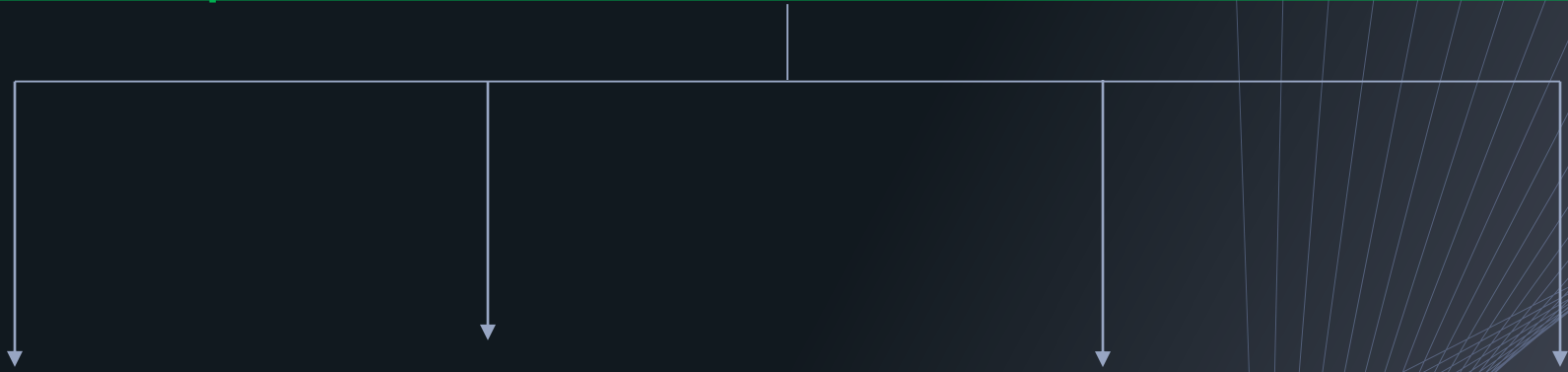
Chapter 21
Transaction Processing

References/Resources:
R. Elmasri, S.B. Navathe, “Fundamentals of Database Systems”, 6th Edition, Pearson Education

Types of problems aries due to concurrent transaction



Committed: Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that all the operations in the transaction are completed successfully and their effect is recorded permanently in the database,

Aborted: Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that if the transaction fails then the transaction does not have any effect on the database or any other transactions.

If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

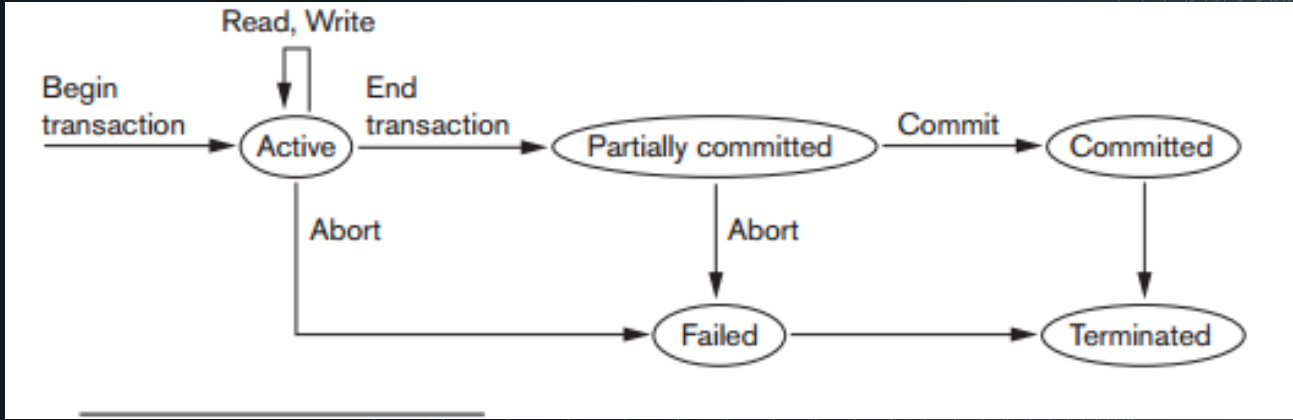
What causes a transaction to fails/ Types of failures

- A computer failure (system crash)
- A transaction or system error
- Local errors or exception conditions detected by the transaction
- Disk failure
- Physical problem and catastrophes

Transaction and its States

A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.
For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.

Transaction states diagram:



Recovery manager keeps track of the following operations:

- BEGIN_TRANSACTION. This marks the beginning of transaction execution.
- READ or WRITE. These specify read or write operations on the database items that are executed as part of a transaction.
- END_TRANSACTION. This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates concurrency control or for some other reason.

- COMMIT_TRANSACTION. This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.

- ROLLBACK (or ABORT). This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be **undone**.

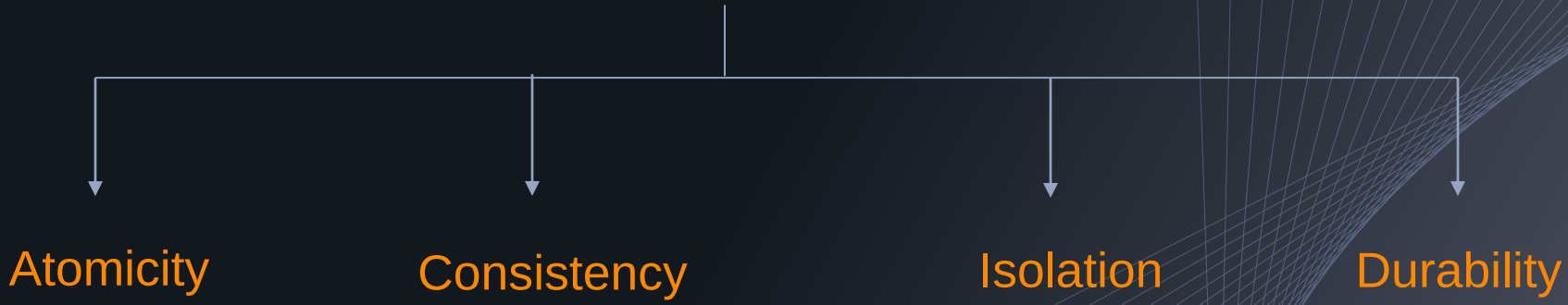
System Log

To be able to recover from failures that affect transactions, the system maintains a **log** to keep track of all transaction operations that affect the values of database items. The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.

The following are the types of entries—called log records—that are written to the log file and the corresponding action for each log record. T refers to a unique transaction-id

1. [**start_transaction**, T]. Indicates that transaction T has started execution.
2. [**write_item**, T, X, ***old_value***, ***new_value***]. Indicates that transaction T has changed the value of database item X from *old_value* to *new_value*.
3. [**read_item**, T, X]. Indicates that transaction T has read the value of database item X.
4. [**commit**, T]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [**abort**, T]. Indicates that transaction T has been aborted.

Four Desirable Properties of Transactions/ACID Property



Atomicity

A transaction is an atomic unit of processing; it is either performed entirely or not performed at all.

Consistency Preservation

A correct execution (completely executed from beginning to end without any interference from other transaction) of the transaction must take the database from one consistent state to another.

Isolation

A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently.

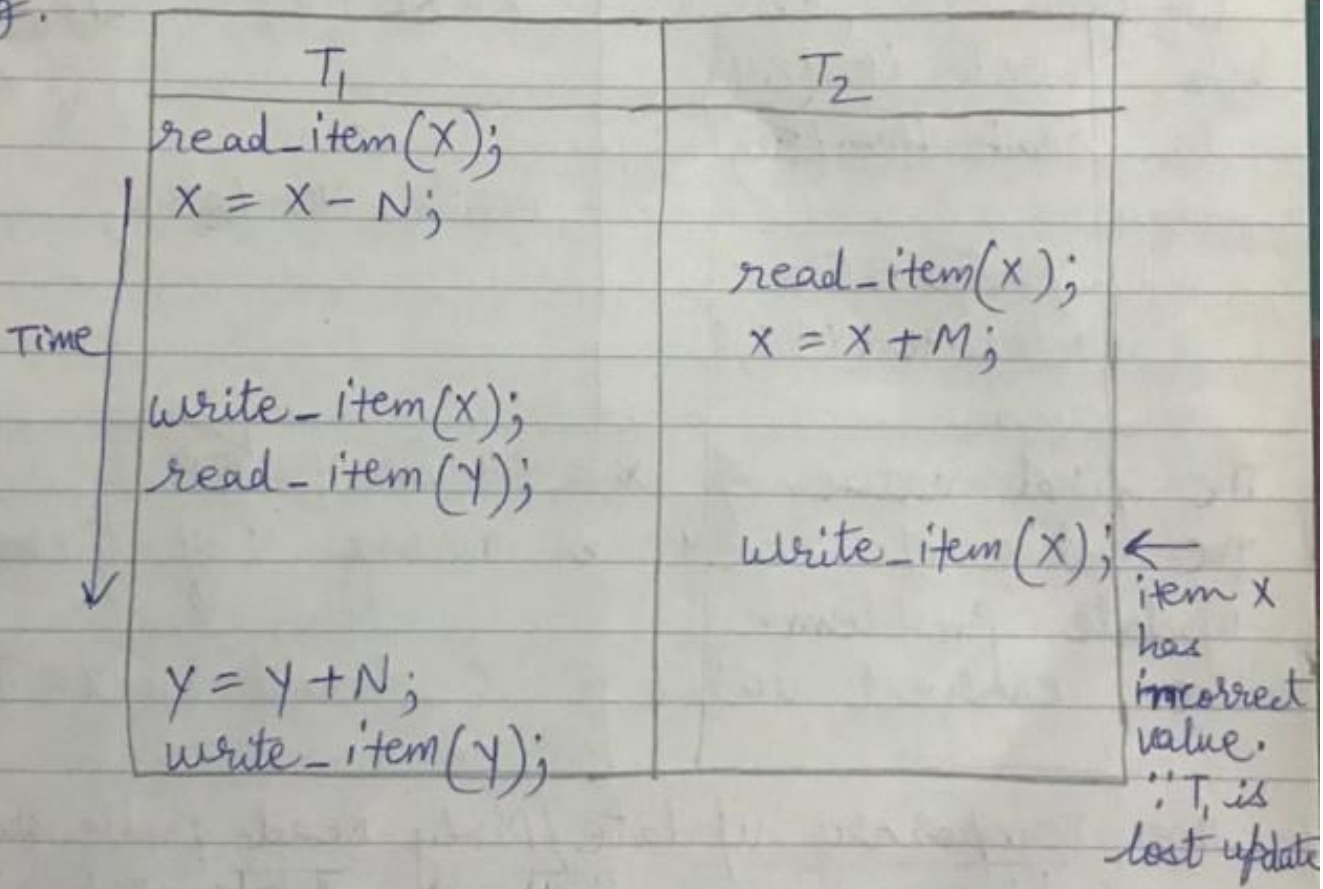
Durability

Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

(a) The Lost update Problem

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.

Eg:



$X = 80$, $N = 5$, $M = 4$, $Y = 50$

T_1	T_2
read_item (80);	
$X = 80 - 5$;	
(T_1 doesn't write new value of $X = 75$)	
	read_item (80); (since T_1 doesn't perform write. ∴ T_2 reads old value of $X = 80$)
	$X = 80 + 4$;
value update by T_2 → write_item (84)	
read_item (50)	
	write_item (84);
$Y = 50 + 5$;	
write_item (55);	

The final value of $X = 84$ & $Y = 55$.
The value of $X = 84$ is wrong ∵ of T_1 Lost update problem.
The correct value of X should by $X = 79$

(b). The Temporary update / Dirty Read Problem

This problem occurs when one transaction updates a database item & then the transaction fails for some reason. Meanwhile the updated item is read by another transaction before it is changed back to its original value.

Eg:

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);	

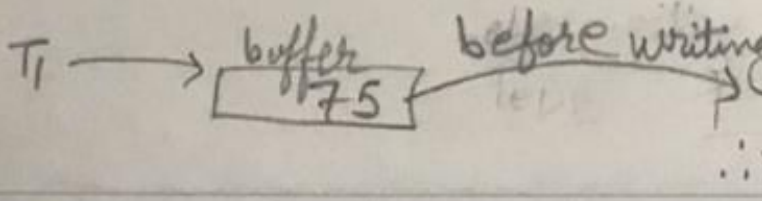
Time ↓

T_1 fails & must change the value of x back to its old value; T_2 reads the temp. incorrect value of x

$x = 80$, $N = 5$, $M = 4$, $Y = 50$

T_1	T_2
read_item(80);	
x = 80 - 5;	
write_item(75);	
	read_item(75);
	x = 75 + 4;
	write_item(79);
read_item(50);	

T_1 updates x & then fails. so the system will restore the original value of $x = 80$



in the database. Meanwhile T_2 reads the temp. value of $X = 75$ (which is not recorded permanently in the DB)

The value of X read by T_2 is called dirty data, ∵ it has been created by T_1 that has not ~~been~~ completed & committed yet. Hence, this problem is also called dirty read problem.

(c). The Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a no. of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated & others after they are updated.

Eg:

T_1	T_2
	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);
Y = Y+N;
write_item(Y);
	read_item(Y);
	Sum = Sum+Y;

T_3 : Total no. of reservations in flights (both)
T_3 reads X after subtraction & read Y before addition. ∴, T_3 is off by N.

(d). The Unrepeatable Read Problem

Unrepeatable read, where a transaction T reads the same item twice & the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item.