

# Chapter 21

## Introduction to Protocols for Concurrency Control in Databases

# Outline

## **Databases Concurrency Control**

- 1 Purpose of Concurrency Control
- 2 Two-Phase locking
- 3 Timestamp
- 4 **Validation (Optimistic)**

# Database Concurrency Control

## 1 Purpose of Concurrency Control

- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Example: In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

# Database Concurrency Control

Locking is an operation which secures (a) permission to Read or (b) permission to Write a data item for a transaction. Example: Lock (X). Data item X is locked on behalf of the requesting transaction.

Unlocking is an operation which removes these permissions from the data item. Example: Unlock (X). Data item X is made available to all other transactions.

Lock and Unlock are Atomic operations.

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

Two locks modes (a) shared (read) and (b) exclusive (write).

Shared mode: shared lock (X). More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.

Exclusive mode: Write lock (X). Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.

Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

Lock Manager: Managing locks on data items.

Lock table: Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

Database requires that all transactions should be well-formed. A transaction is well-formed if:

- It must lock the data item before it reads or writes to it.
- It must not lock an already locked data items and it must not try to unlock a free data item.

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

The following code performs the lock operation:

```
B: if LOCK (X) = 0 (*item is unlocked*)  
    then LOCK (X) ← 1 (*lock the item*)  
    else begin  
        wait (until lock (X) = 0) and  
        the lock manager wakes up the transaction);  
        goto B  
    end;
```



# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

The following code performs the unlock operation:

LOCK (X)  $\leftarrow$  0 (\*unlock the item\*)

if any transactions are waiting then

wake up one of the waiting the transactions;

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

The following code performs the read lock operation:

```
B: if LOCK (X) = “unlocked” then
    begin LOCK (X) ← “read-locked”;
        no_of_reads (X) ← 1;
    end
else if LOCK (X) ← “read-locked” then
    no_of_reads (X) ← no_of_reads (X) + 1
else begin wait (until LOCK (X) = “unlocked” and
    the lock manager wakes up the transaction);
    go to B
end;
```

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

The following code performs the write lock operation:

```
B: if LOCK (X) = “unlocked”  
    then LOCK (X) ← “write-locked”;  
    else begin wait (until LOCK (X) = “unlocked” and  
                    the lock manager wakes up the transaction);  
        go to B  
    end;
```

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

The following code performs the unlock operation:

```
if LOCK (X) = “write-locked” then
    begin LOCK (X) ← “unlocked”;
        wakes up one of the transactions, if any
    end
else if LOCK (X) ← “read-locked” then
    begin
        no_of_reads (X) ← no_of_reads (X) -1
        if no_of_reads (X) = 0 then
            begin
                LOCK (X) = “unlocked”;
                wake up one of the transactions, if any
            end
        end
    end;
end;
```

# Database Concurrency Control

## Two-Phase Locking Techniques: Essential components

### Lock conversion

#### Lock upgrade: existing read lock to write lock

if  $T_i$  has a read-lock (X) and  $T_j$  has no read-lock (X) ( $i \neq j$ ) then

    convert read-lock (X) to write-lock (X)

else

    force  $T_i$  to wait until  $T_j$  unlocks X

#### Lock downgrade: existing write lock to read lock

$T_i$  has a write-lock (X) (\*no transaction can have any lock on X\*)

    convert write-lock (X) to read-lock (X)

# Database Concurrency Control

## Two-Phase Locking Techniques :The algorithm

**Two Phases:** (a) Locking (Growing) (b) Unlocking (Shrinking).

**Locking (Growing) Phase:** A transaction applies locks (read or write) on desired data items one at a time.

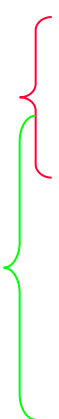
**Unlocking (Shrinking) Phase:** A transaction unlocks its locked data items one at a time.

**Requirement:** For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.


# Database Concurrency Control

## Two-Phase Locking Techniques: The algorithm

### T1

read\_lock (Y);  
read\_item (Y);  
unlock (Y);  
write\_lock (X);  
read\_item (X);  
X:=X+Y;  
write\_item (X);  
unlock (X);

### T2

read\_lock (X);  
read\_item (X);  
unlock (X);  
Write\_lock (Y);  
read\_item (Y);  
Y:=X+Y;  
write\_item (Y);  
unlock (Y);

### Result

Initial values: X=20; Y=30  
Result of serial execution  
T1 followed by T2  
X=50, Y=80.  
Result of serial execution  
T2 followed by T1  
X=70, Y=50

# Database Concurrency Control

## Two-Phase Locking Techniques: The algorithm

	<u>T1</u>	<u>T2</u>	<u>Result</u>
Time ↓	read_lock (Y); read_item (Y); <b>unlock (Y);</b>		X=50; Y=50 Nonserializable because it. violated two-phase policy.
	<b>write_lock (X);</b> read_item (X); X:=X+Y; write_item (X); unlock (X);	read_lock (X); read_item (X); <b>unlock (X);</b> <b>write_lock (Y);</b> read_item (Y); Y:=X+Y; write_item (Y); unlock (Y);	



# Database Concurrency Control

## Two-Phase Locking Techniques: The algorithm

T'1

{ read\_lock (Y);  
read\_item (Y);  
write\_lock (X);  
unlock (Y);  
{ read\_item (X);  
X:=X+Y;  
write\_item (X);  
unlock (X);

T'2

{ read\_lock (X);  
read\_item (X);  
Write\_lock (Y);  
unlock (X);  
{ read\_item (Y);  
Y:=X+Y;  
write\_item (Y);  
unlock (Y);

T1 and T2 follow two-phase policy but they are subject to deadlock, which must be dealt with.

# Database Concurrency Control

## Two-Phase Locking Techniques: The algorithm

**Two-phase policy generates two locking algorithms (a) Basic and (b) Conservative.**

**Conservative: Prevents deadlock by locking all desired data items before transaction begins execution.**

**Basic: Transaction locks data items incrementally. This may cause deadlock which is dealt with.**

**Strict: A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.**

# Database Concurrency Control

- **Two-Phase Locking Techniques: Problems**
  - Does not permit all possible serializable schedules
  - Deadlocks and starvations

# Database Concurrency Control

## Dealing with Deadlock and Starvation

### Deadlock

T'1

read\_lock (Y);  
read\_item (Y);

write\_lock (X);  
(waits for X)

T'2

read\_lock (X);  
read\_item (Y);

write\_lock (Y);  
(waits for Y)

T1 and T2 did follow two-phase policy but they are deadlock

**Deadlock (T'1 and T'2)**

# Database Concurrency Control

## Dealing with Deadlock and Starvation

### Deadlock prevention

A transaction locks all data items it refers to before it begins execution. This way of locking prevents deadlock since a transaction never waits for a data item. The conservative two-phase locking uses this approach.

# Database Concurrency Control

## Dealing with Deadlock and Starvation

### Deadlock detection and resolution

In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.

A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like:  $T_i$  waits for  $T_j$  waits for  $T_k$  waits for  $T_i$  or  $T_j$  occurs, then this creates a cycle. One of the transaction of the cycle is selected and rolled back.

# Database Concurrency Control

## Dealing with Deadlock and Starvation

### Deadlock avoidance

There are many variations of two-phase locking algorithm. Some avoid deadlock by not letting the cycle to complete. That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.

# Database Concurrency Control

## Dealing with Deadlock and Starvation

### Starvation

Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further. In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back. This limitation is inherent in all priority based scheduling mechanisms. In Wound-Wait scheme a younger transaction may always be wounded (aborted) by a long running older transaction which may create starvation.



# Database Concurrency Control

## Timestamp based concurrency control algorithm

### Timestamp

A monotonically increasing variable (integer) indicating the age of an operation or a transaction. A larger timestamp value indicates a more recent event or operation.

Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.

No deadlocks.

# Database Concurrency Control

## Timestamp based concurrency control algorithm

### Basic Timestamp Ordering

1. Transaction T issues a write\_item(X) operation:
  - a. If  $\text{read\_TS}(X) > \text{TS}(T)$  or if  $\text{write\_TS}(X) > \text{TS}(T)$ , then a younger transaction has already read the data item so abort and roll-back T and reject the operation.
  - b. If the condition in part (a) does not exist, then execute write\_item(X) of T and set write\_TS(X) to TS(T).
2. Transaction T issues a read\_item(X) operation:
  - a. If  $\text{write\_TS}(X) > \text{TS}(T)$ , then a younger transaction has already written to the data item so abort and roll-back T and reject the operation.
  - b. If  $\text{write\_TS}(X) \leq \text{TS}(T)$ , then execute read\_item(X) of T and set read\_TS(X) to the larger of TS(T) and the current read\_TS(X).

Problems: cascading rollback, cyclic restart (starvation)

~~T1 (100) ————— Write\_ts (X)~~  
~~T2 (120) ————— Write\_ts (X)=120~~

# Database Concurrency Control

## Timestamp based concurrency control algorithm

### Strict Timestamp Ordering

1. Transaction T issues a write\_item(X) operation:
  - a. If  $TS(T) > read\_TS(X)$ , then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).
2. Transaction T issues a read\_item(X) operation:
  - a. If  $TS(T) > write\_TS(X)$ , then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

# Database Concurrency Control

## Timestamp based concurrency control algorithm

### Thomas's Write Rule

1. If  $\text{read\_TS}(X) > \text{TS}(T)$  then abort and roll-back  $T$  and reject the operation.
2. If  $\text{write\_TS}(X) > \text{TS}(T)$ , then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
3. If the conditions given in 1 and 2 above do not occur, then execute  $\text{write\_item}(X)$  of  $T$  and set  $\text{write\_TS}(X)$  to  $\text{TS}(T)$ .

# Database Concurrency Control

## Validation (Optimistic) Concurrency Control Schemes

In this technique only at the time of commit serializability is checked and transactions are aborted in case of non-serializable schedules.

Three phases:

Read phase: A transaction can read values of committed data items. However, updates are applied only to local copies (versions) of the data items (in database cache).

# Database Concurrency Control

## Validation (Optimistic) Concurrency Control Schemes

Validation phase: Serializability is checked before transactions write their updates to the database.

This phase for  $T_i$  checks that, for each transaction  $T_j$  that is either committed or is in its validation phase, one of the following conditions holds:

1.  $T_j$  completes its write phase before  $T_i$  starts its read phase.
2.  $T_i$  starts its write phase after  $T_j$  completes its write phase, and the `read_set` of  $T_i$  has no items in common with the `write_set` of  $T_j$
3. Both the `read_set` and `write_set` of  $T_i$  have no items in common with the `write_set` of  $T_j$ , and  $T_j$  completes its read phase.

# Database Concurrency Control

## Validation (Optimistic) Concurrency Control Schemes

When validating  $T_i$ , the first condition is checked first for each transaction  $T_j$ , since (1) is the simplest condition to check. If (1) is false then (2) is checked and if (2) is false then (3) is checked. If none of these conditions holds, the validation fails and  $T_i$  is aborted.

Write phase: On a successful validation transactions' updates are applied to the database; otherwise, transactions are restarted.

# Database Concurrency Control

## Granularity of data items and Multiple Granularity Locking

A lockable unit of data defines its granularity. Granularity can be coarse (entire database) or it can be fine (a tuple or an attribute of a relation). Data item granularity significantly affects concurrency control performance. Thus, the degree of concurrency is low for coarse granularity and high for fine granularity. Example of data item granularity:

1. A field of a database record (an attribute of a tuple).
2. A database record (a tuple or a relation).
3. A disk block.
4. An entire file.
5. The entire atabase.



# Database Concurrency Control

## Granularity of data items and Multiple Granularity Locking

The following diagram illustrates a hierarchy of granularity from coarse (database) to fine (record).

