



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept
ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

Database Management Systems

Module 46: Transactions/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept
ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- Need for indexing database tables
- Understood the ordered indexes
- Recap of Balanced BST for optimal in-memory search data structures
- Issues of external search data structures for persistent data
- Explored 2-3-4 Tree as a precursor to B/B+-Tree
- Understood the B⁺ Tree and B Tree for Index files and data files
- Explored Static and Dynamic Hashing
- Compared Ordered Indexing and Hashing
- Studied the use of Bitmap Indices
- Learnt to create indexes in SQL
- Learnt a set of Ground Rules for Indexing



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules

Example

Module Summary

- To understand the concept of transaction – ‘doing a task in a database’ and its state
- To explore issues in concurrent execution of transactions



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules

Example

Module Summary

- Transaction Concept
- Transaction State
- Concurrent Executions



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

**Transaction
Concept**

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

Transaction Concept



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- A **transaction** is a *unit* of program execution that accesses and, possibly updates, various data items
- For example, transaction to transfer \$50 from account A to account B:
 1. **read**(A)
 2. $A := A - 50$
 3. **write**(A)
 4. **read**(B)
 5. $B := B + 50$
 6. **write**(B)
- Two main issues to deal with:
 - **Failures** of various kinds, such as hardware failures and system crashes
 - **Concurrent execution** of multiple transactions



Required Properties of a Transaction: ACID: Atomicity

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- **Atomicity Requirement**

- If the transaction fails after step 3 and before step 6, money will be “lost” leading to an inconsistent database state
 - ▷ Failure could be due to software or hardware
- The system should ensure that updates of a partially executed transaction are not reflected in the database

Transaction to transfer \$50 from account A to account B:

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)



Required Properties of a Transaction: ACID: Consistency

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

● Consistency Requirement

- $A + B$ must be unchanged by the execution of the transaction
- In general, consistency requirements include
 - ▷ Explicitly specified integrity constraints
 - primary keys and foreign keys
 - ▷ Implicit integrity constraints
 - sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction, when starting to execute, must see a consistent database
- During transaction execution the database may be temporarily inconsistent
- When the transaction completes successfully the database must be consistent
 - ▷ Erroneous transaction logic can lead to inconsistency

Transaction to transfer
\$50 from account A to
account B:

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)



Required Properties of a Transaction: ACID: Isolation

Module 46

Partha Pratim
Das

Week Recap

Objectives &
OutlineTransaction
Concept

ACID

Transaction
StatesState Transition
DiagramConcurrent
ExecutionsSchedules
Example

Module Summary

● Isolation Requirement

- If between steps 3 and 6 (of the fund transfer transaction), 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)	read (A), read (B), print ($A + B$)
2. $A := A - 50$	
3. write (A)	
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



Required Properties of a Transaction: ACID: Durability

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- **Durability Requirement**

- Once the user has been notified that the transaction has completed (that is, the transfer of the \$50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures

Transaction to transfer \$50 from account A to account B:

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)



ACID Properties

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

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

- **Atomicity:** Atomicity guarantees that each **transaction is treated as a single *unit***, which **either succeeds completely, or fails completely**
 - If any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged
 - Atomicity must be guaranteed in every situation, including power failures, errors and crashes
- **Consistency:** Consistency ensures that a transaction can only bring the database **from one valid state to another**, maintaining database invariants
 - Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof
- **Isolation:** Transactions are often executed concurrently (multiple transactions reading and writing to a table at the same time)
 - **Isolation ensures that concurrent execution** of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially
- **Durability:** Durability guarantees that **once a transaction has been committed, it will remain committed** even in the case of a system failure (like power outage or crash)
 - This usually means that completed transactions (or their effects) are recorded in non-volatile memory



ACID Properties: Quick Reckoner

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

A - Atomicity

All or Nothing Transactions

C - Consistency

Guarantees Committed Transaction State

I - Isolation

Transactions are Independent

D - Durability

Committed Data is Never Lost

(c) <http://blog.sqlauthority.com>



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

**Transaction
States**

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

Transaction States



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules

Example

Module Summary

- Every transaction can be in one of the following states (like Process States in OS)
 - **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. Two options after it has been aborted:
 - Restart the transaction: Can be done only if no internal logical error
 - Kill the transaction
 - **Committed**
 - ▷ After successful completion
 - **Terminated**
 - ▷ After it has been committed or aborted (killed)



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept

ACID

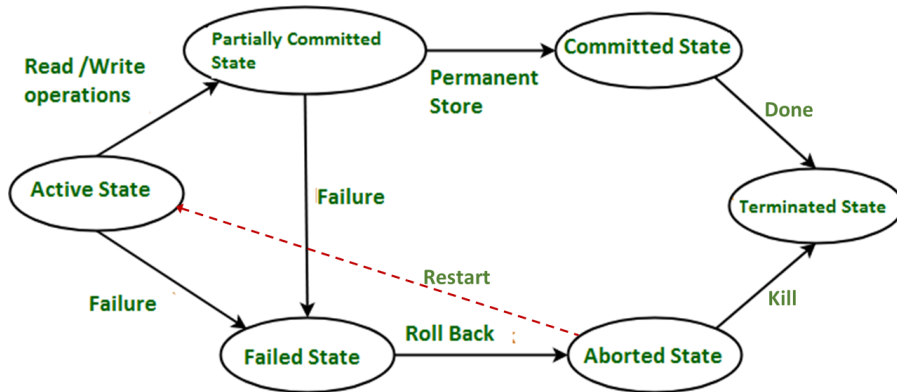
Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary



Concurrent Executions

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept
ACID

Transaction
States

State Transition
Diagram

**Concurrent
Executions**

Schedules
Example

Module Summary

Concurrent Executions



Concurrent Executions

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept
ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
 - **Increased processor and disk utilization**, leading to better transaction *throughput*
 - ▷ For example, one transaction can be using the CPU while another is reading from or writing to the disk
 - **Reduced average response time** for transactions: short transactions need not wait behind long ones
- **Concurrency Control Schemes**: Mechanisms to achieve isolation
 - To control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database



Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept
ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- **Schedule**: A sequence 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
 - Must *preserve the order* in which the instructions appear in each individual transaction
- A transaction that successfully completes its execution will have a **commit** instructions as the last statement
 - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an **abort** instruction as the last statement



Schedule 1

Module 46

Partha Pratim
Das

Week Recap

Objectives &
OutlineTransaction
Concept

ACID

Transaction
StatesState Transition
DiagramConcurrent
Executions

Schedules

Example

Module Summary

- Let T_1 transfer \$50 from A to B , and T_2 transfer 10% of the balance from A to B
- An example of a **serial** schedule in which T_1 is followed by T_2 :

T_1	T_2
<code>read (A)</code> <code>A := A - 50</code> <code>write (A)</code> <code>read (B)</code> <code>B := B + 50</code> <code>write (B)</code> <code>commit</code>	<code>read (A)</code> <code>temp := A * 0.1</code> <code>A := A - temp</code> <code>write (A)</code> <code>read (B)</code> <code>B := B + temp</code> <code>write (B)</code> <code>commit</code>

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
50	200	250	T1, write A	
50	250	300	T1, write B	@ Commit
45	250	295	T2, write A	
45	255	300	T2, write B	@Commit

Consistent @ Commit

Inconsistent @ Transit

Inconsistent @ Commit



Module 46

Partha Pratim
Das

Week Recap

Objectives &
OutlineTransaction
Concept

ACID

Transaction
StatesState Transition
DiagramConcurrent
Executions

Schedules

Example

Module Summary

- A **serial** schedule in which T_2 is followed by T_1 :

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

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
90	210	300	T2, write B	@ Commit
40	210	250	T1, write A	
40	260	300	T1, write B	@Commit

Consistent @ Commit

Inconsistent @ Transit

Inconsistent @ Commit

Values of A & B are different from
Schedule 1 – yet consistent



Schedule 3

Module 46

Partha Pratim
Das

Week Recap

Objectives &
OutlineTransaction
Concept

ACID

Transaction
StatesState Transition
DiagramConcurrent
ExecutionsSchedules
Example

Module Summary

- 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

Schedule 3

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) commit	
	read (B) $B := B + temp$ write (B) commit

Schedule 1

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

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
50	200	250	T1, write A	
45	200	245	T2, write A	
45	250	295	T1, write B	@ Commit
45	255	300	T2, write B	@Commit

Consistent @ Commit

Inconsistent @ Transit

Inconsistent @ Commit

Note – In schedules 1, 2 and 3, the sum " $A + B$ " is preserved



Schedule 4

Module 46

Partha Pratim
Das

Week Recap

Objectives &
OutlineTransaction
Concept

ACID

Transaction
StatesState Transition
DiagramConcurrent
Executions

Schedules

Example

Module Summary

- The following concurrent schedule does not preserve the sum of " $A + B$ "

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

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
50	200	250	T1, write A	
50	250	300	T1, write B	@ Commit
50	210	260	T2, write B	@ Commit

Consistent @ Commit

Inconsistent @ Transit

Inconsistent @ Commit



Module Summary

Module 46

Partha Pratim
Das

Week Recap

Objectives &
Outline

Transaction
Concept
ACID

Transaction
States

State Transition
Diagram

Concurrent
Executions

Schedules
Example

Module Summary

- A task in a database is done as a transaction that passes through several states
- Transactions are executed in concurrent fashion for better throughput
- Concurrent execution of transactions raise serializability issues that need to be addressed
- All schedules may not satisfy ACID properties

Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors.

Edited and new slides are marked with “PPD”.