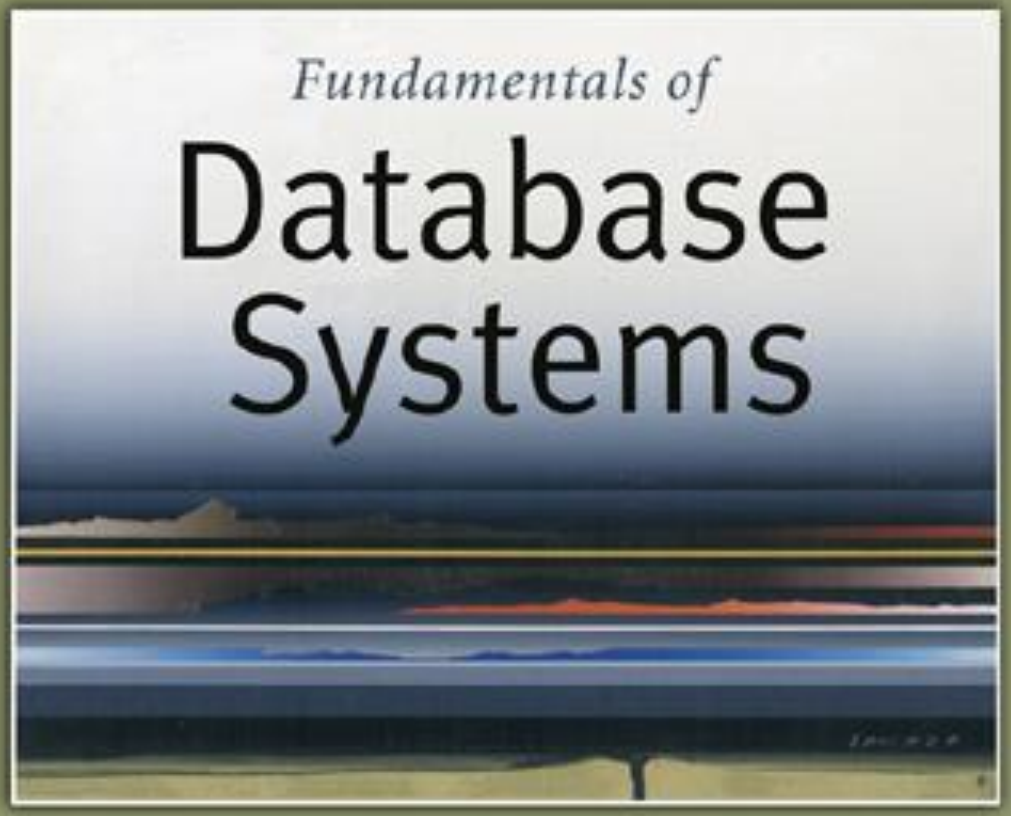
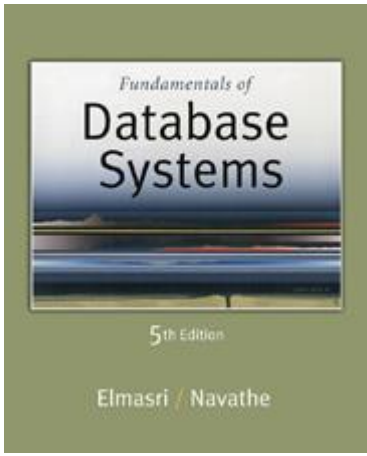


5th Edition

Elmasri / Navathe

Chapter 17

Introduction to Transaction Processing Concepts and Theory



Chapter Outline

1 Introduction to Transaction Processing

2 Transaction and System Concepts

3 Desirable Properties of Transactions

4 Characterizing Schedules based on Recoverability

5 Characterizing Schedules based on Serializability

6 Transaction Support in SQL

1 Introduction to Transaction Processing (1)

- **Single-User System**:
  - At most one user at a time can use the system.
- **Multiuser System**:
  - Many users can access the system concurrently.

Multiple user can access DB simultaneously because of the concept of **multi programming** (which allow the OS to execute multiple programs at the same time) **HOW ?**

A single CPU can execute at most one process at a time.

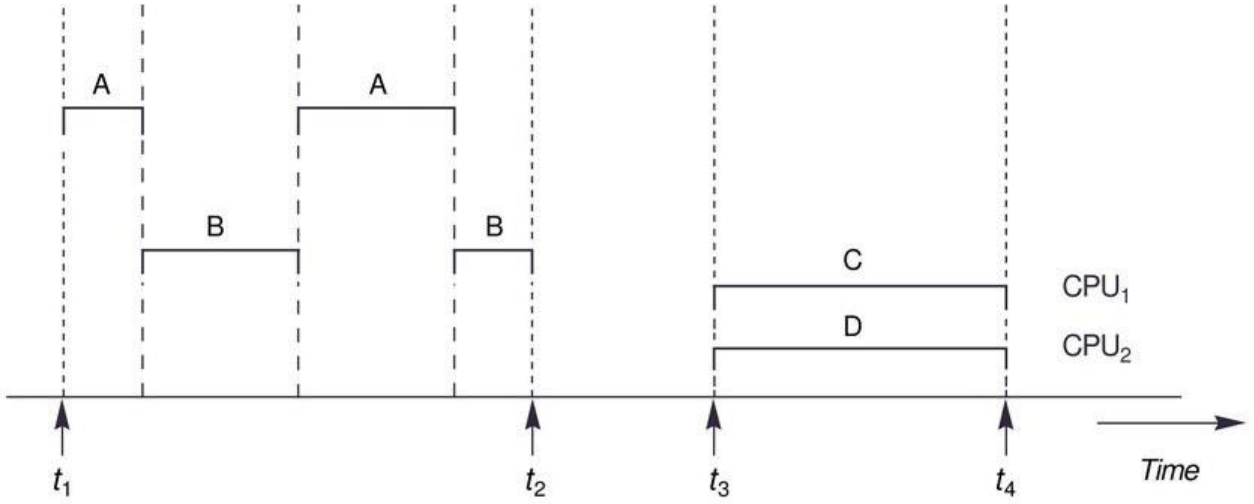
1 Introduction to Transaction Processing (1)

- Multi programming OS execute some commands from one process, then suspend that process and execute some commands from the next process, and so on.
- Hence concurrent execution of the processes is interleaved.

N.B: That doesn't slow down system, as that occur while the input/output operation where the processor is on hold. So instead of having the processor on hold otherwise do other transaction.

1 Introduction to Transaction Processing (1)

- **Interleaved processing**:
  - Concurrent execution of processes is interleaved in a single CPU
- **Parallel processing**:
  - Processes are concurrently executed in multiple CPUs.



Introduction to Transaction Processing (2)

- A **Transaction**:
  - Logical unit of database processing that includes one or more access operations (read -retrieval, write - insert or update, delete).
- A transaction (set of operations) may be stand-alone specified in a high-level language like SQL submitted interactively or may be embedded within a program.
- **Transaction boundaries**:
  - Begin and End transaction.
- An **application program** may contain several transactions separated by the Begin and End transaction boundaries.
  - Read-only transaction
  - Read-write transaction

Introduction to Transaction Processing (3)

SIMPLE MODEL OF A DATABASE (for purposes of discussing transactions):

- **A database** is a collection of named data items
- **Granularity** of data - a field, a record , or a whole disk block (Concepts are independent of granularity)
- Basic operations are **read** and **write**
  - **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 (4)

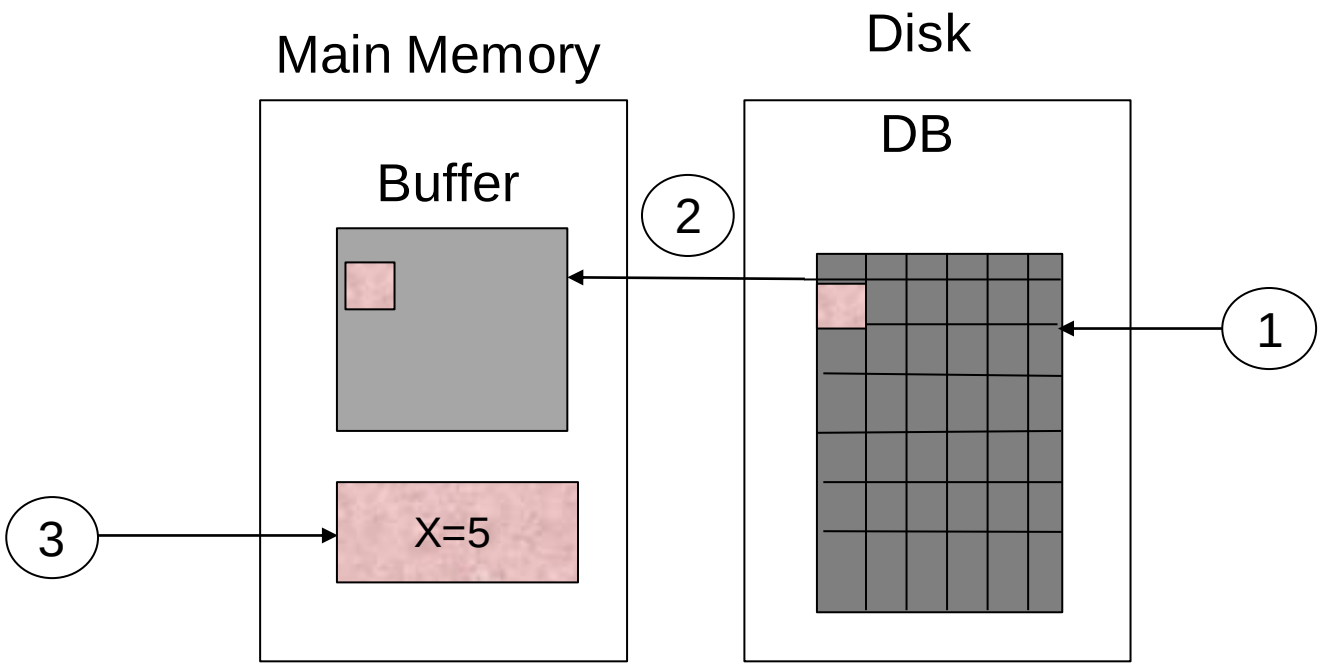
READ AND WRITE OPERATIONS:

- Basic unit of data transfer from the disk to the computer main memory is **one block**. In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a 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 (4)

READ OPERATIONS:



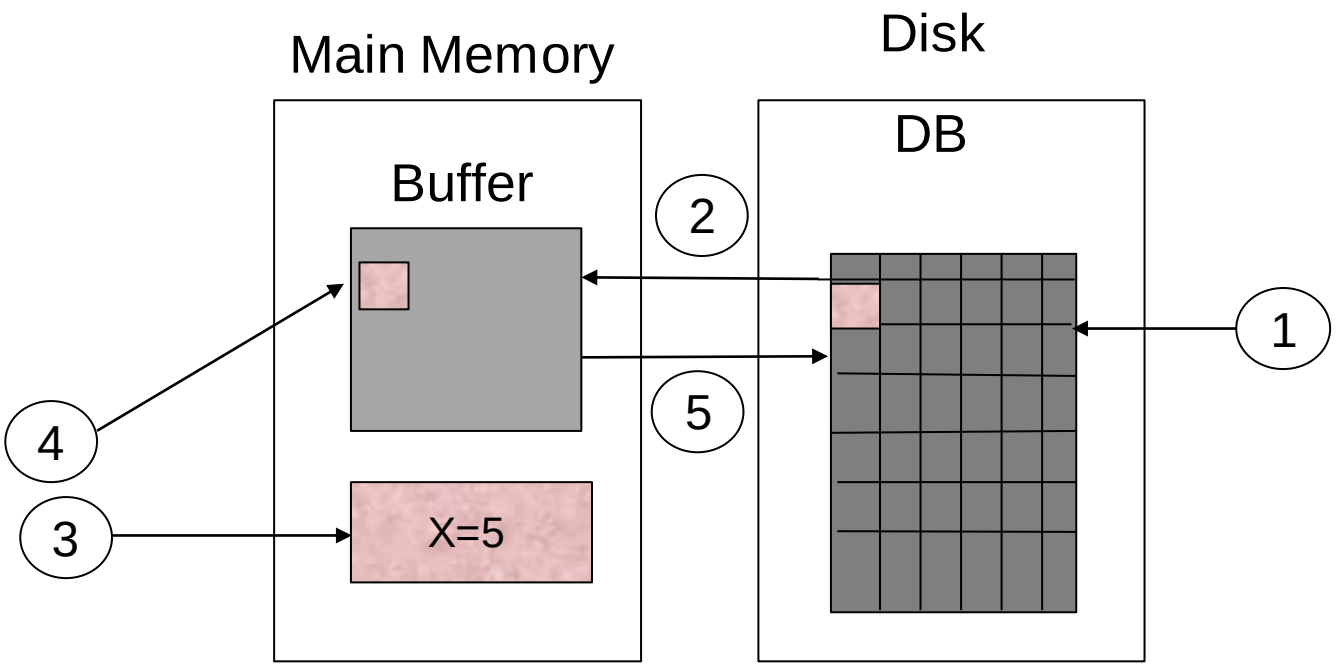
Introduction to Transaction Processing (5)

READ AND WRITE OPERATIONS (contd.):

- **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).

Introduction to Transaction Processing (4)

WRITE OPERATIONS:



Two sample transactions

- FIGURE 17.2 Two sample transactions:
  - (a) Transaction T1
  - (b) Transaction T2

(a) T_1

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

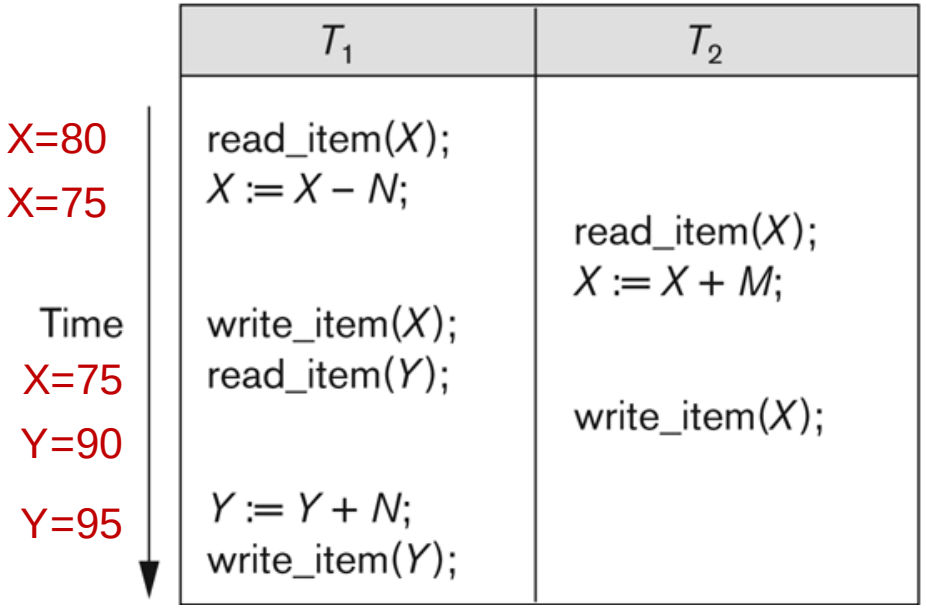
(b) T_2

```
read_item (X);
X:=X+M;
write_item (X);
```

Introduction to Transaction Processing (6)

Why Concurrency Control is needed:

- **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.



Assume:
X=80
Y=90
N=5
M=4

X=80
X=84
(didn't write yet)

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

Wrote without read
X=84 (placed on disk as it's the last write operation)

Introduction to Transaction Processing (6)

Why Concurrency Control is needed:

- **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.

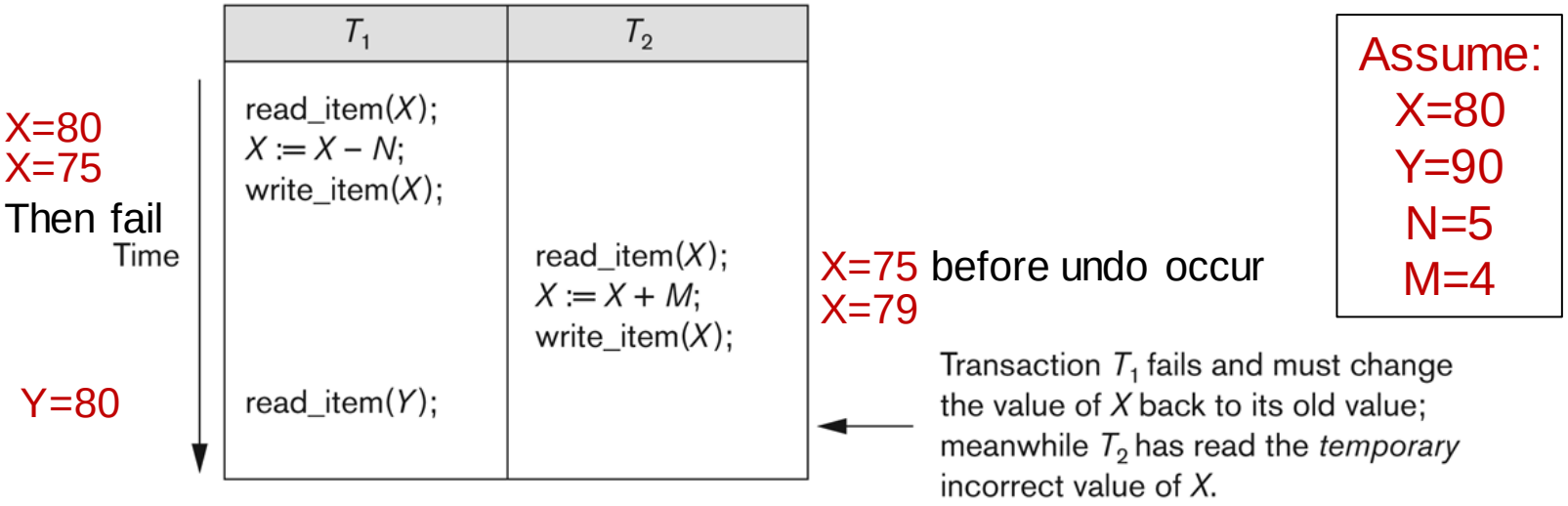
		X	
Transaction A	**Transaction B**	**A**	**B**
Read (X)		X = 100	
X = X+15		115	
	Read (X)		X = 100
	x = x-25		X = 75
	WRITE (X)		X = 75
WRITE (X)		X = 115	

Assume:
X=100

Introduction to Transaction Processing (6)

Why Concurrency Control is needed:

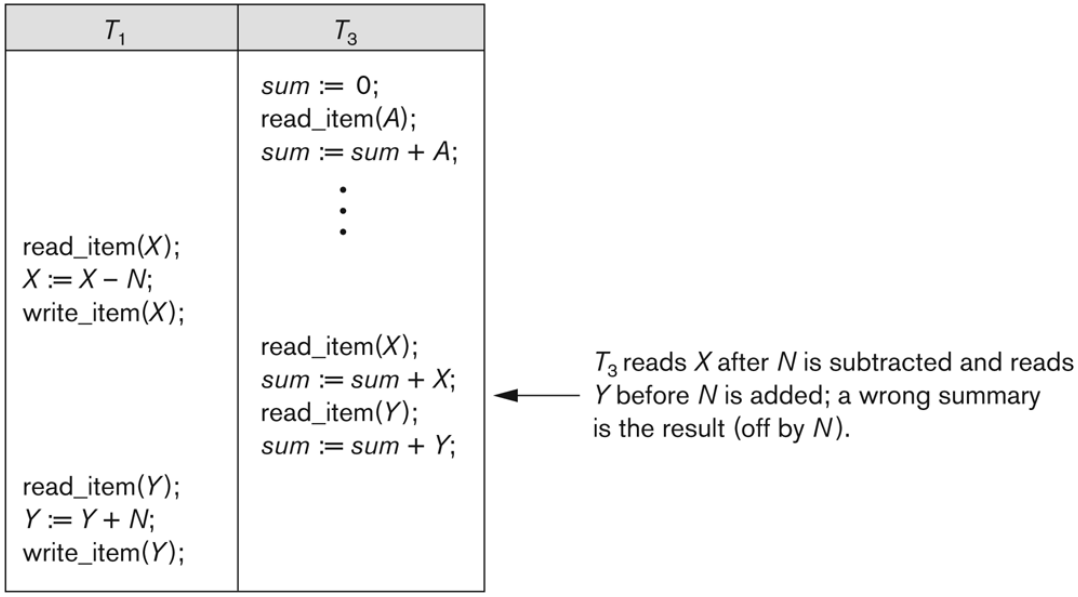
- **The Temporary Update (or Dirty Read) Problem**
  - This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 17.1.4).
  - The updated item is accessed by another transaction before it is changed back to its original value.



Introduction to Transaction Processing (6)

Why Concurrency Control is needed:

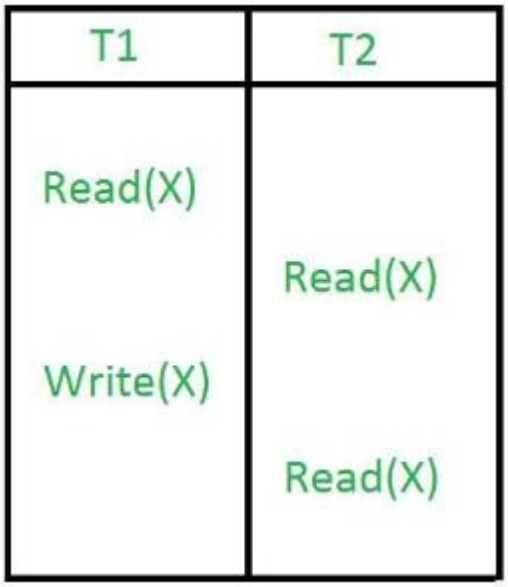
- **The Incorrect Summary Problem**
  - If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.



Introduction to Transaction Processing (6)

Why Concurrency Control is needed:

- **Unrepeatable Read Problem**
  - The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable. .



Introduction to Transaction Processing (12)

Why **recovery** is needed:

(What causes a Transaction to fail)

1. A computer failure (system crash):

   A hardware or software error or network error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.

2. A transaction or system error:

   Some operation in the transaction may cause it to fail, such as integer overflow (include value more than the character accepted in the data item) 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.

Introduction to Transaction Processing (13)

Why **recovery** is needed (Contd.):

(What causes a Transaction to fail)

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 from that account, to be canceled.

   A programmed abort in the transaction causes it 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 18).

Introduction to Transaction Processing (14)

Why **recovery** is needed (contd.):

(What causes a Transaction to fail)

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 may happen during a read or a write operation of the transaction.

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.

2 Transaction and System Concepts (1)

- A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.
  - For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- **Transaction states**:
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

Transaction and System Concepts (2)

- 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.
    - At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database or whether the transaction has to be aborted because it violates concurrency control or for some other reason.

Transaction and System Concepts (3)

- Recovery manager keeps track of the following operations (cont):
  - **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 that the transaction may have applied to the database must be undone.

Transaction and System Concepts (4)

- Recovery techniques use the following operators:
  - **undo**: Similar to rollback except that it applies to a single operation rather than to a whole transaction.
  - **redo**: This specifies that certain *transaction operations* must be *redone* to ensure that all the operations of a committed transaction have been applied successfully to the database.

State transition diagram illustrating the states for transaction execution

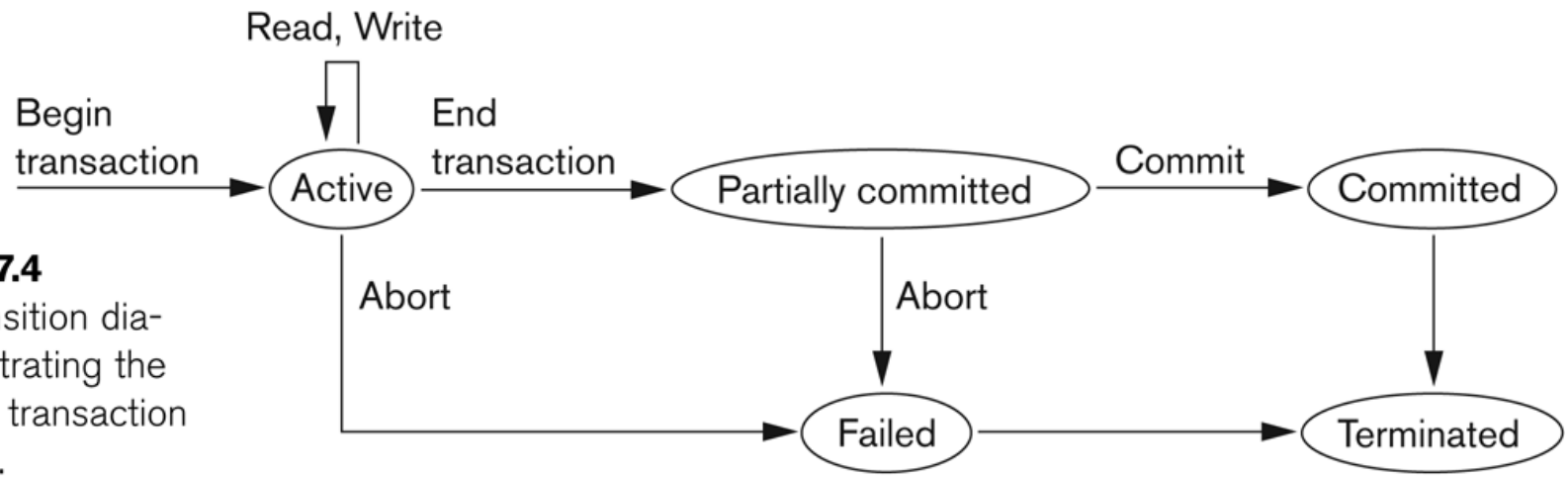


Figure 17.4
State transition diagram illustrating the states for transaction execution.

Failed or aborted transactions may be restarted later- automatically of after resubmitting by the user- as brand-new transactions.

Transaction and System Concepts (6)

- The System Log
  - **Log** or **Journal**: The log keeps track of all transaction operations that affect the values of database items.
    - This information may be needed to permit recovery from transaction failures.
    - 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 is periodically backed up to archival storage (tape) to guard against such catastrophic failures.

Transaction and System Concepts (7)

- The System Log (cont):
  - T in the following discussion refers to a unique **transaction-id** that is generated automatically by the system and is used to identify each transaction:
  - Types of log record:
    - [start_transaction,T]: Records that 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 (recorded permanently) to the database.
    - [abort,T]: Records that transaction T has been aborted.

Transaction and System Concepts (8)

- The System Log (cont):
  - Protocols for recovery that *avoid cascading rollbacks do not require that read operations be written to the system log*, whereas other protocols require these entries for recovery.
  - Strict protocols require simpler write entries that do not include new_value (see Section 17.4).

Transaction and System Concepts (9)

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 described in Chapter 19.
  1. Because the log contains a record of every write operation that changes the value of some database item, it is possible to **undo** the effect of these write operations of a transaction T by tracing backward through the log and resetting all items changed by a write operation of T to their old_values.
  2. We can also **redo** the effect of the write operations of a transaction T by tracing forward through the log and setting all items changed by a write operation of T (that did not get done permanently) to their new_values.

Transaction and System Concepts (10)

Commit Point of a Transaction:

- **Definition a Commit Point:**
  - 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.
  - 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:**
  - Needed for transactions that have a [start_transaction,T] entry into the log but **no** commit entry [commit,T] into the log.

Transaction and System Concepts (11)

Commit Point of a Transaction (cont):

- **Redoing transactions:**
  - Transactions that have written their commit entry in the log must also have recorded all their write operations in the log; otherwise they would not be committed, so their effect on the database can be redone from the log entries. (Notice that the log file must be kept on disk.
  - 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.

Transaction and System Concepts (11)

- Database State: A collection of all the stored data item (values) in the database at a given point in time.
- A consistent state: satisfies the constraints specified in the schema as well as any other constraints on the database that should hold.
- A DB program should be written in a way that guarantees that if the DB is in consistent state before executing the transaction, it will be in a consistent state after the complete execution of the transaction, assuming that no interfere with other transaction occur.

Questions?