



DATABASE SYSTEMS

Design,
Implementation,
and Management
12e

Carlos Coronel | Steven Morris

Chapter 10

Transaction Management and Concurrency Control

Learning Objectives

- In this chapter, you will learn:
 - About database transactions and their properties
 - What concurrency control is and what role it plays in maintaining the database's integrity
 - What locking methods are and how they work
 - How stamping methods are used for concurrency control
 - How optimistic methods are used for concurrency control
 - How database recovery management is used to maintain database integrity

Learning Objectives

- In this chapter, you will learn:
 - How stamping methods are used for concurrency control
 - How optimistic methods are used for concurrency control
 - How database recovery management is used to maintain database integrity

What is a Transaction?

- Logical unit of work that must be entirely completed or aborted
- Consists of:
 - SELECT statement
 - Series of related UPDATE statements
 - Series of INSERT statements
 - Combination of SELECT, UPDATE, and INSERT statements

What is a Transaction?

- **Consistent database state:** All data integrity constraints are satisfied
 - Must begin with the database in a known consistent state to ensure consistency
- Formed by two or more database requests
 - **Database requests:** Equivalent of a single SQL statement in an application program or transaction

Evaluating Transaction Results

- Not all transactions update database
 - SQL code represents a transaction because it accesses a database
- Improper or incomplete transactions can have devastating effect on database integrity
 - Users can define enforceable constraints based on business rules
 - Other integrity rules are automatically enforced by the DBMS

Transaction Properties

Atomicity

- All operations of a transaction must be completed
 - If not, the transaction is aborted

Consistency

- Permanence of database's consistent state

Isolation

- Data used during transaction cannot be used by second transaction until the first is completed

Durability

- Ensures that once transactions are committed, they cannot be undone or lost

Serializability

- Ensures that the schedule for the concurrent execution of several transactions should yield consistent results

Transaction Management with SQL

- SQL statements that provide transaction support
 - COMMIT
 - ROLLBACK
- Transaction sequence must continue until:
 - COMMIT statement is reached
 - ROLLBACK statement is reached
 - End of program is reached
 - Program is abnormally terminated

The Transaction Log

- Keeps track of all transactions that update the database
- DBMS uses the information stored in a log for:
 - Recovery requirement triggered by a ROLLBACK statement
 - A program's abnormal termination
 - A system failure

Table 10.1 – A Transaction Log

TABLE 10.1

A TRANSACTION LOG

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	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				



TRL_ID = Transaction log record ID

TRX_NUM = Transaction number

PTR = Pointer to a transaction log record ID

(Note: The transaction number is automatically assigned by the DBMS.)

Concurrency Control

- Coordination of the simultaneous transactions execution in a multiuser database system
- Objective - Ensures serializability of transactions in a multiuser database environment

Problems in Concurrency Control

Lost update

- Occurs in two concurrent transactions when:
 - Same data element is updated
 - One of the updates is lost

Uncommitted data

- Occurs when:
 - Two transactions are executed concurrently
 - First transaction is rolled back after the second transaction has already accessed uncommitted data

Inconsistent retrievals

- Occurs when a transaction accesses data before and after one or more other transactions finish working with such data

Consider the following pair of transactions, starting with 35 units of an item: purchase 100 units, then sell 30 units:

TRANSACTION	COMPUTATION
T1: Purchase 100 units	$\text{PROD_QOH} = \text{PROD_QOH} + 100$
T2: Sell 30 units	$\text{PROD_QOH} = \text{PROD_QOH} - 30$

Cengage Learning © 2015

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	$\text{PROD_QOH} = 35 + 100$	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	$\text{PROD_QOH} = 135 - 30$	
6	T2	Write PROD_QOH	105

Cengage Learning © 2015

'Lost update' problem:

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	$\text{PROD_QOH} = 35 + 100$	
4	T2	$\text{PROD_QOH} = 35 - 30$	
5	T1	Write PROD_QOH (lost update)	135
6	T2	Write PROD_QOH	5

Cengage Learning © 2015

Consider this rollback situation:

TRANSACTION	COMPUTATION
T1: Purchase 100 units	$\text{PROD_QOH} = \text{PROD_QOH} + 100$ (Rolled back)
T2: Sell 30 units	$\text{PROD_QOH} = \text{PROD_QOH} - 30$

Cengage Learning © 2015

Proper rollback:

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 - 30	
7	T2	Write PROD_QOH	5

Cengage Learning © 2015

'Uncommitted data' problem (improper rollback):

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	$PROD_QOH = 35 + 100$	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	$PROD_QOH = 135 - 30$	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Cengage Learning © 2015

Consider the following fix (of a typo made earlier):

TRANSACTION T1	TRANSACTION T2
SELECT SUM(PROD_QOH) FROM PRODUCT	UPDATE PRODUCT SET PROD_QOH = PROD_QOH + 10 WHERE PROD_CODE = 1546-QQ2
	UPDATE PRODUCT SET PROD_QOH = PROD_QOH - 10 WHERE PROD_CODE = 1558-QW1
	COMMIT;

Cengage Learning © 2015

Here, T1 is a transaction that sums up QOH; T2 is a 'correction' transaction.

Proper retrieval of modified data:

PROD_CODE	BEFORE	AFTER
	PROD_QOH	PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QW1	23	(23 – 10) → 13
2232-QTY	8	8
2232-QWE	6	6
Total	92	92

Cengage Learning © 2015

'Inconsistent retrievals' problem (improper retrieval of modified data):

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 - 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

Cengage Learning © 2015

T1 should be doing $65+13$ (which would be correct), but instead does $65+23$ (which makes it incorrect).

The Scheduler

- Establishes the order in which the operations are executed within concurrent transactions
 - Interleaves the execution of database operations to ensure serializability and isolation of transactions
- Based on concurrent control algorithms to determine the appropriate order
- Creates serialization schedule
 - **Serializable schedule:** Interleaved execution of transactions yields the same results as the serial execution of the transactions

Concurrency Control with Locking Methods

- Locking methods - Facilitate isolation of data items used in concurrently executing transactions
- **Lock:** Guarantees exclusive use of a data item to a current transaction
- **Pessimistic locking:** Use of locks based on the assumption that conflict between transactions is likely
- **Lock manager:** Responsible for assigning and policing the locks used by the transactions

Lock Granularity

- Indicates the level of lock use
- Levels of locking
 - **Database-level lock**
 - **Table-level lock**
 - **Page-level lock**
 - **Page or diskpage:** Directly addressable section of a disk
 - **Row-level lock**
 - **Field-level lock**

Figure 10.3 - Database-Level Locking Sequence

FIGURE 10.3 DATABASE-LEVEL LOCKING SEQUENCE

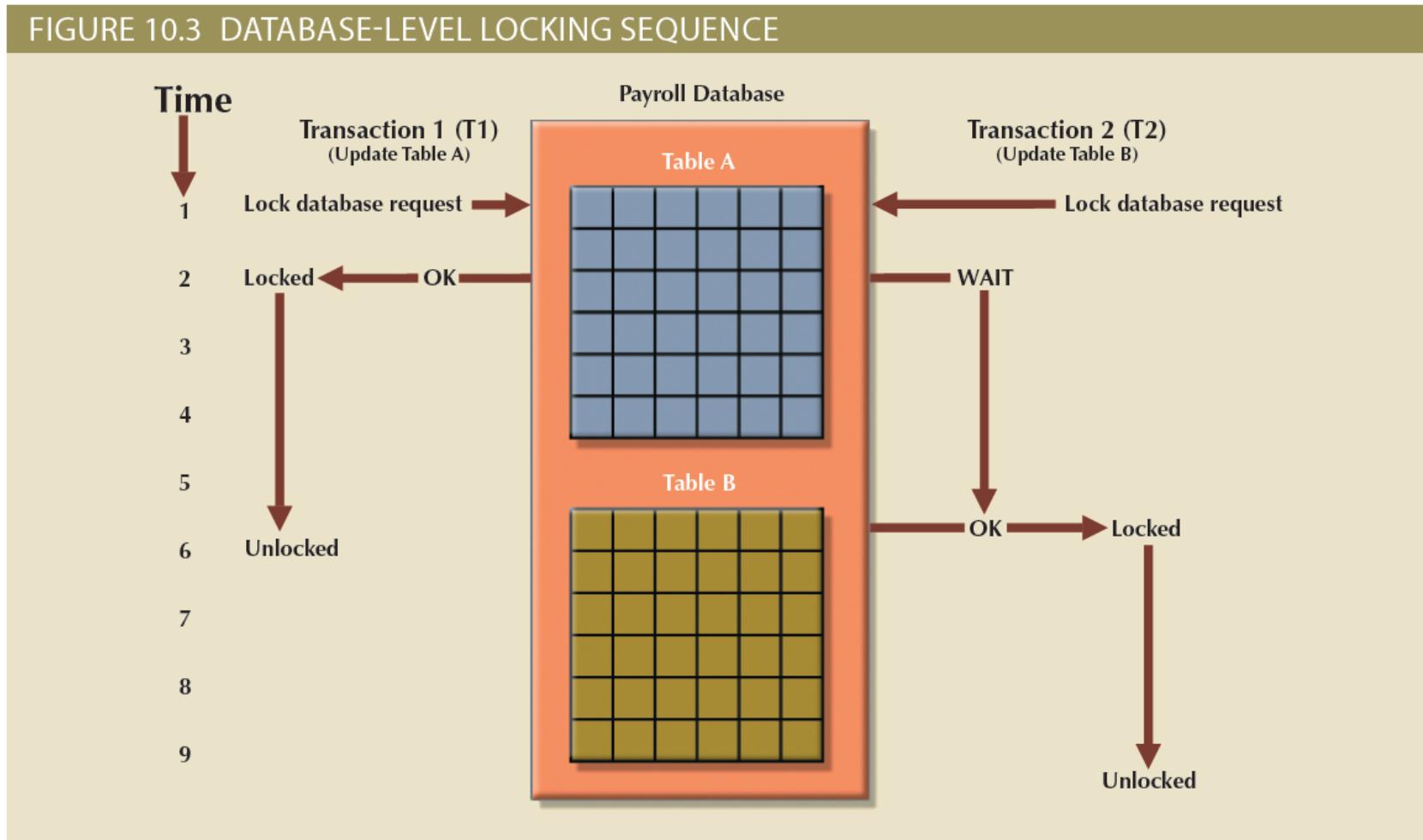


Figure 10.4 - An Example of a Table-Level Lock

FIGURE 10.4 AN EXAMPLE OF A TABLE-LEVEL LOCK

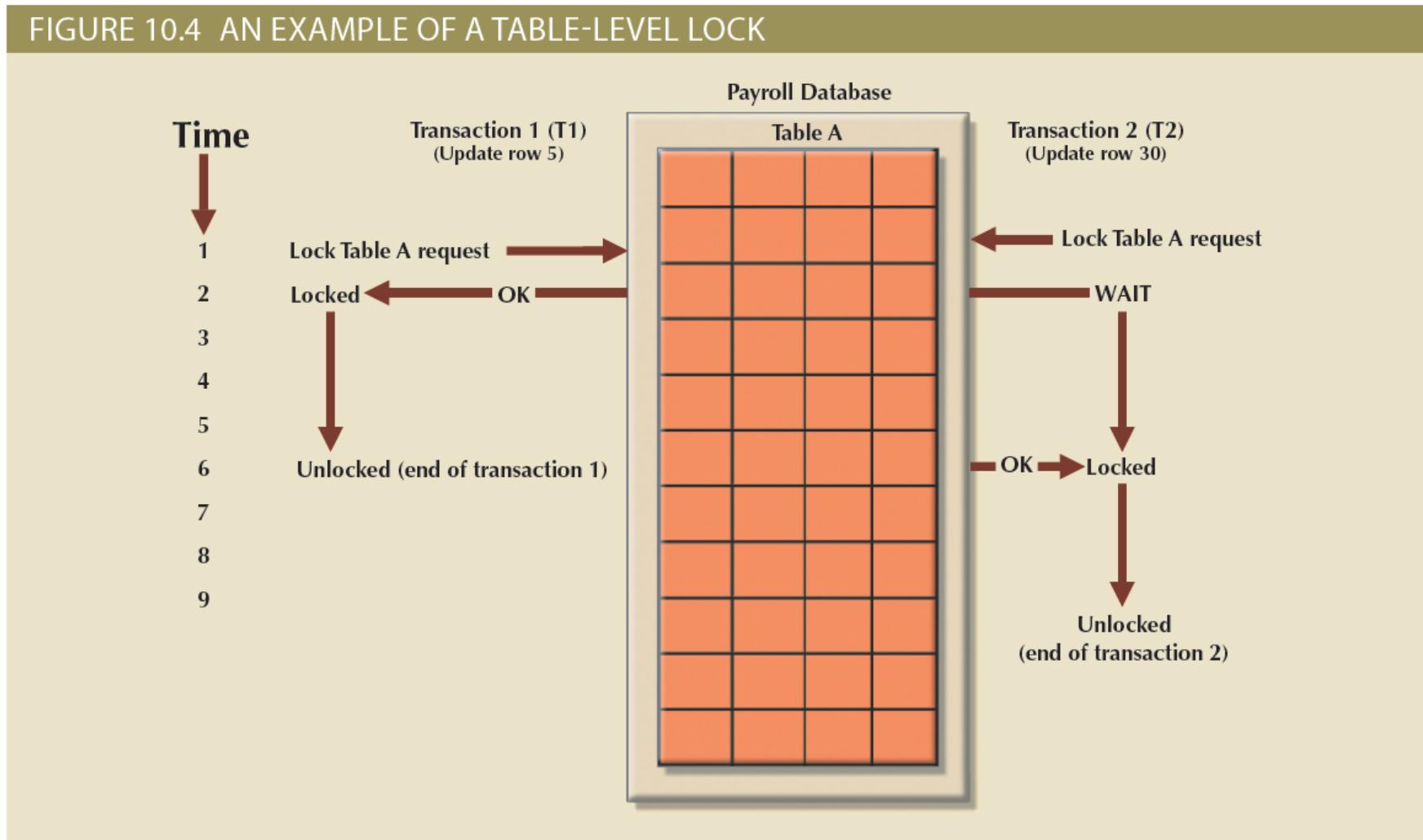


Figure 10.5 - An Example of a Page-Level Lock

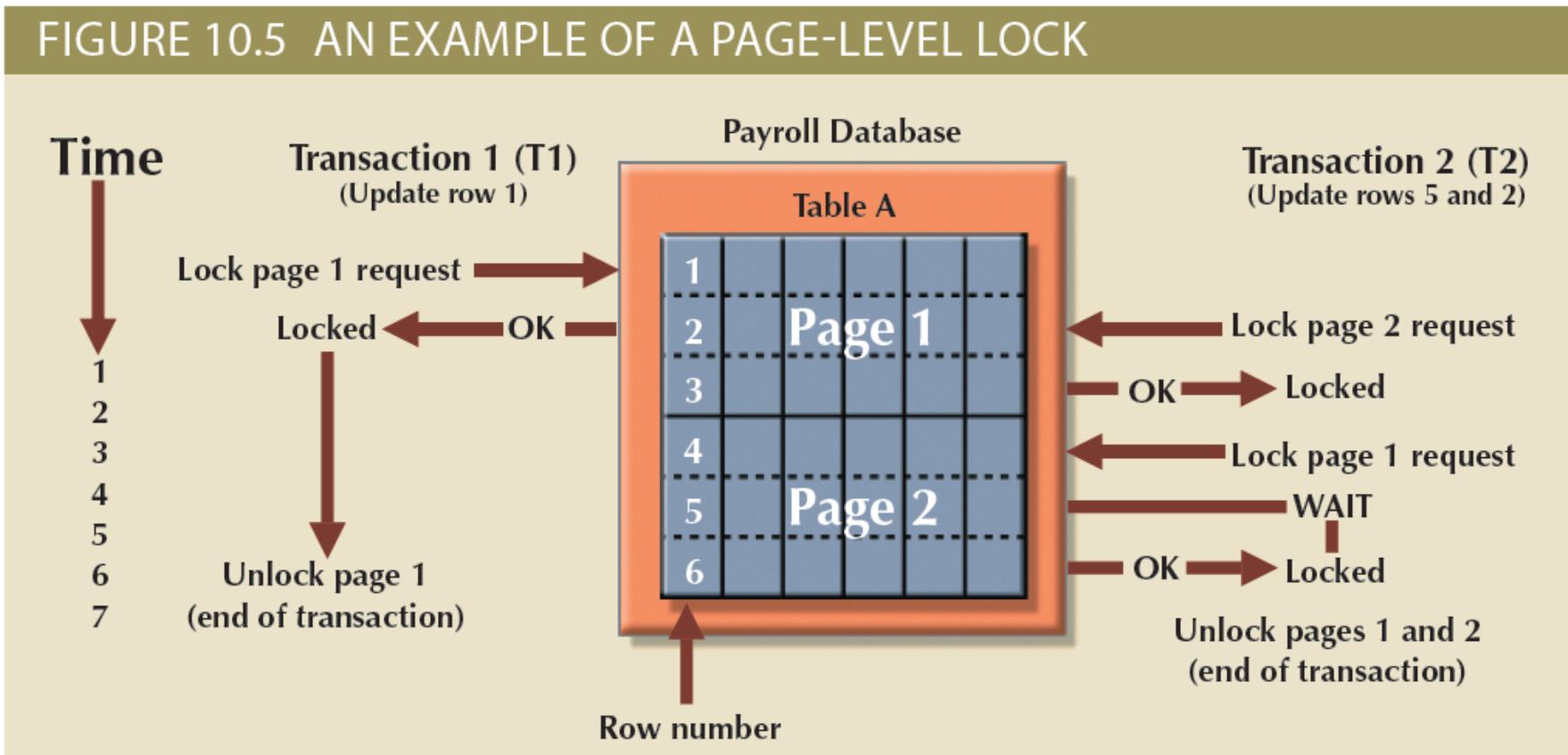
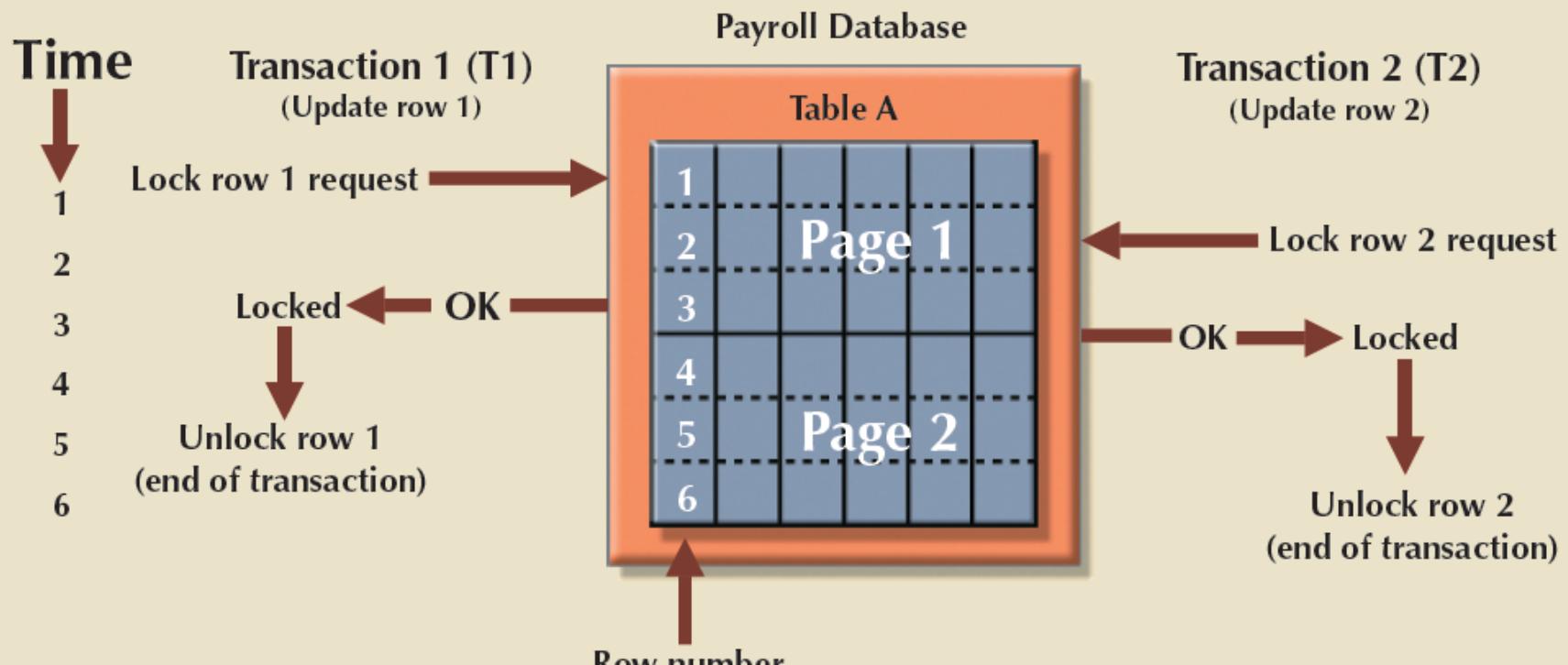


Figure 10.6 - An Example of a Row-Level Lock

FIGURE 10.6 AN EXAMPLE OF A ROW-LEVEL LOCK



Lock Types

Binary lock

- Has two states, locked (1) and unlocked (0)
 - If an object is locked by a transaction, no other transaction can use that object
 - If an object is unlocked, any transaction can lock the object for its use

Exclusive lock

- Exists when access is reserved for the transaction that locked the object

Shared lock

- Exists when concurrent transactions are granted read access on the basis of a common lock

Problems in Using Locks

- Resulting transaction schedule might not be serializable
- Schedule might create **deadlocks**

Two-Phase Locking (2PL)

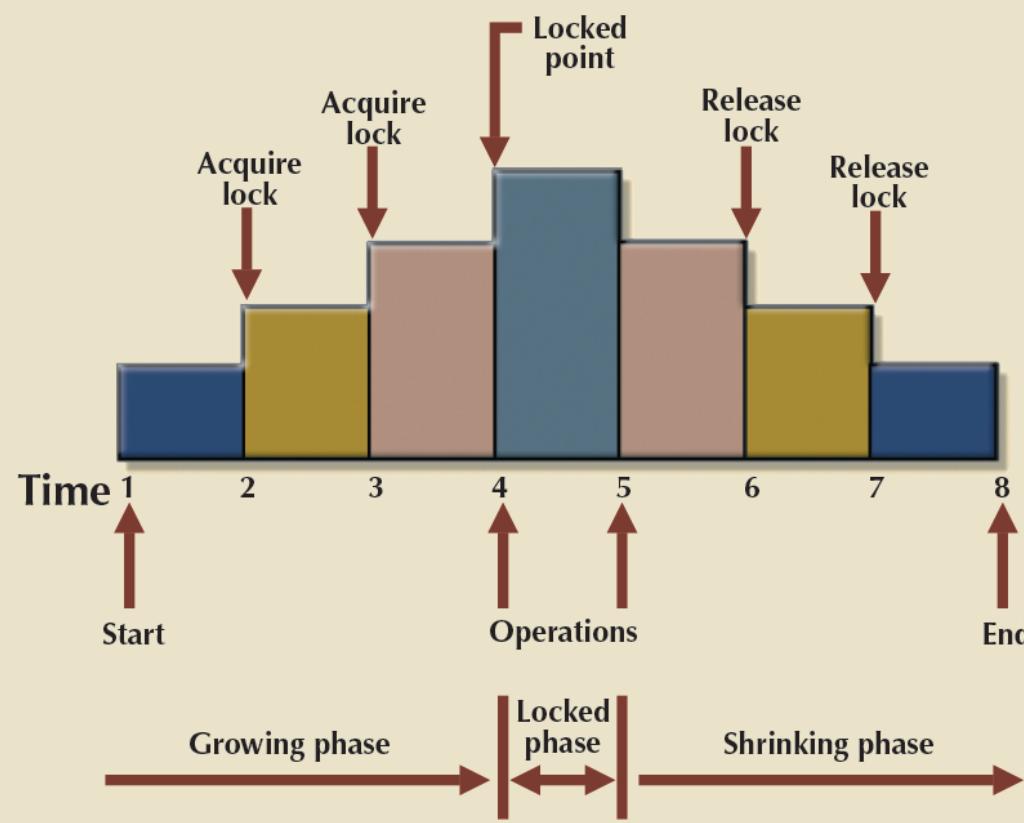
- Defines how transactions acquire and relinquish locks
- Guarantees serializability but does not prevent deadlocks
- Phases
 - Growing phase - Transaction acquires all required locks without unlocking any data
 - Shrinking phase - Transaction releases all locks and cannot obtain any new lock

Two-Phase Locking (2PL)

- Governing 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

Figure 10.7 - Two-Phase Locking Protocol

FIGURE 10.7 TWO-PHASE LOCKING PROTOCOL



Deadlocks

- Occurs when two transactions wait indefinitely for each other to unlock data
 - Known as **deadly embrace**
- Control techniques
 - Deadlock prevention
 - Deadlock detection
 - Deadlock avoidance
- Choice of deadlock control method depends on database environment

Table 10.13 - How a Deadlock Condition is Created

TABLE 10.13

HOW A DEADLOCK CONDITION IS CREATED

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

Deadlock



Time Stamping

- Assigns global, unique time stamp to each transaction
 - Produces explicit order in which transactions are submitted to DBMS
- Properties
 - **Uniqueness:** Ensures no equal time stamp values exist
 - **Monotonicity:** Ensures time stamp values always increases

Time Stamping

■ Disadvantages

- Each value stored in the database requires two additional stamp fields
- Increases memory needs
- Increases the database's processing overhead
- Demands a lot of system resources

Table 10.14 - Wait/Die and Wound/Wait Concurrency Control Schemes

TABLE 10.14

WAIT/DIE AND WOUND/WAIT CONCURRENCY CONTROL SCHEMES			
TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT/DIE SCHEME	WOUND/WAIT SCHEME
T1 (11548789)	T2 (19562545)	<ul style="list-style-type: none">• T1 waits until T2 is completed and T2 releases its locks.	<ul style="list-style-type: none">• T1 preempts (rolls back) T2.• T2 is rescheduled using the same timestamp.
T2 (19562545)	T1 (11548789)	<ul style="list-style-type: none">• T2 dies (rolls back).• T2 is rescheduled using the same timestamp.	<ul style="list-style-type: none">• 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 of Optimistic Approach

Read

- Transaction:
 - Reads the database
 - Executes the needed computations
 - Makes the updates to a private copy of the database values

Validation

- Transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database

Write

- Changes are permanently applied to the database

Table 10.15 - Transaction Isolation Levels

TRANSACTION ISOLATION LEVELS						
	ISOLATION LEVEL	ALLOWED			COMMENT	
		DIRTY READ	NONREPEATABLE READ	PHANTOM READ		
Less restrictive 	Read Uncommitted	Y	Y	Y	The transaction reads uncommitted data, allows nonrepeatable reads and, phantom reads.	
	Read Committed	N	Y	Y	Does not allow uncommitted data reads but allows nonrepeatable reads and phantom reads.	
	Repeatable Read	N	N	Y	Only allows phantom reads.	
	Serializable	N	N	N	Does not allow dirty reads, nonrepeatable reads, or phantom reads.	
Oracle / SQL Server Only	Read Only / Snapshot	N	N	N	Supported by Oracle and SQL Server. The transaction can only see the changes that were committed at the time the transaction started.	

Database Recovery Management

- **Database recovery:** Restores database from a given state to a previously consistent state
- Recovery transactions are based on the atomic transaction property
 - **Atomic transaction property:** All portions of a transaction must be treated as a single logical unit of work
 - If transaction operation cannot be completed:
 - Transaction must be aborted
 - Changes to database must be rolled back

Concepts that Affect Transaction Recovery

Write-ahead log protocol

- Ensures that transaction logs are always written before the data are updated

Redundant transaction logs

- Ensure that a physical disk failure will not impair the DBMS's ability to recover data

Buffers

- Temporary storage areas in a primary memory

Checkpoints

- Allows DBMS to write all its updated buffers in memory to disk

Techniques Used in Transaction Recovery Procedures

Deferred-write technique or deferred update

- Only transaction log is updated

Write-through technique or immediate update

- Database is immediately updated by transaction operations during transaction's execution

Recovery Process in Deferred-Write Technique

- Identify the last check point in the transaction log
- If transaction was committed before the last check point
 - Nothing needs to be done
- If transaction was committed after the last check point
 - Transaction log is used to redo the transaction
- If transaction had a ROLLBACK operation after the last check point
 - Nothing needs to be done

Recovery Process in Write-Through Technique

- Identify the last checkpoint in the transaction log
- If transaction was committed before the last check point
 - Nothing needs to be done
- If transaction was committed after the last checkpoint
 - Transaction must be redone
- If transaction had a ROLLBACK operation after the last check point
 - Transaction log is used to ROLLBACK the operations