Transaction Management

MRS. SANGEETA BHANDARI
ASSISTANT PROFESSOR

HANS RAJ MAHILA MAHA VIDYALAYA,
JALANDHAR

Transactions

- What's a transaction?
- What properties transactions should have?
- Why do we want concurrent execution of user programs?
- What are the problems when interleaving transactions?
- How might we overcome these?

What is a Transaction?

- A collection of operations performed on the database which are executed as a single unit that may or may not change the contents of the database so as to ensure the consistency and maintain integrity constraints

What is a Transaction?

- Any action that reads from and/or writes to a database may consist of
  - Simple SELECT statement to generate a list of table contents
  - A series of related UPDATE statements to change the values of attributes in various tables
  - A series of INSERT statements to add rows to one or more tables
  - A combination of SELECT, UPDATE, and INSERT statements

What is a Transaction?

- A *logical* unit of work that must be either entirely completed or aborted
- Successful transaction changes the database from one *consistent* state to another
  - One in which all data integrity constraints are satisfied
- Most real-world database transactions are formed by two or more database requests
  - The equivalent of a single SQL statement in an application program or transaction

What is a Transaction?

- Not all transactions update the database
- SQL code represents a transaction because database was accessed
- Improper or incomplete transactions can have a devastating effect on database integrity
  - Some DBMSs provide means by which user can define enforceable constraints based on business rules
  - Other integrity rules are enforced automatically by the DBMS when table structures are properly defined, thereby letting the DBMS validate some transactions

What is a Transaction?

- For example, a transaction may involve
  - The creation of a new invoice
  - Insertion of an row in the LINE table
  - Decreasing the quantity on hand by 1
  - Updating the customer balance
  - Creating a new account transaction row
- If the system fails between the first and last step, the database will no longer be in a consistent state

Transaction Properties

- Atomicity
  - Requires that *all* operations (SQL requests) of a transaction be completed; if not, then the transaction is aborted
  - A transaction is treated as a single, indivisible, logical unit of work
  - Atomicity is maintained in the presence of disk failure, CPU failure and application software failure
  - Atomicity is maintained in the presence of deadlocks

Transaction Properties

- A DBMS ensures atomicity by **undoing** the actions of partial transactions
- To enable this, the DBMS maintains a record, called a **log**, of all writes to the database
- The component of a DBMS responsible for this is called the **recovery manager**

Transaction Properties

- Consistency
  - Means no violation of integrity rules
  - A transaction is said to be consistent if the database before the start of the transaction and after the completion of transaction is in consistent state
  - Preservation of consistency is the responsibility of database programmer
  - For example, consistency criterion that my inter-account-transfer transaction does not change the total amount of money in the accounts!

Transaction Properties

- Isolation
  - Ensures that the concurrent execution of several transactions yields consistent results
  - Multiple, concurrent transactions appear as if they executed in serial order (one after another)
  - Data used during execution of a transaction cannot be used by second transaction until first one is completed
- For example, if transactions T1 and T2 are executed concurrently, the net effect is equivalent to executing
  - T1 followed by T2, **or**
  - T2 followed by T1

Transaction Properties

- Durability
  - Indicates permanence of database's consistent state
  - When a transaction is complete, the database reaches a consistent state. That state can not be lost even if the system fails
  - The system must ensure that once the transaction commits, its effect on the database state is not lost inspite of subsequent faqilures.

Transaction Properties

- DBMS uses the log to ensure durability
- If the system crashed before the changes made by a completed transaction are written to disk, the log is used to remember and restore these changes when the system is restarted

Why concurrent transactions ?

- Usually large systems are used for Database processing
- Many users needs to access the database at the same time
- Different concurrent users may or may not be aware of each other's presence
- Concurrent execution of user programs is essential for good DBMS performance

Why concurrent transactions ?

- Disk access is frequent and slow ☹
- Overlapping I/O and CPU activity reduces the amount of time disks and processor are idle
- Want to keep the CPU busy ☺
- Increase the system throughput.

Transactions and schedules

- A transaction is seen by the DBMS as a series, or list, of actions
  - Includes read and write of objects
  - We'll write this as R(o) and W(o) (sometimes $R_T(o)$ and $W_T(o)$)
- For example

  T1: [R(a), W(a), R(c), W(c)]

  T2: [R(b), W(b)]
- In addition, a transaction should specify as its final action either **commit**, or **abort**

Transactions and schedules

- A **schedule** is a list of actions from a set of transactions
  - A well-formed schedule is one where the actions of a particular transaction T are in the same order as they appear in T
- For example
  - $[R_{T1}(a), W_{T1}(a), R_{T2}(b), W_{T2}(b), R_{T1}(c), W_{T1}(c)]$ is a well-formed schedule
  - $[R_{T1}(c), W_{T1}(c), R_{T2}(b), W_{T2}(b), R_{T1}(a), W_{T1}(a)]$ is not a well-formed schedule

Transactions and schedules

- A **complete schedule** is one that contains an abort or commit action for every transaction that occurs in the schedule
- A **serial schedule** is one where the actions of different transactions are not interleaved

Serializability

- A **serializable schedule** is a schedule whose effect on any consistent database instance is identical to that of some complete serial schedule

Anomalies with Concurrent execution

- Two actions on the same data object **conflict** if at least one of them is a write
- We'll now consider three ways in which a schedule involving two consistency-preserving transactions can leave a consistent database inconsistent

Dirty Read/WR Conflicts

- Transaction T2 reads a database object that has been modified by T1 which has not committed
  - T1 transfers Rs 100 from account a to account b
  - T2 displays the sum of balance amount of account a and b
  - If the operations of T1 and T2 are performed in the following way, it may lead to inconsistent results

T1: R(a),W(a), R(b),W(b),C

T2: R(a),W(a),R(b),W(b),C

Unrepeatable Read/RW Conflict

- Transaction T2 could change the value of an object that has been read by a transaction T1, while T1 is still in progress

T1: R(a), R(a), W(a), C
T2: R(a),W(a),C

T1: R(a), W(a),C
T2: R(a), W(a),C

Lost Update/WW Conflict

- Transaction T2 could overwrite the value of an object which has already been modified by T1, while T1 is still in progress

  T1: [W(Priya), W(Amit)] "Set both salaries at Rs.1m"

  T2: [W(Amit), W(Priya)] "Set both salaries at $1m"

- But:

T1: W(Priya), W(Amit)

T2: W(Amit), W(Priya)

Cascading aborts

- Things are more complicated when transactions can abort
- T1, transfer money from account a to b
- T2, print the sum of balance of account a and b
- Following schedule will create problem

T1:R(a), W(a), **Abort**

T2: R(a),W(a),R(b),W(b),C

Transaction Management with SQL

- ANSI has defined standards that govern SQL database transactions
- Transaction support is provided by two SQL statements: COMMIT and ROLLBACK
- ANSI standards require that, when a transaction sequence is initiated by a user or an application program,it must continue through all succeeding SQL statements until one of four events occurs

Transaction Management with SQL

1. A COMMIT statement is reached- all changes are permanently recorded within the database
2. A ROLLBACK is reached – all changes are aborted and the database is restored to a previous consistent state
3. The end of the program is successfully reached – equivalent to a COMMIT
4. The program abnormally terminates and a rollback occurs

Concurrency Control

- The coordination of the simultaneous execution of transactions in a multiprocessing database is known as ***concurrency control***

- The objective of concurrency control is to ensure the serializability of transactions in a multiuser database environment

Scheduler

- Special DBMS program: establishes order of operations within which concurrent transactions are executed
- Interleaves the execution of database operations to ensure serializability and isolation of transactions
  - To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms such as locking and time stamping

Scheduler

- Ensures computer's central processing unit (CPU) is used efficiently
- Facilitates data isolation to ensure that two transactions do not update the same data element at the same time

Concurrency Control

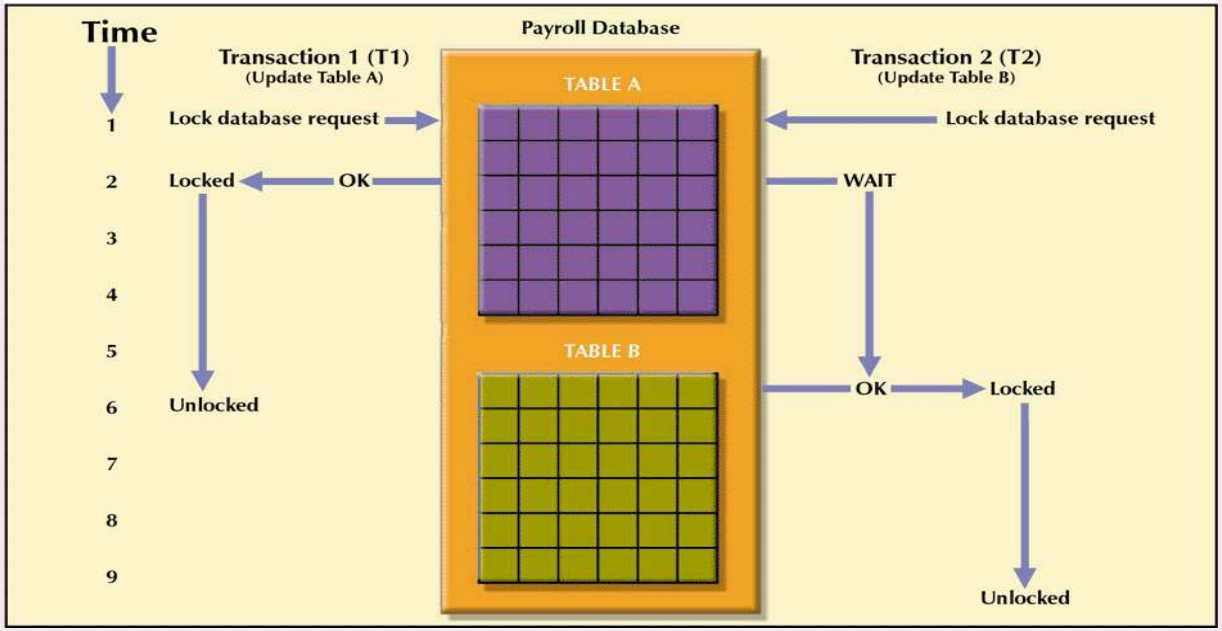
- Lock
  - Guarantees exclusive use of a data item to a current transaction
    - T2 does not have access to a data item that is currently being used by T1
    - Transaction acquires a lock prior to data access; the lock is released when the transaction is complete
  - Required to prevent another transaction from reading inconsistent data
- Lock manager
  - Responsible for assigning and policing the locks used by the transactions

Lock Granularity

- Indicates the level of lock use
- Locking can take place at the following levels:
  - Database-level lock
    - Entire database is locked
  - Table-level lock
    - Entire table is locked
  - Page-level lock
    - Entire diskpage is locked
  - Row-level lock
    - Allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page
  - Field-level lock
    - Allows concurrent transactions to access the same row, as long as they require the use of different fields (attributes) within that row

Lock at Database Level

FIGURE 9.3 A DATABASE-LEVEL LOCKING SEQUENCE

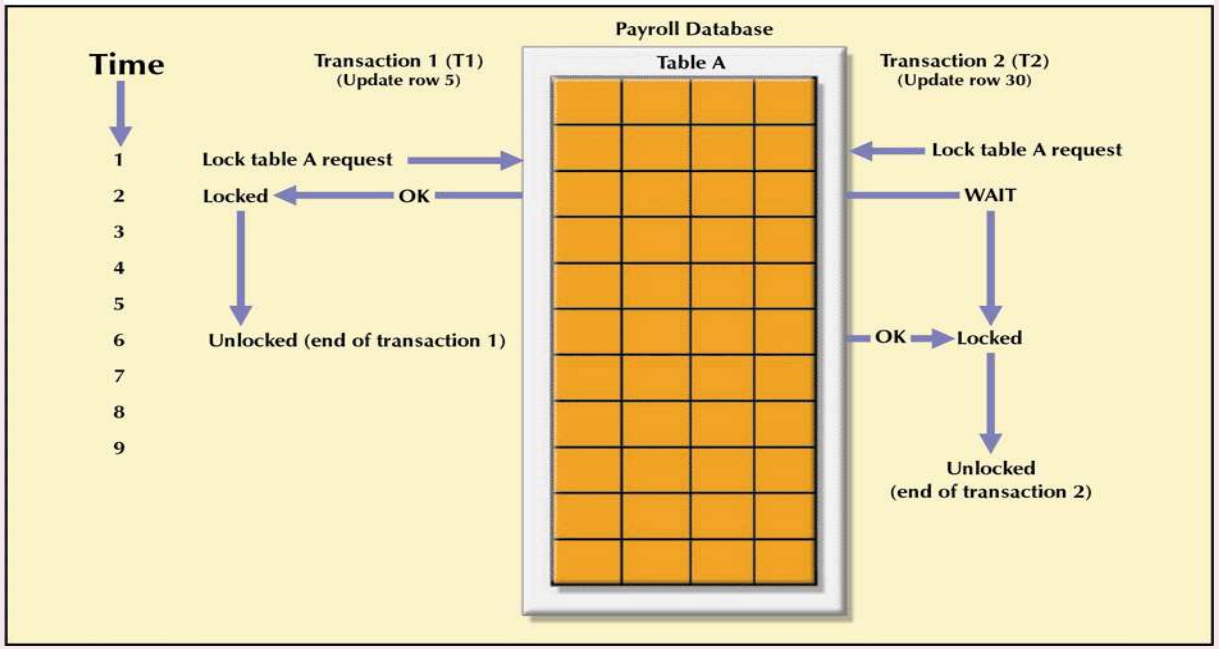


- Good for batch processing but unsuitable for online multi-user DBMSs
- T1 and T2 can not access the same database concurrently even if they use different tables

Table level Lock

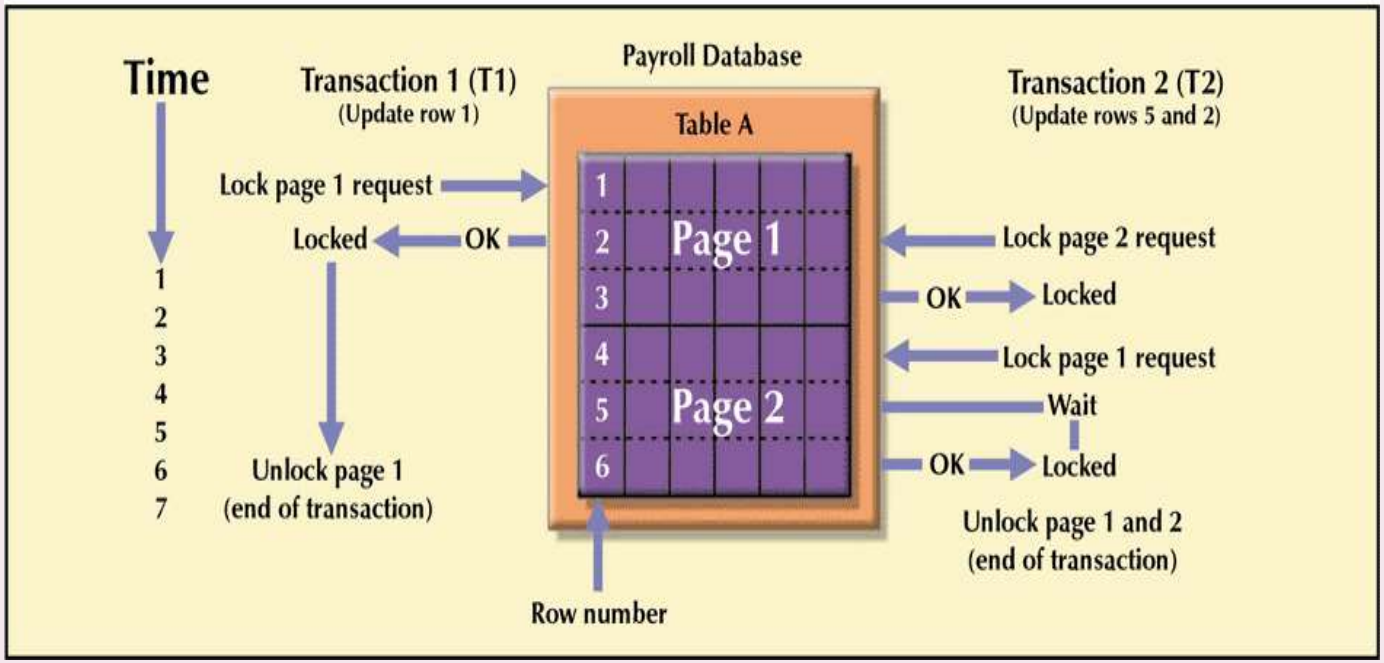
- T1 and T2 can access the same database concurrently as long as they use different tables
- Can cause bottlenecks when many transactions are trying to access the same table (even if the transactions want to access different parts of the table and would not interfere with each other)
- Not suitable for multi-user DBMSs

FIGURE 9.4 AN EXAMPLE OF A TABLE-LEVEL LOCK



Page level Lock

FIGURE 9.5 AN EXAMPLE OF A PAGE-LEVEL LOCK

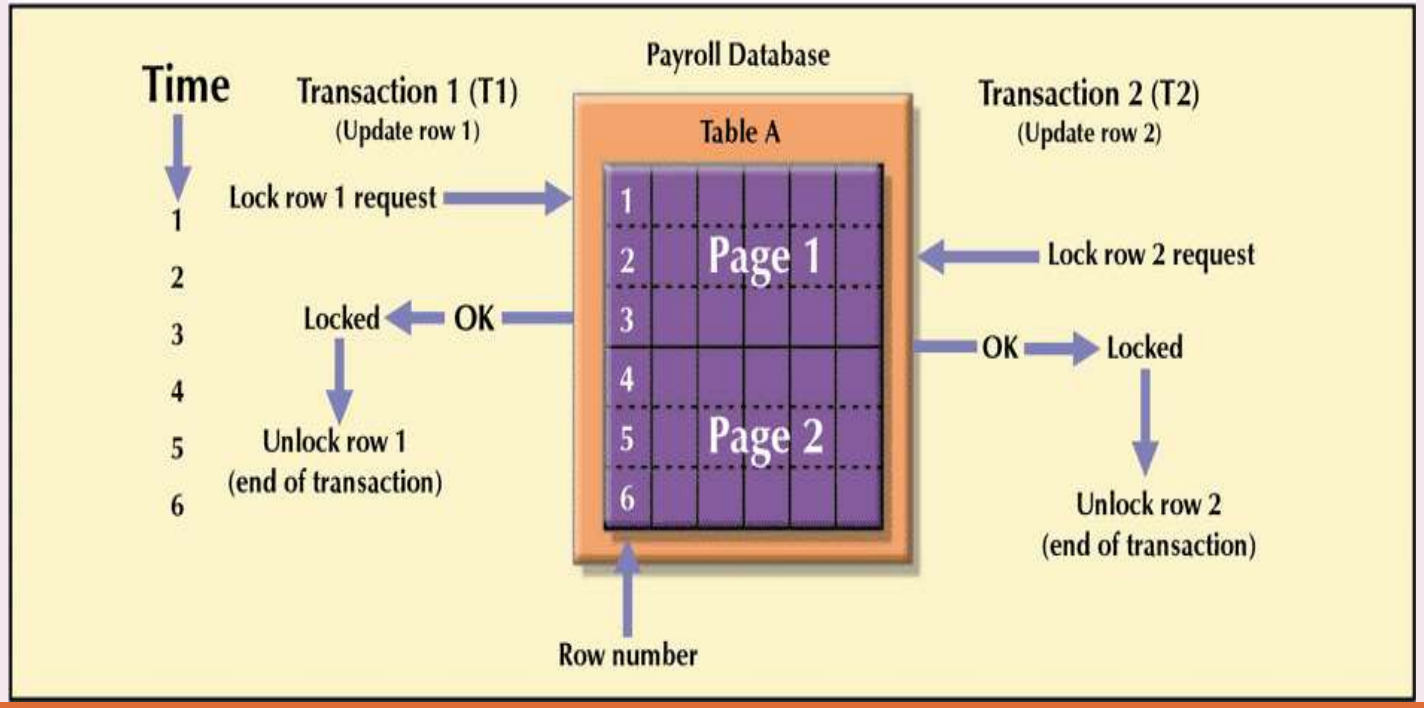


- An entire disk page is locked (a table can span several pages and each page can contain several rows of one or more tables)
- Most frequently used multi-user DBMS locking method

Row Level Lock

- Concurrent transactions can access different rows of the same table even if the rows are located on the same page
- Improves data availability but with high overhead (each row has a lock that must be read and written to)

FIGURE 9.6 AN EXAMPLE OF A ROW-LEVEL LOCK



Field Level Lock

- Allows concurrent transactions to access the same row as long as they require the use of different fields with that row
- Most flexible lock buy requires an extremely high level of overhead

Binary Locks

- Has only two states: locked (1) or unlocked (0)
- Eliminates “Lost Update” problem – the lock is not released until the write statement is completed
- Considered too restrictive to yield optimal concurrency conditions as it locks even for two READs when no update is being done

Binary Lock Example

- Can not use PROD_QOH until it has been properly updated

TABLE 9.10 AN EXAMPLE OF A BINARY LOCK

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Lock PRODUCT	
2	T1	Read PROD_QOH	15
3	T1	PROD_QOH = 15 + 10	
4	T1	Write PROD_QOH	25
5	T1	Unlock PRODUCT	
6	T2	Lock PRODUCT	
7	T2	Read PROD_QOH	23
8	T2	PROD_QOH = 23 – 10	
9	T2	Write PROD_QOH	13
10	T2	Unlock PRODUCT	

Exclusive/Shared Lock

- Exclusive lock
  - Access is specifically reserved for the transaction that locked the object
  - Must be used when the potential for conflict exists – when a transaction wants to update a data item and no locks are currently held on that data item by another transaction
  - *Granted if and only if no other locks are held on the data item*

Exclusive/Shared Lock

- Shared lock
  - Concurrent transactions are granted Read access on the basis of a common lock
  - Issued when a transaction wants to read data and no exclusive lock is held on that data item
    - Multiple transactions can each have a shared lock on the same data item if they are all just reading it
- Mutual Exclusive Rule
  - Only one transaction at a time can own an exclusive lock in the same object

Exclusive/Shared Lock

- Increased overhead
  - The type of lock held must be known before a lock can be granted
  - Three lock operations exist:
    - READ_LOCK to check the type of lock
    - WRITE_LOCK to issue the lock
    - UNLOCK to release the lock
  - A lock can be upgraded from share to exclusive and downgraded from exclusive to share
- Two possible major problems may occur
  - The resulting transaction schedule may not be serializable
  - The schedule may create deadlocks

2PL-Two Phase Locking Protocol

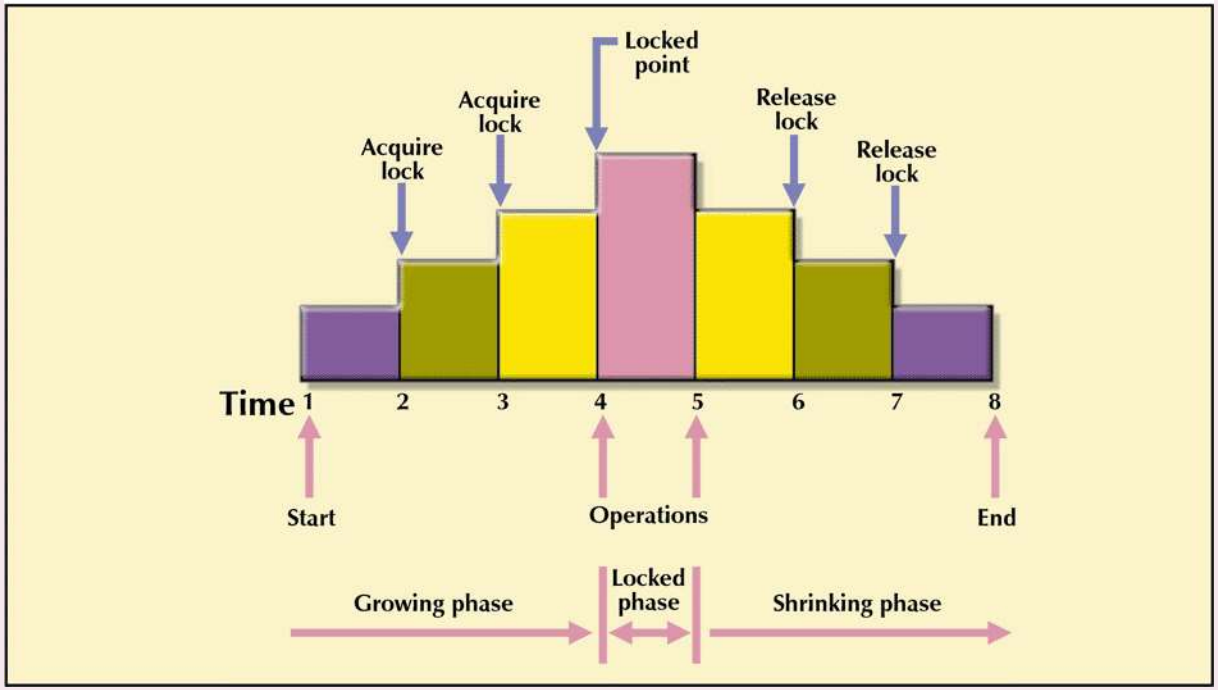
- Defines how transactions acquire and relinquish locks
- Guarantees serializability, but it does not prevent deadlocks
  - *Growing phase*, in which a transaction acquires all the required locks without unlocking any data
  - *Shrinking phase*, in which a transaction releases all locks and cannot obtain any new lock

2PL-Two Phase Locking Protocol

- Governed by the following rules:
  - Two transactions cannot have conflicting locks
  - No unlock operation can precede a lock operation in the same transaction
  - No data are affected until all locks are obtained—that is, until the transaction is in its locked point

2PL-Two Phase Locking Protocol

FIGURE 9.7 TWO-PHASE LOCKING PROTOCOL



Strict 2PL

- If a transaction Ti is aborted, then all actions must be undone
  - Also, if Tj reads object last written by Ti, then Tj must be aborted!
- Most systems avoid **cascading aborts** by releasing locks only at commit time (strict protocols)
  - If Ti writes an object, then Tj can only read this after Ti finishes
- In order to undo changes, the DBMS maintains a **log** which records every write

Thanks, Happy Learning