CS430: Introduction to Database Management Systems

Transaction Management

Dr. Chetan Jaiswal,
Department of Computer Science,
Truman State University
cjaiswal@truman.edu

Transaction Management

- Introduction to transaction
- Transaction structure and properties
- Transformation execution and issues
- Concurrency control
- Transaction commit

Transaction Management

- **What is a transaction?**
  - A mechanism for applying the desired modifications to the final database. A final database stores the consistent value of data
  - Database management systems accepts user queries. They are converted to transactions for execution
  - An update query may be converted to more than one transaction. For example, your debit/credit query may be converted to (a) debit transaction and (b) credit transaction

Transaction Management

- **Example of a transaction (a small transaction)**

```
Begin_Transaction (BT);
    Get message (* from terminal*);
    Extract Account_no, Teller, Branch, $$ from message;
    Find Account (Account_no) in database;
    If not found or Account_bal < $$ or ≤ 0 then send negative message
    else
    begin
       Account_bal := Account_bal -Amount;
       Post history record on Account (Amount);
       Cash_drawer (Teller) :=  Amount;
       Branch_bal (Branch) := Branch_bal (Branch) - Amount;
       Put message ('New balance = Branch_bal);
    End Transaction (ET);
  Commit;
```

Transaction Management

- **Example of a transaction (a large transaction)**

```
SELECT          *
FROM            Account, History
WHERE           Account.Account_no = History.Account_no and
                History_date  = Last_report
GROUPED BY      Account.Account_no
ASCENDING BY    Account.Account_address;
```

Transaction Management

Database System View of a Transaction

- A database system interprets a transaction not as an application program but a logical sequence of low-level operations *read* and *write* (referred to as primitives). For example, the system will look at the debit_credit transaction as follows

read (X)	*(X is the withdrawal amount)*
read (A)	*(A is source account)*
A:= A – X	*(Debit X from the account A)*
write (A)	*(Write new account value to the database)*
read (Y)	*(Y is the credit amount)*
read (B)	*(B is destination account)*
B := B + Y	*(Add Y to the existing account A)*
write (B)	*(Write new value of B to the database)*

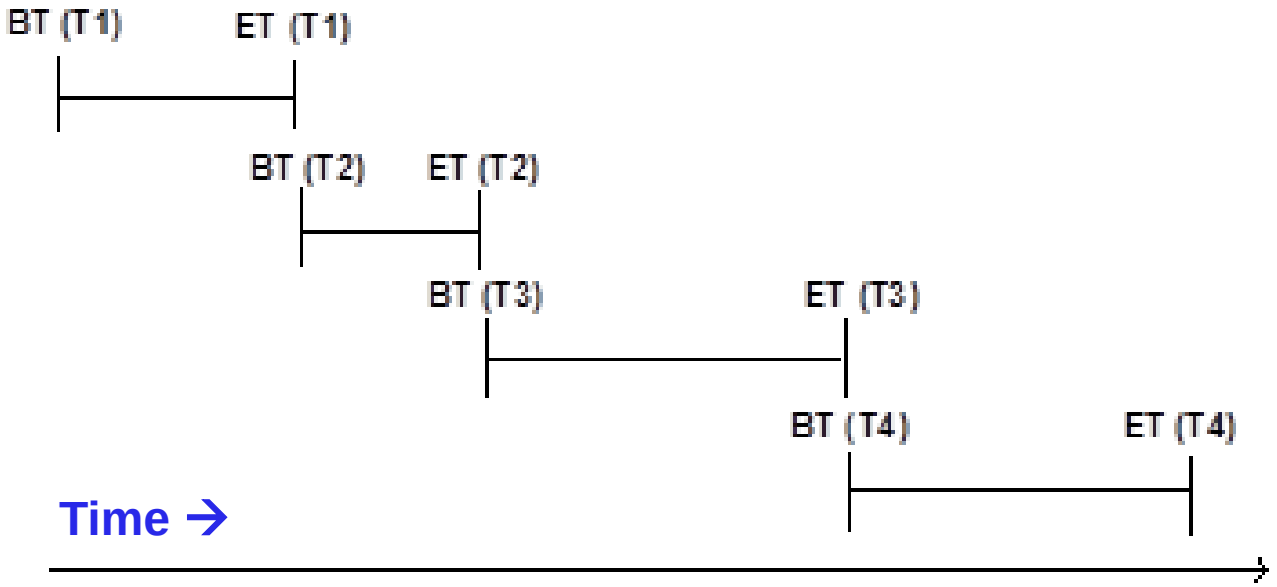
- Note: read (x) and write (x) are I/O operations.

Transaction Management

- **Transaction execution**
  - In a database system many transactions are executed. There are two ways of executing a set of transactions
    - Serially
    - Concurrently

Transaction Management

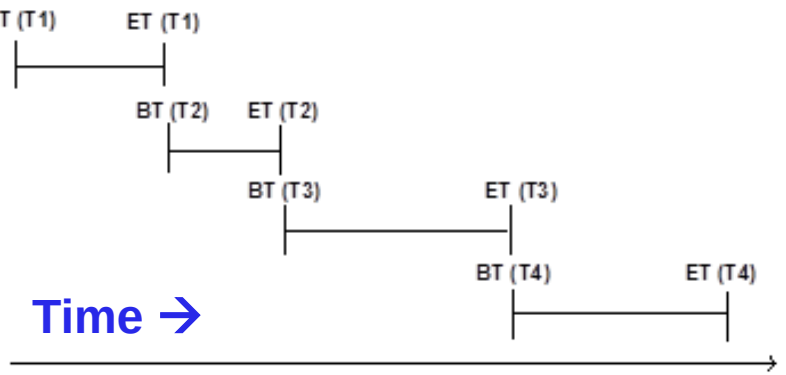
- **Serial execution**
  - **Only one transaction executing in the system, thus, no data or resource sharing. So, the execution is interference-free.**



Serial execution of transactions

Transaction Management

- **Serial execution**
  - Good points: Correct execution, i.e., if the input is correct then output will be correct
  - Bad points: Very inefficient resource and data utilization. The only way to improve resource utilization is to execute transactions concurrently or simultaneously



Serial execution of transactions

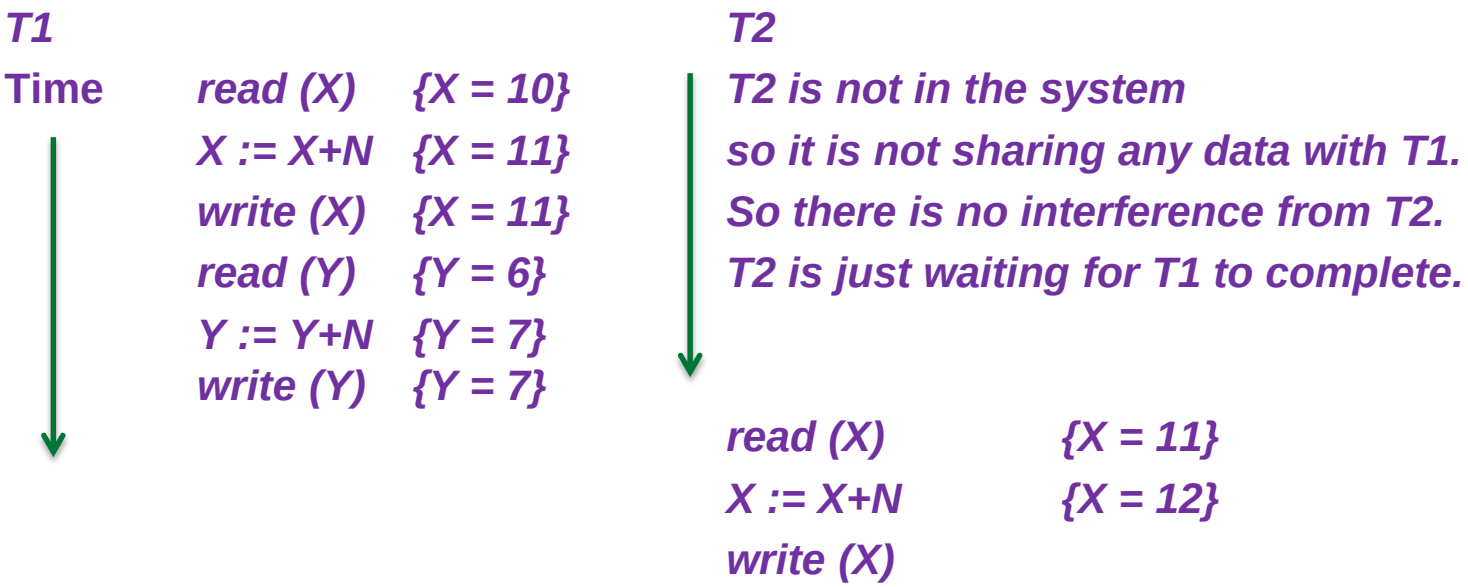
Transaction Management

- **Serial execution example**
  - Suppose data item X = 10, Y = 6, and N = 1.
  - Transactions T1 and T2 use these data items
  - Code of T1 and T2

T1	*T2*
read (X)	*read (X)*
X := X+N	*X := X+N*
write (X)	*write (X)*
read (Y)	
Y := Y+N	
write (Y)	

Transaction Management

- **Serial execution example**
  - **Suppose data item X = 10, Y = 6, and N = 1.**
  - **Transactions T1 and T2 use these data items**



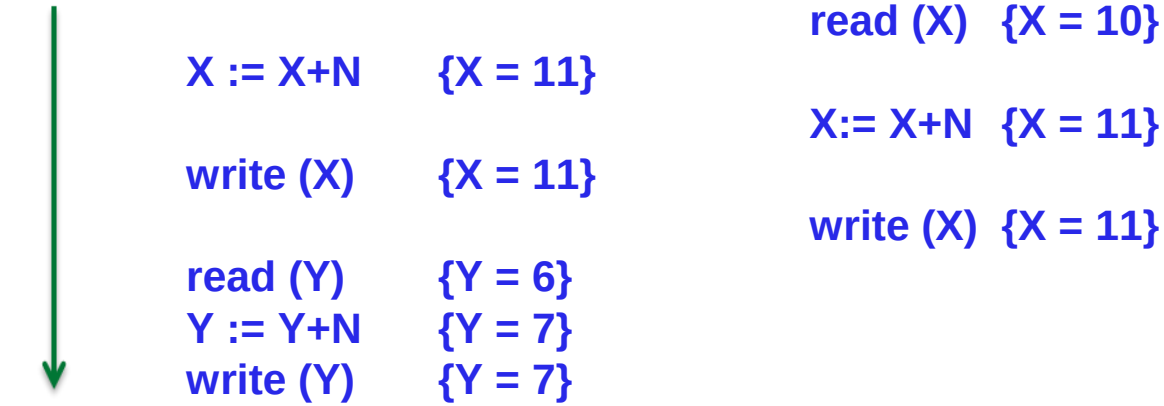
Final values of X, Y, Z, and N at the end of T1 and T2: X = 12 and Y = 7.

Transaction Management

- **Serial execution observation**
  - **Transaction T2 has to wait until T1 completes. This is waste of resources. We can improve the situation by interleave the execution of transactions' operation (read and write), i.e., run transactions *concurrently* or simultaneously.**

Transaction Management

- **Concurrent execution**
  - In this scheme the individual operations of transactions, i.e., reads and writes are interleaved in some order. We execute T1 and T2 concurrently as follows. Here reads and writes of T1 and T2 are interleaved.



Final values at the end of T1 and T2: X = 11, and Y = 7

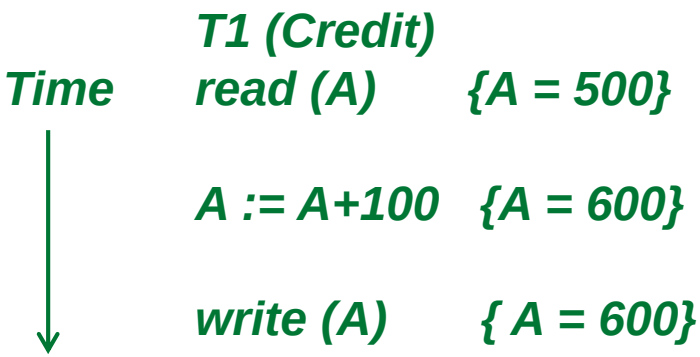
Transaction Management

- **Concurrent execution problems**
  - This improves resource utilization, but gives incorrect results. The correct value of X is 12 but in concurrent execution X = 11, which is incorrect
  - The reason for this error is incorrect sharing of X by T1 and T2. In serial execution T2 read the value of X written by T1 (i.e., 11) but in concurrent execution T2 reads the same value of X (i.e., 10) as T1 did and the update made by T1 was overwritten by T2's update. This is the reason that the final value of X is one less that what is produced by serial execution

Transaction Management

Concurrent execution problems

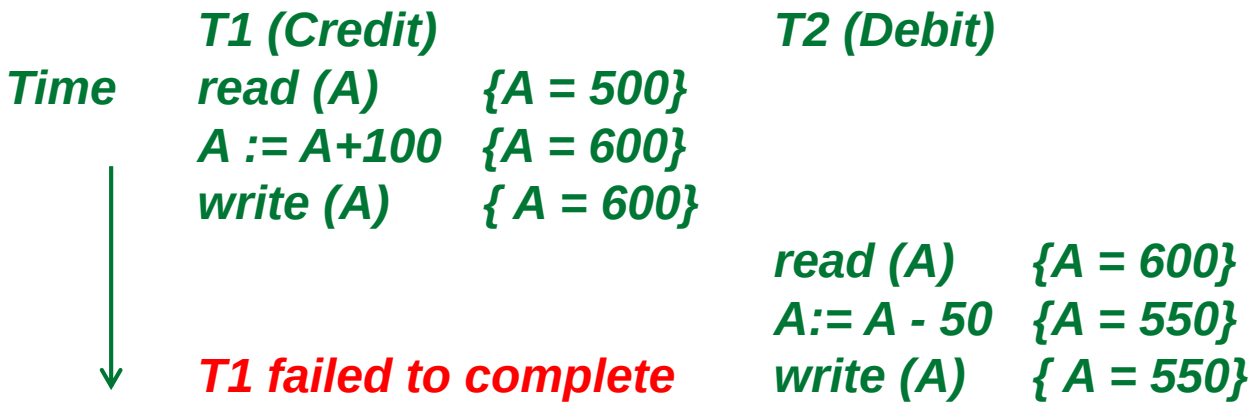
- There are more ways concurrent execution of transactions produce incorrect results. There are three problems:
  - Lost update: The update of one T1is overwritten by T2
  - Example: T1 credits $100 to account A. Initial value of A = 500
  - T2 debits $50 from account A
  - Correct result after credit and debit = 550



Final value of A = 450. The credit of T1 is missing (lost update) from the account.

Transaction Management

- **Concurrent execution problems**
  - **Dirty read: Reading of a non-existent value of A by T2. If T1 updates A which is then read by T2, then if T1 aborts T2 will have read a value of A which never existed**



T1 modified A = 600. T2 reads A = 600. But T1 failed and its effect is removed from the database, so A is restored to its old value, i.e., A = 500. A = 600 is a nonexistent value but read (reading dirty data) by T2

Transaction Management

Concurrent execution problems

- **Unrepeatable read: If T2 reads A, which is then altered by T1 and T1 commits. When T2 re-reads A it will find different value of A in its second read or A non-repeatable read occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.**
- **Phantom read: A phantom read occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.**

Transaction Management

- **Serialization of concurrent transactions**
  - In serial execution these problems (dirty read, unrepeatable read, and lost update) do not arise since serial execution does not share data items. This means we can use the results of serial execution as a measure of correctness and concurrent execution for improving resource utilization.
  - We do this using the process called *serialization* of concurrent transactions
  - Serialization of concurrent transactions: Process of managing the execution of a set of transactions in such a way that their concurrent execution produces the same end result as if they were run serially

Transaction Management

- **Properties of transactions**
  - Before we study serialization scheme we discuss the properties of transactions, which are essential to serialize concurrent execution of transactions. A transaction has four properties: Atomicity, Consistency, Isolation, and Durability
  - Atomicity: This property has two states
    - Done - a transaction must complete successfully and its effect should be installed in the database.
    - Never started - If a transaction fails during execution then all its modifications must be removed (undone) to bring back the database to the last consistent state, i.e., remove the effect (updates) of failed transaction

Transaction Management

- **Properties of transactions**
  - **Consistency: If the transaction code is correct then a transaction, at the end of its execution, must leave the database consistent**
  - **Isolation: A transaction must execute without interference from other concurrent transactions and its intermediate modifications to data must not be visible to other transactions**
  - **Durability or permanency: The effect of a completed transaction must persist in the database, i.e., its updates must be available to other transaction**

Transaction Management

- **Lock types**
  - To eliminate Lost update, Dirty read, and Unrepeatable read problems and to implement atomicity, consistency, and isolation we need two additional operations (other than read and write), which are called Lock and Unlock, which are applied on a data item. A data item can be a tuple or an entire relation (granularity)
  - Lock (X): T1 applies Lock on data item X. X is locked for T1 and it is not available to any other transaction
  - Unlock (X): T1 Unlocks X. X is available to other transactions

http://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm

Transaction Management

Lock modes

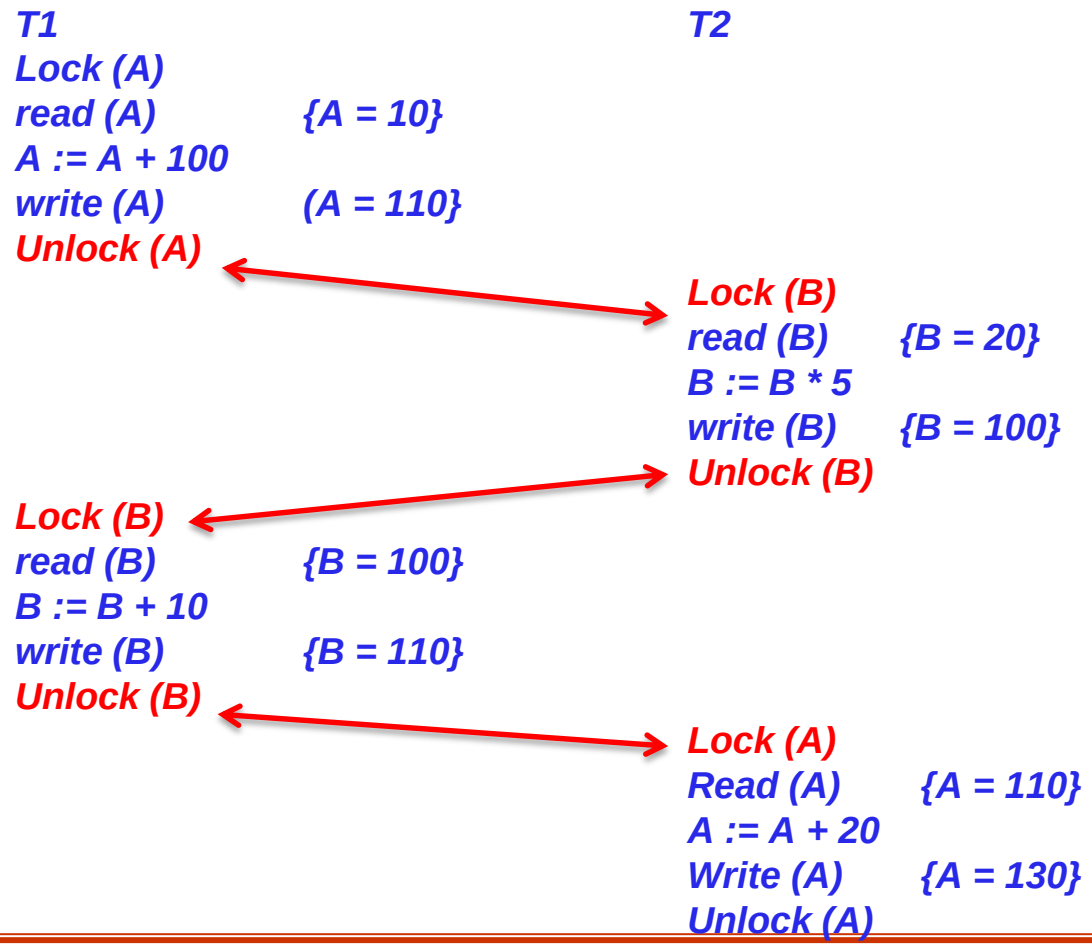
- **Shared mode: A Read operation does not change the value of a data item. Hence it can be read by two different transactions simultaneously under share mode. So a to read a data item T1 will do: *Share lock (X), then Read (X), and finally Unlock (X)***
- **Exclusive mode: A write operation changes the value of the data item. Hence two write operations from two different transactions or a write from T1 and a read from T2 are not allowed. A data item can be modified only under Exclusive mode. To modify a data item T1 will do: *Exclusive lock (X), then Write (X) and finally Unlock (X)***
- **Thus, when T1 executes a lock(X) to read the data item X then this lock allows all other concurrent transactions (T2, T3, …, Tn) to apply lock(X) and read X.**

Transaction Management

- **Well-formed transaction**
  - A transaction is well-formed if it does not lock a locked data item and it does not try to unlock an unlocked data item
  - A transaction is well-formed if it does not mix lock and unlock operations (lock(X) → unlock(X) → lock(Y) → unlock(Y)…)

Transaction Management

- **Well-formed transaction - Example(correct?)**



The final value of A = 130 and B = 110. This is not correct because a serial execution of T1 and then T2 will produce A = 130 and B = 150. This means the above method of locking and unlocking is not correct. The correct way of locking must follow two-phase scheme.

Transaction Management

- **Two-Phase Scheme (Protocol)**
  - **A transaction must not lock any data item once it has unlocked some data item. This scheme has two phases:**
    - **Growing phase: In this phase a transaction applies locks on desired data items (Ex: lock(x) → lock(y) → … →)**
    - **Shrinking phase: A transaction unlocks all data items it locked in the shrinking phase (Ex: unlock(x) → unlock(y) → … →)**

Transaction Management

- **Two-Phase Scheme (Protocol)**
  - A transaction must not lock any data item once it has unlocked some data item. This scheme has two phases:
    - Growing phase: In this phase a transaction applies locks on desired data items (Ex: lock(x) → lock(y) → … →)
    - Shrinking phase: A transaction unlocks all data items it locked in the shrinking phase (Ex: unlock(x) → unlock(y) → … →)
  - Serialization of concurrent transactions requires that all transactions must be *well-formed* and *two-phase*. The mechanism that serializes concurrent transactions is called *Concurrency Control Mechanisms* (CCMs)

https://dev.mysql.com/doc/refman/5.0/en/innodb-transaction-model.html

Transaction Management

- **Concurrency Control Mechanisms (CCMs)**
  - A CCM is a software module that enforces serialization rules of concurrent transactions. There are many types of CCMS, we will begin with CCMs that uses two-Phase protocol to serialize the execution of concurrent transactions
  - A CCM assume that the transaction code is free from programming errors and guarantee that all transactions will be well-formed and would follow two-phase locking policy

Transaction Management

- **Concurrency Control Mechanisms (CCMs)**
  - The execution of a transaction has three phases:
  - Growing (locking)
  - Execution (data modification)
  - Shrinking (releasing locks)

Transaction Management

- **Transaction schedule**
  - **Schedule: A history of concurrent execution of a set of transactions.**
  - **In a schedule read and write operations of concurrent transactions are listed in the order they are processed.**
  - **We will represent a Read operation by letter r or R and a Write operation by letter w or W. Thus, Read (x) = r(x) and Write(x) = w(x). Further, a read operation by T1 will be represented as r1(x) and a write operation will be w1(x).**
  - **Schedule example**
  - ***T1R1(A) T1R1(B) T1R1(C) T1W1(A) T1W1(B) T1W1(C) T2R2(A) T2W2(A)***

Transaction Management

- **Concurrency Control Mechanisms (CCMs)**
  - Under two phase strategy, lock and unlock operations can be applied in four different ways
  - Simultaneous Locking and Simultaneous Release
  - Incremental Locking and Simultaneous Release
  - Simultaneous Locking and Incremental Release
  - Incremental Locking and Incremental Release

Transaction Management

- **Lock table**
  - Database system maintain a lock table. When a transaction locks a data item then this information is entered in the table.

No.	T-ID	Data item	Lock mode	Link to the next data
1	1	Account 1	Shared (r)	Back - 1, Next - 2
2	1	Account 2	Exclusive(w)	Back - 1, Next - nil
3	2	Account 2	Denied	Back - nil, Next - nil
4	3	Employee	Exclusive(w)	Back - 4, Next - 5
5	3	Department	Shared (r)	Back - 4, Next - nil
6				

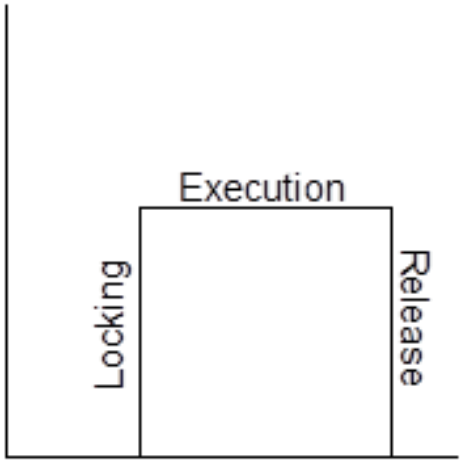
https://dev.mysql.com/doc/refman/5.0/en/internal-locking.html

Transaction Management

- **Simultaneous Locking and Simultaneous Release**
  - **All concurrent transactions go through *Growing* phase (apply locks on desired data items), *Execution* phase and finally *Shrinking* (release/unlocking) phase. Each phase, i.e., locking, execution, and unlocking is atomic. Graphically this CCM can be illustrated as follows:**

Transaction Management

Simultaneous Locking and Simultaneous Release



Transaction execution

Time	T1	T2
	Lock (A)	
	Lock (B)	
	Read (A)	
	Read (B)	
	A := A + 2	
	B := B*3	
	Write (A)	
	Write (B)	
	Unlock (A)	
	Unlock(B)	
		T2 waits for T1 to unlock A and B
		Lock (A)
		Lock (B)
		Read (A)
		Read (B)
		A := A + 2
		B := B*3
		Write (A)
		Write (B)
		Unlock (A)
		Unlock(B)

Schedule: T1R1(A)T1R1(B) T1W1(A)T1W1(B)T2R2(A)T2R2(B)T2W2(A)T2W2(B)

Transaction Management

- **Simultaneous Locking and Simultaneous Release**
  - Implementation: CPU must lock all data items referenced by the transaction before execution can begin. Locking is an atomic operation, that is, if a data item could not be locked then no data item is locked.
  - Example: T1 needs A, B, and C
  - Lock(A) – granted. Lock (B) - granted. Lock(c) – failed. T1 dis not lock any data item
  - Advantage: Simple implementation. It is good for transactions that use nearly all data items it locks. It could be good for batch transactions because they tend to access and use almost all data items they lock

Transaction Management

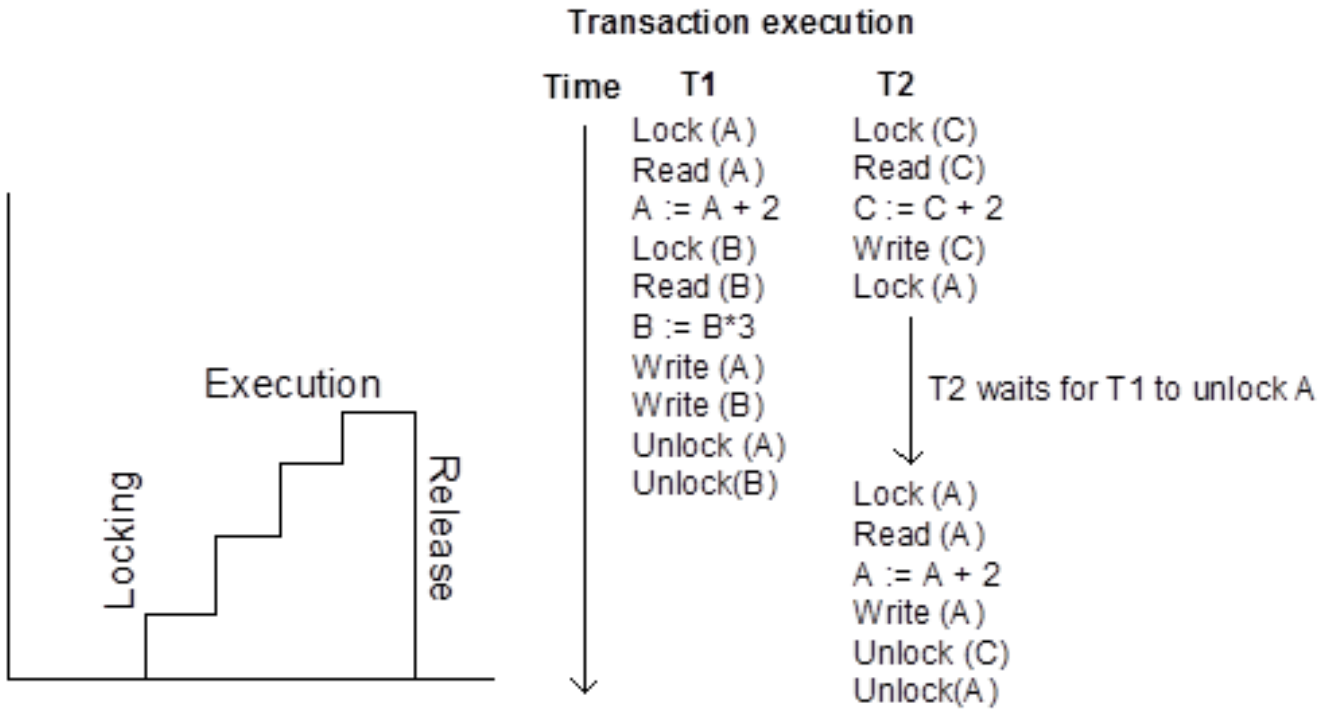
- **Simultaneous Locking and Simultaneous Release**
  - **Disadvantage**
  - **Redundant locking**
    - **Example: T1 has code like - *if A = B then lock C else lock D.* T1 refers to data items A, B, C, and D, but uses only A, B, and C or A, B, and D. So one of the locks C or D is redundant.**
  - **Not possible for interactive transactions since the next data requirement may depend on user response**

Transaction Management

- **Incremental Locking and Simultaneous Release**
  - This algorithm is widely known as *General Waiting*.
  - Locking: lock if the data item is free and enter execution phase. If data item is locked by another transaction then wait. Get next item if first data is processed.
  - It is a need-based execution. As a result, the locking phase and the execution phase are mixed together.
  - At the end of execution phase transaction commits.
  - Unlocking: During commit all locked data items are unlocked simultaneously (atomically)

Incremental Locking and Simultaneous Release

- Graphically

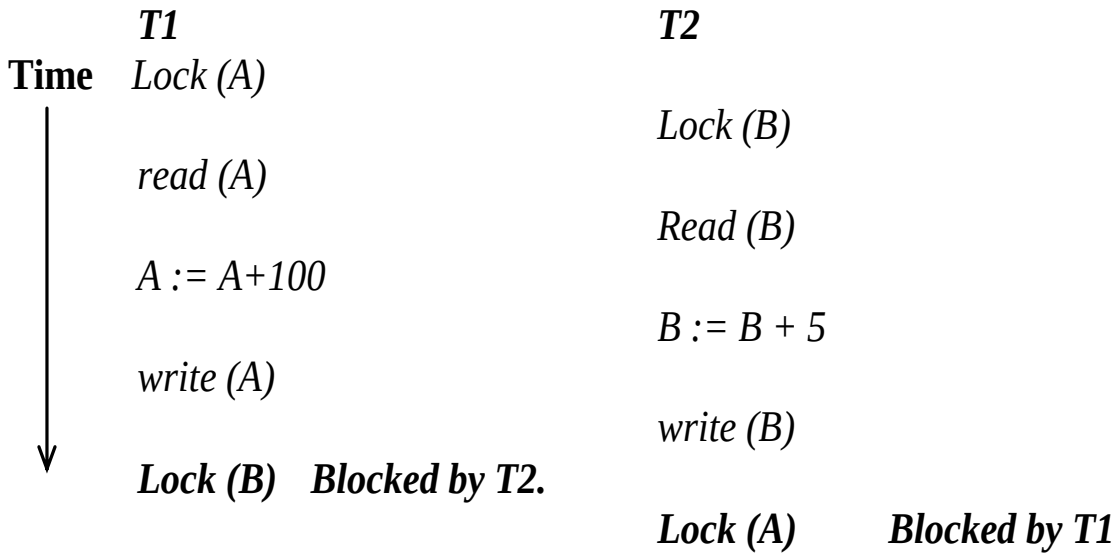


T1R1(A) T2R2(C) T2W2(C) T1R1(B) T1W1(A) T1W1(B) T2R2(A) T2W2(A)

Transaction Management

Incremental Locking and Simultaneous Release

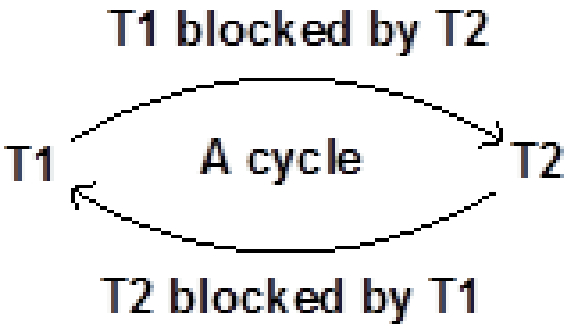
- **Advantages: No redundant locks. Transaction may wait less for a data item**
- **Disadvantages: Deadlock may happen. Some entities may remain locked even after they have been modified. For example, item A remains locked even its modification is over**



Transaction Management

- **Incremental Locking and Simultaneous Release**
  - Deadlock or a Cycle
  - Neither T1 nor T2 can proceed further. This means no transaction can complete, thus, this will not produce a serializable schedule. Thus the following schedule is not serializable

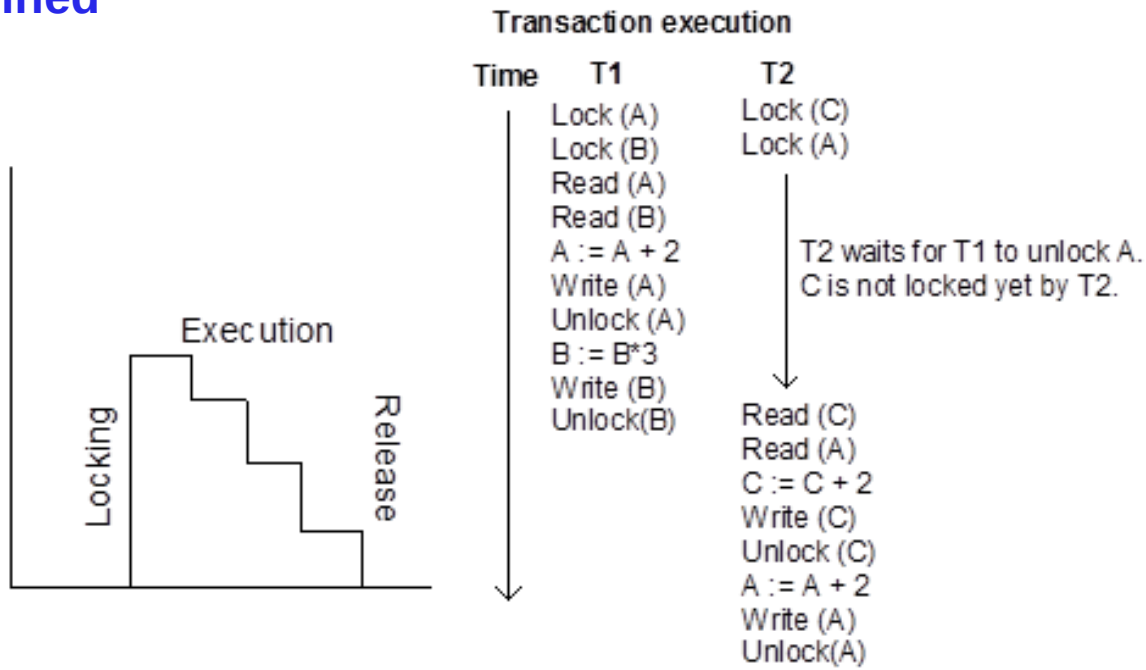
    T1R(A) T1W(A) T2R(B) T2W(B) T1R(B) T2R(A)



Transaction Management

Simultaneous Locking and Incremental Release

- Locks are applied simultaneously. As soon as modification of an item is over, it is unlocked. Execution and release phase are combined



T1R1(A) T1R1(B) T1W1(A) T1W1(B) T2R2(C) T2R2(A) T2W2(C) T2W2(A)

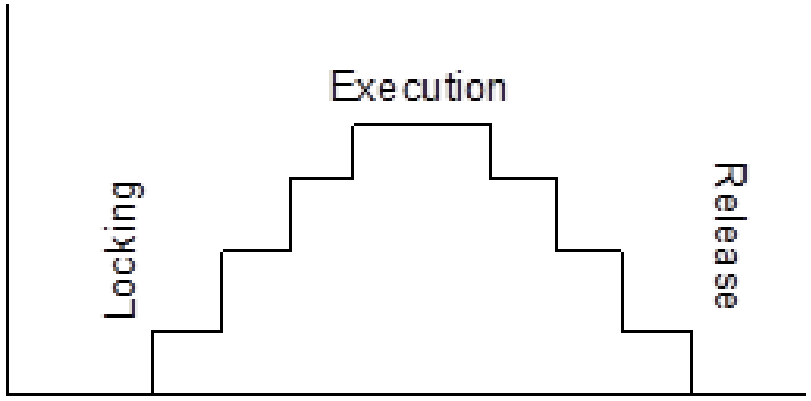
Transaction Management

- **Simultaneous Locking and Incremental Release**
  - **Advantages: Reduces transaction waiting time by incremental release. No deadlock**
  - **Disadvantages: It suffers with cascade roll-back (the roll-back of one transaction cascades to roll-backs of other dependent transactions) and redundant locking**

Transaction Management

Incremental Locking and Incremental Release

- It combines locking and execution phases and execution and release phases. Locks are acquired incrementally and released incrementally. Graphically:



Transaction execution

Time	T1	T2
↓	Lock (A)	Lock (C)
	Read (A)	Read (C)
	A := A + 2	C := C + 2
	Write (A)	Write (C)
	Lock (B)	Lock (A)
	Read (B)	↓ T2 waits for T1 to unlock A
	Unlock (A)	Read (A)
	B := B*3	Unlock (C)
	Write (B)	A := A + 2
	Unlock (B)	Write (A)
		Unlock (A)

Transaction Management

- **Incremental Locking and Incremental Release**
  - **Advantage: Reduced transaction waiting time so higher number of concurrent transactions in the system.**
  - **Disadvantages: Allows deadlock and cascade roll-back to happen**

Transaction Management

- **Conclusions**
  - **Incremental locking and simultaneous release (General waiting) is the most commonly used CCM. It allows deadlock to happen, therefore, deadlock detection and resolution is required. There are many ways for detecting deadlock and many ways for selecting a transaction to be rolled-back to resolve a deadlock. However, numerous performance studies (experimental and analytical) show that deadlocks are not frequent, so deadlock detection and resolution are not expensive**

Transaction Management

- **Timestamp**
  - We have studied several CCMs based on two-phase locking policy. Locking dynamically defines the execution order of transactions. This means that during execution of concurrent transactions, the order of execution is decided by locking
  - The execution can be defined using timestamps. This approach is known as Timestamping

Transaction Management

- **Transaction Timestamping**
  - **Timestamp: An increasing integer number**
  - **Transaction timestamp (ts): Associated with each transaction. The associated timestamp determines a transaction's age, i.e., when the scheduler scheduled it for execution. Note that a user may have issues it long before the scheduler received it. Thus, the time the user issued it does not count**
  - **A transaction's timestamp is unique, i.e., no two transactions' timestamp can be the same**

Transaction Management

- **Data item Timestamping**
  - **Data item timestamp: Timestamp associated with each data item. When a transaction modifies a data item it stamps the data item with its timestamp, indicating that the data item was modified by the transaction**

Transaction Management

- **Timestamping use**
  - **Example and convention**
    - T1: 1 is the value of timestamp of transaction T1
    - T2: 2 is the timestamp of transaction T2
    - E1: Entity E1 was last modified by T1
    - T1 is older than T2, i.e., the scheduler scheduled T1 for execution before T2
  - **We will study two CCMs based on timestamp**

Transaction Management

- **Simple Timestamp scheme**
  - **Processing a data item Dx by a transaction Ty**

If x (data item D's ts) < y (transaction ts) then
begin
access and modify the data item D;
overwrite x with y (transaction's ts), i.e., x := y
end
else roll-back Ty;

Example (Timestamps of all data items = 0)

Transaction	Needs data item	
T1	A	B C
T2	B	C D
T3	A	B F

T1 begins ts(A) < 1, T1 processes A, ts(A) = 1
T2 begins ts(B) < 2, T2 processes B, ts(B) = 2
T3 begins ts(A) < 3, T3 processes A, ts(A) = 3
T1 Asks for ts(B) > ts(T1), not OK T1 must roll-back.

Transaction Management

- **Simple Timestamp scheme**
  - **In the example, the younger transaction (T2) has already modified B, which cannot happen in a serial execution. This operation will produce a non-serializable schedule, therefore, T1 must be rolled-back. But rolling-back T1 will undo A and make A's ts = 0. T3 accessed A after T1, so T3 must also be rolled-back. T1 and T3 will then start again with higher timestamps. Remember, the new timestamps for T1 and T3 must be larger than the most recent timestamp**

Transaction Management

- **Simple Timestamp scheme**
  - **Problem: This CCM is unable to differentiate between a read and a read (sharable operations). Result: If T1 reads data item A after T2 did, T1 must be rolled-back since the timestamp of A will be 2 (updated by T2's read). This roll-back is unnecessary**
  - **Solution: Each entity is associated with two timestamps: one read timestamp (rt) and one write timestamp (wt)**
  - **A read operation is managed by the read timestamp and a write by the write timestamp**

Transaction Management

- **Basic Timestamp scheme**
  - **Read operation:**

Get wt (write timestamp of the entity)
If wt < ts then
begin
overwrite rt by the larger of the ts and rt;
read data item
end
else roll-back the transaction

Transaction Management

- **Basic Timestamp scheme**
  - **Write operation:**

Get rt of the data item

If rt < ts then

begin

overwrite wt by ts;

modify data item

end

else roll-back the transaction;

Transaction Management

- One more locking scheme is very popular

Multiversion Concurrency Control

http://research.microsoft.com/en-us/people/philbe/chapter5.pdf