Chapter 9

TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

In this chapter, you will learn:

- Read/Write conflict scenario
- Locking Methods
- Lock Granularity
- Lock Types
- Concurrency Control with Time Stamping Methods
- Data Recovery Management

Read/Write Conflict Scenarios: Conflicting Database Operations Matrix

TABLE 9.9 READ/WRITE CONFLICT SCENARIOS: CONFLICTING DATABASE OPERATIONS MATRIX

	TRANSACTIONS		
	T1	T2	RESULT
Operations	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

Concurrency Control with Locking Methods

- Lock
  - Guarantees exclusive use of a data item to a current transaction
  - 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
  - Table
  - Page
  - Row
  - Field (attribute)

Lock Granularity (continued)

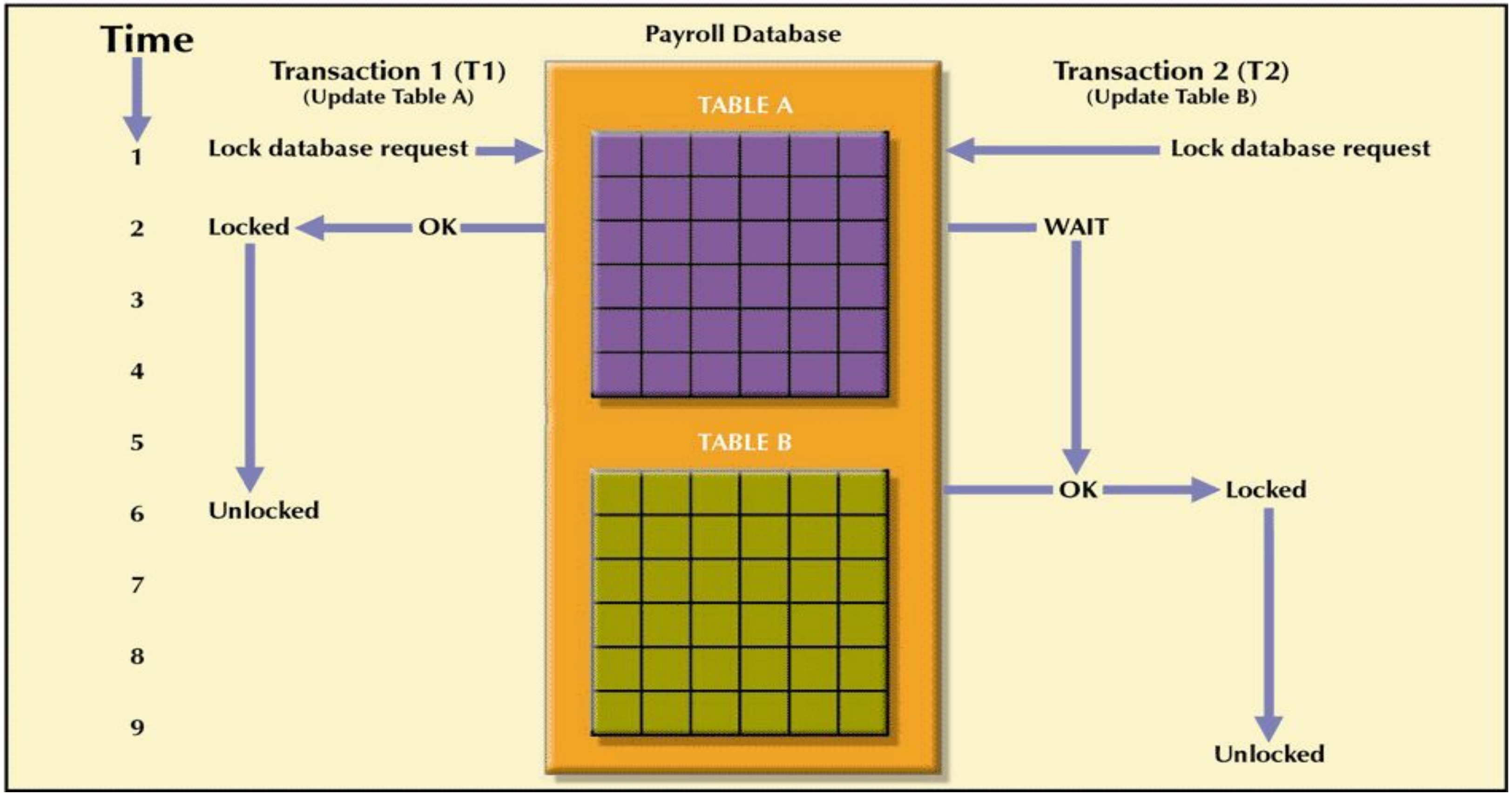
- Database-level lock
  - Entire database is locked
- Table-level lock
  - Entire table is locked
- Page-level lock
  - Entire diskpage is locked

Lock Granularity (continued)

- 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

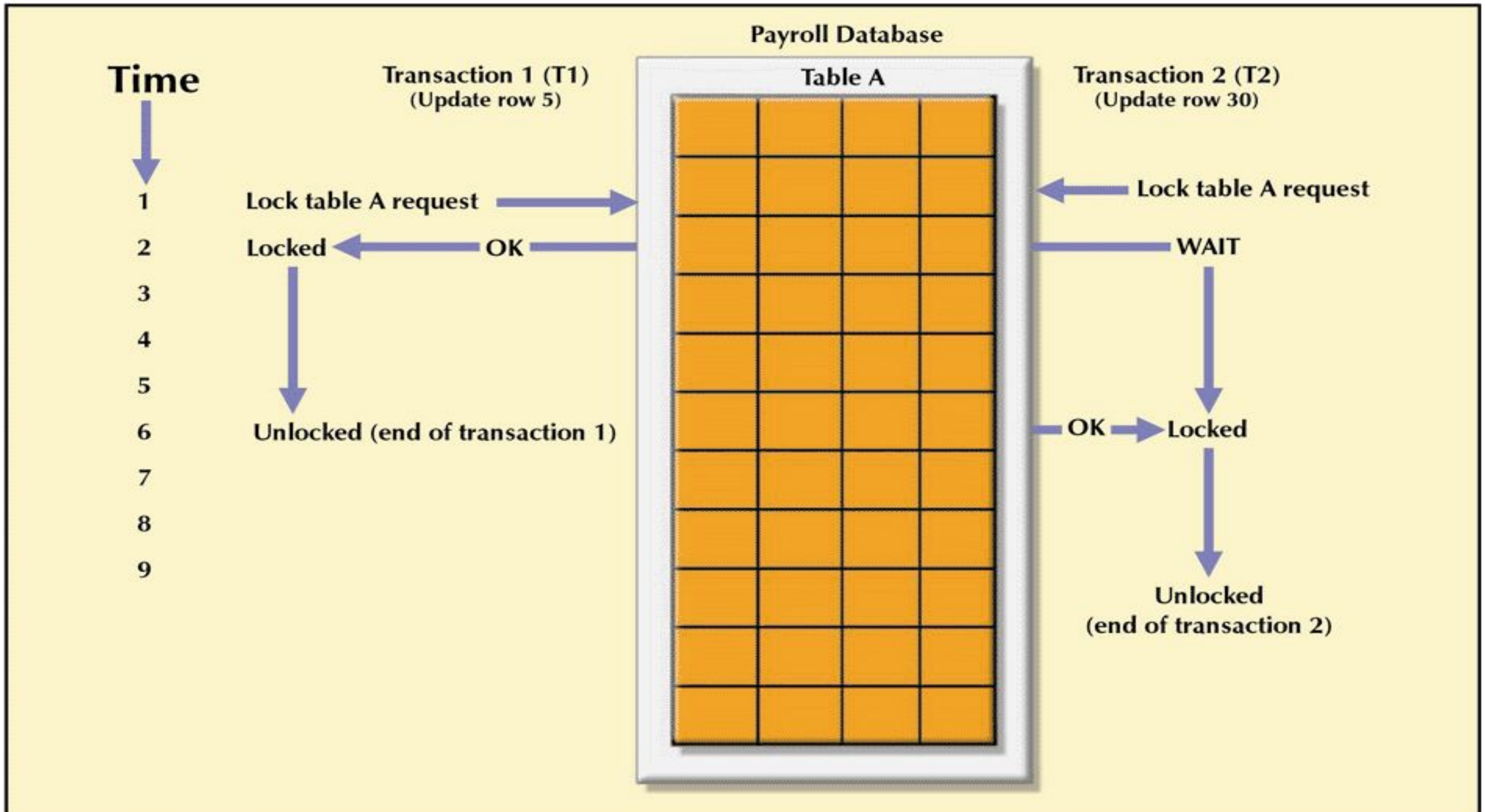
A Database-Level Locking Sequence

FIGURE 9.3 A DATABASE-LEVEL LOCKING SEQUENCE



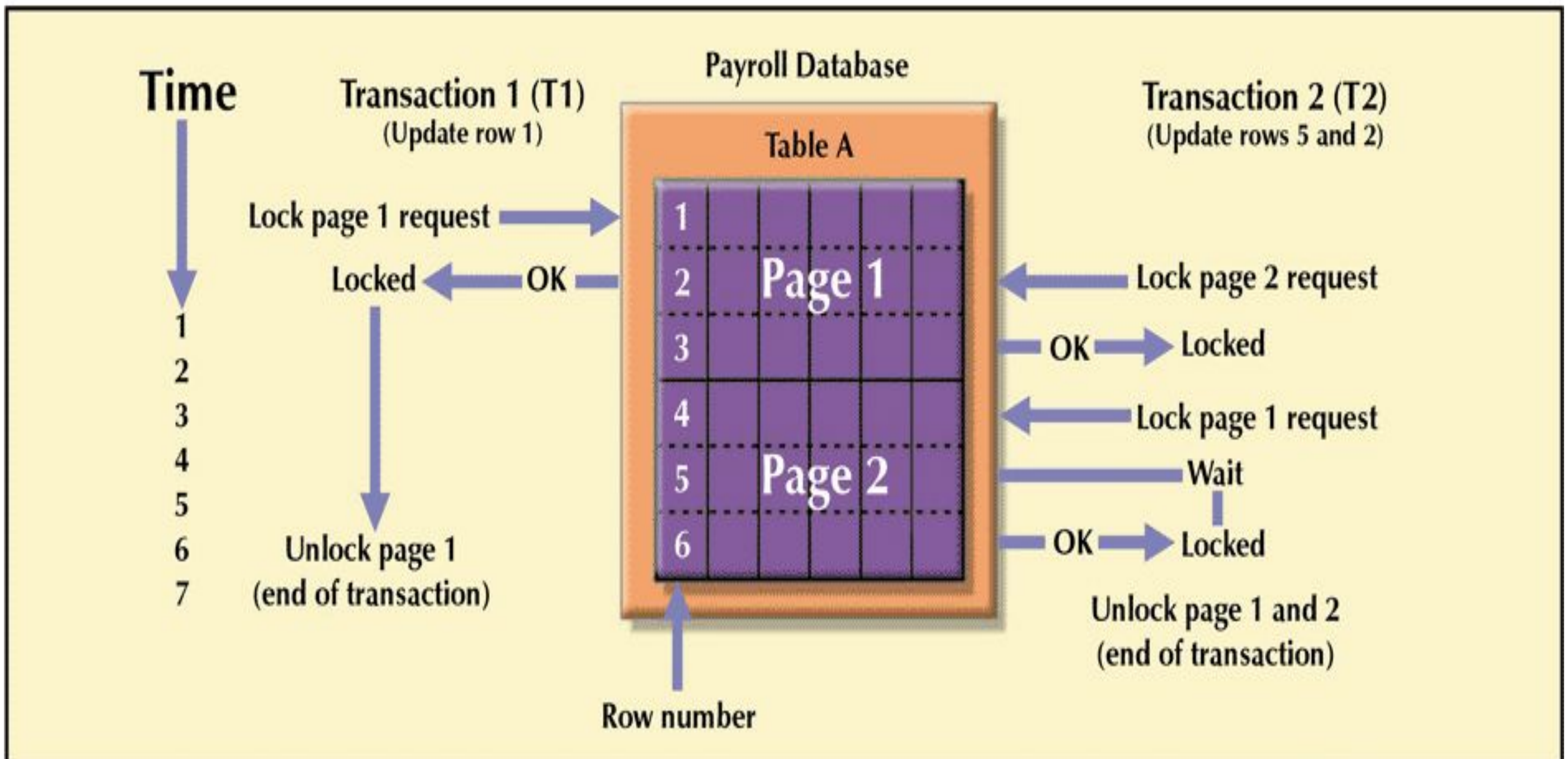
An Example of a Table-Level Lock

FIGURE 9.4 An Example of a Table-Level Lock



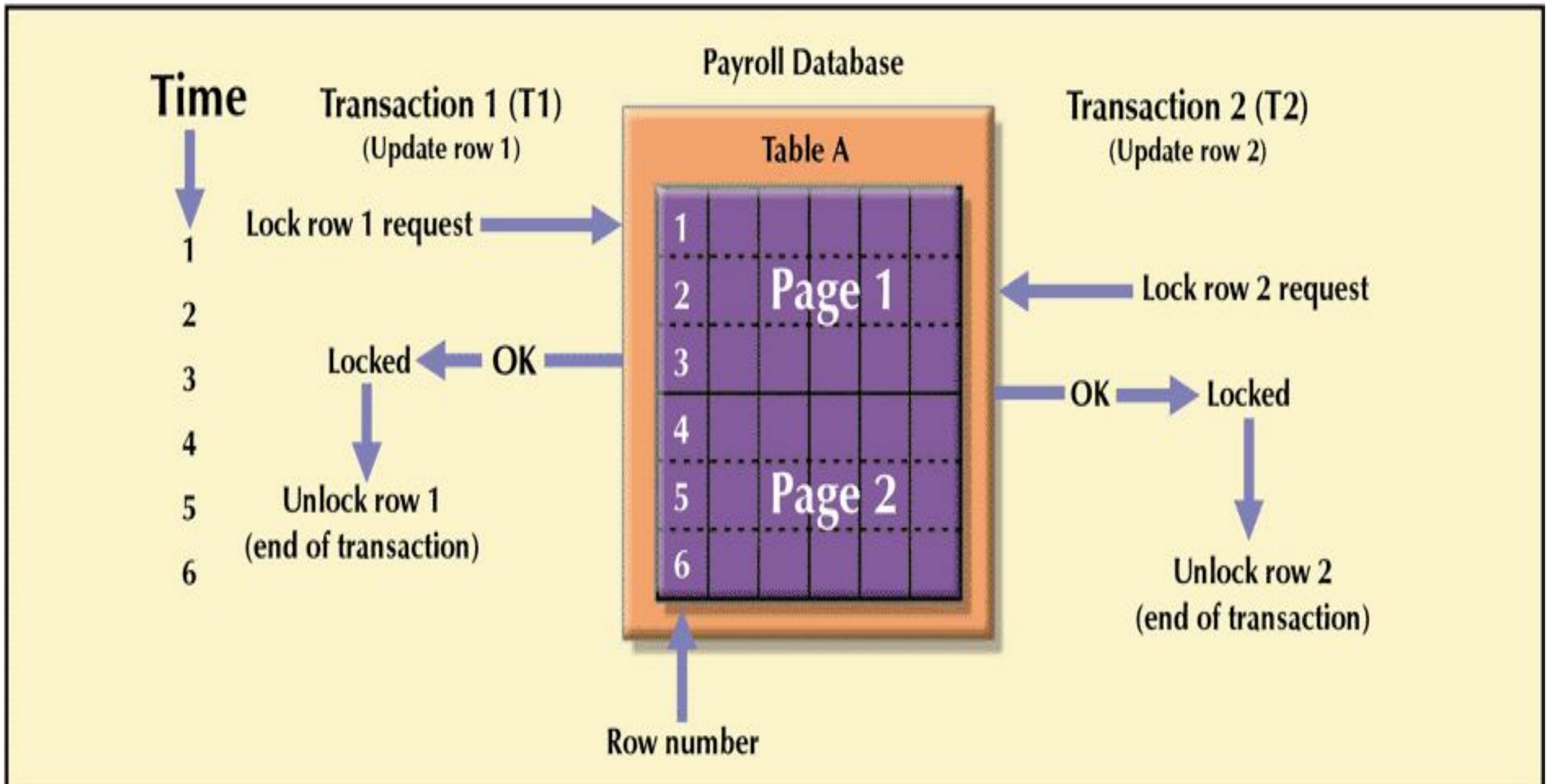
Example of a Page-Level Lock

FIGURE 9.5 AN EXAMPLE OF A PAGE-LEVEL LOCK



An Example of a Row-Level Lock

FIGURE 9.6 AN EXAMPLE OF A ROW-LEVEL LOCK



Lock Types

- Binary lock
  - Has only two states: locked (1) or unlocked (0)
- Exclusive lock
  - Access is specifically reserved for the transaction that locked the object
  - Must be used when the potential for conflict exists
- Shared lock
  - Concurrent transactions are granted Read access on the basis of a common lock

An Example of a Binary Lock

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	

Two-Phase Locking to Ensure Serializability

- 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

Two-Phase Locking to Ensure Serializability (continued)

- 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

Two-Phase Locking Example

(a)

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

(b) Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result of serial schedule T_2 followed by T_1: X=70, Y=50

- Transaction that support 2pl

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);

Two-Phase Locking to Ensure Serializability (continued)

Suppose two-phase locking does not ensure serializability. Then there exists a set of transactions $T0,T1$ ... $Tn-1$ which obey 2PL and which produce a nonserializable schedule. A non-serializable schedule implies a cycle in the

precedence graph, and we shall show that 2PL cannot produce such cycles.

Without loss of generality, assume the following cycle exists in the precedence

graph: $T0 \rightarrow T1 \rightarrow T2 \rightarrow \ldots \rightarrow Tn-1 \rightarrow T0$. Let αi be the time at which Ti obtains

its last lock (i.e. Ti's lock point). Then for all transactions such that $Ti \rightarrow Tj$,

$\alpha i < \alpha j$. Then for the cycle we have

$\alpha 0 < \alpha 1 < \alpha 2 < \ldots < \alpha n-1 < \alpha 0$

Since $\alpha 0 < \alpha 0$ is a contradiction, no such cycle can exist. Hence 2PL cannot

produce non-serializable schedules. Because of the property that for all trans-

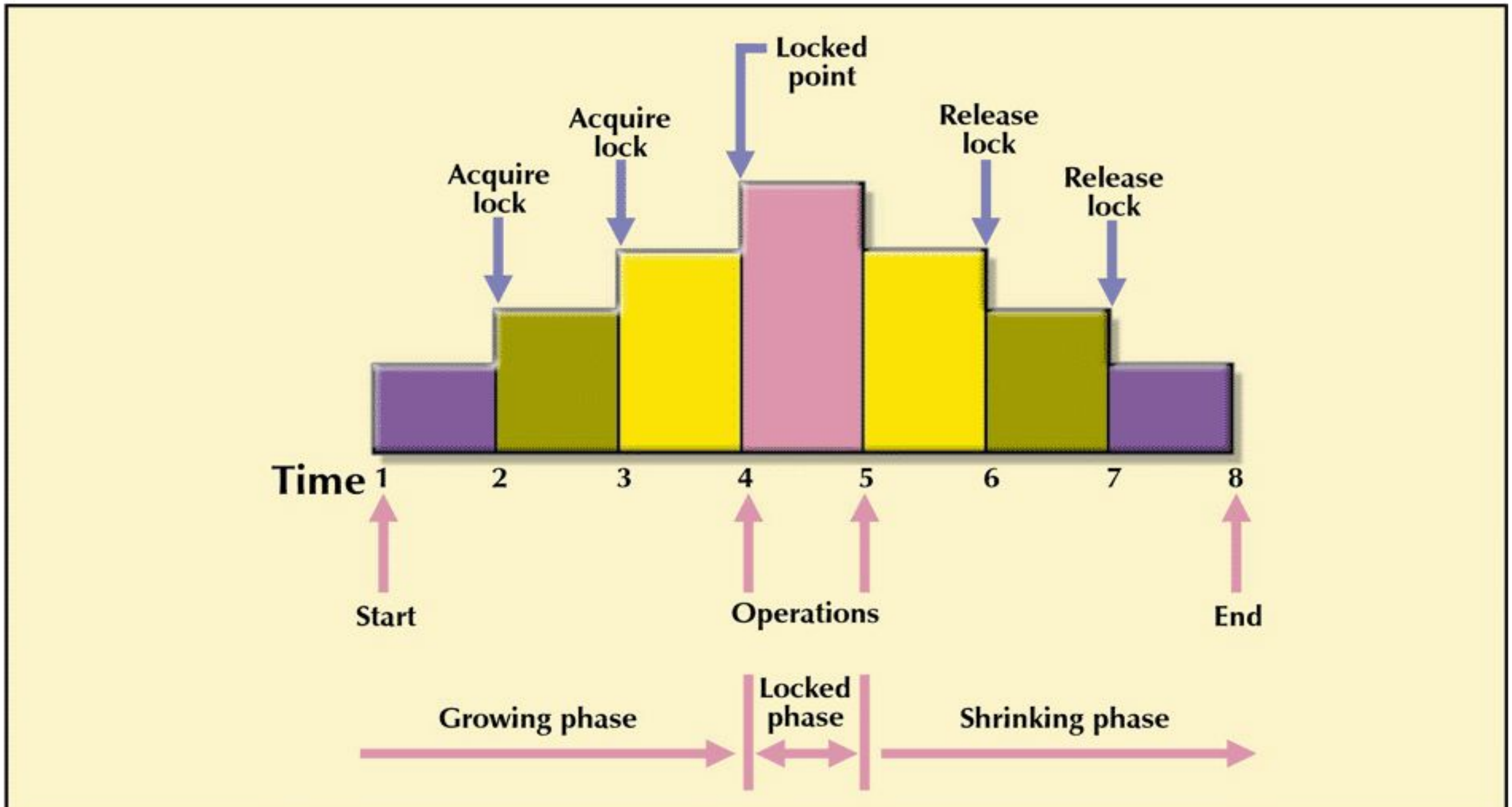
actions such that $Ti \rightarrow Tj$, $\alpha i < \alpha j$, the lock point ordering of the transactions

is also a topological sort ordering of the precedence graph. Thus transactions

can be serialized according to their lock points.

Two-Phase Locking Protocol

FIGURE 9.7 TWO-PHASE LOCKING PROTOCOL



Deadlocks

- Condition that occurs when two transactions wait for each other to unlock data
- Possible only if one of the transactions wants to obtain an exclusive lock on a data item
  - No deadlock condition can exist among *shared* locks
- Control through
  - Prevention
  - Detection
  - Avoidance

How a Deadlock Condition Is Created

TABLE 9.11 How a Deadlock Condition Is Created

TIME	TRANSACTION	REPLY	LOCK STATUS		
0			Data X		Data Y
1	T1:LOCK(X)	OK	Unlocked		Unlocked
2	T2: LOCK(Y)	OK	Locked		Unlocked
3	T1:LOCK(Y)	WAIT	Locked		Locked
4	T2:LOCK(X)	WAIT	Locked	**Deadlock**	Locked
5	T1:LOCK(Y)	WAIT	Locked		Locked
6	T2:LOCK(X)	WAIT	Locked	Deadlock	Locked
7	T1:LOCK(Y)	WAIT	Locked		Locked
8	T2:LOCK(X)	WAIT	Locked		Locked
9	T1:LOCK(Y)	WAIT	Locked		Locked
...	....	....	....		....
...	....	....	....		....
...	....	....	....		....
...	....	....	....		....

Concurrency Control with Time Stamping Methods

- Assigns a global unique time stamp to each transaction
- Produces an explicit order in which transactions are submitted to the DBMS
- Uniqueness
  - Ensures that no equal time stamp values can exist
- Monotonicity
  - Ensures that time stamp values always increase

Deadlock Prevention Techniques

- **Wait-die scheme**: It is a non-preemptive technique for deadlock prevention. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (That is T_i is older than T_j), otherwise T_i is rolled back (dies)
- **For example:**
- Suppose that transaction T_{22}, T_{23}, T_{24} have time-stamps 5, 10 and 15 respectively. If T_{22} requests a data item held by T_{23} then T_{22} will wait. If T_{24} requests a data item held by T_{23}, then T_{24} will be rolled back.

Wait/Die and Wound/Wait Schemes

- **Wound-wait scheme**: It is a preemptive technique for deadlock prevention. It is a counterpart to the wait-die scheme. When Transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j, otherwise T_j is rolled back (T_j is wounded by T_i)
- **For example:**
- Suppose that Transactions T_{22}, T_{23}, T_{24} have time-stamps 5, 10 and 15 respectively . If T_{22} requests a data item held by T_{23}, then data item will be preempted from T_{23} and T_{23} will be rolled back. If T_{24} requests a data item held by T_{23}, then T_{24} will wait.

Wait/Die and Wound/Wait Concurrency Control Schemes

TABLE 9.12 Wait/Die and Wound/Wait Concurrency Control Schemes

TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT/DIE SCHEME	WOUND/WAIT SCHEME
T1 (11548789)	T2 (19562545)	• T1 waits until T2 is completed and T2 releases its locks.	• T1 preempts (rolls back) T2. T2 is rescheduled using the same time stamp.
T2 (19562545)	T1 (11548789)	• T2 Dies (rolls back). • T2 is rescheduled using the same time stamp.	• T2 waits until T1 is completed and T1 releases its locks.

Concurrency Control with Optimistic Methods

- Optimistic approach
  - Based on the assumption that the majority of database operations do not conflict
  - Does not require locking or time stamping techniques
  - Transaction is executed without restrictions until it is committed
  - Phases are read, validation, and write

Database Recovery Management

- Database recovery
  - Restores database from a given state, usually inconsistent, to a previously consistent state
  - Based on the atomic transaction property
    - All portions of the transaction must be treated as a single logical unit of work, in which all operations must be applied and completed to produce a consistent database
  - If transaction operation cannot be completed, transaction must be aborted, and any changes to the database must be rolled back (undone)

Transaction Recovery

- Makes use of deferred-write and write-through
- Deferred write
  - Transaction operations do not immediately update the physical database
  - Only the transaction log is updated
  - Database is physically updated only after the transaction reaches its commit point using the transaction log information

Transaction Recovery (continued)

- Write-through
  - Database is immediately updated by transaction operations during the transaction's execution, even before the transaction reaches its commit point

A Transaction Log for Transaction Recovery Examples

TABLE 9.13 A Transaction Log for Transaction Recovery Examples

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	****Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ...
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ...
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ...
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	****Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				
					* * * * * C *R*A* S* H * * * *				

Summary

- Transaction
  - Sequence of database operations that access the database
  - Represents real-world events
  - Must be a logical unit of work
    - No portion of the transaction can exist by itself
  - Takes a database from one consistent state to another
    - One in which all data integrity constraints are satisfied

Summary (continued)

- SQL provides support for transactions through the use of two statements: COMMIT and ROLLBACK
- Concurrency control coordinates the simultaneous execution of transactions
- Scheduler is responsible for establishing order in which concurrent transaction operations are executed

Summary (continued)

- Lock guarantees unique access to a data item by a transaction
- Database recovery restores the database from a given state to a previous consistent state

Summary (continued)

- Dirty read:
- Nonrepeatable reads
- Phantom Reads

Trans

Isolation Levels: Summary

weak

	dirty reads	nonrepeatable reads	phantoms
Read Uncommitted	Y	Y	Y
Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N

strong

- Phantom read: This means that if you execute a query at time T1 and re-execute it at time T2,
- additional rows may have been added to the database, which will affect your results. This
- differs from the nonrepeatable read in that with a phantom read, data you already read has not
- been changed, but rather that more data satisfies your query criteria than before.

Queries ?