Database Technology

Topic 8: Introduction to Transaction Processing

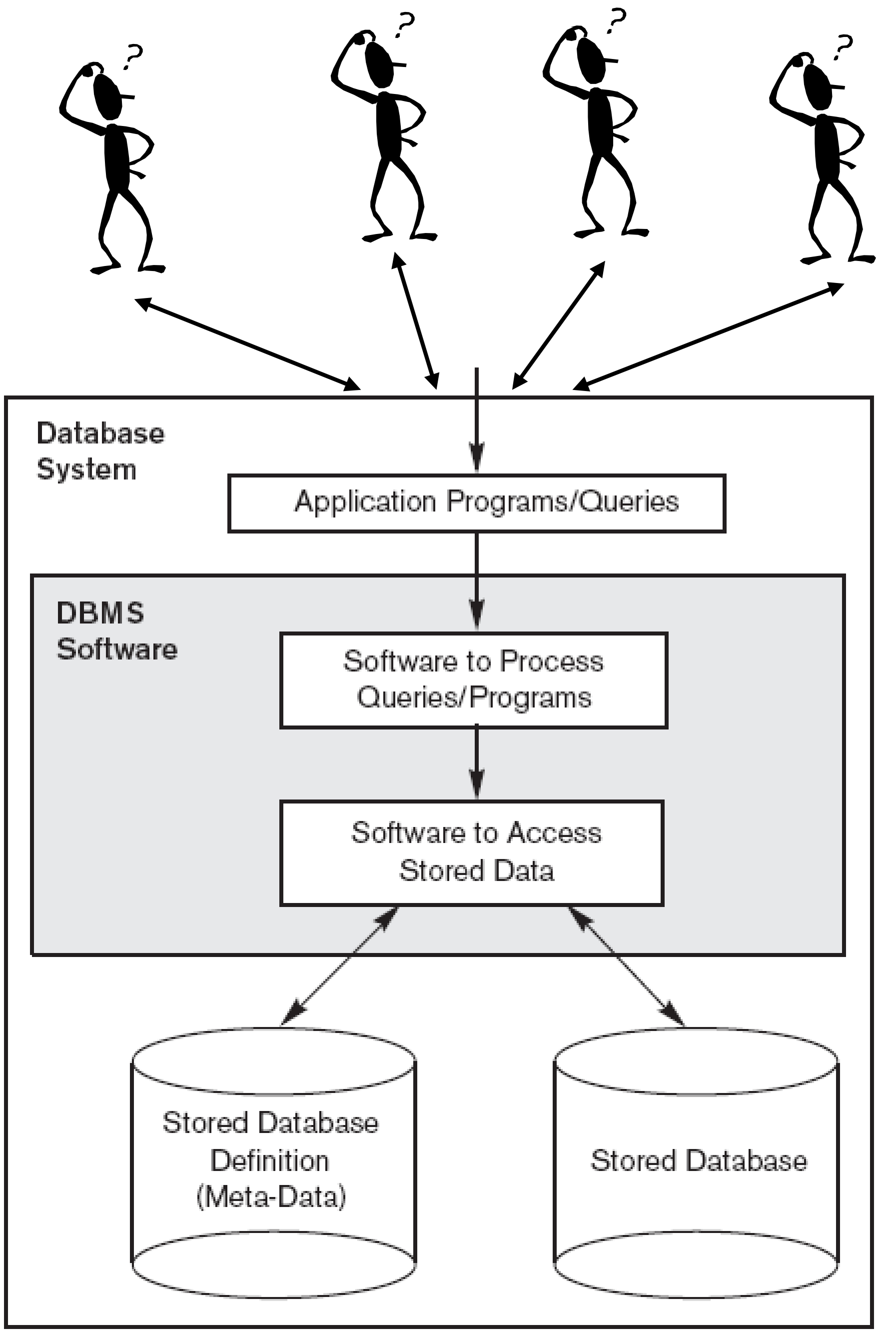
Olaf Hartig

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



Basic Terminology

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)

Some More Terminology

- Online Transaction Processing (OLTP) systems: large multi-user database systems supporting thousands of concurrent transactions (user processes) per minute
- 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

Standalone versus Embedded TAs

- Transactions may be *standalone*
  - specified in a high-level language like SQL, submitted interactively
- More typically, transactions are *embedded* within application programs
  - Application program may include specification of several transactions, separated by Begin and End transaction boundaries
  - Transaction code can be executed several times (e.g., in a loop), spawning multiple transactions

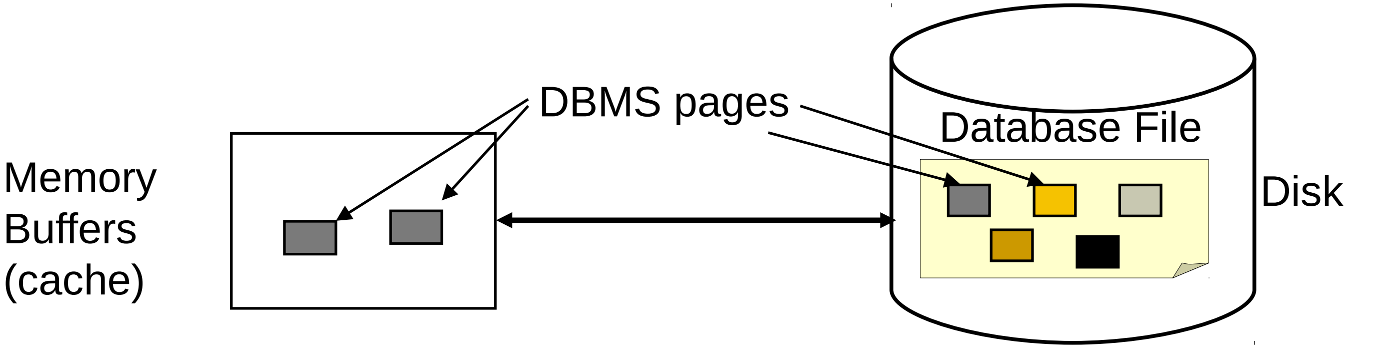
Transaction Processing Model

Simple Database Model

- Database: simply, a collection of named items
- Granularity (size) of these data items is unimportant
  - May be a field, a tuple, or a disk block, etc
  - Transaction processing concepts
    are independent of granularity

Basic Operations

- read_item(X): reads item X into a program variable (for simplicity, assume that the variable is also named X)
- write_item(X): write the value of program variable X into the database item named X
- These operations take some amount of time to execute
- Basic unit of data transfer between the disk and the computer main memory is a disk block



Steps of Read / Write Operations

- read_item(X) consists of the following steps:
  1. Find address of the disk block that contains item X
  2. Copy the disk block into a buffer in main memory (if the block is not already in main memory)
  3. Copy item X from the buffer to the program variable X
- write_item(X) consists of the following steps:
  1. Find address of the disk block that contains item X
  2. Copy the disk block into a buffer in main memory (if the block is not already in main memory)
  3. Copy item X from the program variable named X into its correct location in the buffer
  4. Store the updated block from the buffer back to disk (either immediately or at some later point in time)

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
  - Money lost if crash between 3–6 or between c–d
  - Updates lost if write to disk not performed before crash
- 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

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

Desirable Properties

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