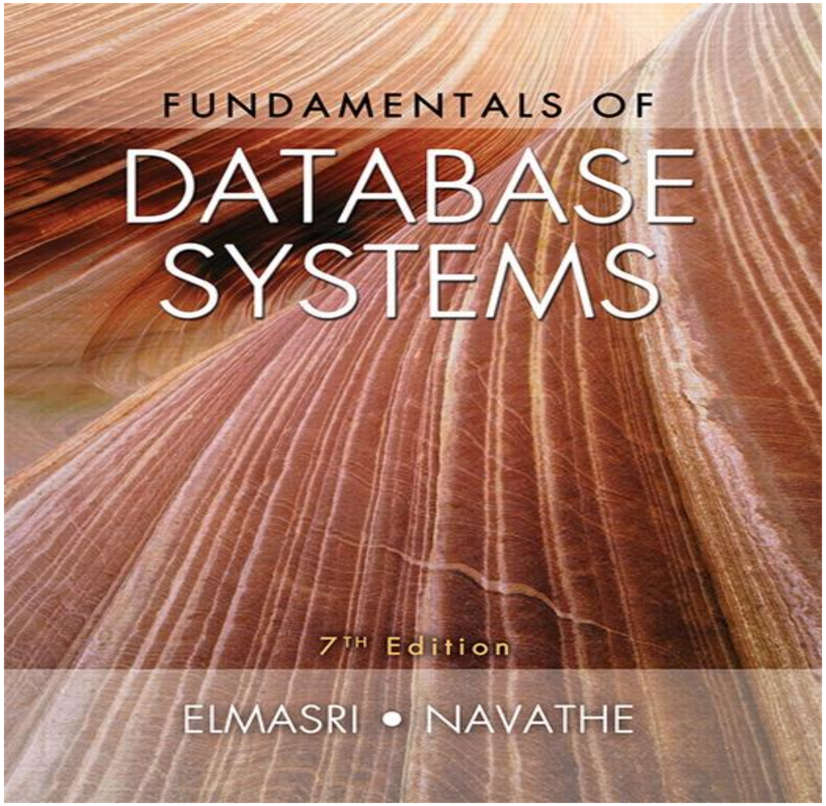
Comp-4150: Advanced and Practical Database Systems

- Ramez Elmasri , Shamkant B. Navathe(2016) Fundamentals of Database Systems (7th Edition), Pearson, isbn 10: 0-13-397077-9; isbn-13:978-0-13-397077-7.

CHAPTER 20

Introduction to Transaction Processing Concepts and Theory



Chapter 20: Introduction to Transaction Processing Concepts and Theory
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
- 7. Other Concurrency Control Issues

Chapter 20: Introduction

- Transaction
  - Is used to Describe logical units of database processing (an example transaction is making an airline reservation, registering for a course)
- Transaction processing systems
  - Are Systems with large databases and hundreds of concurrent users (eg. Users are, airline agents, students in a student information system)
  - Systems Require high availability and fast response time for many concurrent users.
  - How are concurrent transactions executed to ensure correctness? When transactions fail, how does the database system recover?

20.1.1 Single User Versus Multiuser Systems

- Single-user DBMS
  - At most one user at a time can use the system
  - Example: home computer
- Multiuser DBMS
  - Many users can access the system (database) concurrently
  - Example: airline reservations system, student information system.
  - Multiprogramming allows the operating system with one CPU execute multiple programs (that is, multiple transactions or processes) concurrently, at the same time by interleaving the execution of the processes as processes A and B of Fig. 20.1.
  - Parallel execution may also be done with more than one CPU as shown for processes C and D on Fig. 20.1.

20.1.1 Single User Versus Multiuser Systems (cont'd.)

- Multiprogramming
  - Allows operating system to execute multiple processes concurrently
  - Executes commands from one process, then suspends that process and executes commands from another process, etc.

20.1.1 Single User Versus Multiuser Systems (cont'd.)

- Interleaved processing as for processes A and B
- Parallel processing
  - Processes C and D in figure below

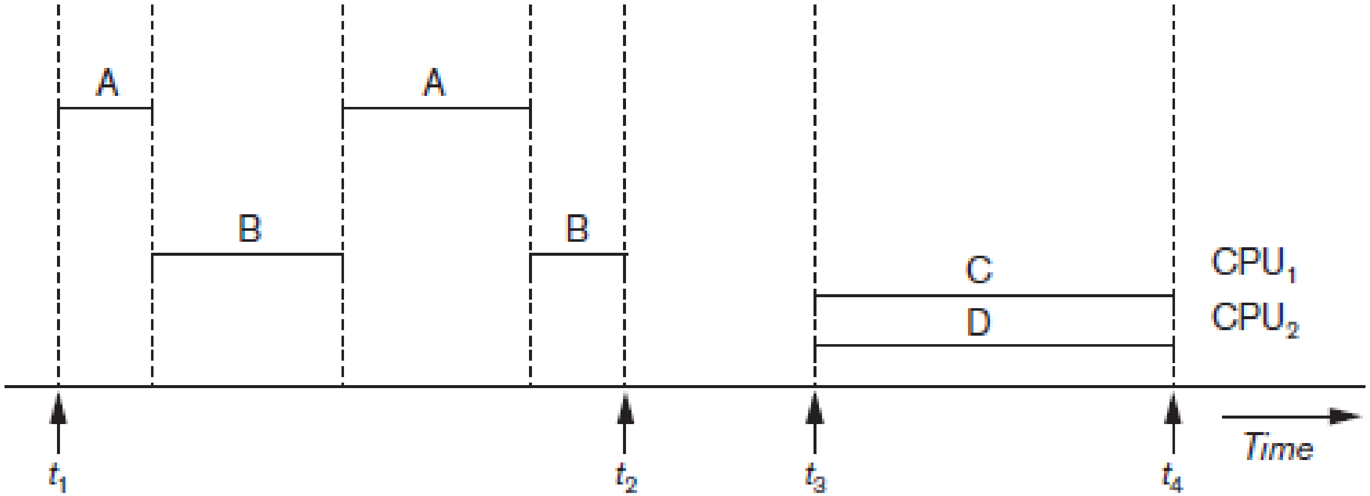


Figure 20.1 Interleaved processing versus parallel processing of concurrent transactions

20.1.2. Transactions

- A Transaction: is an executing program
  - That Forms a logical unit of database processing (eg register for a course)
  - It can include one or more database access operations with insert, delete, update or retrieve operations expressed through a query languages as SQL or through a database application program.
- Begin and end transaction statements
  - Can be used to Specify transaction boundaries in an application. An application can contain many transactions.
- A transaction can be Read-only transaction (if only Reads needed)
- It can also be Read-write transaction for both Read and Update operations on database objects (e.g., Number of students registered in a course).

20.1.2. Database Items

- Database is represented as a collection of named data items
- Size of a data item called data granularity
- Data item can be a
  - Record
  - Disk block
  - Attribute value of a record
- Transaction processing concepts work independent of data item granularity
- Basic transaction operations are expressed using the two operations read_item(X) and write_item(X) for reading a database item and to write a database item name X respectively.

20.1.2. Read and Write Operations

- read_item(X)
  - Reads a database item named X into a program variable named X
  - Process includes finding the address of the disk block, and copying to and from a memory buffer
- write_item(X)
  - Writes the value of program variable X into the database item named X
  - Process includes finding the address of the disk block, copying to and from a memory buffer, and storing the updated disk block back to disk

20.1.2. Read and Write Operations (cont'd.)

- Read set of a transaction
  - Set of all items read
- Write set of a transaction
  - Set of all items written

(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 20.2 Two sample transactions (a) Transaction *T*1 (b) Transaction *T*2

20.1.2. DBMS Buffers

- DBMS will maintain several main memory data buffers in the database cache
- When buffers are occupied, a buffer replacement policy is used to choose which buffer will be replaced
  - Example policy: least recently used

20.1.3. Concurrency Control

- Transactions submitted by various users may execute concurrently
  - Access and update the same database items
  - Some form of concurrency control is needed
- The lost update problem
  - Occurs when two transactions that access the same database items have operations interleaved
  - Results in incorrect value of some database items

20.1.3. The Lost Update Problem

(a)

Time ↓	T_1	T_2
	read_item(X); $X := X - N$;	
		read_item(X); $X := X + M$;
	write_item(X); read_item(Y);	
		write_item(X); ← Item X has an incorrect value because its update by T_1 is *lost* (overwritten).
	$Y := Y + N$; write_item(Y);	

Figure 20.3 Some problems that occur when concurrent execution is uncontrolled (a) The lost update problem

20.1.3. The Temporary Update Problem

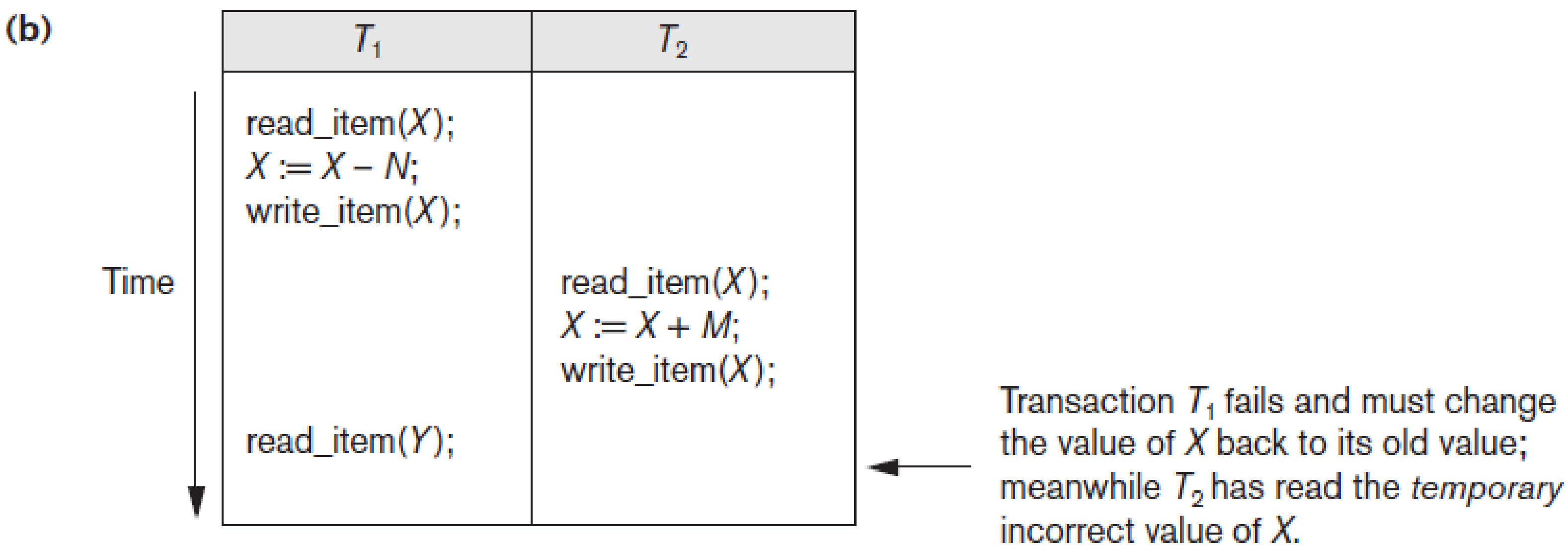


Figure 20.3 (cont'd.) Some problems that occur when concurrent execution is uncontrolled (b) The temporary update problem

20.1.3. The Incorrect Summary Problem

(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*).

Figure 20.3 (cont'd.) Some problems that occur when concurrent execution is uncontrolled (c) The incorrect summary problem

20.1.3. The Unrepeatable Read Problem

- Transaction T reads the same item twice
- Value is changed by another transaction T′ between the two reads
- T receives different values for the two reads of the same item

20.1.4. Why Recovery is Needed

- Committed transaction
  - Effect recorded permanently in the database
- Aborted transaction
  - Does not affect the database
- Types of transaction failures
  - Computer failure (system crash)
  - Transaction or system error
  - Local errors or exception conditions detected by the transaction

20.1.4. Why Recovery is Needed (cont'd.)

- Types of transaction failures (cont'd.)
  - Concurrency control enforcement
  - Disk failure
  - Physical problems or catastrophes
- System must keep sufficient information to recover quickly from the failure
  - Disk failure or other catastrophes have long recovery times

20.2 Transaction and System Concepts

- System must keep track of when each transaction starts, terminates, commits, and/or aborts
  - BEGIN_TRANSACTION
  - READ or WRITE
  - END_TRANSACTION
  - COMMIT_TRANSACTION
  - ROLLBACK (or ABORT)

20.2 Transaction and System Concepts (cont'd.)

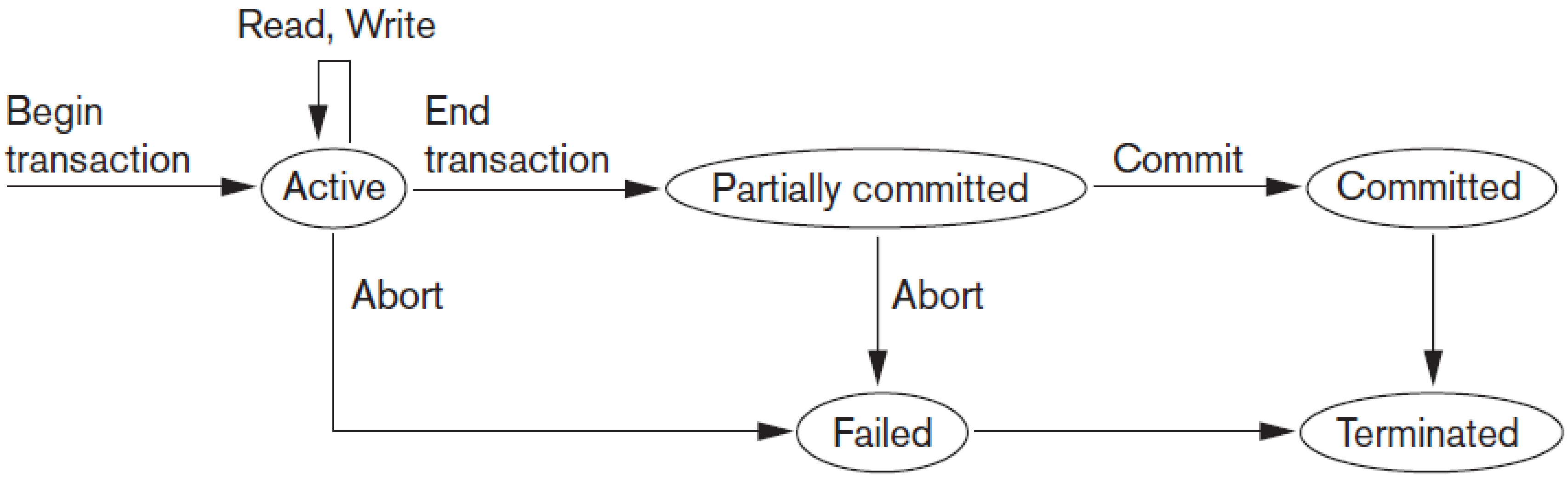


Figure 20.4 State transition diagram illustrating the states for transaction execution

20.2.2 The System Log

- System log keeps track of transaction operations
- Sequential, append-only file
- Not affected by failure (except disk or catastrophic failure)
- Log buffer
  - Main memory buffer
  - When full, appended to end of log file on disk
- Log file is backed up periodically
- Undo and redo operations based on log possible

20.2.3 Commit Point of a Transaction

- Occurs when all operations that access the database have completed successfully
  - And effect of operations recorded in the log
- Transaction writes a commit record into the log
  - If system failure occurs, can search for transactions with recorded start_transaction but no commit record
- Force-writing the log buffer to disk
  - Writing log buffer to disk before transaction reaches commit point

20.2.4. DBMS-Specific Buffer Replacement Policies

- Page replacement policy
  - Selects particular buffers to be replaced when all are full
- Domain separation (DS) method
  - Each domain handles one type of disk pages
    - Index pages
    - Data file pages
    - Log file pages
  - Number of available buffers for each domain is predetermined

20.2.4. DBMS-Specific Buffer Replacement Policies (cont'd.)

- Hot set method
  - Useful in queries that scan a set of pages repeatedly
  - Does not replace the set in the buffers until processing is completed
- The DBMIN method
  - Predetermines the pattern of page references for each algorithm for a particular type of database operation
    - Calculates locality set using query locality set model (QLSM)

20.3 Desirable Properties of Transactions

- ACID properties
  - Atomicity
    - Transaction performed in its entirety or not at all
  - Consistency preservation
    - Takes database from one consistent state to another
  - Isolation
    - Not interfered with by other transactions
  - Durability or permanency
    - Changes must persist in the database

20.3 Desirable Properties of Transactions (cont'd.)

- Levels of isolation
  - Level 0 isolation does not overwrite the dirty reads of higher-level transactions
  - Level 1 isolation has no lost updates
  - Level 2 isolation has no lost updates and no dirty reads
  - Level 3 (true) isolation has repeatable reads
    - In addition to level 2 properties
  - Snapshot isolation

20.4 Characterizing Schedules Based on Recoverability

- Schedule or history
  - Order of execution of operations from all transactions
  - Operations from different transactions can be interleaved in the schedule
- Total ordering of operations in a schedule
  - For any two operations in the schedule, one must occur before the other

20.4 Characterizing Schedules Based on Recoverability (cont'd.)

- Two conflicting operations in a schedule
  - Operations belong to different transactions
  - Operations access the same item X
  - At least one of the operations is a write_item(X)
- Two operations conflict if changing their order results in a different outcome
- Read-write conflict
- Write-write conflict

20.4 Characterizing Schedules Based on Recoverability (cont'd.)

- Recoverable schedules
  - Recovery is possible
- Nonrecoverable schedules should not be permitted by the DBMS
- No committed transaction ever needs to be rolled back
- Cascading rollback may occur in some recoverable schedules
  - Uncommitted transaction may need to be rolled back

20.4 Characterizing Schedules Based on Recoverability (cont'd.)

- Cascadeless schedule
  - Avoids cascading rollback
- Strict schedule
  - Transactions can neither read nor write an item X until the last transaction that wrote X has committed or aborted
  - Simpler recovery process
    - Restore the before image

20.5 Characterizing Schedules Based on Serializability

- Serializable schedules
  - Always considered to be correct when concurrent transactions are executing
  - Places simultaneous transactions in series
    - Transaction T_1 before T_2, or vice versa

(a)

T_1	T_2

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

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

Time ↓

Schedule A

(b)

T_1	T_2

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

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

Time ↓

Schedule B

(c)

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Time ↓

Schedule C

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

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

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

Time ↓

Schedule D

Figure 20.5 Examples of serial and nonserial schedules involving transactions *T*1 and *T*2 (a) Serial schedule A: *T*1 followed by *T*2 (b) Serial schedule B: *T*2 followed by *T*1 (c) Two nonserial schedules C and D with interleaving of operations

20.5 Characterizing Schedules Based on Serializability (cont'd.)

- Problem with serial schedules
  - Limit concurrency by prohibiting interleaving of operations
  - Unacceptable in practice
  - Solution: determine which schedules are equivalent to a serial schedule and allow those to occur
- Serializable schedule of *n* transactions
  - Equivalent to some serial schedule of same *n* transactions

20.5 Characterizing Schedules Based on Serializability (cont'd.)

- Result equivalent schedules
  - Produce the same final state of the database
    - May be accidental
  - Cannot be used alone to define equivalence of schedules

S_1

read_item(X);
$X := X + 10$;
write_item(X);

S_2
read_item(X); $X := X * 1.1$; write_item (X);

Figure 20.6 Two schedules that are result equivalent for the initial value of X = 100 but are not result equivalent in general

20.5 Characterizing Schedules Based on Serializability (cont'd.)

- Conflict equivalence
  - Relative order of any two conflicting operations is the same in both schedules
- Serializable schedules
  - Schedule S is serializable if it is conflict equivalent to some serial schedule S'.

20.5 Characterizing Schedules Based on Serializability (cont'd.)

- Testing for serializability of a schedule

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a read_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
3. For each case in S where T_j executes a write_item(X) after T_i executes a read_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

Algorithm 20.1 Testing conflict serializability of a schedule S

20.5 Characterizing Schedules Based on Serializability (cont'd.)

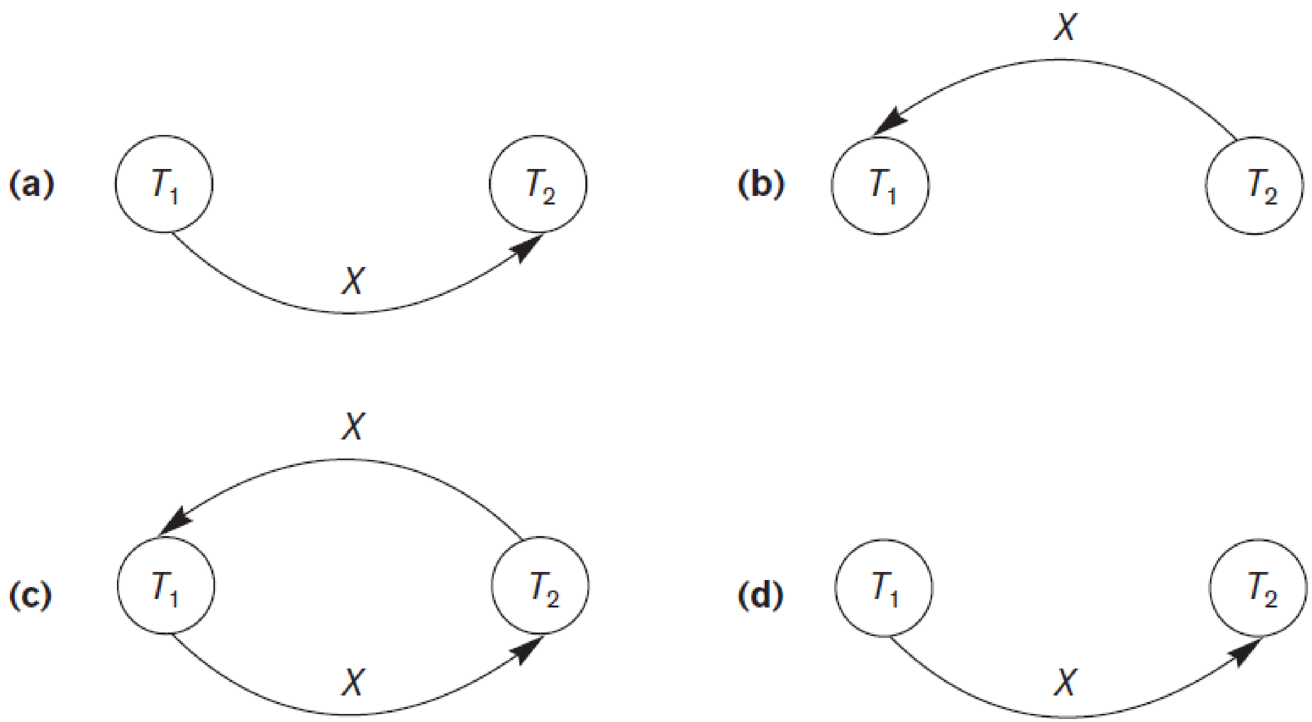


Figure 20.7 Constructing the precedence graphs for schedules A to D from Figure 20.5 to test for conflict serializability (a) Precedence graph for serial schedule A (b) Precedence graph for serial schedule B (c) Precedence graph for schedule C (not serializable) (d) Precedence graph for schedule D (serializable, equivalent to schedule A)

20.5 How Serializability is Used for Concurrency Control

- Being serializable is different from being serial
- Serializable schedule gives benefit of concurrent execution
  - Without giving up any correctness
- Difficult to test for serializability in practice
  - Factors such as system load, time of transaction submission, and process priority affect ordering of operations
- DBMS enforces protocols
  - Set of rules to ensure serializability

20.5 View Equivalence and View Serializability

- View equivalence of two schedules
  - As long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results
  - Read operations said to see the same view in both schedules
- View serializable schedule
  - View equivalent to a serial schedule

20.5 View Equivalence and View Serializability (cont'd.)

- Conflict serializability similar to view serializability if constrained write assumption (no blind writes) applies
- Unconstrained write assumption
  - Value written by an operation can be independent of its old value
- Debit-credit transactions
  - Less-stringent conditions than conflict serializability or view serializability

20.6 Transaction Support in SQL

- No explicit Begin_Transaction statement
- Every transaction must have an explicit end statement
  - COMMIT
  - ROLLBACK
- Access mode is READ ONLY or READ WRITE
- Diagnostic area size option
  - Integer value indicating number of conditions held simultaneously in the diagnostic area

20.6 Transaction Support in SQL (cont'd.)

- Isolation level option
  - Dirty read
  - Nonrepeatable read
  - Phantoms

	Type of Violation		
Isolation Level	Dirty Read	Nonrepeatable Read	Phantom
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Table 20.1 Possible violations based on isolation levels as defined in SQL

20.6 Transaction Support in SQL (cont'd.)

- Snapshot isolation
  - Used in some commercial DBMSs
  - Transaction sees data items that it reads based on the committed values of the items in the database snapshot when transaction starts
  - Ensures phantom record problem will not occur

20.7 Summary

- Single and multiuser database transactions
- Uncontrolled execution of concurrent transactions
- System log
- Failure recovery
- Committed transaction
- Schedule (history) defines execution sequence
  - Schedule recoverability
  - Schedule equivalence
- Serializability of schedules