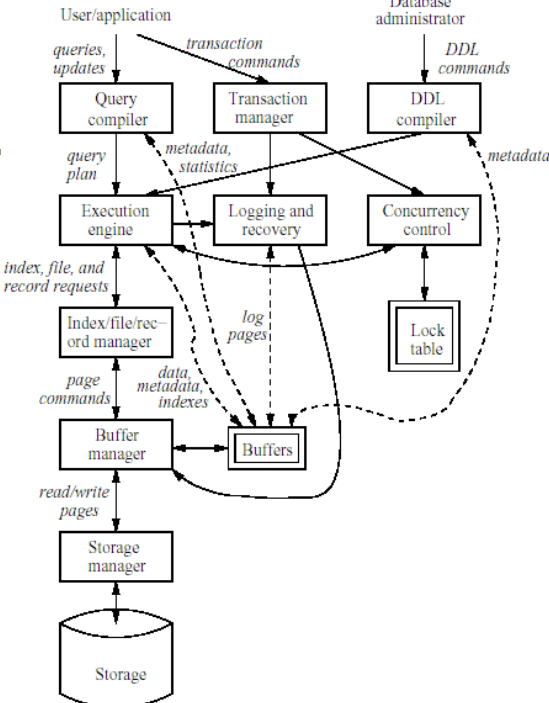


## What's New?

- Query Compiler?
  - DDL Compiler?
- Transaction Manager?
- Execution Engine?
- Logging/Recovery?
- Concurrency Control?
- Index/File/Record?
- Buffer Manager?
- Storage Manager?



The diagram illustrates the database architecture. At the top, a **User/application** sends *queries, updates* to the **Query compiler** and *transaction commands* to the **Transaction manager**. A **Database administrator** sends *DDL commands* to the **DDL compiler**. The **Query compiler** sends a *query plan* to the **Execution engine**. The **DDL compiler** sends *metadata* to the **Execution engine**. The **Transaction manager** sends *metadata, statistics* to the **Execution engine** and *log pages* to the **Logging and recovery** component. The **Execution engine** sends *index, file, and record requests* to the **Index/file/record manager**. The **Index/file/record manager** sends *page commands* to the **Buffer manager**. The **Buffer manager** interacts with **Buffers** and sends *data, metadata, indexes* to the **Storage manager**. The **Storage manager** sends *read/write pages* to the **Storage** (represented by a cylinder). The **Logging and recovery** component sends *log pages* to the **Buffers**. The **Concurrency control** component sends *lock* information to the **Lock table**. Dashed lines indicate metadata flow from the **DDL compiler** to the **Execution engine**, **Logging and recovery**, and **Concurrency control**.



# CS542: Distributed Database Concurrency Control

16 January 2009  
Chris Clifton





## Concurrency Control



- To improve performance, we want to allow (maximize) concurrent access to data
- If there are no updates, there is no problem,
- However, in the presence of updates there are potential problems:
  - Lost Update
  - Inconsistent Retrieval

4




## Examples of Incorrect Behavior



TRANSFER  
Read Acc1  
Read Acc2  
Adjust balances  
Write Acc1  
Write Acc2

WITHDRAW  
Read Acc1  
Adjust Balance  
Write Acc1

5



## Examples of Incorrect Behavior


---

Initially:  
Both accounts have \$100

	TRANSFER	
	Read Acc1	
(\$100)	Read Acc2	
(\$100)		
	Adjust balances	
	Write Acc1	
	Write Acc2	
(\$150)		
(\$50)		
		WITHDRAW
		Read Acc1
		Adjust Balance
		Write Acc1
		(\$100)
		(\$80)
		LOST UPDATE!!

Finally: Acc1 has \$150,  
Acc2 has \$50 – FREE MONEY!!!

6




## Examples of Incorrect Behavior

---

<p>TRANSFER</p> <p>Read Acc1</p> <p>Write Acc1</p> <p>Read Acc2</p> <p>Write Acc2</p>	<p>CHECK BALANCE</p> <p>Read Acc1</p> <p>Read Acc2</p>
---	--

7



## Examples of Incorrect Behavior


Initially:  
Both accounts have \$100

	TRANSFER	
(\$100)	Read Acc1 Write Acc1	
(\$150)		CHECK BALANCE
	Read Acc2 Write Acc2	Read Acc1 (\$150) Read Acc2 (\$100)
(\$100)		
(\$50)		

INCONSISTENT RETRIEVAL!!

Check Balance Finds \$250 in all Accounts: CREATING MONEY!!

8



## How to Allow CORRECT interleaving?

- It is difficult to judge each interleaving to decide whether something has gone wrong for the given application (depends too heavily on application semantics).
- SOLUTION: Transaction Model.
  - The DBMS guarantees the correct interleaving of transactions.

9



## TRANSACTIONS



- A *Transaction* is an execution of a program that accesses a shared database. The goal of concurrency control and recovery is to ensure that transactions execute *Atomically*:
  - Each txn accesses shared data without interfering with other transaction, and
  - If a txn terminates normally, then all of its effects are made permanent; otherwise it has no effect at all.

10



## Database Systems



- A *database* consists of a set of named data items.
- The *database state* is the set of values of these data items.
- The DBMS supports *operations* (e.g., read and write of data items).
- DBMS executes txns *atomically*, i.e. behaves as though it were sequential (may or may not be).

11



## Database Systems (contd.)

- The DBMS also supports *transaction operations*: Start, Commit, Abort.
- Transactions must end with a Commit or Abort. (the Start may be implicit).
- Assume that the DBMS begins in a consistent state.
- The correct atomic, execution of a transaction takes the DBMS from one consistent state to another. (May be inconsistent during the txn execution).

12



## Transaction

```

Procedure TRANSFER begin
  Start;
  input(from, to, amount);
  temp ← Read(Accounts[from]);
  if temp < amount then begin
    output("insufficient funds")
    Abort;
  end
  else begin
    Write(Accounts[from], temp-amount);
    temp ← Read(Accounts[to]);
    Write(Accounts[to], temp+amount);
    Commit;
    output("Transfer Completed");
  end;
  return;
end

```

13



## Transactions



- **Active txn** – one that has not yet committed or aborted.
- **Abort** – may be from application semantics, or system imposed.
- **Commit**: effects must persist **forever**.
  - Also important for read-only transactions!
- **Abort**: **Completely** undo any effect.
- **Messages**: txns communicate **only** via data stored in the DBMS!!

14



## ACID properties



- **Atomicity**: All or nothing
- **Consistency**: defined by the execution of the txn
- **Isolation**: txn should see a consistent state at all times
- **Durability**: permanence of committed txn actions.

15



## Recoverability



- Ensure: ALL the effects of committed txns and NONE of aborted txns
- Easy if there are no aborts.
- Aborting requires: *undoing* the effects of the txn: I.e. updates and effects on others.
- *Cascading aborts*: aborting a txn implies that all those that read the data it wrote should also be aborted.

16



## Recoverability



- Committing a txn means it can NEVER be aborted.
- Thus we shouldn't commit if we could have a cascading abort!
- A *Recoverable* execution is one which allows commits only when all txns that wrote data items read by the committing txn have committed.

17





## Recoverability



- A txn  $T_j$  reads data item  $x$  from txn  $T_i$ , if
  - $T_j$  reads  $x$  after  $T_i$  has written to it;
  - $T_i$  does not abort before  $T_j$  reads  $x$ ; and
  - Every txn (if any) that writes  $x$  between the time  $T_i$  writes it and  $T_j$  reads it, abort before  $T_j$  reads it.
- An execution is recoverable if, for every txn  $T$  that commits,  $T$ 's commit follows the commit of every txn from which  $T$  read.

18



## Avoid Cascading Aborts



- Recoverability may require cascading aborts.
- These are expensive operations – should be avoided.
- A DBMS Avoids Cascading Aborts (ACA) if it does not allow a txn to read a data item that has been modified by an uncommitted transaction.

19



## Before Images

- Undo is typically done using *before-images*, e.g.

Write<sub>1</sub>(x,1)

Write<sub>1</sub>(y,3)

Write<sub>2</sub>(y,1)

Commit<sub>1</sub>

Read<sub>2</sub>(x)

Abort<sub>2</sub>

20



## Strict Execution

- However, if we have:

Write<sub>1</sub>(x,2); Write<sub>2</sub>(x,3); Abort<sub>1</sub>; Abort<sub>2</sub>;

- To avoid the corruption of before images, we enforce Strict executions:
- A DBMS that does not read or overwrite a data item that has been modified by an uncommitted transaction is STRICT.

21



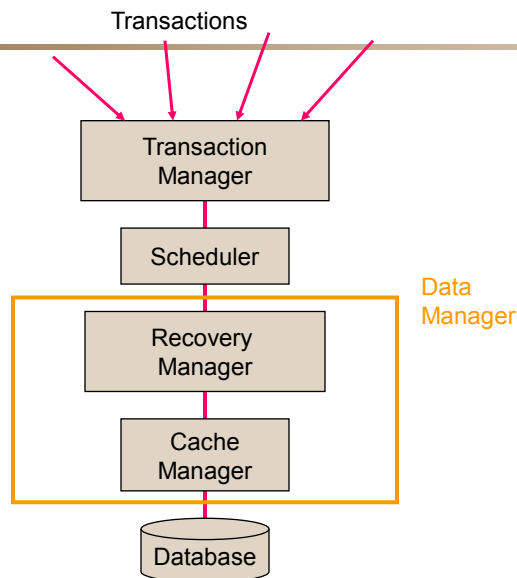
## SERIALIZABILITY

- By the definition of transactions, the sequential (SERIAL) execution of transactions is correct.
- A SERIAL execution is one with no interleaving of transactions.
- How can we interleave txns while ensuring that the result (effect) is the same as some acceptable sequential execution – such an execution is called SERIALIZABLE.

22



## DBMS Model



23



## DBMS Model



- Abstract Model.
- Centralized system (For NOW).
- **Cache Manager**: manages the cache and stable storage. Fetch, Flush.
- **Recovery Manager**: ensures atomicity and durability. Handles system failures, media failures – restores after crash.

24



## Scheduler



- Ensures concurrency control.
- By controlling the order of execution of operations submitted to it. Ensure that ordering is serializable and recoverable.
- **Execute, Reject, Delay** operations.
- Sees only operations:
  - type of operation,
  - data object being operated upon,
  - ID of the executing transaction.

25



## DBMS Model



- The Transaction Manager assigns txn Ids, passes operation onto scheduler.
- There is no guarantee of order of execution of operations at any level: must be ensured through checks at txn submission, scheduler, and the recovery manager.

26