Chapter 20

Introduction to Transaction Processing Concepts and Theory

Introduction to Transaction Processing

- **Transaction:** An executing program (process) that includes one or more database access operations
  - Read operations (database retrieval, such as SQL SELECT)
  - Write operations (modify database, such as SQL INSERT, UPDATE, DELETE)
  - Transaction: A logical unit of database processing
  - Example: Bank balance transfer of $100 dollars from an account to a saving account in a BANK database
- **Note:** Each execution of a program is a *distinct transaction* with different parameters
  - Bank transfer program parameters: savings account number, checking account number, transfer amount

Introduction to Transaction Processing (cont.)

- A transaction (set of operations) may be:
  - stand-alone, specified in a high-level language like SQL submitted interactively, or
  - consist of database operations embedded within a program (most transactions)
- **Transaction boundaries**: Begin and End transaction.
  - Note: An **application program** may contain several transactions separated by Begin and End transaction boundaries

Introduction to Transaction Processing (cont.)

- **Transaction Processing Systems:** Large multi-user database systems supporting thousands of *concurrent transactions* (user processes) per minute
- **Two Modes of Concurrency**
  - **Interleaved processing**: concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing**: processes are concurrently executed in multiple CPUs (Figure 21.1)
  - Basic transaction processing theory assumes interleaved concurrency

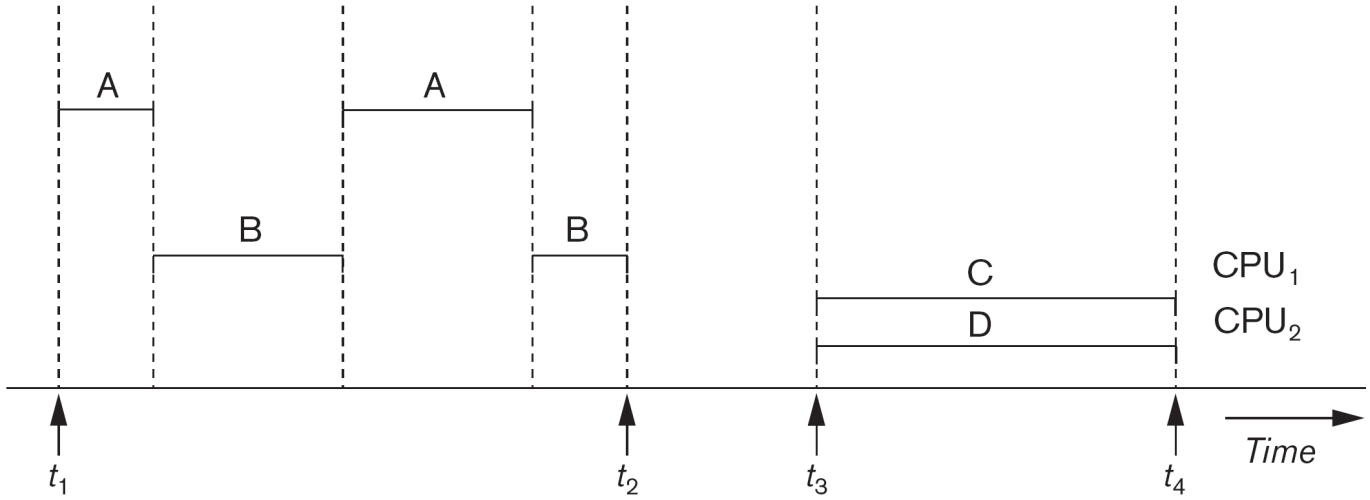


Figure 21.1
Interleaved processing versus parallel processing of concurrent transactions.

Introduction to Transaction Processing (cont.)

For transaction processing purposes, a simple database model is used:

- **A database -** collection of named data items
- **Granularity (size) of a data item** - a field (data item value), a record, or a whole disk block
  - TP concepts are independent of granularity
- Basic operations on an item X:
  - **read_item(X)**: Reads a database item named X into a program variable. To simplify our notation, we assume that *the program variable is also named X.*
  - **write_item(X)**: Writes the value of program variable X into the database item named X.

Introduction to Transaction Processing (cont.)

READ AND WRITE OPERATIONS:

- Basic unit of data transfer from the disk to the computer main memory is one disk block (or page). A data item X (what is read or written) will usually be the field of some record in the database, although it may be a larger unit such as a whole record or even a whole block.
- **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 (if that disk block is not already in some main memory buffer).
  - Copy item X from the buffer to the program variable named X.

Introduction to Transaction Processing (cont.)

READ AND WRITE OPERATIONS (cont.):

- **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 it 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).

Transaction Notation

- Figure 21.2 (next slide) shows two examples of 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;
- bi and ei specify transaction boundaries (begin and end)
- i specifies a unique transaction identifier (TId)

(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);

Figure 21.2
Two sample transactions. (a) Transaction T_1. (b) Transaction T_2.

Why we need concurrency control

Without Concurrency Control, problems may occur with concurrent transactions:

- **Lost Update Problem.**

  Occurs when two transactions update the same data item, but both read the same original value before update

The Temporary Update (or Dirty Read) Problem.

This occurs when one transaction T1 updates a database item X, which is accessed (read) by another transaction T2; then T1 fails for some reason; X was (read) by T2 before its value is changed back (rolled back or UNDONE) after T1 fails

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

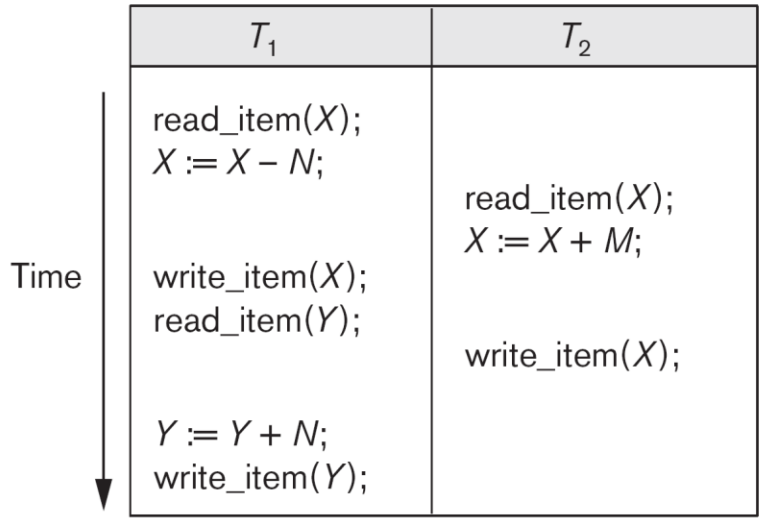
Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

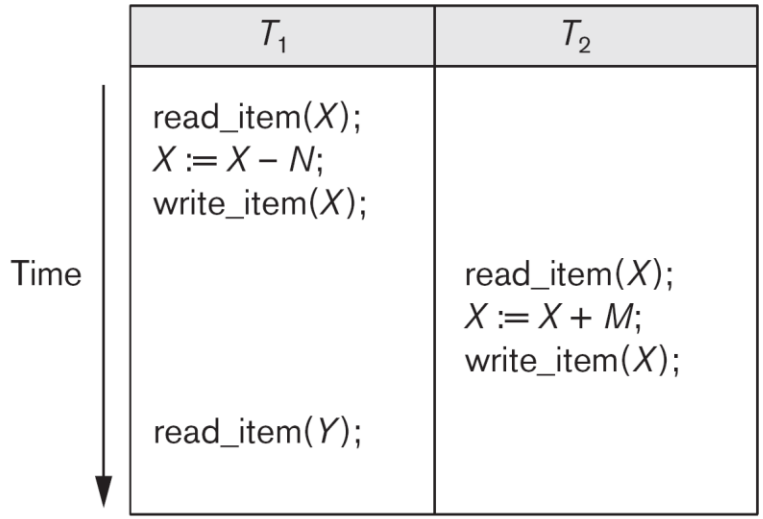
- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

(a)



Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

(b)



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.

Figure 21.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

Why we need concurrency control (cont.)

- **The Incorrect Summary Problem .**

  One transaction is calculating an aggregate summary function on a number of records (for example, sum (total) of all bank account balances) while other transactions are updating some of these records (for example, transferring a large amount between two accounts, see Figure 21.3(c)); the aggregate function may read 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);	

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

Why we need concurrency control (cont.)

- **The Unrepeatable Read Problem .**

  A transaction T1 may read an item (say, available seats on a flight); later, T1 may read the same item again and get a different value because another transaction T2 has updated the item (reserved seats on the flight) between the two reads by T1

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Why recovery is needed

Causes of transaction failure:

1. **A computer failure (system crash):** A hardware or software error occurs during transaction execution. If the hardware crashes, the contents of the computer's internal main memory may be lost.
2. **A transaction or system error :** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

Why recovery is needed (cont.)

3. **Local errors or exception conditions** detected by the transaction:

   - certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled
   - a programmed abort causes the transaction to fail.

4. **Concurrency control enforcement:** The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock (see Chapter 22).

Why recovery is needed (cont.)

5. **Disk failure:** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This kind of failure and item 6 are more severe than items 1 through 4.
6. **Physical problems and catastrophes:** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

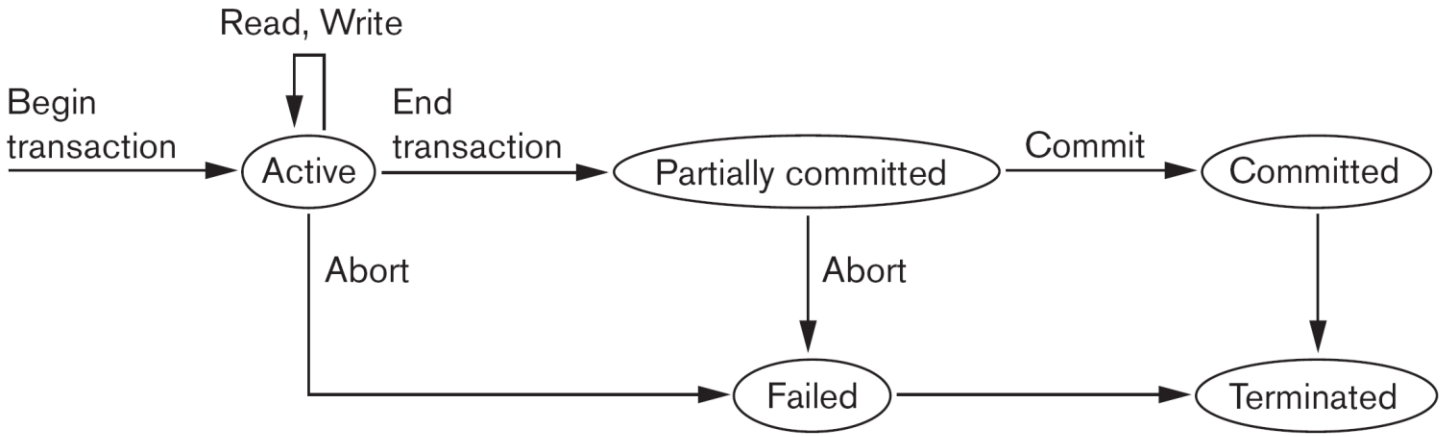
Transaction and System Concepts

A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all. A transaction passes through several states (Figure 21.4, similar to process states in operating systems).

Transaction states:

- Active state (executing read, write operations)
- Partially committed state (ended but waiting for system checks to determine success or failure)
- Committed state (transaction succeeded)
- Failed state (transaction failed, must be rolled back)
- Terminated State (transaction leaves system)

Figure 21.4
State transition diagram illustrating the states for transaction execution.



Transaction and System Concepts (cont.)

DBMS Recovery Manager needs system to keep track of the following operations (in the system **log file**):

- **begin_transaction:** Start of transaction execution.
- **read or write:** Read or write operations on the database items that are executed as part of a transaction.
- **end_transaction:** Specifies end of read and write transaction operations have ended. System may still have to check whether the changes (writes) introduced by transaction can be *permanently applied to the database* (**commit** transaction); or whether the transaction has to be *rolled back* (**abort** transaction) because it violates concurrency control or for some other reason.

Transaction and System Concepts (cont.)

Recovery manager keeps track of the following operations (cont.):

- **commit_transaction:** Signals *successful end* of transaction; any changes (writes) executed by transaction can be safely **committed** to the database and will not be undone.
- **abort_transaction (or rollback):** Signals transaction has *ended unsuccessfully*; any changes or effects that the transaction may have applied to the database must be *undone.*

Transaction and System Concepts (cont.)

System operations used during recovery (see Chapter 23):

- **undo(X):** Similar to rollback except that it applies to a single write operation rather than to a whole transaction.
- **redo(X):** This specifies that a *write operation* of a committed transaction must be *redone* to ensure that it has been applied permanently to the database on disk.

Transaction and System Concepts (cont.)

Commit Point of a Transaction:

- **Definition:** A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully *and* the effect of all the transaction operations on the database has been recorded in the log file (on disk). The transaction is then said to be **committed**.

Desirable Properties of Transactions

Called ACID properties – Atomicity, Consistency, Isolation, Durability:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another.

Desirable Properties of Transactions (cont.)

ACID properties (cont.):

- **Isolation**: Even though transactions are executing concurrently, they should appear to be executed in isolation – that is, their final effect should be as if each transaction was executed in isolation from start to finish.

- **Durability or permanency**: Once a transaction is committed, its changes (writes) applied to the database must never be lost because of subsequent failure.

Desirable Properties of Transactions (cont.)

- **Atomicity**: Enforced by the recovery protocol.
- **Consistency preservation**: Specifies that each transaction does a correct action on the database *on its own*. Application programmers and DBMS constraint enforcement are responsible for this.
- **Isolation**: Responsibility of the concurrency control protocol.
- **Durability or permanency**: Enforced by the recovery protocol.

Introduction to Transaction Support in SQL

- A single SQL statement is always considered to be atomic. Either the statement completes execution without error, or it fails and leaves the database unchanged.
- With SQL, there is no explicit Begin Transaction statement. Transaction initiation is done implicitly when particular SQL statements are encountered.
- Every transaction must have an explicit end statement, which is either a COMMIT or ROLLBACK.