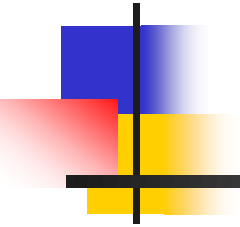


Transaction Processing Concepts





1. Introduction To transaction Processing

- 1.1 Single User VS Multi User Systems
 - One criteria to classify Database is according to number of user that concurrently connect to the system.
 - Single User: only one user use the system in each time
 - Multi User: many users use the system in the same time



1.2 Transactions, Read and Write Operations, and DBMS Buffers

■ What is a Transaction?

- Transaction is an **executing program** that forms a **logical unit of database processing**.
- A transaction include one or more database access operations.
- The database operation can be embedded within an application or can be specified by high level query language.
- Specified boundary by Begin and End transaction statements
- If the database operations in a transaction do not update the database, it is called "**Read-only transaction**"



Example of transaction

- Let T_i be a transaction that transfer money from account A (5000) to account B. The transaction can be defined as

■ T_i :	read (A)	(withdraw from A)
	$A := A - 5000$	
	write (A);	(update A)
	read (B)	(deposit B)
	$B := B + 5000$	
	write (B)	(update B)



Why recovery is needed?

- Transaction submitted for execution
- **DBMS is responsible** for making sure that either
 - **All operations in transaction** are **completed successfully** and the changes is recorded permanently in the database.
 - The DBMS **must not permit** some operations of a transaction T to be applied to the DB while others of T are not.
 - (will cause inconsistency)



Failures Type

Generally classified as

- Transaction Failure
- System Failure
- Media Failure



Reasons for a transaction fails in the middle of execution

- A **computer failure** (System crash) – media failures
- A **transaction or system error**: logical program error
- **Load error or exception conditions detected by the transaction** : no data for the transaction
- **Concurrency control enforcement**: by concurrency control method
- **Disk failure**
- **Physical problems and catastrophes**: ex. Power failure, fire, overwrite disk



Transaction or system error

- Some operation in transaction may cause it to fail, such as integer overflow or divide by zero.
- May occur because of erroneous parameter values
- Logical programming
- User may interrupt during execution



Local error or exception conditions detected by transaction

- During transaction execution, conditions may occur that necessitate cancellation of the transaction
- Ex. Data for the transaction may not found
- Exception should be programmed (not be consider failure)



Concurrency control enforcement

- The concurrency control method may decide to abort the transaction,
 - Because of violent serializability
 - Because several transaction are in state of deadlock



Transaction states and additional Operation

- For **recovery purpose**, the system needs to keep track of when the transaction
 - starts,
 - terminates,
 - and commits or aborts.
- **What information that the recovery manager keep track?**



Transaction states and additional Operation

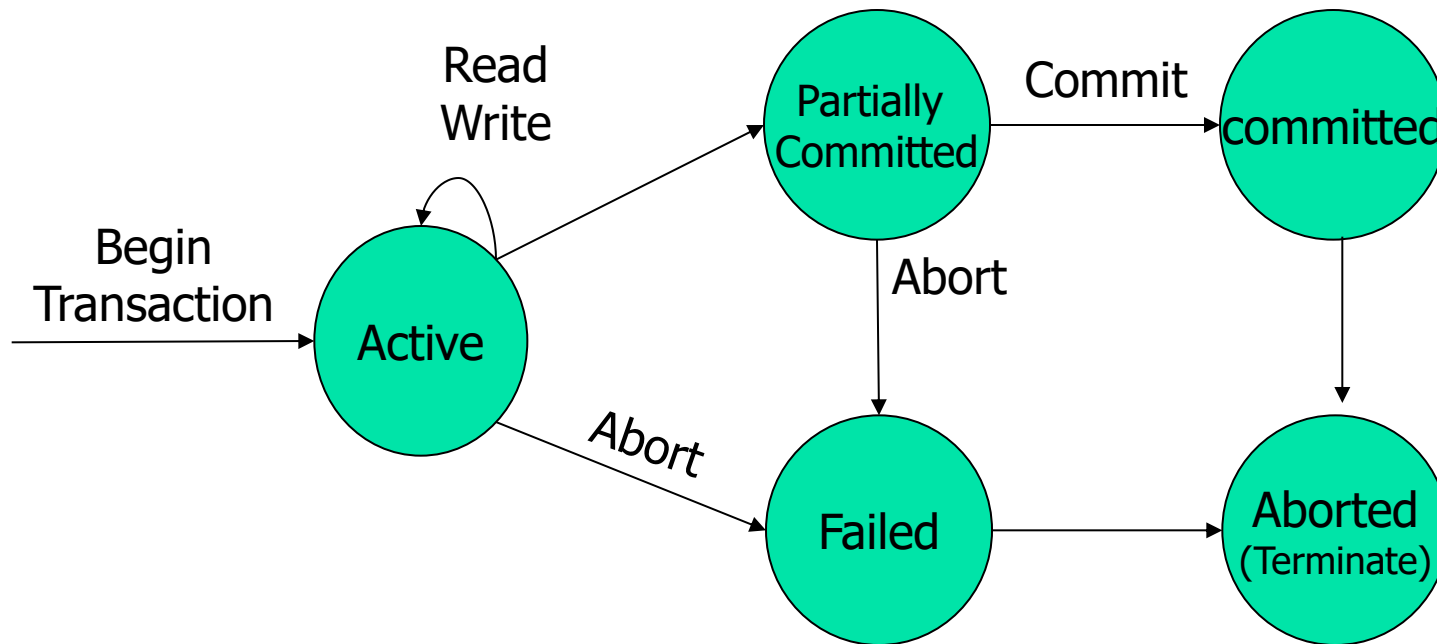
- The recovery manager keep track of the followings
 - **Begin_transaction:** mark the beginning of transaction execute
 - **Read or write:** specified operations on the database item that executes as part of transaction
 - **End_transaction:** specifies that operations have ended and marks the end of execution (Necessary to check)
 - The change can be **committed**
 - Or whether the transaction has to **aborted**
 - **Commit_Transaction:** successful end (will not undo)
 - **Rollback:** unsuccessful end (undone)



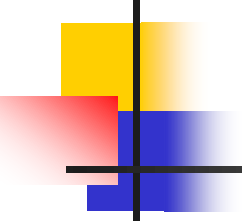
State of transaction

- **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 backed and the database has been restored to its state prior to the start of transaction.
- **Committed**, after successful completion

State diagram of a transition



A transaction must be in one of these states.

- 
-
- The transaction has **committed** only if it has entered the **committed state**.
 - The transaction has **aborted** only if it has entered the **aborted state**.
 - The transaction is said to have **terminated** if has either **committed** or **aborted**.



The System Log

- The system maintain log by keep track of all transactions that effect the database.
- Log is kept on Disk.
- Effected only by disk or catastrophic failure
- Keep Log records



Log records

T is transaction ID

- (Start_transaction, T) start transaction
- (Write_item, T, X, old_value, new_value)
transaction write/update item x
- (Read_item, T, X) transaction read item X
- (Commit, T) complete operation of T
- (Abort, T) terminate transaction T



System Log (cont.)

- Log file keep track
 - start transaction → complete transaction
- System fail occur
- Restart system, system will recovery
 - Redo transaction that already commit
 - Undo no commit transaction



Commit point of a transaction

- **When** is the transaction T **reach its commit** point?
- **Answer** is “when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log.”
- The transaction is said to be “**committed**”



(Cont.)

- At committed point
 - Write [commit] in log file
- Failure occur
 - Search in log file looking for all transactions T, that have write [Start_Transaction ,T]
 - If no commit, Rollback transaction
 - If commit found, Redo transaction



Desirable properties of transaction : ACID properties

- To ensure integrity of data, we require that the database system maintain the following properties of the transactions:
 - **Atomicity.**
 - **Consistency preservation.**
 - **Isolation.**
 - **Durability or permanency.**

A diagram consisting of five circles arranged horizontally. The first circle is solid light purple and contains the word 'ACID' in bold black text. The second circle is an outline. The third circle is solid light purple. The fourth circle is an outline. The fifth circle is solid light purple.

ACID

- **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)
- **Isolation.** Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j , it appears to T_i that
 - either T_j finished execution before T_i started,
 - or T_j started execution after T_i finished.
 - Thus, each transaction is unaware of other transactions executing concurrently in the system.

(Execution of transaction should not be interfered with by any other transactions executing concurrently)
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
(The changes must not be lost because of any failure)



Consistency

- **Consistency.**

- The consistency requirement here is that the **sum of A and B** be unchanged by the execution of the transaction.
- Without consistency requirement, money could be created or destroyed by the transaction.
- It can be verified,
 - If the database is consistency before an execution of the transaction, the database remains consistent after the execution of the transaction.



Atomicity

- **Atomicity.** Either **all operations of the transaction** are reflected properly in the database, or none are.
 - State before the execution of transaction T_i
 - The value of A = 50,000
 - The value of B = 100
 - Failure occur (ex. Hardware failure)
 - Failure happen after the **WRITE(A)** operation
 - (at this moment $A = 50000 - 5000 = 45000$)
 - And the value of B = 100 (**inconsistency state**)
 - **In consistency state A = 45000 and B = (5100)**



(cont.)

- Idea behind ensuring atomicity is following:
 - The database system keeps track of the old values of any data on which a transaction performs a write
 - If the transaction does not complete, the DBMS restores the old values to make it appear as though the transaction have never execute.



Durability or permanency

- **Durability or permanency.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
- **These changes must not be lost because of any failure**
- ensures that, transaction has been committed, that transaction's updates do not get lost, even if there is a system failure



Isolation

- **Isolation.** Even though multiple transactions may execute concurrently, the **system guarantees that**,
 - for every pair of transactions T_i and T_j ,
 - it appears to T_i that either T_j finished execution before T_i started,
 - or T_j started execution after T_i finished.
 - Thus, each transaction is unaware of other transactions executing concurrently in the system.
 - **(Execution of transaction should not be interfered with by any other transactions executing concurrently)**



Concurrency Control



Concurrent Executions

- Transaction processing permit
 - Multiple transactions to run concurrently.
 - Multiple transactions to update data concurrently
- Cause
 - Complications with **consistency of data**



Reason for allowing concurrency

- Improved **throughput** of transactions and **system resource** utilization
- Reduced waiting time of transactions



Possible Problems

- Lost update problem
- Temporary update problem
- Incorrect summary problem



Example transaction

- Transfer money from account A to B

- Read_item(A)
- $A := A - 50$
- Write_item(A)
- Read_item(B)
- $B := B + 50$
- Write_item(B)

- Transfer 10% of A to Account B

- Read_item(A)
- $temp := 0.1 * A$
- $A := A - temp$
- Write_item(A)
- Read_item(B)
- $B := B + temp$
- Write_item(B)

Lost update problem

A = 1000, B = 2000

T1	T2
Read_item(A) A = 1000	Read_item(A) A = 950
A := A - 50 A = 950	temp := 0.1 * A temp = 95
Write_item(A) A = 950	A := A - temp A = 950 - 95 = 855
Read_item(B) B = 2000	Write_item(A) A = 855
B := B + 50 B = 2050	Read_item(B) B = 2000
Write_item(B) B = 2050	B := B + temp B = 2095
	Write_item(B) B = 2095

Temporary update problem

R = 3000

T1	T2
-	Write_item(R) R = 1000
Read_item(R) R = 1000	-
-	RollBack R = 3000

Inconsistency problem

A = 40 ,

B = 50,

C = 30

T1

Read_item(A)

A = 40

SUM = Sum+A

Sum = 40

Read_item(B)

B = 50

SUM = A + B

SUM = 40+50
= 90

T2

A+B+C = 40+50+30 = 120

Read_item(C) C = 30

C = C - 10 C = 30-10 = 20

Write_item(C) C = 20

Read_item(A) A = 40

A = A + 10

Write_item(A) A = 50

COMMIT

After
A+B+C = 50+50+20 = 120

Read_item(C)

C = 20

SUM = SUM + C

Sum = 90 + 20 = 110