Transaction Processing

Mirunalini.P

SSNCE

May 18, 2022

Table of Contents

Session Objective

- Transaction Concepts
- Read /Write Operations
- Concurrency problem
- Transaction state

Session Outcome

At the end of this session, participants will be able to

- Understand transaction processing concepts

Transaction Processing Concepts

- The concept of transaction provides a mechanism for describing **logical units of a database Processing**.
- Transaction processing systems include **large databases and hundreds of concurrent users executing database transactions**
- Examples of these systems are:
  - Airline reservations
  - Banking
  - Credit card processing
  - Supermarket checkout

Transaction Processing Concepts

- These system requires **high availability and fast response time** for hundreds of concurrent users.
- Each transaction must be completed in its entirety to ensure correctness.
- A transaction is implemented by a computer program that includes database commands such as retrievals, insertions, deletions and updates.

Transaction Processing Concepts

Database system is classified according to the number of users who use the system concurrently.

- **Single-User System:** At most one user at a time can use the system.
- **Multiuser System:** Many users can access the system concurrently.
    - Concurrent database usage is possible with **multiprogramming**
    - It allows the operating system of the computer to execute multiple programs or processes at the same time.

Multiprogramming

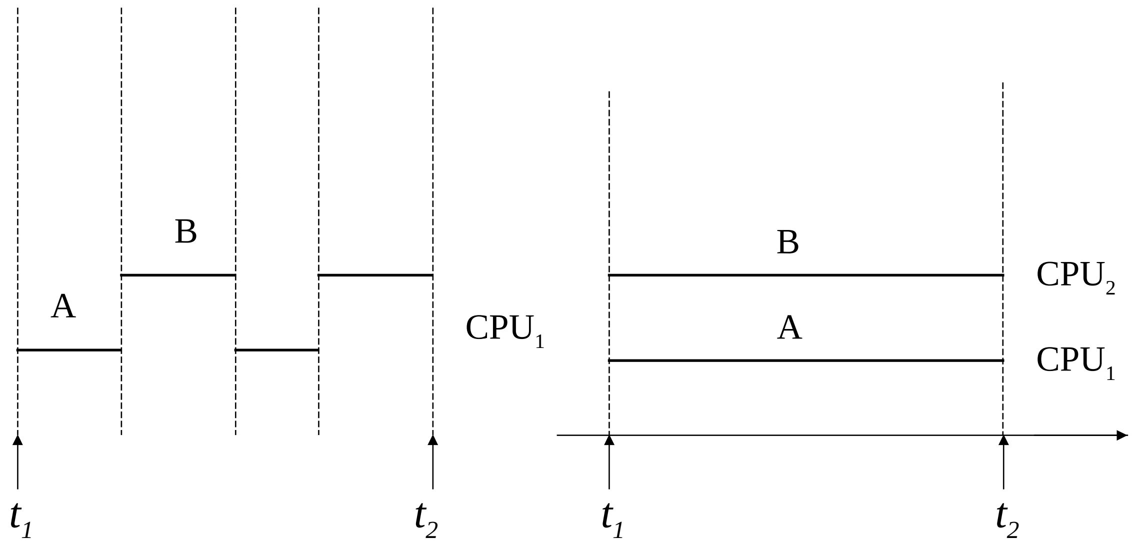
Interleaved processing:

- Multiprogramming OS execute some commands from one process then suspend that process and execute some commands from the next process.
- A process is resumed whenever it gets its turn to use the CPU again.
- Concurrent execution of processes is interleaved in a single CPU.
- Keeps CPU busy when a process requires an input/output operations.
- The CPU switched to execute another process rather than remaining idle-during I/O time.
- Prevent long process from delaying other processes.

Parallel processing: Processes are concurrently executed in multiple CPUs.

Basic transaction processing theory assumes interleaved concurrency

Interleaved Versus Parallel Processing



Interleaved processing

Parallel processing

Transactions

- A transaction is an executing program that forms a logical unit of database processing.
- A transaction includes one or more database access operations which includes insertion,deletion,modification or retrieval of operations.
- The transactions can be embedded within an application program or can be specified interactively using query language.
- Begin transaction and end transaction acts as a transaction boundaries.
- if the transaction does not update the database but only retrieve the data called as **read-only** transactions otherwise called as **read-write** transactions.

Transactions

- A simple database model will be used to represent transaction processing concepts
- A database is a collection of named data items.
- The size of the data item is called its granularity.
- Granularity of data - a field, a record , or a whole disk block
- Transaction processing concepts are independent of granularity
- Basic operations are read and write:
  - **Read_item(X):** Reads a database item named X into a program variable X.
  - **Write_item(X):** Writes the value of program variable X into the database item named X.

Read and Write Opearations

- Basic unit of data transfer from the disk to the computer main memory is one block.
- In general, a data item (what is to be read or written) will be the field of some record in the database
- **Read_item(X)** command includes the following steps:
  - Find the address of the disk block that contains item X.
  - Copy that disk block into a buffer in main memory
  - Copy item X from the buffer to the program variable named X.

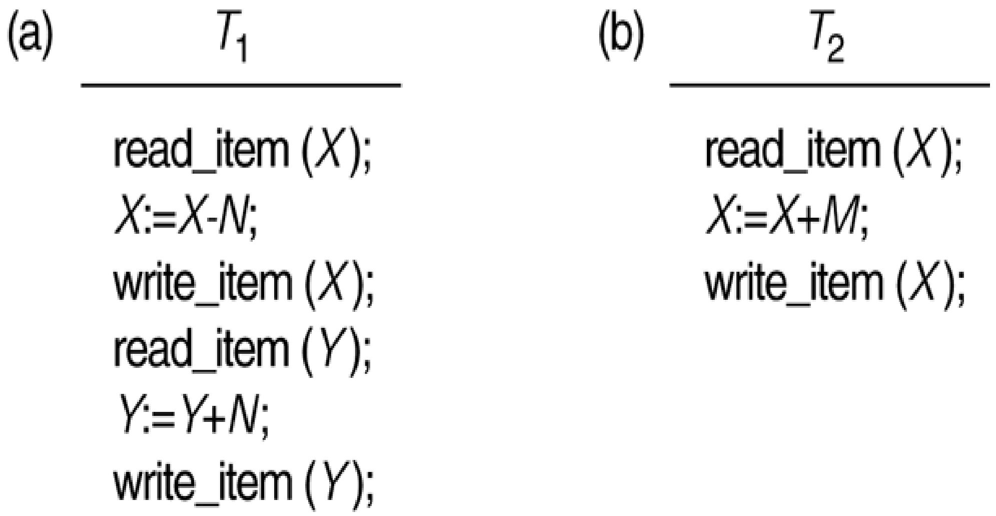
Read and Write Opearations

Write_item(X) command includes the following steps:

- Find the address of the disk block that contains item X.
- Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
- Copy item X from the program variable named X into its correct location in the buffer.
- Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Two sample transactions

T1 and T2 transactions



(a)

T_1
read_item (*X*);
X:=*X*-*N*;
write_item (*X*);
read_item (*Y*);
Y:=*Y*+*N*;
write_item (*Y*);

(b)

T_2
read_item (*X*);
X:=*X*+*M*;
write_item (*X*);

Two sample transactions

Notation focuses on the read and write operations
Can also write in shorthand notation:

T1: b1; r1(X); w1(X); r1(Y); w1(Y); e1;

T2: b2; r2(Y); w2(Y); e2;

b_i and e_i specify transaction boundaries (begin and end)

i specifies a unique transaction identifier (T_Id)

Concurrency control and database recovery mechanism are mainy concerned with the database commands in a transaction.

Sources of Database Inconsistency

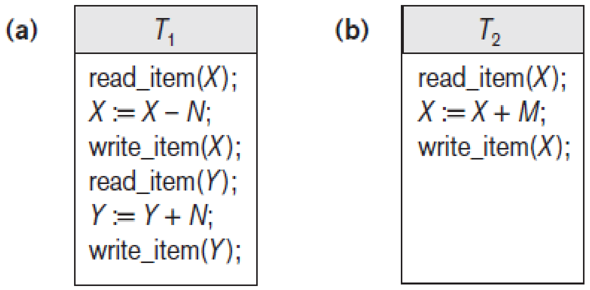
Uncontrolled execution of database transactions in a multi_ user environment can lead to database inconsistency

There are number of possible sources for database inconsistency
The typical ones are:

- **Lost update problem**
- **Dirty read problem**
- **Incorrect Summary Problem**

Why Concurrency Control is needed?

T1 and T2 transactions



(a)

T_1
read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

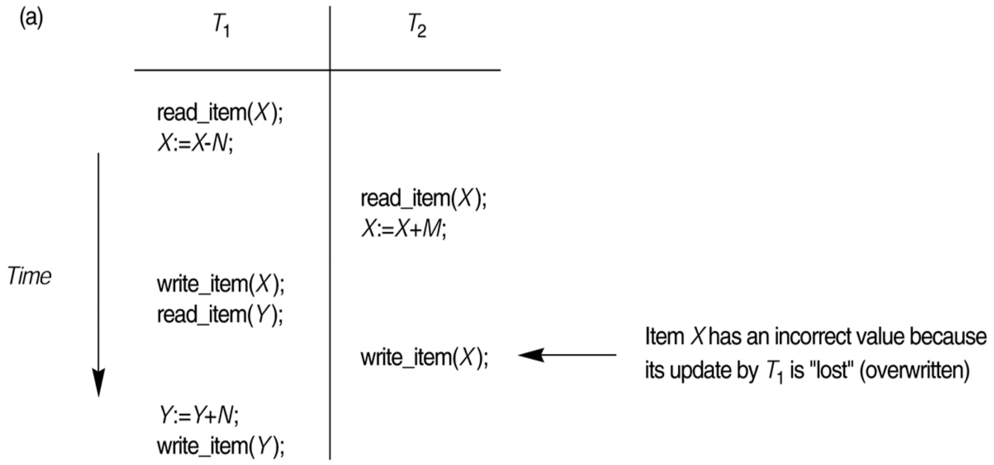
(b)

T_2
read_item(X);
$X := X + M$;
write_item(X);

Transaction T1 transfers some amount from acct1 to another acct2
N & M : Amount
X: Database item to store acct1 amt
Y: Database item to store acct2 amt

The Lost Update Problem

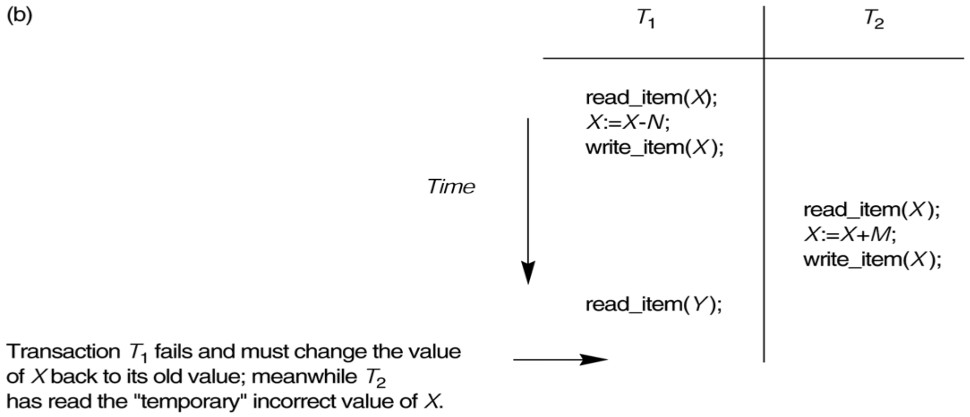
This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.



Lost Update : T2 reads the value of X before T1 changes it in the database

The Temporary Update (or Dirty Read) Problem

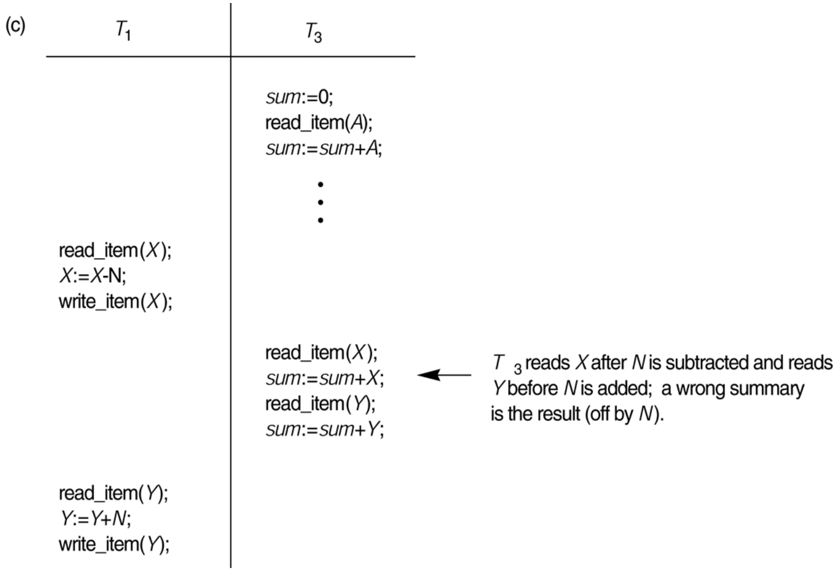
This occurs when one transaction updates a database item and then the transaction fails for some reason.
The updated item is accessed by another transaction before it is changed back to its original value.


(b)
T_1
T_2
read_item(X);
X:=X-N;
write_item(X);
Time
read_item(X);
X:=X+M;
write_item(X);
read_item(Y);
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X.

Temporary Update : T2 reads the temporary value of X before T1 commits (dirty read)

Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.



(c)

T_1	T_3

	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*);
	sum:=*sum*+*Y*;

read_item(*Y*);
Y:=*Y*+*N*;
write_item(*Y*);	

$\leftarrow$ T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).

The Unrepeatable Read Problem

- When a transaction reads the same item twice and the item is changed by another transaction T' between the two reads.
- T receives different values for its two read of the same item.

Why recovery is needed?

Whenever transaction is submitted it might be:

Committed Transactions: All operations of transactions are completed successfully and they are recorded permanently in the db.

Aborted: The transaction does not have any effect on the db, due to some failure in transactions.

if the transaction fails after executing some of the operations but before executing all of them, the operations already executed must be **undone** and should not have no lasting effect.

Types of Failures:

Several reasons for the transactions to fail in the middle of execution:

- **Computer failure (system crash):** Hardware, software, network error during transaction execution
- **Transaction or system error:** Some operations may cause overflow, division by zero, erroneous parameter values, logical programming error, transaction interruption during execution
- **Local errors or exception conditions detected by the transaction:** Certain conditions may cancel transactions due to lack of data. Eg : Insufficient Fund
- **Concurrency control enforcement:** Dead lock, timeout and Serializability

- **Disk failure:** Disk blocks may lose due to read or write malfunction, disk read/write head crash
- **Physical problems or catastrophes:** Includes power or air-conditioning failure, theft, overwriting disks or tapes by mistake and mounting wrong tape by the operator.

DBMS has a Recovery Subsystem to protect database against system failures

Transaction States

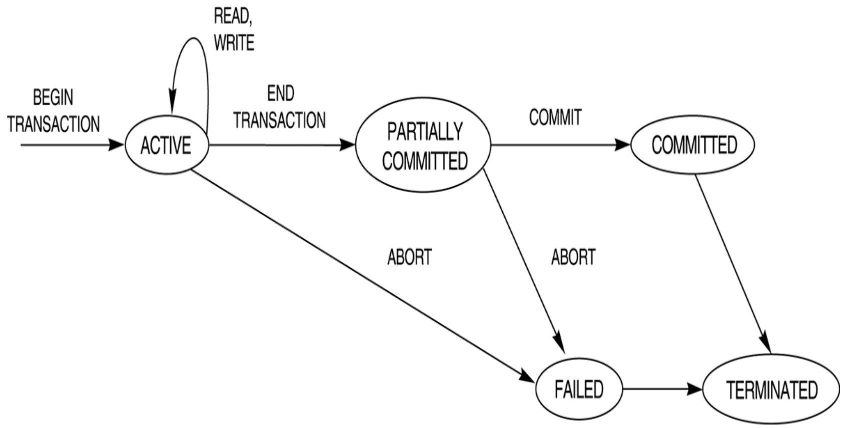
A transaction is an atomic unit of work that is either completed in its entirety or not done at all.

Transaction states:

- **Active state**: Transaction goes into active state immediately after it starts execution.
- **Partially Committed State**: When the transaction ends it moves to the partially committed state.
- **Committed State**: If the transaction reaches its commit point, the transaction enters the committed state.
- **Failed state**: If the transaction is aborted during its active state, it enters into the failed state.
- **Terminated State**: Corresponds to the transaction leaving the system.

State Transition Diagram

State transition diagram: states for transaction execution



Transaction States

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 limit of transaction execution.
    - It may be necessary to check whether the transaction is commited transactions or aborted

Transaction States

Recovery manager keeps track of the following operations:

- **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 by the transaction that are applied to the database must be **undone**.

System Log

- To able to recover from transaction failures, the system maintains a log.
- The log keeps track of all transaction operations that affect the values of database items.
- It also keeps track of transaction info needed to permit recovery from failures.
- Log entries are first added to the log main memory buffer. When the log buffer is filled, or when certain other conditions occur, the log buffer is appended to the end of the log file on disk.
- The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.
- In addition, the log file is periodically backed up to archival storage (tape) to guard against such catastrophic failures.

System Log

The following are the types of entries—called log records—that are written to the log file for the corresponding action : T refers to unique transaction ID.

- **[start_transaction,T]:** Records the transaction T has started execution.
- **[write_item,T,X,old_value,new_value]:** Records that transaction T has changed the value of database item X from old_value to new_value
- **[read_item,T,X]**: Records that transaction T has read the value of database item X.
- **[commit,T]:** Records that transaction T has completed successfully, and affirms that its effect can be committed to the database
- **[abort,T]:** Records that transaction T has been aborted.

System Log

- To Recovery using log records: If the system crashes, we can recover to a consistent database state by examining the log and using one of the techniques:
- Need to either **redo or undo** everything that happened since last commit point.
- **UNDO** by resetting all items changed by a write operation of T to their **old_values** by tracing backward through the log records.
- **REDO**:by setting all items changed by a write operation of T to their **new_values**.
  Redo of an operation may be necessary if a transaction has its updates recorded in the log but a failure occurs before the these new_values have been written to the actual database on disk from the main memory buffers

Commit Point of a Transaction

Commit Point of a Transaction

- A transaction T reaches its **commit point** when all its operations have been executed successfully and the effect of all the transaction operations has been recorded in the log.
- Beyond, the commit point, the transaction is said to be **committed**, and its effect is assumed to be permanently recorded in the database.
- The transaction then writes an entry [commit,T] into the log.

Roll Back of Transactions

- If system failure occurs, search the log for [start_ transaction,T] entry into the log and but have not written their commit entry [commit,T] into the log.
- The transactions have to be **rolled back to undo** their effect on the database during the reovery process.
- Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, effect of the database can be **redone** from their log records.

Commit Point of a Transaction

- The log file must be kept on disk.
- It is common to keep one or more blocks of the log file in main memory buffers (log buffer), until they are filled with log entries and then to write them back to disk only once, rather than writing to disk every time a log entry is added.
- At the time of a system crash, only the log entries that have been written back to disk are considered in the recovery process because the contents of main memory may be lost.
- **Force writing a log**: Before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called **force-writing** the log file before committing a transaction.

ACID Properties

Transaction should possess serveral properties often called as the ACID properties that should be enforced by the concurrency control and recovery methods.

- **Atomicity:**A transaction is an atomic unit of processing; It is either performed in its entirety or not performed at all.
  It is the responsibility of the **transaction recovery subsystem**.
- **Consistency preservation:** Every transaction should execute from beginning to end without interference of other transactions. A correct execution of the transaction must take the database from one consistent state to another.
  It is the responsibility of the **programmer or the DBMS module** that enforces integrity constraints.

ACID Properties

- **Isolation**:
  A transaction should not make its updates visible to other transactions until it is committed.
  This property, when enforced strictly, solves the temporary update problem and elminates cascading rollbacks.
  - Level 0: no dirty read
  - Level 1: no lost updates
  - Level 2: no lost updates and no dirty read
  - Level 3: Level 2 + repeatable reads

  It is enforced by concurrency control subsystem of the DBMS
- **Durability or permanency:**
  Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.
  It is the responsibility of the recovery subsystem of the DBMS .
  Recovery protocols enforces atomicity and durability

Example

Transfer 50 rupees from account a to account b

read(a); a:=a-50; write(a); read(b); b=b+50; write(b)

Atomicity: *a* must not be debited without crediting *b*
Consistency: Sum of a and b remains constant
Isolation : Account *a* is making T1 and T2 transactions to account *b* and *c*, but both are executing independently without affecting each other. It is known as Isolation.
Durability: if *b* is notified of credit, it must persist even if the database crashes

Reference

Fundamentals of Database systems 7^{th} Edition by Ramez Elmasri.