Database Technology

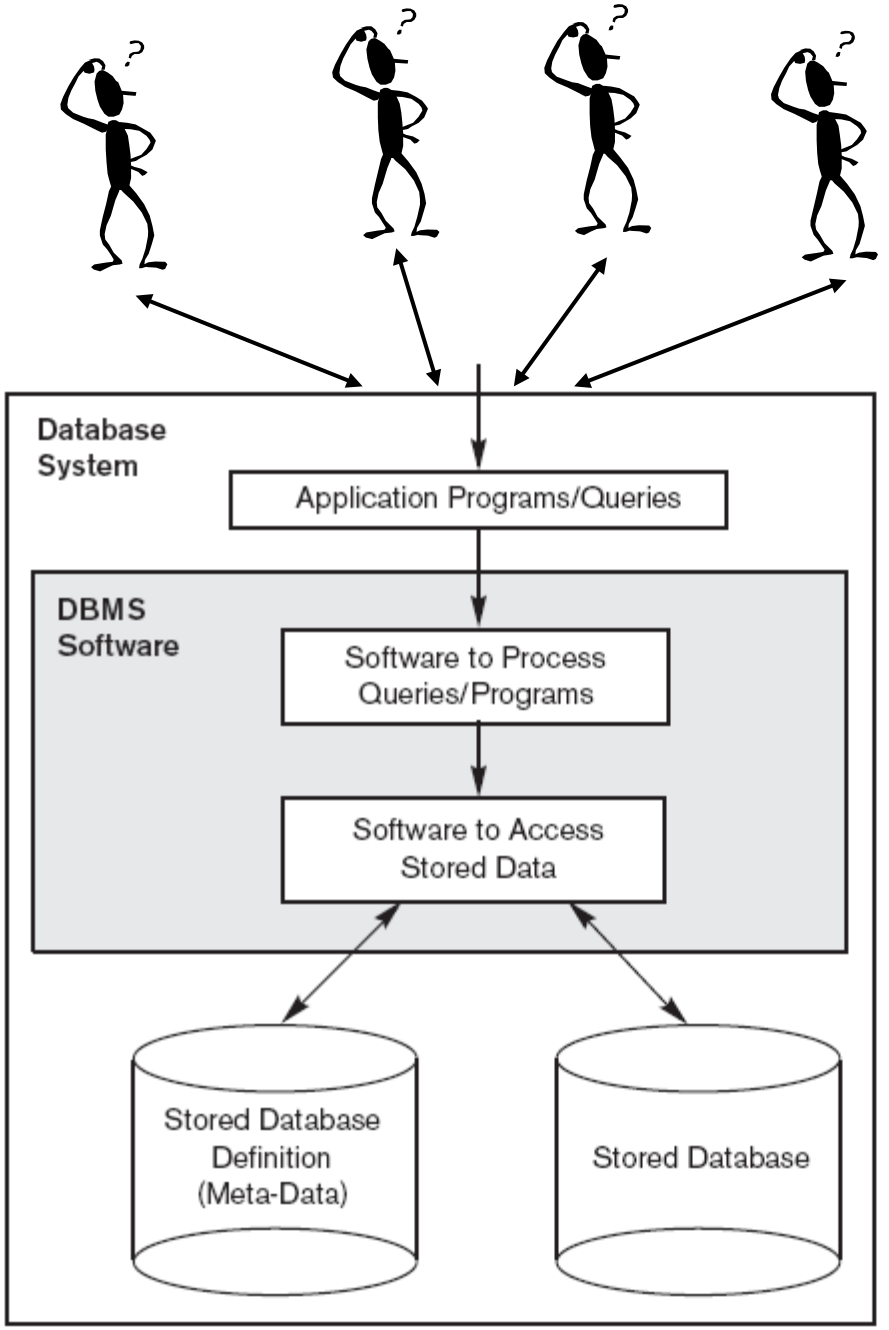
Topic 9: Introduction to Transaction Processing

Olaf Hartig
olaf.hartig@liu.se

LINKÖPING UNIVERSITY

Motivation

- A DB is a shared resource accessed by many users and processes concurrently
- Not managing concurrent access to a shared resource will cause problems (not unlike in operating systems)
- Transaction processing is about avoiding problems caused by
  - concurrency
  - failure



What can go wrong?

- Consider two concurrently executing transactions:

	at ATM window #1		at ATM window #2
1	read_item(savings);	a	read_item(checking);
2	savings = savings - $100;	b	checking = checking - $20;
3	write_item(savings);	c	write_item(checking);
4	read_item(checking);	d	dispense $20 to customer;
5	checking = checking + $100;		
6	write_item(checking);		

- System might crash after a TA begins and before it ends
  - Updates lost if write to disk not performed before crash
  - Money lost if crash between 3–6 or between c–d
- Checking account might have incorrect amount recorded
  - Quiz on next slide ...

Quiz

- If the initial value of checking is $500, what value does it have after the following interleaved execution completes?

	at ATM window #1	**at ATM window #2**
1	read_item(savings);	
2	savings = savings - $100;	
3		read_item(checking);
4	write_item(savings);	
5	read_item(checking);	
6		checking = checking - $20;
7		write_item(checking);
8	checking = checking + $100;	
9	write_item(checking);	
10		dispense $20 to customer;

A: $480 **B:** $500 **C:** $580 **D:** $600

What can go wrong?

- Consider two concurrently executing transactions:

	at ATM window #1		at ATM window #2
1	read_item(savings);	a	read_item(checking);
2	savings = savings - $100;	b	checking = checking - $20;
3	write_item(savings);	c	write_item(checking);
4	read_item(checking);	d	dispense $20 to customer;
5	checking = checking + $100;		
6	write_item(checking);		

- System might crash after a TA begins and before it ends
  - Updates lost if write to disk not performed before crash
  - Money lost if crash between 3–6 or between c–d
- Checking account might have incorrect amount recorded
  - $20 withdrawal lost if T2 executed between 4–6
  - $100 deposit lost if T1 executed between a–c

Basic Terminology and Desirable Properties

Transaction

- An application-specified, *atomic* and *durable* unit of work (a process) that comprises one or more database access operations
- Example from a banking database: Transfer $100 from a checking account to a savings account
- Characteristic operations
  - Read (database retrieval, such as SQL SELECT)
  - Write (modify DB, such as INSERT, UPDATE, DELETE)

Transaction (cont'd)

- Transaction boundaries:
  - Begin_transaction
  - End_transaction
- Transactions can end in one of two states:
  - Commit: transaction completes successfully and all of its results are made permanent
  - Abort: transaction does not complete and none of its actions are reflected in the database

ACID Properties

- **A**tomicity: a transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all
- **C**onsistency preservation: a correct execution of a TA must take the DB from one consistent state to another
- **I**solation: even though TAs are executing concurrently, they should appear to be executed in isolation; that is, their final effect should be as if each TA was executed alone from start to end
- **D**urability: once a TA is committed, its changes applied to the database must never be lost due to subsequent failure

Enforcement of ACID Properties

- Subsystems of a DBMS that are responsible for enforcing the ACID properties:
  - *Database constraint subsystem* (and application program correctness) responsible for ***C***
  - *Concurrency control subsystem* responsible for ***I***
  - *Recovery subsystem* responsible for ***A*** and ***D***

Transaction Support in SQL

Transaction Support in SQL

- Single SQL statement always considered to be atomic
  - i.e., either the statement completes execution without error or it fails and leaves the database unchanged
- No explicit Begin_transaction statement
  - Begin_transaction implicit at first SQL statement, and at next SQL statement after previous TA terminates
- Every transaction must have an end statement
  - COMMIT - the DBMS must assure that the effects are permanent
  - ROLLBACK - the DBMS must assure that the effects are as if the TA had not yet begun
  - Some systems have an *auto-commit* feature enabled: treats each single statement as if followed by COMMIT

www.liu.se

LINKÖPING UNIVERSITY