Database Management Systems

Associate Professor Dr. Raed Ibraheem Hamed

University of Human Development, College of Science and Technology
Computer Science Department

2015 – 2016

Points to Cover

- Transaction Concept
- Transaction properties
- 4 Properties of a Transaction
- Transaction Management with SQL
- Transaction Log
- Concurrency Control
- Transaction State
- Schedules
- Example Schedule 1,2,3

Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.

- **E.g. transaction to transfer €50 from account A to account B:**
  1. read_from_acoount(A)
  2. A := A – 50
  3. write_to_account(A)
  4. read_from_accont(B)
  5. B := B + 50
  6. write_to_account(B)

Transaction properties

- **Isolation requirement** — if between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum A + B will be less than it should be).

  T1 **T2**

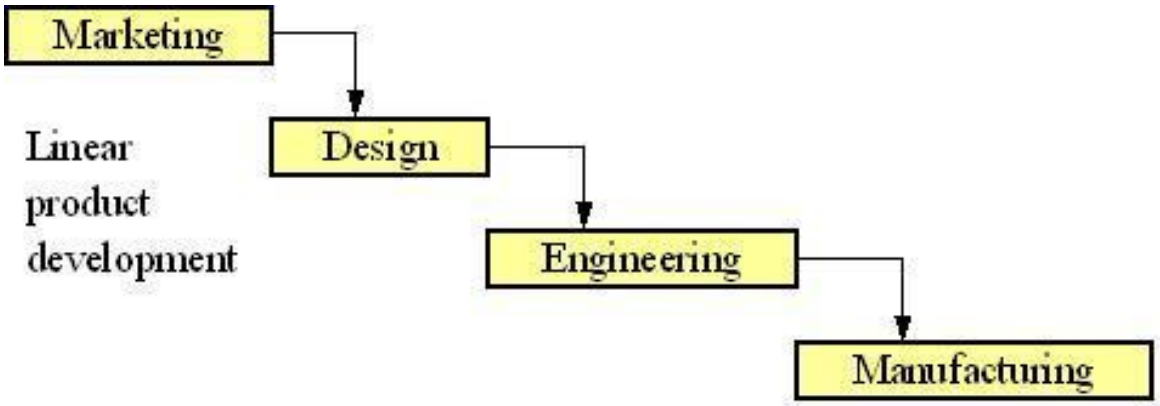
  1. **read**(A)
  2. $A := A - 50$
  3. **write**(A)

     …….. read(A), read(B), print(A+B)
  4. **read**(B)
  5. $B := B + 50$
  6. **write**(B)

- Isolation can be ensured trivially by running transactions **serially**
  - that is, one after the other.
- However, executing multiple transactions concurrently has significant benefits, as we will see later.

Concurrent Engineering

Concurrent is a work methodology based on the parallelization of tasks (i.e. performing tasks concurrently), Concurrent engineering more effective than sequential process



Concurrent

4 Properties of a Transaction

- **Atomic – All or Nothing**

  All parts of the transaction must be completed or it must be aborted and move back.

- **Consistent**

  Each user is responsible to ensure that their transaction would leave the database in a consistent state.

4 Properties of a Transaction

- **Isolation**

  The final effects of multiple simultaneous transactions must be the same as if they were executed one right after the other.

- **Durability**

  If a transaction has been committed, the DBMS must ensure that its effects are permanently recorded in the database (even if the system crashes)

Transaction Management with SQL

- SQL Statements → Commit / Rollback
- When a transaction sequence is initiated it must continue through all succeeding SQL statements until:
  1. A Commit Statement is Reached
  2. A Rollback Statement is Reached
  3. The End of the Program is Reached (Commit)
  4. The Program is Abnormally Terminated (Rollback)

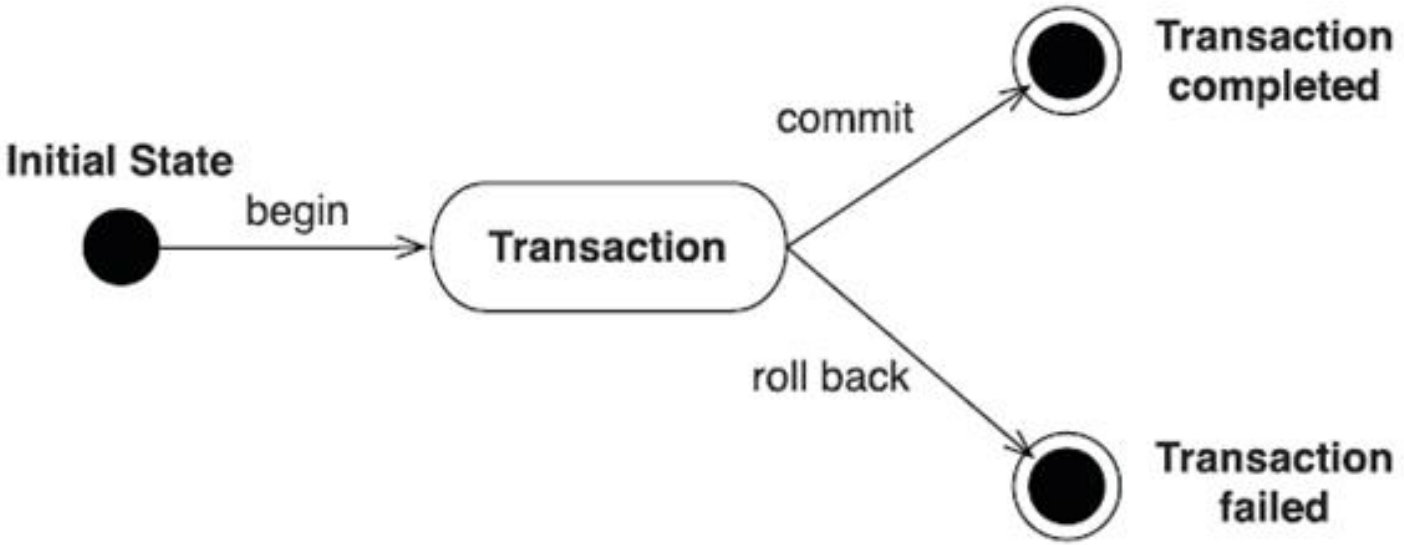
Transaction Log

- Keeps track of all transactions that update the database
  1. Record for the beginning of the transaction
  2. Type of operation (insert / update / delete)
  3. Names of objects/tables affected by the transaction
  4. Before and After Values for Updated Fields
  5. Pointers to Previous and Next Transaction Log Entries for the same transaction
  6. The Ending of the Transaction (Commit)
- Used for recovery in case of a Rollback

Concurrency Control

- Coordination of simultaneous transactions execution in a multiprocessing database system
- Ensure transaction serializability in a multi-user database
- Lack of Concurrency Control can create data integrity and consistency problems:
  - Lost Updates
  - Uncommitted Data
  - Inconsistent Retrievals

Transaction Life Cycle



Serializability

Definition : It is sequences of actions (read, write, commit, abort) from set of transactions (which obey the sequence of operations with in the transactions).

Serial schedules : The schedules which gets execute one after other is serial schedules.

Interleaved schedules : The schedules which allows the parts of multiple transactions to get execute.

Serializability

Serial Schedule	
T1	T2
R1(A)	
W1(A)	
R1(B)	
R1(B)	
C1	
	R2(A)
	W2(A)
	R2(B)
	W2(B)
	C2

schedule-1

Interleaved schedule	
T1	T2
R1(A)	
W1(A)	
	R2(A)
	W2(A)
R1(B)	
R1(B)	
C1	
	R2(B)
	W2(B)
	C2

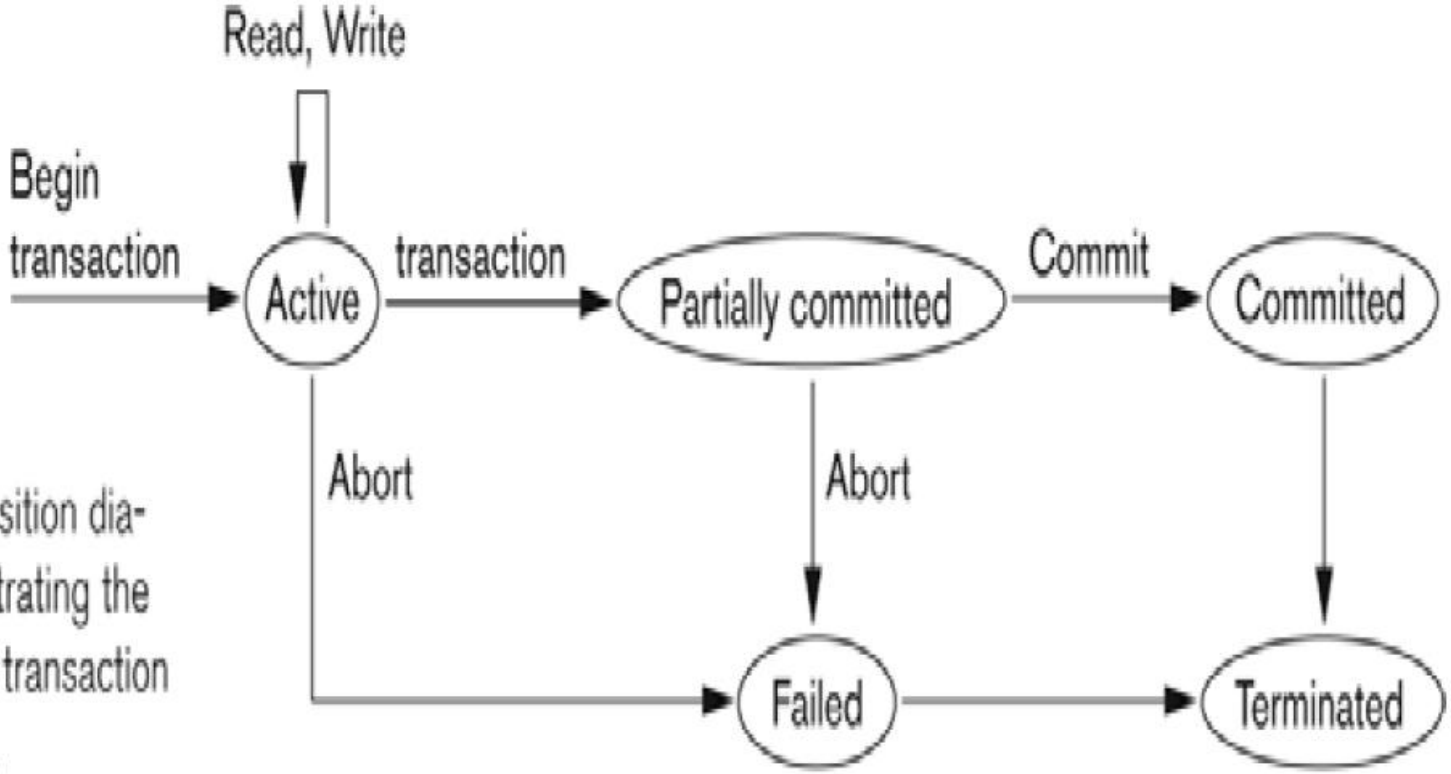
schedule-2

Transaction States

- **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 back and the database restored to its state prior to the start of the transaction.
- **Committed** – after successful completion.
- To guarantee atomicity, external observable action should all be performed (in order) after the transaction is committed.

Transaction States

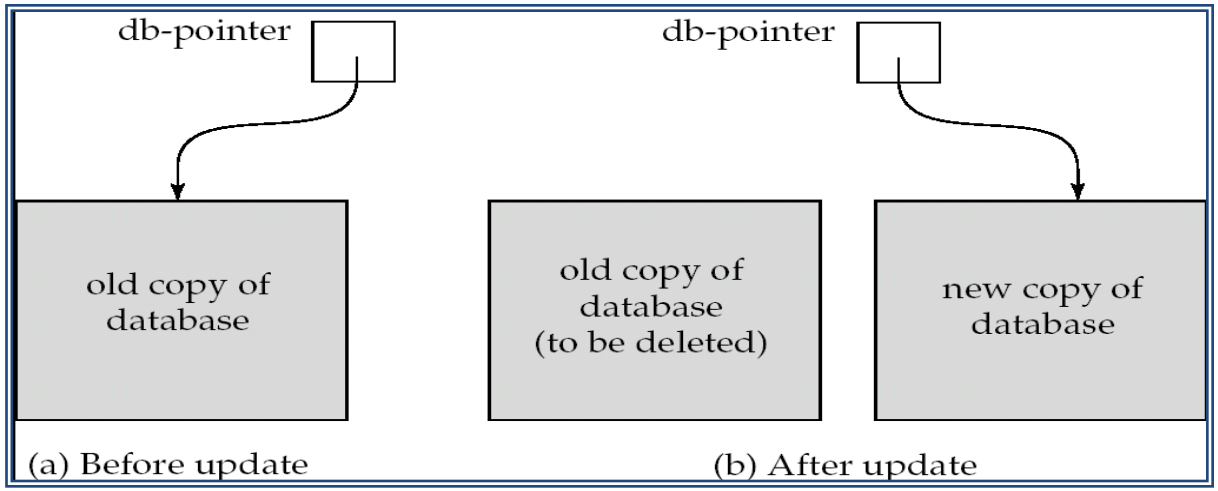
State transition diagram illustrating the states for transaction execution:



State transition diagram illustrating the states for transaction execution.

Implementation of Atomicity and Durability

- The **recovery-management** component of a database system implements the support for atomicity and durability.
- E.g. the ***shadow-database*** scheme:
  - all updates are made on a *shadow copy* of the database
    - **db_pointer** is made to point to the updated shadow copy after
      - the transaction reaches partial commit and
      - all updated pages have been flushed to disk.



Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - a schedule for a set of transactions must consist of all instructions of those transactions
- A transaction that successfully completes its execution will have a commit instructions as the last statement
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement
- The goal is to find schedules that preserve the consistency.

Example Schedule 1

- Let T_1 transfer €50 from A to B, and T_2 transfer 10% of the balance from A to B.
- A serial schedule in which T_1 is followed by T_2 :

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Example Schedule 2

A serial schedule where T_2 is followed by T_1

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

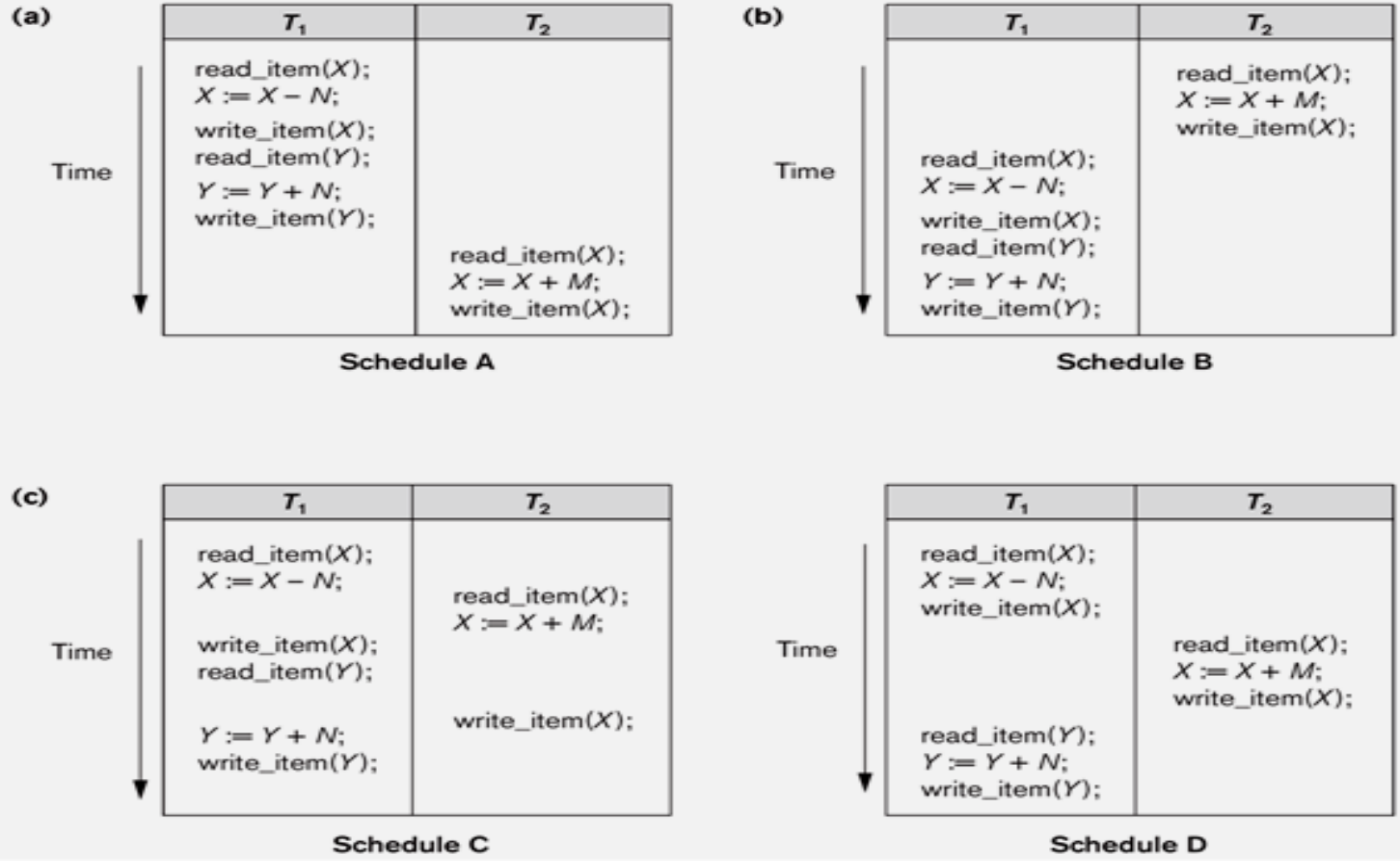
Example Schedule 3

- Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is *equivalent* to Schedule 1.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Transaction schedules

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.



Thank you

???