DATABASE SYSTEMS (Transaction Management)

WEEK 12 LECTURE 1

AQSA IFTIKHAR

Topics to Cover

- What is Transaction?
- Transaction States
- Properties of Transactions
- Concurrency control and why it is needed?
- The lost update
- The dirty read problem
- The inconsistent analysis problem
- Why recovery is needed?
- Types of failures

Transaction

- An action, or series of actions, carried out by a single user or application program, that reads or updates the contents of the database.
- A transaction can be defined as an indivisible unit of work comprised of several operations, all or none of which must be performed in order to preserve data integrity.
- A transaction includes one or more database access operations—these can include insertion, deletion, modification (update), or retrieval operations.

Transaction

Using this simplified database model, the basic database access operations that a transaction can include are as follows:

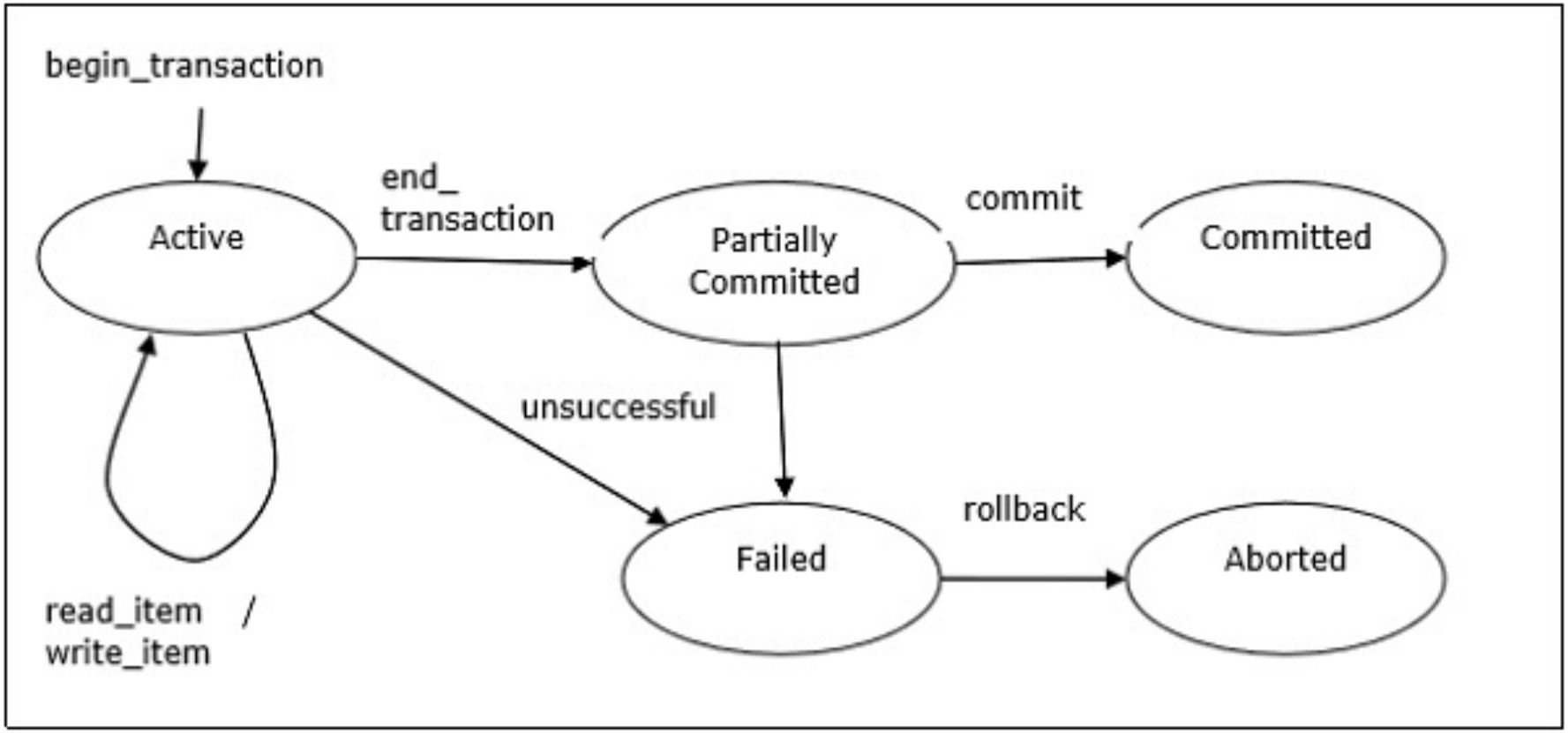
- **read_item(*X*).** Reads a database item named *X* into a program variable.

read(**staffNo** = x, salary)

- **write_item(*X*).** Writes the value of program variable *X* into the database item named *X*.

write(**staffNo** = x, salary)

Transaction States



Transaction States

- A transaction can have one of two outcomes. If it completes successfully, the transaction is said to have **committed** and the database reaches a new consistent state.
- On the other hand, if the transaction does not execute successfully, the transaction is **aborted**. If a transaction is aborted, the database must be restored to the consistent state it was in before the transaction started.
- Such a transaction is **rolled back** or **undone**. A committed transaction cannot be aborted.

Transaction States

- **PARTIALLY COMMITTED,** which occurs after the final statement has been executed.
- At this point, it may be found that the transaction has violated an integrity constraint and the transaction has to be aborted.
- Alternatively, the system may fail and any data updated by the transaction may not have been safely recorded on secondary storage.
- In such cases, the transaction would go into the **FAILED** state and would have to be aborted.
- **FAILED,** which occurs if the transaction cannot be committed or the transaction is aborted while in the ACTIVE state, perhaps due to the user aborting the transaction.

Properties of Transactions (ACID)

- **Atomicity.** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
- **Consistency preservation.** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.
- **Isolation.** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
- **Durability or permanency.** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

Concurrency Control

The process of managing simultaneous operations on the database without having them interfere with one another.

Why Concurrency Control Is Needed?

- A major objective in developing a database is to enable many users to access shared data concurrently.
- Concurrent access is relatively easy if all users are only reading data, as there is no way that they can interfere with one another.
- However, when two or more users are accessing the database simultaneously and at least one is updating data, there may be interference that can result in inconsistencies.

Why Concurrency Control Is Needed?

- The system begins executing the first transaction until it reaches an I/O operation.
- While the I/O is being performed, the CPU suspends the first transaction and executes commands from the second transaction.
- When the second transaction reaches an I/O operation, control then returns to the first transaction and its operations are resumed from the point at which it was suspended.

Why Concurrency Control Is Needed?

- The first transaction continues until it again reaches another I/O operation.
- In this way, the operations of the two transactions are **interleaved** to achieve concurrent execution. In addition, **throughput** (the amount of work that is accomplished in a given time interval) is improved as the CPU is executing other transaction instead of being in an idle state waiting for I/O operations to complete.

Problems due to Concurrency

We examine three examples of potential problems caused by concurrency:

- **the lost update problem**,
- **the uncommitted dependency problem (dirty read problem)**
- **the inconsistent analysis problem**

The Lost Update Problem

- A lost update occurs when two different transactions are trying to update the same column on the same row within a database at the same time.
- Typically, one transaction updates a particular column in a particular row, while another that began very shortly afterward did not see this update before updating the same value itself.
- The result of the first transaction is then “lost”, as it is simply overwritten by the second transaction.

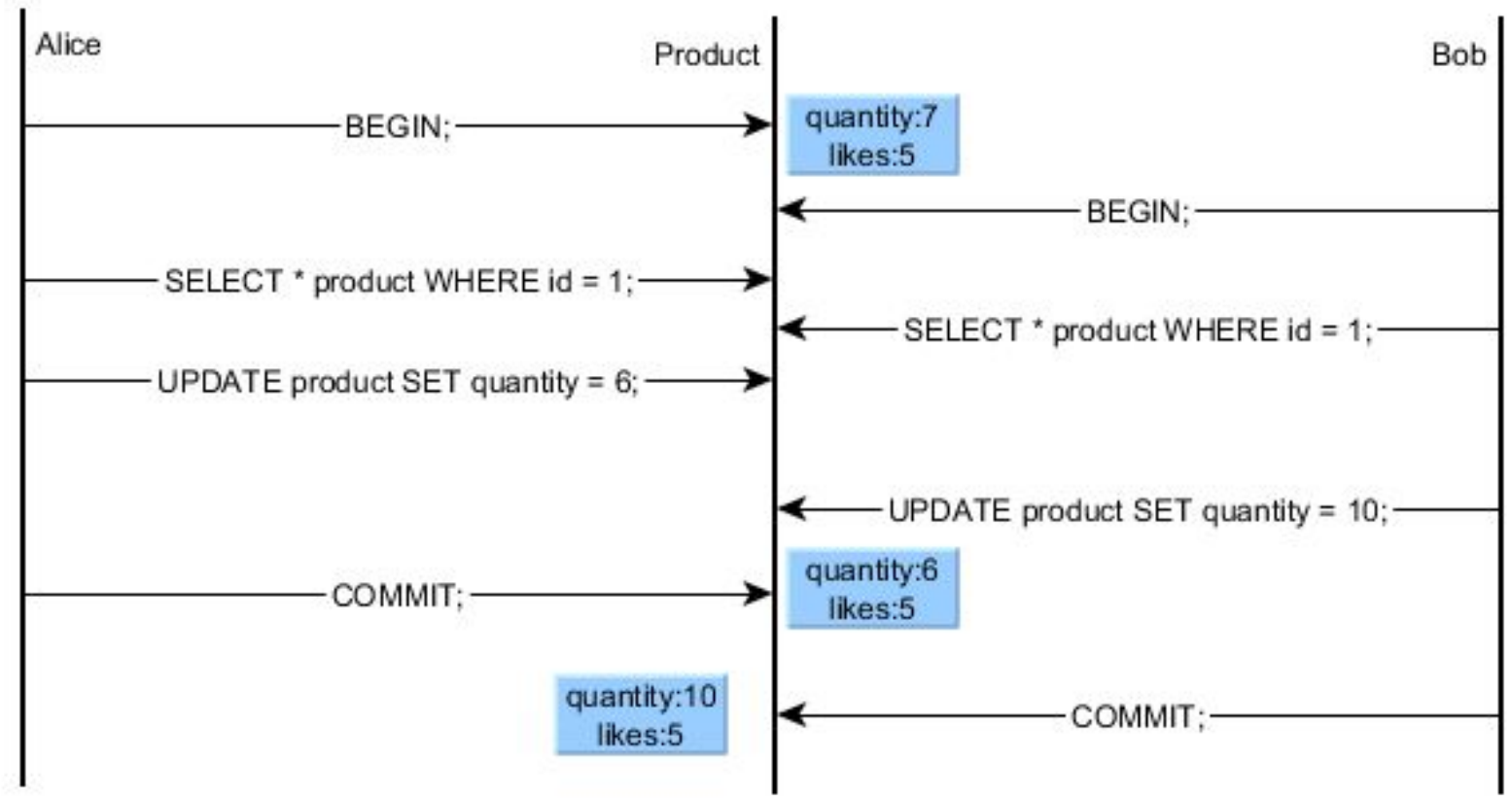
The Lost Update Problem
Example 1

For example, consider X=100

T1	T2	
R(X)		X = 100
X= X+20		X = 100 + 20 = 120
	R(X)	X = 100
	X = X*10	X = 100 * 10 = 1000
W(X)		X = 120
	W(X)	X =1000

Update by T1 is lost.

The Lost Update Problem
Example 2



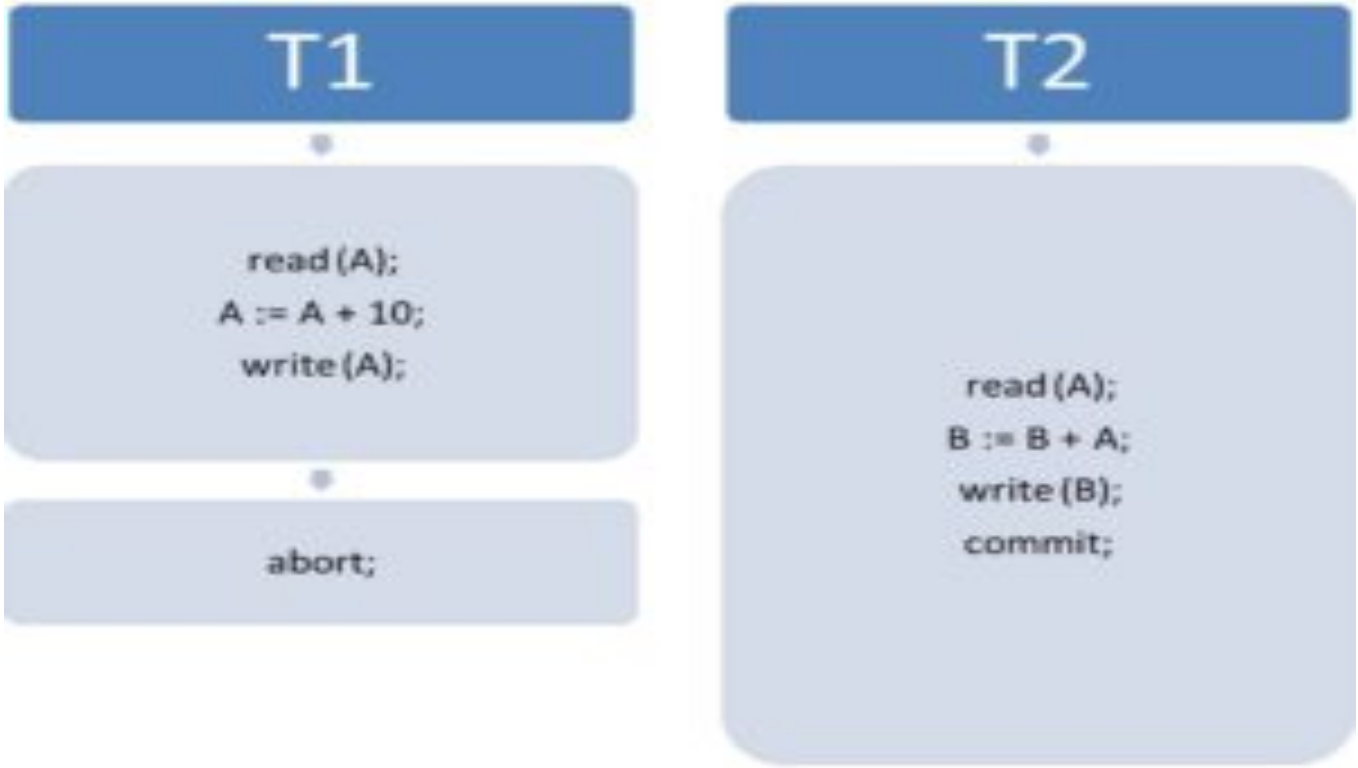
The Uncommitted Dependency (dirty read) Problem

- A **dirty read** occurs when one transaction is permitted to **read** data that is being modified by another transaction which is running concurrently but which has not yet committed itself.

The Uncommitted Dependency (dirty read) Problem Example 1

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	⋮	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write(bal_x)		190
t_8	commit		190

The Uncommitted Dependency (dirty read) Problem Example 2



The Inconsistent Analysis Problem

- This problem is caused when one of the transactions is executing an aggregate operation on several data items, and other transactions are updating one or more of those data items. This causes a inconsistent database state.
- The problem of inconsistent analysis occurs when a transaction reads several values from the database but a second transaction updates some of them during the execution of the first.

The Inconsistent Analysis Problem Example 1

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

The Inconsistent Analysis Problem Example 2

Transaction T1	Transaction T2	A = 1000, B = 1000, C = 1000
	sum = 0;	sum = 0
	avg = 0;	avg = 0
	read(C);	T2 read: C = 1000
	sum := sum + C;	sum = 1000
read(A);		T1 read: A = 1000
A := A – 50;		
write(A);		T1 write: A = 950
	read(A);	T2 read: A = 950
	sum := sum + A;	sum = 1950
	read(B);	t2 read: B = 1000
	sum := sum + B;	sum = 2950
	avg := sum/3;	avg = 983.33
	commit;	
read(B);		T2 read: B = 1000
B := B + 50;		
write(B);		T2 write: B = 1050
commit;		

Why Recovery is Needed?

- Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or that the transaction does not have any effect on the database or any other transactions.

Why Recovery is Needed?

- In the first case, the transaction is said to be **committed**, whereas in the second case, the transaction is **aborted**.
- The DBMS must not permit some operations of a transaction *T* to be applied to the database while other operations of *T* are not, because *the whole transaction* is a logical unit of database processing.
- If a transaction **fails** after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

Types of Failures

Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

- **1. A computer failure (system crash).** A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.
- **2. A transaction or system error.** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.

Types of Failures

- **3. Local errors or exception conditions detected by the transaction.** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction.
- For example, data for the transaction may not be found. An exception condition,4 such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception could be programmed in the transaction itself, and in such a case would not be considered as a transaction failure

Types of Failures

- **4. Concurrency control enforcement.** The concurrency control method may abort a transaction to resolve a state of deadlock among several transactions. Transactions aborted because of deadlocks are typically restarted automatically at a later time.
- **5. Disk failure.** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
- **6. Physical problems and catastrophes.** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

NEXT LECTURE

- Serializability
- Recoverability
- Algorithms to remove concurrency problems