ADVANCED DATABASE SYSTEM

Chapter:4 – Concurrency Controlling Techniques

Compiled by: Dureso.M

Outlines to be Covered

- In this chapter we are going to cover the following basic concept of concurrency control techniques

✓The basic concept of concurrency controlling

✓We will try to deal with four different concurrency control mechanisms.

Pessimistic approach

1. Locking based methods
2. Time stamping based methods
3. Optimistic methods(Reading Assignment)
4. Multiversion (Reading Assignment)

INTRODUCTION TO CONCURRENCY CONTROL

- **Concurrency Control** is the process of managing simultaneous operations on the database without having them interfere with one another. Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.
- In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control techniques to ensure **atomicity, isolation,** and **serializability** of concurrent transactions.

CONT…

- **Three basic concurrency control techniques:**

1. **Locking methods**
2. **Time stamping methods**
3. **Optimistic methods (Reading Assignment)**

- **Locking** and **Time stamping** are pessimistic approaches since they delay transactions.
- Both Locking and Time stamping are conservative approaches: delay transactions in case they conflict with other transactions.
- The **optimistic approach** allows us to proceed and check conflicts at the end.
- Optimistic methods assume conflict is rare and only check for conflicts at commit.

Locking Based methods

- The **locking method** is a mechanism for preventing simultaneous access on a shared resource for a critical operation
- **A LOCK** is a mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. Locks are one way of enforcing concurrency control policies.
- Transaction uses locks to deny access to other transactions and so prevent incorrect updates.
- Lock prevents another transaction from modifying item or even reading it, in the case of a write lock/ exclusive lock.

CONT…

- Lock (X): If a transaction T1 applies Lock on data item X, then X is locked and it is not available to any other transaction.
- Unlock (X): T1 Unlocks X. X is available to other transactions.

Types of a Lock

1. **Shared lock/ Read lock**
2. **Exclusive lock/ write lock**

1. Shared lock/Read Lock: A Read operation does not change the value of a data item. Hence a data item can be read by two different transactions simultaneously under share lock mode. So only to read a data item T1will do: Share lock (X), then Read (X), and finally Unlock (X)

CONT…

2. **Exclusive lock/write lock:** 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 T2are not allowed. A data item can be modified only under Exclusive lock. To modify a data item T1will do: **Exclusive lock** (X), then Write (X) and finally **Unlock** (X).

❖ When these locks are applied, then a transaction must behave in a special way. This special behavior of a transaction is referred to as **well-formed**.

❖ 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.

CONT…

Locking - Basic Rules

- ✓If transaction has a **shared lock** on an item, it can read but not update the item.
- ✓If a transaction has an **exclusive lock** on an item, it can both read and update the item.
- ✓Reads operation cannot form a conflict, so that more than one transaction can hold shared locks simultaneously on same item.
- ✓**Exclusive lock** gives transaction **exclusive** right to access that item.
- ✓Some systems allow transaction to upgrade a shared lock to an exclusive lock, or vice-versa

CONT…

Examples: T1 and T2 are two transactions. They are executed under locking as follows. T1 locks '**A**' in **exclusive** mode. When T2 wants to **lock** '**A**' , it finds it is locked by T1 so T2 waits for **Unlock** on '**A**' by T1. When '**A**' is released then T2 **locks** '**A**' and begins execution.

✓Suppose a lock on a data item is applied, the data item is processed and it is unlocked immediately after **reading/writing** is completed as follows.

→Initial values of **A = 10** and **B = 20.**

CONT...

Serial Execution of T1 and then T2	
T_1	T_2
Lock (A)	
read (A) {A = 10}	
A := A + 100	
write (A) (A = 110}	
Unlock (A)	
Lock (B)	
read (B) {B = 20}	
B := B + 10	
write (B) {B =30}	
Unlock (B)	
	Lock (B)
	read (B) {B = 30}
	B := B * 5
	write (B) {B = 150}
	Unlock (B)
	Lock (A)
	Read (A) {A = 110}
	A := A + 20
	Write (A) {A = 130}
	Unlock (A)
Final Result: A=130 B=150	

Concurrent Execution of T1 and T2	
T_1	T_2
Lock (A)	
read (A) {A = 10}	
A := A + 100	
write (A) (A = 110}	
Unlock (A)	
	Lock (B)
	read (B) {B = 20}
	B := B * 5
	write (B) {B = 100}
	Unlock (B)
Lock (B)	
read (B) {B = 100}	
B := B + 10	
write (B) {B = 110}	
Unlock (B)	
	Lock (A)
	Read (A) {A = 110}
	A := A + 20
	Write (A) {A = 130}
	Unlock (A)
Final Result: A=130 B=110	

CONT…

- ✓The final result of the two transactions using the two types of transaction execution (**serial** and **concurrent**) is not the same. This indicates that the above method of **locking** and **unlocking** is not correct.
- ✓This is because although such kind of locking and unlocking data items increases the concurrency of execution it violates the **isolation** and **atomicity** properties of transactions.
- ✓Immediate unlocking is not **trustworthy**. Thus, to preserve **consistency** we have to use another approach to locking, **the following** scheme helps us to overcome that issue.

CONT…

❖Generally there are **three types** of lock protocols available for lock based concurrency control methods:

1. Simple locking Protocol
2. Two phase locking protocol
3. Graph based protocol

1. Simplistic Lock Protocol

✓**Simplistic lock-based** protocols allow transactions to obtain a **lock** on every object before a '**write**' operation is performed. Transactions may **unlock** the data item after completing the '**write**' operation.

✓Lock item before access and unlock item when access is completed.

CONT…

2. Pre - claiming Lock Protocol

- ✓**Pre-claiming protocols** evaluate their operations and create a list of data items on which they need **locks**. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand.
- ✓If all the **locks are granted**, the transaction executes and **releases** all the **locks** when all its operations are over.
- ✓If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

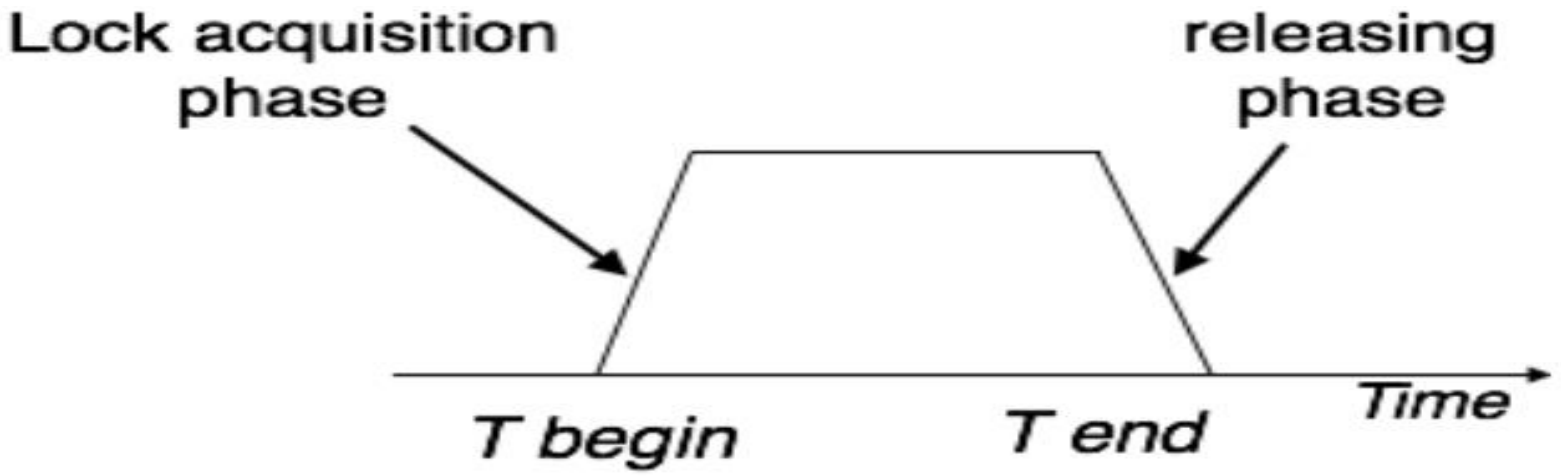
CONT…

3. Two - Phase Locking – 2PL- protocol

- ✓This locking protocol divides the execution phase of a transaction into **three parts**.
- ✓In the **first part**, when the transaction starts executing, it seeks permission for the locks it requires.
- ✓The **second part** is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the **third part** starts. In this part, the transaction cannot demand any new locks; it only **releases** the acquired locks.

CONT…

Two - Phase Locking – 2PL



[Image: Two Phase Locking]

CONT…

- ✓**Two-phase locking** has two phases, one is **growing phase**, where all the locks are being acquired by the transaction; and the second phase is **shrinking phase**, where the locks held by the transaction are being released.
- ✓To claim an **exclusive (write) lock**, a transaction must first acquire a **shared (read) lock** and then upgrade it to an **exclusive lock**.

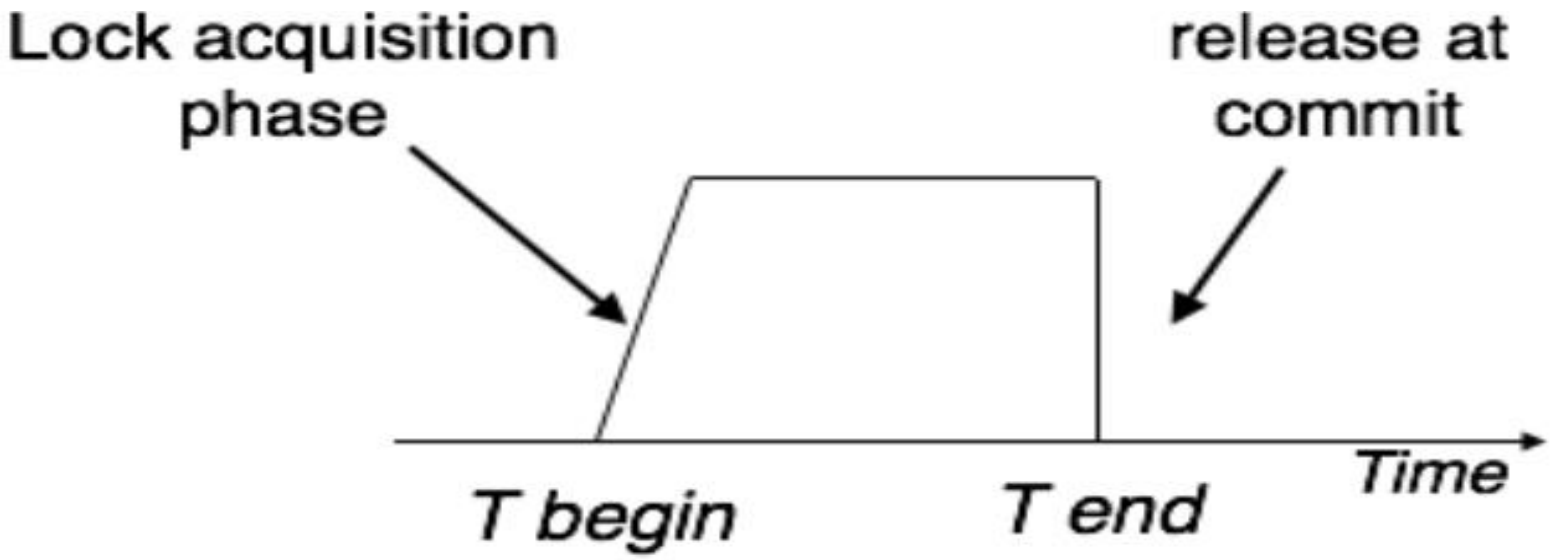
CONT…

4. Strict Two - Phase Locking

✓The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

CONT…

Strict Two - Phase Locking



[Image: Strict Two Phase Locking]

CONT…

✓Problem of locking methods

1. Deadlock

- Deadlock that may result when two (or more) transactions are each waiting for locks held by the other to be released.
- In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

CONT…

- ✓**For example,** assume a set of transactions {**T0, T1, T2, ...,Tn**}. **T0** needs a resource **X** to complete its task. Resource **X** is held by **T1**, and **T1** is waiting for a resource **Y**, which is held by **T2**. **T2** is waiting for resource **Z**, which is held by **T0**.
- ✓Thus, all the processes wait for each other to release resources. In this situation, **none** of the processes can **finish** their task. This situation is **known as a deadlock.**
- ✓The transactions involved in the deadlock are either **rolled back** or **restarted**.

CONT…

❖**Two general techniques for handling deadlock:**

- ✓ Deadlock prevention.
- ✓ Deadlock detection and recovery

✓**Deadlock Prevention** To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute.

✓The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

✓There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

CONT…

1. Wait - Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur:

- ✓ If **TS(Ti) < TS(Tj)** — that is **Ti,** which is requesting a conflicting lock, is older than **Tj** — then **Ti** is allowed to wait until the data-item is available.
- ✓ If **TS(Ti) > TS(Tj)** — that is **Ti** is younger than **Tj** — then **Ti** dies. **Ti** is restarted later with a random delay but with the same timestamp. This scheme allows the older transaction to wait but kills the younger one.

CONT…

2. Wound - Wait Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by another transaction, one of the two possibilities may occur:

- ✓ If **TS(Ti) < TS(Tj),** then **Ti** forces **Tj** to be rolled back — that is **Ti** wounds **Tj**. **Tj** is restarted later with a random delay but with the same timestamp.
- ✓ If **TS(Ti) > TS(Tj),** then **Ti** is forced to wait until the resource is available. This scheme allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

CONT…

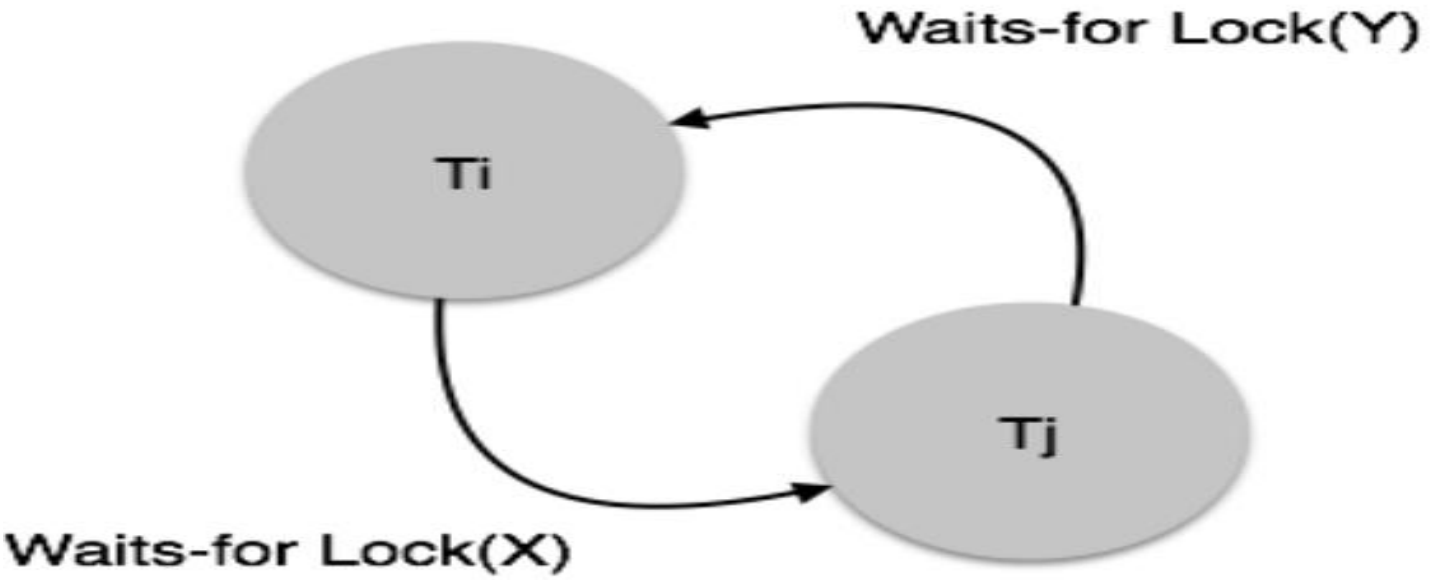
- ✓**Deadlock Avoidance:** Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance.
- ✓Methods like "**wait-for graph**" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource.
- ✓In a **bulky system**, **deadlock prevention** techniques may work well.

CONT…

- ✓**Wait - for Graph:** This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created.
- ✓When a transaction **Ti** requests for a lock on an item, let say **X**, which is held by some other transaction **Tj,** a directed edge is created from **Ti** to **Tj**. If **Tj** releases item **X,** the edge between them is dropped and **Ti** locks the data item.
- ✓The system maintains this **wait-for graph** for every transaction waiting for some data items held by others. The system keeps checking if there's any **cycle** in the graph.

CONT…

Wait For Graph



[*Image: Wait-for Graph*]

CONT…

Here, we can use any of the two following approaches:

- ✓First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- ✓The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the victim and the process is known as victim selection.

CONT…

❖Time Stamp Based Methods

- ✓The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.
- ✓Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.
- ✓Every transaction has a timestamp associated with it, and the ordering is determined by the **age** of the transaction.

CONT…

❖Time Stamp Based Methods

- ✓A transaction created at **0002** clock time would be older than all other transactions that come after it. For example, any transaction '**y**' entering the system at **0004** is two seconds younger and the **priority** would be given to the **older one.**
- ✓In addition, every data item is given the latest **read and write-timestamp**. This lets the system know when the last '**read and write'** operation was performed on the data item.

CONT…

❖Time stamp Ordering Protocol

- ✓The timestamp-ordering protocol ensures **serializability** among transactions in their conflicting **read and write** operations.
- ✓This is the responsibility of the Time stamp ordering protocol that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.
  - ✓ The timestamp of transaction Ti is denoted by **TS(Ti).**
  - ✓ Read timestamp of data-item X is denoted by **R-timestamp(X).**
  - ✓ Write timestamp of data-item X is denoted by **W-timestamp(X).**

CONT…

Timestamp ordering protocol works as follows:

- ✓ **If a transaction Ti issues a read(X) operation:**
- ✓ If **TS(Ti) < W-timestamp(X)**
  - ✓ Operation **rejected.**
- ✓ If **TS(Ti) >= W-timestamp(X)**
  - ✓ Operation **executed.**
- ✓ All data-item timestamps updated.

CONT…

Timestamp ordering protocol works as follows:

- ✓ **If a transaction Ti issues a write(X) operation:**
- ✓ If TS(Ti) < R-timestamp(X)
  - ✓ Operation rejected.
- ✓ If TS(Ti) < W-timestamp(X)
  - ✓ Operation rejected and Ti rolled back.
  - ✓ Otherwise, operation executed.

SUMMARY

- ✓**To summarize what we have learned**
  - ✓We have seen concurrency controlling techniques
    - ✓Locking methods
    - ✓Time stamp based methods
  - ✓Problem of locking methods
    - ✓Deadlock situation

“IF YOU HAVE ANY QUESTION”



“YOU ARE WEL-COME”


Thank You!