Database Management Systems (CSN-351)
Transactions

BTech 3rd Year (CS) + Minor + Audit

Instructor: **R**anita **B**iswas
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, Uttarakhand - 247 667, India

What is a Transaction?

A **transaction** is a unit of program execution that accesses and possibly updates various data items.

Example: Transfer 50 rupees from account A to account B

read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

ACID Properties

Atomicity: Either all operations of the transaction are reflected properly in the database, or none are.

ACID Properties

Atomicity: Either all operations of the transaction are reflected properly in the database, or none are.

Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

ACID Properties

Atomicity: Either all operations of the transaction are reflected properly in the database, or none are.

Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

Isolation: Even though multiple transactions may execute concurrently, the system guarantees that, each transaction is unaware of other transactions executing concurrently in the system.

ACID Properties

Atomicity: Either all operations of the transaction are reflected properly in the database, or none are.

Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

Isolation: Even though multiple transactions may execute concurrently, the system guarantees that, each transaction is unaware of other transactions executing concurrently in the system.

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

Transaction Operations

read(X): transfers the data item X from the database to a variable, also called X, in a buffer in main memory belonging to the transaction that executed the read operation.

write(X): transfers the value in the variable X in the main-memory buffer of the transaction that executed the write to the data item X in the database.

Example

read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

Storage Structure

- Volatile storage

Storage Structure

- Volatile storage
- Nonvolatile storage

Storage Structure

- Volatile storage
- Nonvolatile storage
- Stable storage

States of Transaction

Active: the initial state; the transaction stays in this state while it is executing.

States of Transaction

Active: the initial state; the transaction stays in this state while it is executing.

Partially committed: after the final statement has been executed.

States of Transaction

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.

States of Transaction

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 has been restored to its state prior to the start of the transaction.

States of Transaction

Active: the initial state; the transaction stays in this state while it is executing.

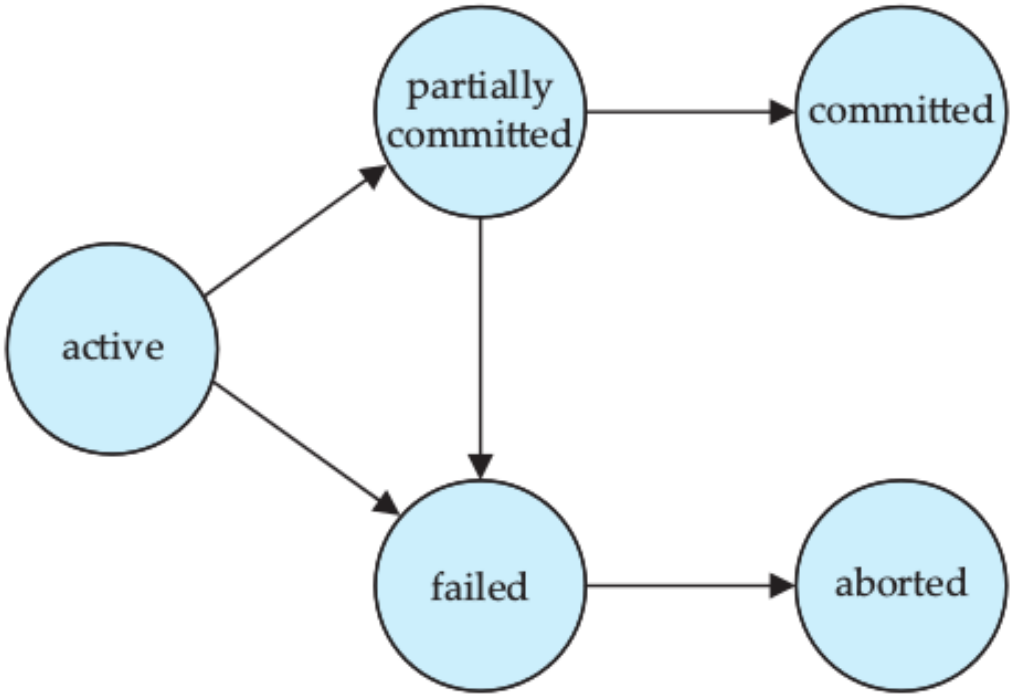
Partially committed: after the final statement has been executed.

Failed: after the discovery that normal execution can no longer proceed.

Aborted: after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed: after successful completion.

States of Transaction



Need for concurrency

Improved **throughput** and resource **utilization**

Need for concurrency

Improved **throughput** and resource **utilization**

Reduced **waiting time**

Example Transactions

T_1: read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

T_2: read(A);
$temp := A * 0.1$;
$A := A - temp$;
write(A);
read(B);
$B := B + temp$;
write(B).

Serial Schedule 1

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

Serial Schedule 2

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	

Concurrent Schedule 1

T_1	T_2
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)	
commit	
	read(B)
	$B := B + temp$
	write(B)
	commit

Concurrent Schedule 2

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	$B := B + temp$
	write(B)
	commit

Example Schedule

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)