



C# Programming in Depth

Prof. Dr. Bertrand Meyer

March 2007 - May 2007

Lecture 11: Transaction processing and
concurrency in ADO.NET
Lisa (Ling) Liu

Transaction processing



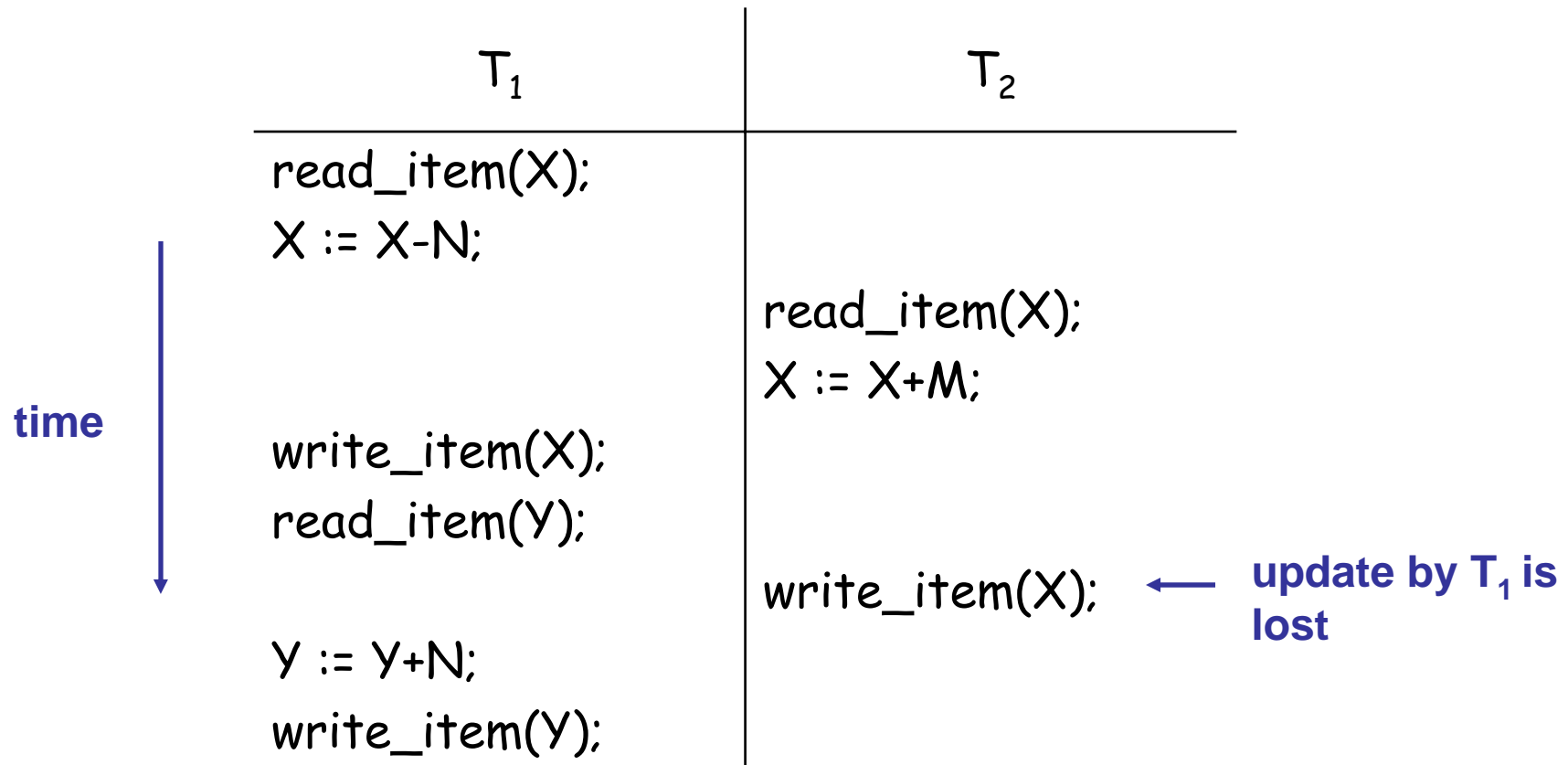
- Transaction
 - The execution of a program that accesses or changes the contents of the database is called a transaction.
 - A transaction is used to represent a logic unit of database operations.
- Two sample transactions:

```
(a) read_item (X);  
    X := X-N;  
    write_item(X);  
    read_item(Y);  
    Y := Y+N;  
    write_item(Y);
```

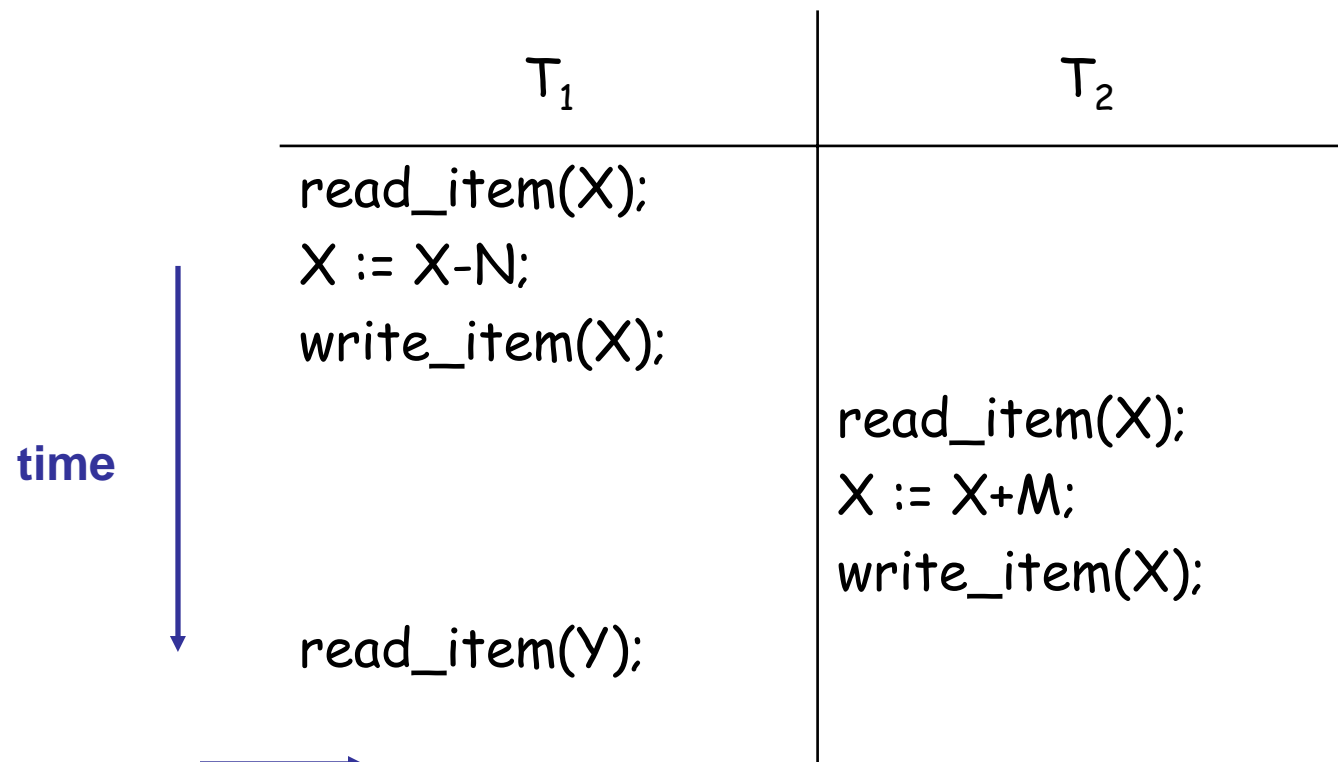
```
(b) read_item(X);  
    X := X+M;  
    write_item(X);
```

What will happen if we don't control the concurrent execution of transactions?

The lost update Problem



The temporary update (or dirty read) problem



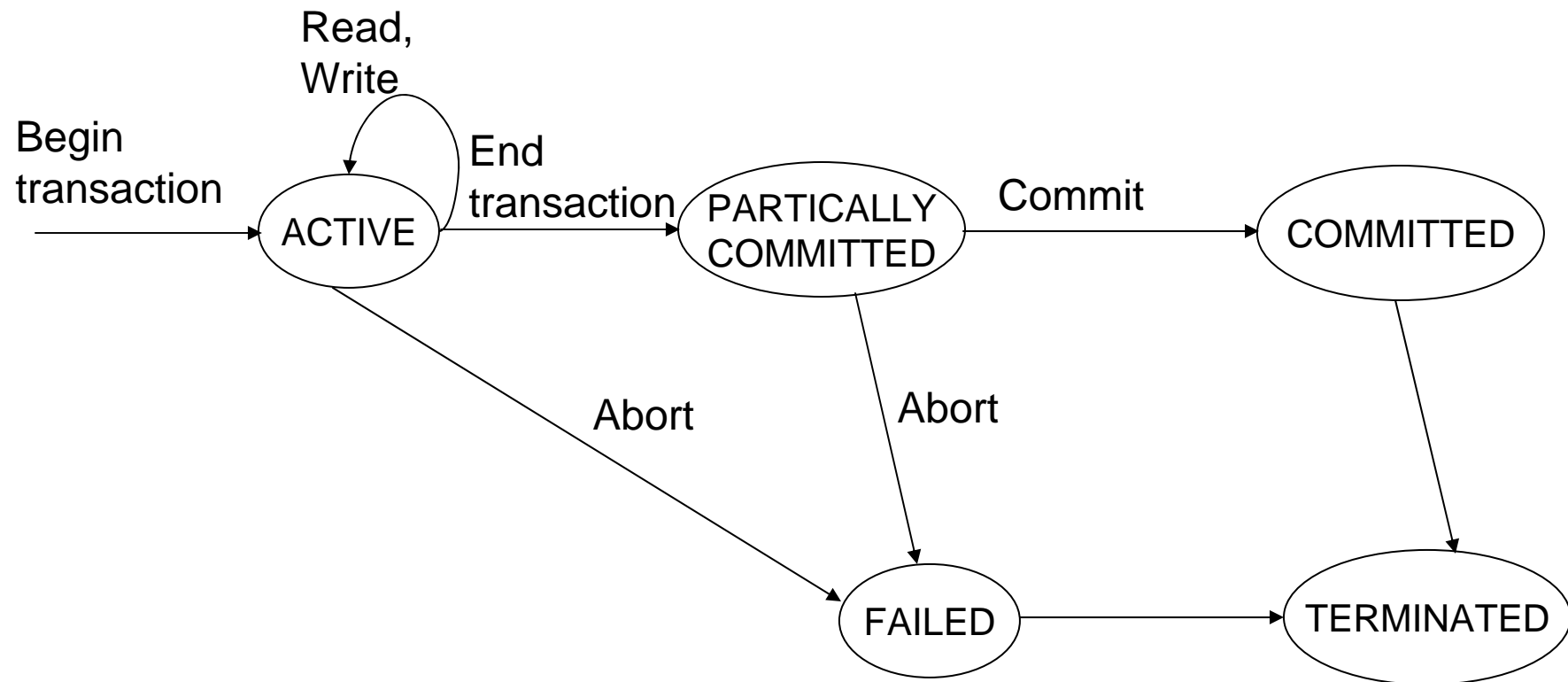
T1 fails and roll back; meanwhile, T₂ has read the “temporary” incorrect value of X

Atomic transaction



- 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
 - The transaction has no effect on the database or any other transactions

Transaction states and operations



Desirable properties of transactions



- **Atomicity**

A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

- **Consistency preservation**

A correct execution of the transaction must take the database from one consistent state to another.

- **Isolation**

A transaction should not make its updates visible to other transactions until it is committed.

- **Durability or permanency**

Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

How to achieve the desirable properties of transactions?



- Atomicity
 - The responsibility of the recovery method

- Consistency
 - The responsibility of the programmers

- Isolation
 - Achieved by the concurrency control method

- Durability
 - The responsibility of the recovery method

Perform a transaction using ADO.NET



```
SqlConnection cn = new SqlConnection(connectionString);
cn.Open();
SqlTransaction myTrans = cn.BeginTransaction();
SqlCommand myCommand = cn.CreateCommand();
myCommand.Transaction = myTrans;
try {
    myCommand.CommandText = "...";
    myCommand.ExecuteNonQuery();
    myCommand.CommandText = "...";
    myCommand.ExecuteNonQuery();
    ...;
    myTrans.Commit();
}
catch (Exception e) {
    myTrans.Rollback();
}
finally {
    cn.Close();
}
```

start a transaction

enlist commands in the current transaction

complete the transaction

Transaction schedule



- A **schedule** (or **history**) S of n transactions T_1, T_2, \dots, T_n is an ordering of the operations of the transactions subject to the constraint that, for each transaction T_i that participates in S the operations of T_i in S must appear in the same order in which they occur in T_i .

- Example:

$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1;$

Conflict operations



- Two operations in a schedule
- Belong to different transactions
- Access the same item X
- One is a `write_item(X)`

- In S_a , following transactions are conflict:
 - $r_1(X)$ and $w_2(X)$
 - $r_2(X)$ and $w_1(X)$
 - $w_1(X)$ and $w_2(X)$

Recoverable schedule



- A schedule S is said to be recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed

S_a : $r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_1; w_1(Y); c_2$; — recoverable

S_c : $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1$; — unrecoverable

Serializability of schedules



Serializability theory

- An important aspect of concurrency control, which attempts to determine which schedules are “correct” and which are not and to develop techniques that allow only correct schedules.

Serial schedule:

- A schedule S is **serial** if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called nonserial.


If we consider the transactions to be independent, we can assume that every serial schedule is considered correct.



Serializable transaction

- A schedule S of n transactions is **serializable** if it is **equivalent** to some serial schedule of the same n transactions
 - **Conflict equivalent**
Two schedules are said to be **conflict equivalent** if the order of any two conflicting operations is the same in both schedules.
 - **Conflict serializable**
A schedule S is conflict serializable if it is (conflict) equivalent to some serial schedule S' .

$S_d: r_1(X); w_1(X); r_2(X); w_2(X); c_2; r_1(Y); w_1(Y); c_1;$
 $S_a: r_1(X); w_1(X); r_1(Y); w_1(Y); c_1; r_2(X); w_2(X); c_2;$



conflict equivalent

View equivalence and view serializability



- Two schedules are said to be *view equivalent* if the following three conditions hold:
 - The same set of transactions participate in S and S' , and S and S' include the same operations of those transactions.
 - For any operation $r_i(X)$ of T_i in S , if the value of X read by the operation has been written by an operation $w_j(X)$ of T_j , the same order between $r_i(X)$ and $w_j(X)$ must hold in S' .
 - If the operation $w_k(Y)$ of T_k is the last operation to write item Y in S , then $w_k(Y)$ of T_k must also be the last operation to write item Y in S' .

View serializable



- A schedule S is said to be view serializable if it is view equivalent to a serial schedule.
 - View serializability is less restrictive than conflict serializability because it allows blind write

Sa: $r_1(X); w_2(X); w_1(X); w_3(X); c_1; c_2; c_3;$ — view serializable

- A serializable schedule is considered correct.

Concurrency control techniques



- Locking data item
- Optimistic protocols
- ...

Types of locks



- Binary locks

- A binary lock can have two states or values: **locked** and **unlocked**.
- When binary locking scheme is used, every transaction must obey the following rules:
 1. A transaction T must issue the operation `lock_item(X)` before any `read_item(X)`
 2. A transaction T must issue the operation `unlock_item(X)` after all `read_item(X)` and `write_item(X)` operations are completed in T
 3. A transaction T will not issue a `lock_item(X)` operation if it already holds the lock on item X
 4. A transaction T will not issue an `unlock_item(X)` operation unless it already holds the lock on item X

Shared and exclusive locks



- read lock
 - also called shared lock that allow mutiple transactions to read the item
- write_lock
 - also called exclusive lock that only allow a single transaction exclusively holds the lock on the item

Multi-mode locking scheme



1. A transaction T must issue the operation `read_lock(X)` or `write_lock(X)` before any `read_item(X)` operation is performed in T
2. A transaction T must issue the operation `write_lock(X)` before any `write_item(X)` operation is performed in T
3. A transaction T must issue the operation `unlock(X)` after all `read_item(X)` and `write_item(X)` operations are completed in T
4. A transaction will not issue a `read_lock(X)` if it already holds a read lock or write lock on item X
5. A transaction T will not issue a `write_lock(X)` if it already holds a read or write lock on X
6. A transaction T will not issue an `unlock(X)` operation unless it already holds a read or write lock on X

Guarantee serializability of a schedule



- Using binary locks or multiply-mode locks in transactions does not guarantee serializability of schedules
- Two-phase locking
 - A transaction is said to follow the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction
 - Two-phase locking guarantees the serializability of schedules

Concurrency control of transactions in ADO.NET



Isolation level:

- **ReadUncommitted:** No shared locks are issued and no exclusive locks are honored
- **ReadCommitted:** This is the default isolation level. Shared locks are issued
- **RepeatableRead:** Binary locks are issued
- **Serializable:** Two-phase blocking
- **Chaos:** not supported by SQL Server
- **Unspecified:** ...

```
SqlConnection cn = new SqlConnection(connectionString);  
cn.Open();  
SqlTransaction myTrans =  
    cn.BeginTransaction(IsolationLevel.RepeatableRead);  
.....
```

set concurrency control scheme



Optimistic concurrency control



- In **optimistic** concurrency control techniques, also known as validation or certification techniques
 - During transaction execution, all updates are applied to local copies of the data items that are kept for the transaction
 - A validation phase checks whether any of the transaction updates violate serializability
 - Not violated, the transaction is committed and the database is updated from the local copies
 - Otherwise, the transaction is aborted

Concurrency control for disconnected data in ADO.NET



- ADO.NET uses optimistic concurrency for disconnected data update
 - Locks are set and held only while the database is being accessed
 - When an update is attempted the original version of a changed row is compared against the existing row in the database. If the two are different, the update fails with a concurrency error
 - Prevent multiple users from attempting to update records at the same instant

Demo of concurrency control for disconnected data



Lock granularity



What is locked?

- Whole database?
- Whole table?
- Page of data?
- Individual record?

Lock granularity supported by SQL Server:

- Database
- Extent
- Table
- Page
- Row
- Key
- Application

Reference



Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, Addison Wesley

Questions

