Database Management System

Module 1: Transaction

Salim Diwani
College of Informatics and Virtual Education
University of Dodoma

Module Outline:

- Transaction Concept
- Transaction State
- Concurrent Execution

Transaction Concepts

A **transaction** is a set of operations used to perform logi
cal unit of **work**. Transaction generally represents chang
e in database. For example, transaction to transfer $50 f
rom account **A** to account **B**:

Transaction	Local buffer	Database
		A = 200
Read (A)	A = 200	A = 200
A: = A – 50	A = 200 – 50 = 150	A = 200
Write (A)	A = 150	A = 150
Read (B)	B = 300	B = 300
B: = B + 50	B = 300 + 50	B = 300
Write (B)	B = 350	350

Transaction Concepts

- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes.
  - Concurrent execution of multiple transactions.

In order to solve the above problem we going to look at **ACID** properties of transactions.

Required Properties of a Transaction

Atomicity Requirement

Though a transaction involves several low level operations but 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 database where the transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution or abortion or failure of the transaction.

Required Properties of a Transaction

Atomicity Requirement

Example

User A wants to withdraw $50 from his account and then transfer it to the account of user B. Each transaction (withdrawing $50 from account A and transferring $50 to account B) is counted as separate. If the first transaction (withdrawing $50) **fails** because (say) the server **crashes** during the transaction, user A cannot transfer the money to user B.

Required Properties of a Transaction

If the transaction fails after step 3 and before step 6, money will be **“lost”** leading to inconsistent database state.

- ➢failure could be due to software or hardware
- ➢The system should ensure that updates of a partially executed transaction are not reflected in the database.

1. read (A)

2. A: = A - 50

3. Write (A)

4. Read (B)

5. B: = B + 50

6. Write (B)

Required Properties of a Transaction

Consistency

This property states that after the transaction is finished, its database must remain in a **consistent state.** There must not be any possibility that some data is incorrectly affected by the execution of transaction. If the database was in a consistent state **before** the execution of the transaction, it must remain in consistent state **after** the execution of the transaction.

Required Properties of a Transaction

Consistency

Example

If user A wants to withdraw $1,000 from his account, b
ut only has a balance of $500, consistency will preven
t him from withdrawing money and the transaction will
be aborted.

Required Properties of a Transaction

- In example, the sum of A and B is unchanged by execution of transaction.
- In general, consistency requirements include:
  - **Explicitly specified integrity constraints**
    - Primary and Foreign keys
  - **Implicit integrity constraints**
    - Sum of balances of all accounts, minus sum of loan amounts must equal values of **cash in hand**.

Required Properties of a Transaction

- A transaction when starting to execute , must see a **consistent database.**
- During transaction execution the database may be temporarily inconsistent
- When transaction completes successfully the database must be in **consistent**
  - **Erroneous** transaction logic can lead to inconsistency.

Required Properties of a Transaction

Isolation Requirements

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.

Required Properties of a Transaction

Example

user A withdraws $100 and user B withdraws $250 from user Z’s account, which has a balance of $1,000. Since both A and B draw from Z’s account, one of the users is required to wait until the other user transaction is completed, avoiding inconsistent data.

If B is required to wait, then B must wait until A’s transaction is completed, and Z’s account balance changes to $900. Now, B can withdraw $250 from this $900 balance.

Required Properties of a Transaction

If between steps **3** and **6** (of the fund transfer transaction), another transaction **T2** 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).

T1	T2
1. Read (A) 2. A: = A - 50 3. Write (A)	
	read(A), read(B), print(A+B)
4. Read (B) 5. B: = B + 50 6. Write (B)	

T1	T2
1. Read (A)100 2. A- 50 (100 – 50) 3. Write A (50)	
	Read (A) (50), read B (100), print (A+B) 150
4. Read (B) 100 5. B+ 50 (100 + 50) 6. 6. Write (B) 150	

Required Properties of a Transaction

- Isolation can be ensured trivially by running transactions serially.
  - That is one after the other
- However, executing multiple transactions concurrently has significant benefits.

Required Properties of a Transaction

Durability Requirement

This property states that in any case all updates made
on the database will persist even if the system fails
and restarts. If a transaction writes or updates some d
ata in database and commits that data will always be t
here in the database. If the transaction commits but da
ta is not written on the disk and the system fails, that d
ata will be updated once the system comes up.

Required Properties of a Transaction

Example

user B may withdraw $100 only after user A’s transacti
on is completed and is updated in the database. If the
system fails before A’s transaction is logged in the dat
abase, A cannot withdraw any money, and Z’s account
returns to its previous consistent state.

Required Properties of a Transaction

Once the user has been notified that the transaction h
as completed (i.e., the transfer of the $50 has taken pl
ace), the updates to the database by the transaction m
ust persist even if there are software or hardware failur
es.

1. read (A)
2. A: = A-50
3. Write (A)
4. Read (B)
5. B: = B+50
6. Write (B)

Summary

A transcation is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

Atomicity:

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

Consistency:

Execution of a transaction in isolation preserves the consistency of the database

Summary

Isolation:

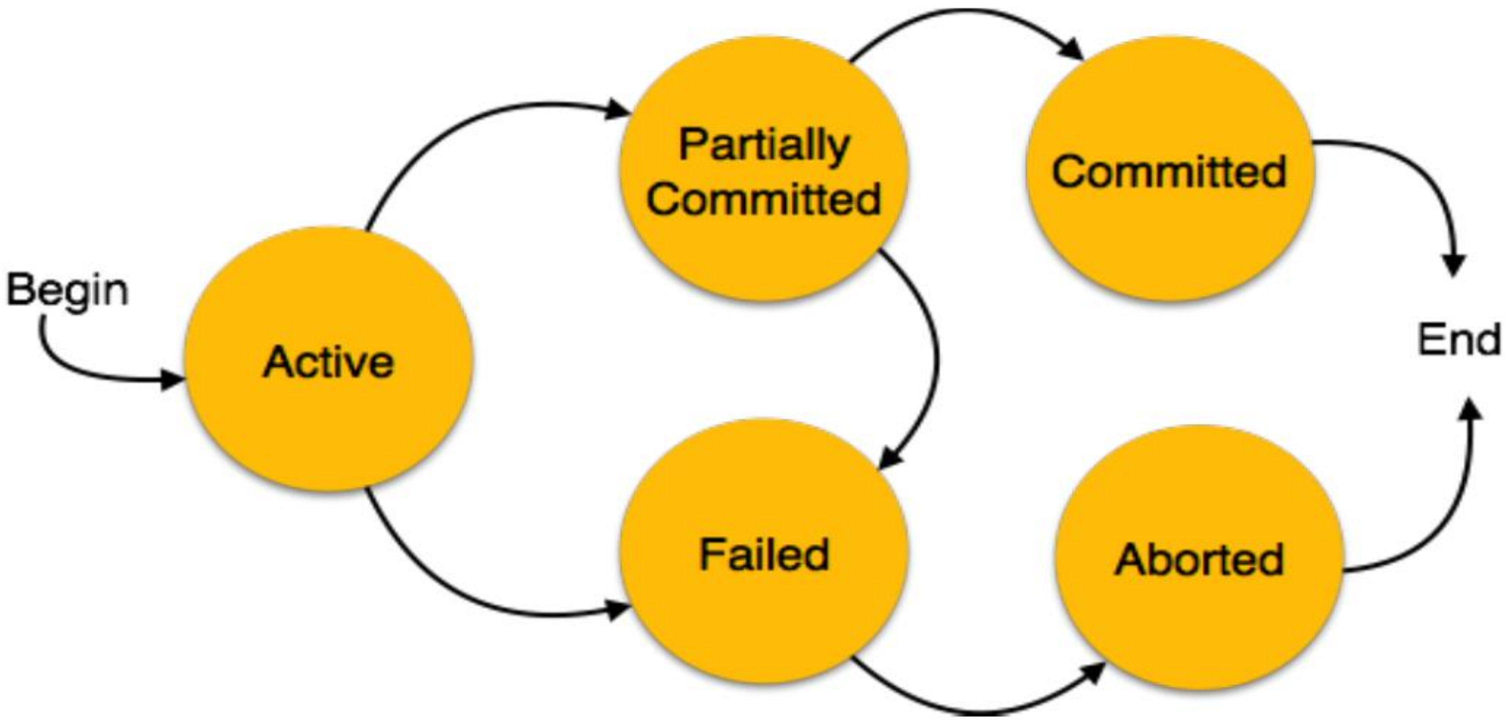
- Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
- That is, for every pair of transactions $\mathbf{T_i}$ and $\mathbf{T_j}$, it appears to $\mathbf{T_i}$ that either $\mathbf{T_j}$, finished executing before $\mathbf{T_i}$ started execution after $\mathbf{T_i}$ finished.

Summary

Durability:

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

Transaction State



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 l
onger proceed.

Transaction State

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
  - Can be done only if no internal logical error
- Kill the transaction

Transaction State

Committed

After successful completion

1. read (A)
2. A: = A-50
3. Write (A)
4. Read (B)
5. B: = B+50
6. Write (B)

Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - **Increased processor and disk utilization**, leading to better transaction throughput
    - For example, one transaction can be using the CPU while another is reading from or writing to the disk
  - **Reduced average response time** for transactions: short transactions need not wait behind long ones

Concurrent Executions

- **Concurrency control schemes** – mechanisms to achieve isolation
  - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the transactions appear in each individual transaction

Schedules

- A transaction that successfully completes its execution will have a **commit** instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- **A transaction** that **fails** to successfully complete its execution will have an **abort** instruction as the last statement

Schedules

- Let T_1 transfer \$50 from A to B, and T_2 transfer 10 % of the balance from A to B

- An example of a serial schedule in which T_1 is followed by T_2:

Schedule1

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

A	B	A + B	transaction	Remarks
100	200	300	@ Start	
50	200	250	T1, write A	
50	250	300	T1, write B	@ Commit
45	250	295	T2, write A	
45	255	300	T2, write B	@ Commit

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

Schedule 2

A serial schedule in which T2 is followed by T1

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	

A	B	A + B	transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
90	210	300	T2, write B	@ Commit
40	210	250	T1, write A	
40	260	300	T1, write B	@ Commit

Consistent @ Commit

Inconsistent @ Transit

Inconsistent @ Commit

Schedule 3 (Class Quiz)

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

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

Schedule 3

A	B	A + B	transaction	Remarks
100	200	300	@ Start	
50	200	250	T1, write A	
45	200	245	T2, write A	
45	250	295	T1, write B	@ Commit
45	255	300	T2, write B	@ Commit

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

Schedule 4 (Class Quiz)

The following concurrent schedule does not preserve the sum of “A + B”

T_1	T_2
	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

Schedule 4

The following concurrent schedule does not preserve the sum of "A + B"

T_1	T_2
	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

A	B	A + B	transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
90	200	290	T1, write A	
90	250	340	T1, write B	@ Commit
90	260	350	T2, write B	@ Commit

Consistent @ Commit

Inconsistent @ Transit

Inconsistent @ Commit

Module Summary

- A task is a database is done as a transaction that passes th
  rough several states
- Transactions are executed in concurrent fashion for better
  throughput
- Concurrent execution of transaction raise serializability is
  sues that need to be addressed
- All schedules may not satisfy ACID properties

Than you...!