)	
Read(B)	
B:= B+50	
Write(B)	
	Read(A)
	temp:= A * 0.1
	A:= A – temp
	Write(A)
	Read(B)
	B:= B + temp
	Write(B)

T1 transaction transfers Rs. 50 from account A to B.

T2 transaction transfers 10% of balance from account A to B.

*Fig. Schedule 1: **Serial schedule** in which T1 is followed by T2.*

A schedule in which transactions are aligned in such a way that **one transaction is executed first**. When the first transaction completes its cycle, then the next transaction is executed. **Transactions are ordered one after the other**. This type of schedule is called a **serial schedule**, as transactions are executed in a serial manner.

Schedules & Serializability

- Example: write(A) must appear before read(B)

T1	T2
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)

Two transactions executing concurrently. Occurrence of parallel execution of transactions.

Fig. A **concurrent schedule** equivalent to schedule .

Schedules & Serializability

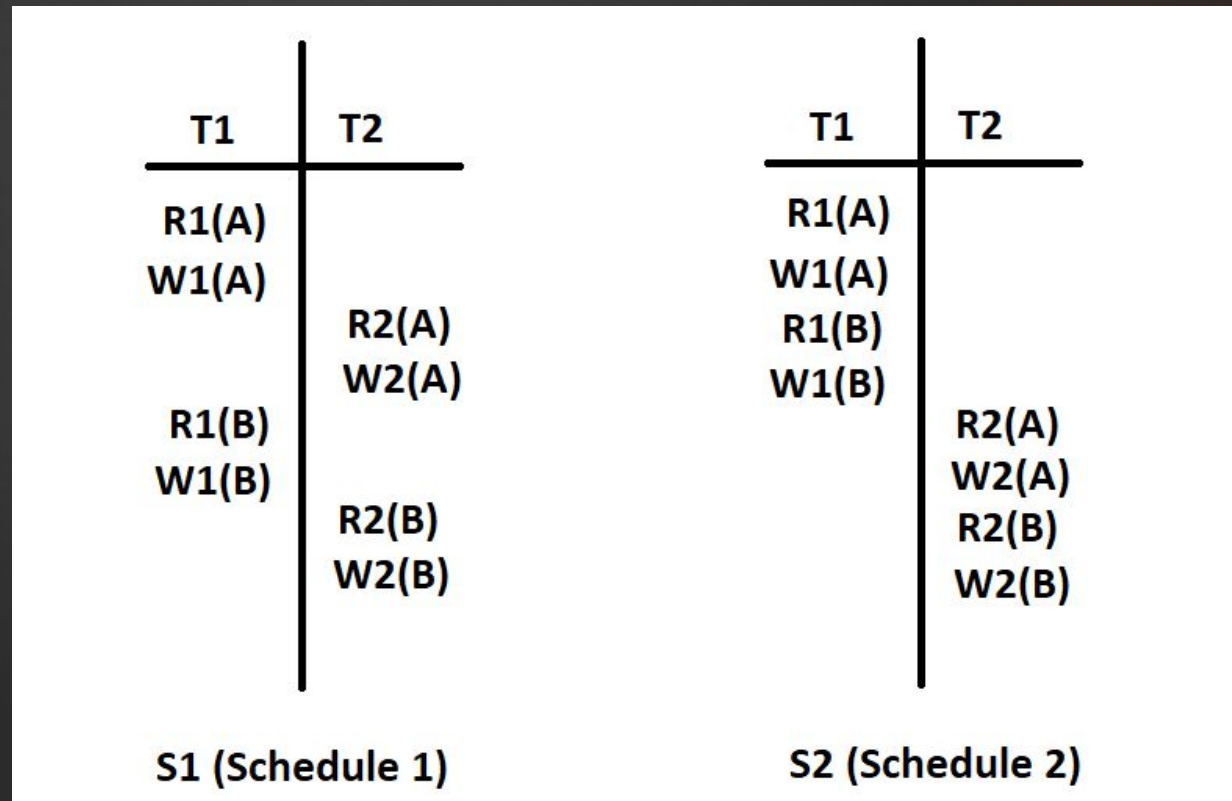
Serializability:

- ▶ is the process which prevents inconsistency & aims to find a non-serial schedule that allows transaction to execute concurrently without being interfered with one another & produce a db state that could be produced by a serial schedule.
- ▶ Db system must control concurrent execution of transactions to ensure that db state remains constant.
- ▶ It is difficult to determine what operation transaction performs & how transactions interact. So, we shall not interpret the type of operations that transaction can perform but we consider only read & write operations.
- ▶ Thus, the only significant operations of a transaction from a scheduling point of view are its read and write instructions.

Schedules & Serializability

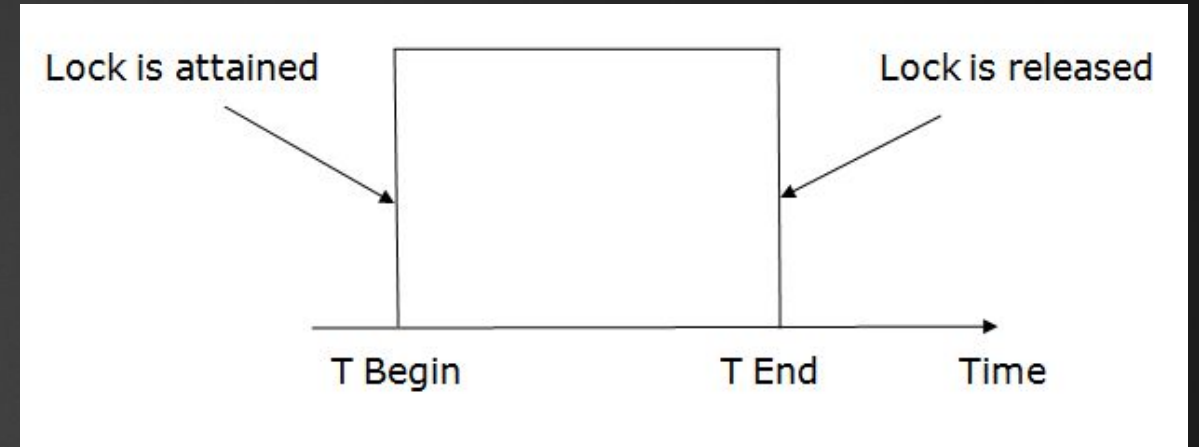
Serializability:

- It ensures that a schedule for executing **concurrent transactions** is **equivalent** to one that executes the transactions *serially* in some order.
- A schedule is called ‘correct’ if we can find a serial schedule that is ‘equivalent’ to it.



Lock based protocols

- Concept of locking for concurrency control



- While one transaction is accessing a data item, no other transaction can modify the data item

<i>State of the lock</i>	<i>Lock request type</i>	
	<i>Shared</i>	<i>Exclusive</i>
<i>Shared</i>	Yes	No
<i>Exclusive</i>	No	No

Lock based protocols


- ▶ Common method is to allow a transaction to **access data only if it is currently holding a lock** on that item.
- ▶ Data item may be locked in various types. Modes of locks are:

- ▶ **Shared:**

If transaction T1 has obtained a shared-mode lock on an item then T1 can read but cannot write on the item.

- ▶ **Exclusive:**

If transaction T1 has obtained an exclusive-mode lock on an item then T1 can both read & write.

<i>State of the lock</i>	<i>Lock request type</i> 	
	<i>Shared</i>	<i>Exclusive</i>
<i>Shared</i>	Yes	No
<i>Exclusive</i>	No	No

Lock based protocols

- ▶ Every transaction **should request a lock** in certain mode on data item depending upon its operation type.
- ▶ Transaction makes request to Concurrency Control Manager (**CCM**).
- ▶ Transaction can continue with operation **only after CCM grants the lock** to the transaction.
- ▶ At any time, **several *shared* mode locks can be held simultaneously** (by different transactions) on a particular data item.
- ▶ An **exclusive mode lock request has to wait until** the currently held shared mode locks are released.
- ▶ A transaction requests a shared lock on data item Q by executing the **lock S(Q)** instruction. Similarly, a transaction requests an exclusive lock through **lock X(Q)** instruction. A transaction can unlock a data item Q by **unlock(Q)** instruction.

Lock based protocols

- Transaction T1 has to wait until all **incompatible** locks held by other transactions have been released. If it accesses / requests for same compatible mode of Shared lock, then it is allowed to access it.

Ex: T1: lock-X(B)

read(B)

B:=B - 50

write(B)

unlock(B)

lock-X(A)

read(A)

A:=A+50

write(A)

unlock(A)

Fig. Transaction T1 with lock & unlock method

Lock based protocols

- ▶ A transaction **must hold a lock on a data item as long as it accesses that item.**

T1	T2
Lock-X(B)	
Read(B)	
B=B-50	
Write(B)	
	Lock-S(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

*T1 is waiting for
T2 to unlock A*

*T2 is waiting
for T1 to unlock
B*

Here, we see a state where neither of these transactions can ever proceed with its normal execution. This situation is called **Deadlock**.

When deadlock occurs, system must **rollback** one of the two transactions. On roll-backing, the locked state on any item will be unlocked & the other transaction can continue its operation.

Fig. Schedule creating deadlock situation



Thank you