Transactions Management

CCI 4301 : Advanced Database Management Systems

Introduction

In this Session, we will cover:

- Definition of a transaction and transaction management
- The ACID properties
- Life cycle of a Transaction {Transaction States}
- DBMS components

Introduction to Transaction Processing

- Database systems are classified according to the number of users who can use the system concurrently.
  - **Single-user DBMS** –support at most one user at a time.
  - **Multiuser DBMS** are database systems that can be used by many users at the same time thus allowing concurrent access the database
- Such systems are called **transaction processing systems**.

Transaction Processing Systems

- Systems with large databases and hundreds of concurrent users executing database transactions
- They require high availability and fast response time for hundreds of concurrent users..
  - Examples : airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications.

Concurrent Transactions

- When multiple transactions are submitted by various users concurrently, it is possible for them to interfere with one another in a way that produces incorrect results, or some transactions may fail.
- These problems lead to what we call concurrency control problem.
- In order to ensure the correct executions of database transactions, we therefore need concurrency control to manage concurrent users and transactions.
- These is broadly referred to as **transaction management**.

What is a Transaction?

- A transaction is defined as **a series of actions, carried out by a single user or application program, which reads or updates the contents of a database.**
  - A transaction is typically implemented by a computer program
  - Includes database operations such as retrievals, insertions, deletions, and updates.
- These database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

What is a Transaction?

- A transaction is a 'logical unit of work' on a database
  - Each transaction does something in the database

- A transaction is a unit of consistent and reliable computation.
  - Transactions are the unit of recovery, consistency, and integrity of a database
  - Transactions transform the database from one consistent state to another consistent state.

Example of a transaction

Suppose a bank employee transfers Ksh. 50 from A's account to B's account. In this transaction, the tasks involved are as follows:

A's Account

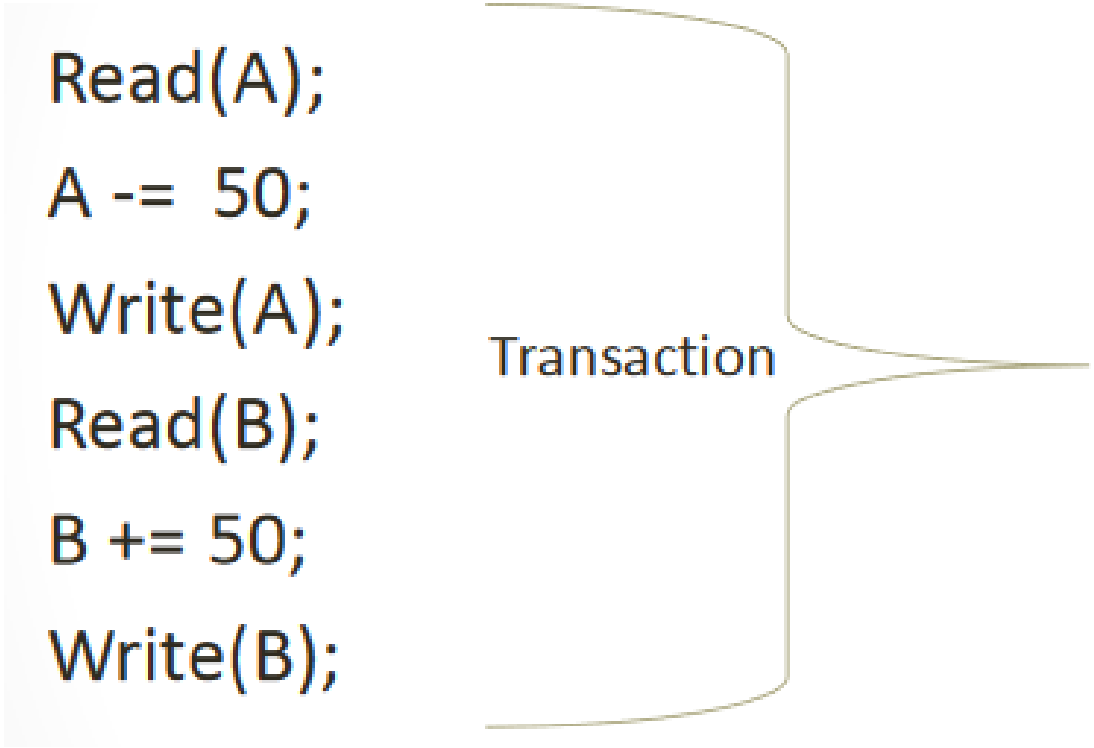
Open_Account(A)
read(A) = Old_Balance
New_Balance = Old_Balance - 50
write(A) = New_Balance
Close_Account(A)

B's Account

Open_Account(B)read(B) = Old_Balance
New_Balance = Old_Balance + 50
write(A) = New_Balance
Close_Account(B)

Example of a Transaction

The initial balances of accounts A and B are sh.200 and sh. 50 respectively. Transfer sh.50 from account A to account B.



What is Transactions Management?

- Transactions Management deals with the problem of always keeping the database in a consistent (or correct) state even when concurrent accesses and failures occur
  - The database can (and usually is) temporarily inconsistent during the execution of a transaction.
  - The important point is that the database should be consistent when the transaction terminates
- Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof

What is Transaction Management?

Motivating Example

Consider a person who wants to transfer sh.50 from a savings account with balance sh.1000 to a checking account with current balance = sh.250.

1)At the ATM, the person starts the process by telling the bank to remove sh.50 from the savings account.

2)The sh.50 is removed from the savings account by the bank.

3)Before the customer can tell the ATM to deposit the sh.50 in the checking account, the ATM "crashes."Where has the sh.50 gone?

It will be lost if the ATM did not support transactions!

What is Transactions Management?

- Transactions Management deals with the problem of always keeping the database in a consistent state even when concurrent accesses and failures occur
  - The database can (and usually is) temporarily inconsistent during the execution of a transaction.
  - The important point is that the database should be consistent when the transaction terminates

Properties of a Transaction (ACID)

- **Atomicity:** either the entire set of operations happens or none of it does
- **Consistency:** the set of operations taken together should move the database from one consistent state to another consistent state
- **Isolation:** each system perceives the system as if no other transactions were running concurrently (even though odds are there are other active transactions)
- **Durability:** results of a completed transaction must be permanent ~even IF the system crashes

Properties of a Transaction
1. Atomicity – ALL or Nothing Rule

- Transactions are either done or not done but NEVER left partially executed
- A transaction is an **indivisible unit** that is either performed in its entirety (completely) or is not performed at all
- A transaction is an **atomic program** that executes on the database and preserves the **consistency** of the database.
  - The input to a transaction is a consistent database, AND the output of the transaction must also be a consistent database.

Properties of a Transaction
2. Consistency ~ Correctness

- This properties says 'Only valid data is saved'
- Consistency occurs when users access a shared resource and the resource exhibits the same characteristics and satisfies all the constraints among all operations.
- Transactions should leave the database in a consistent state. They should transform the database from one consistent state to another consistent state.
- It's the responsibility of the DBMS [*integrity constraints*] and application developers [*enterprise constraints*] to ensure consistency.
  - Consistency property ensures that only valid data is written in the database.

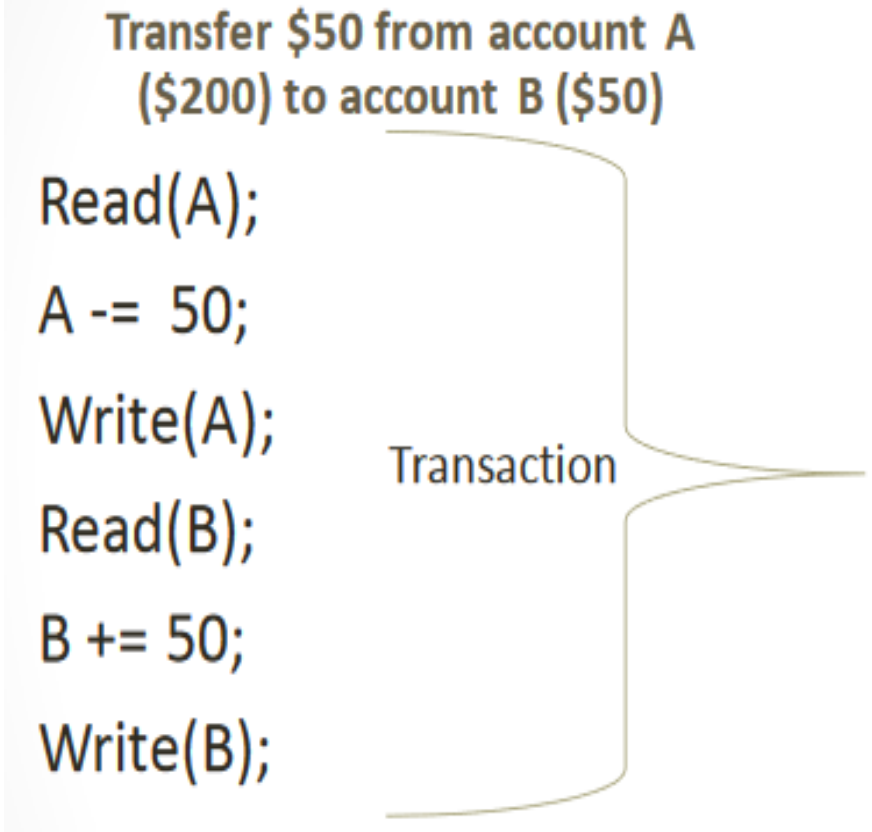
Properties of a Transaction
3. Isolation

- Transaction is hidden from the other transactions until its completed.
- Transactions must behave as if they were executed in isolation
  - As if they were executed independent of one another.
- Partial effects of incomplete transactions should not be visible to other transactions.
- It is the responsibility of the concurrency control subsystem to ensure isolation.

Properties of a Transaction
4. Durability

- All updates done by transaction must be made permanent (commit)
- Effects of completed transactions are resilient against failures
- That is the effects of a successfully completed transaction are permanently recorded in the database and must not be lost because of a subsequent failure.
- It's the duty of the recovery subsystem to ensure durability.

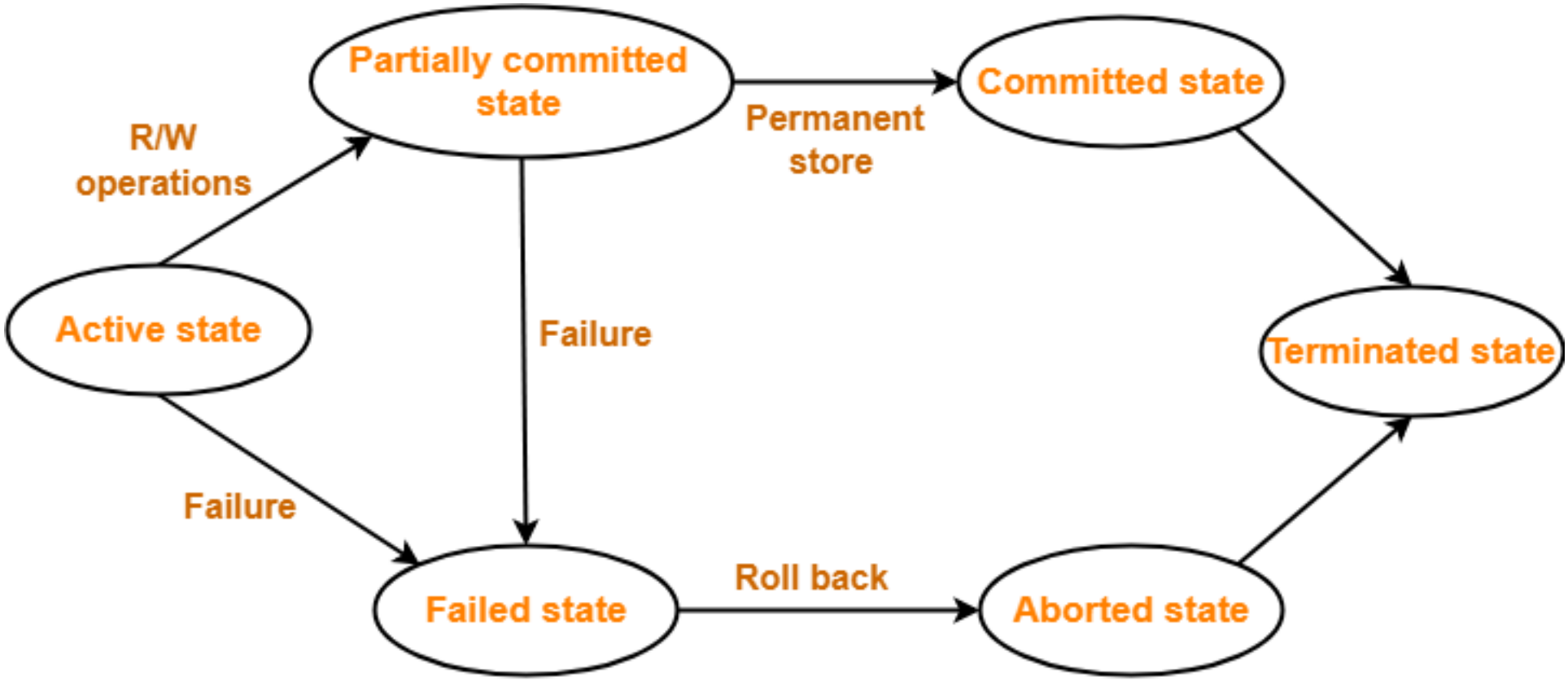
Example of a Transaction



ACID Properties

- **Atomicity** -shouldn't take money from A without giving it to B
- **Consistency** -money isn't lost or gained
- **Isolation** -other queries shouldn't see A or B change until transaction is completed
- **Durability** -the money does not go back to A if transaction was marked as committed

Transaction States in DBMS



Transaction States in DBMS

Active State

- When the instructions of the transaction are running then the transaction is in active state

- If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state"

Transaction States in DBMS

Partially Committed

- After completion of all the read and write operation the changes are made in main memory or local buffer

- If the changes are made permanent on the DataBase then the state will change to “committed state” and in case of failure it will go to the “failed state”

Transaction States in DBMS

Failed State

- When any instruction of the transaction fails, it goes to the "failed state" or if failure occurs in making a permanent change of data on Data Base

Aborted State

- After having any type of failure the transaction goes from "failed state" to "aborted state" and since in previous states, the changes are only made to local **buffer** or **main memory** and hence these changes are deleted or rolled-back

Transaction States in DBMS

Committed State

- It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.

Terminated State

- If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.

Outcomes of a Transaction

(Life of a Transaction)

- A transaction can have two outcomes:
  - ***a. Committed :*** if the transaction completes successfully , the transaction is committed and the database reaches a new consistent state.
    - Changes made to database are visible
  - ***b. Aborted :*** if the transaction does not execute successfully , the transaction is aborted and the database restored back to the consistent state it was before the transaction started(rolled back).
    - Changes made to database are not visible

Life of a Transaction

QUESTION:

1.A committed transaction can not be aborted? YES/NO

2.If we decide that the committed transaction was a mistake, what next?

[Perform another compensation transaction to reverse its effect]

Database Architecture:
DBMS Components

- **Transactions Manager** ~ It is the responsibility of the Transactions Manager of a DBMS to ensure transactions do not interfere with one another and corrupt the database.

  It coordinates transactions on behalf of application program.

- **Scheduler** ~ Responsible for implementing a particular strategy for concurrency control. Also called the Lock Manager (for those systems that use locking as the means of managing concurrency).

  The objective of the scheduler is to maximize concurrency control without allowing concurrently executing transactions to interfere with one another and compromise the integrity of the database.

Database Architecture:
DBMS Components

- **Recovery Manager~** if a failure occurs during the transaction, then the database could be inconsistent. Recovery Manager ensures that the database is restored to the state it was in before the start of the transaction and therefore a consistent state.

- **Buffer Manager ~** Responsible for the transfer of data between disk storage and memory.

References

- https://www.geeksforgeeks.org/types-of-schedules-in-dbms/
- http://www.ccs.neu.edu/home/kathleen/classes/cs3200/9-Transactions.pdf