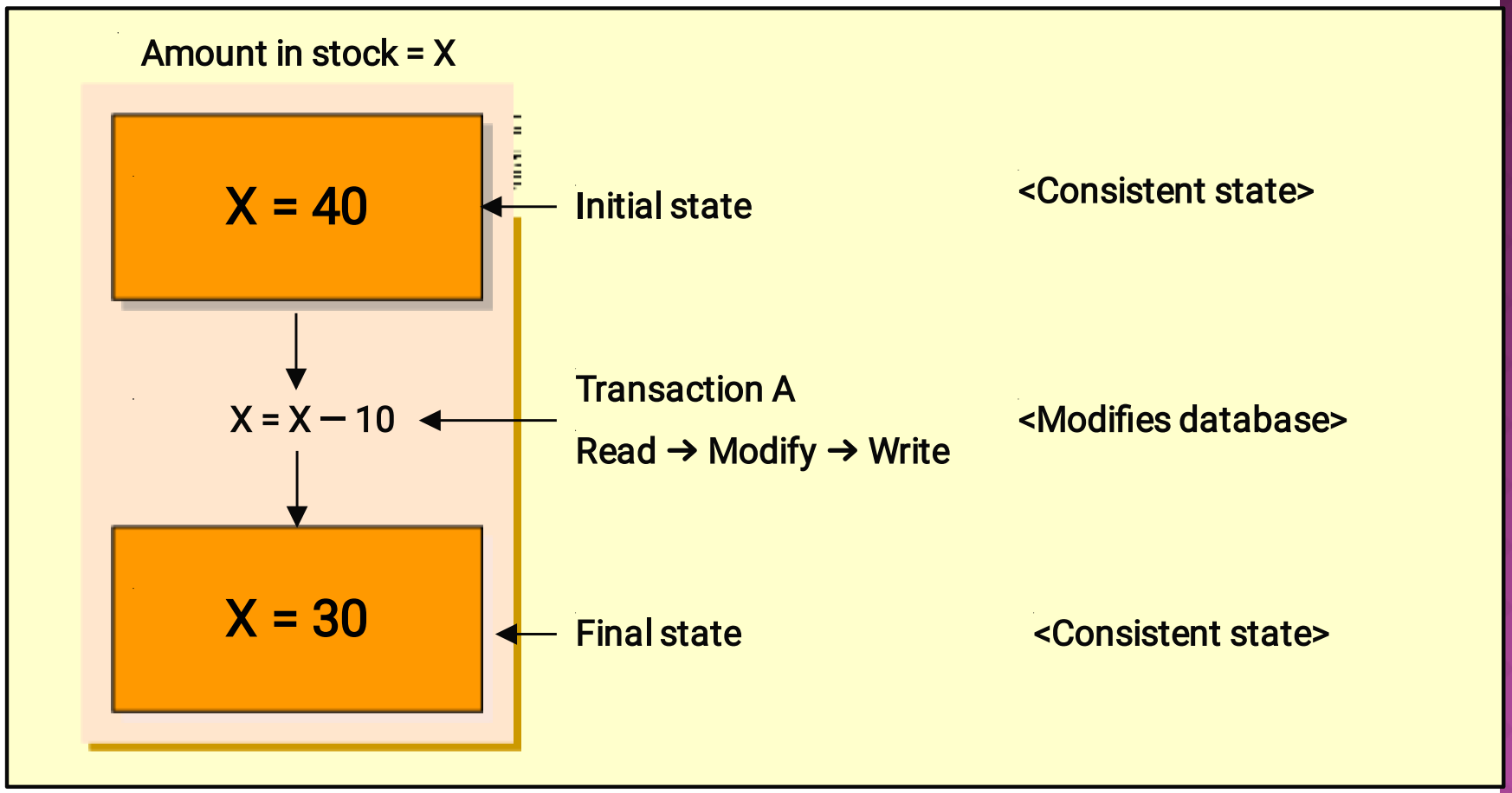
TRANSACTIONS AND CONCURRENCY CONTROL

WHAT IS A TRANSACTION?

- Logical unit of work that must be either entirely completed or aborted
- Successful transaction changes 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
  - Equivalent of a single SQL statement in an application program or transaction

TRANSACTION - EXAMPLE



EVALUATING TRANSACTION RESULTS

- Not all transactions update database
- SQL code represents a transaction because database was accessed
- Improper or incomplete transactions can have devastating effect on database integrity
  - User can define enforceable constraints based on business rules
  - Entity and referential integrity rules are enforced automatically by the DBMS

TRANSACTION PROPERTIES

- Atomicity
  - All operations of a transaction must be completed
- Consistency
  - Permanence of database's consistent state
- Isolation
  - Data used during transaction cannot be used by second transaction until the first is completed

TRANSACTION PROPERTIES (CONT'D.)

- Durability
  - Once transactions are committed, they cannot be undone
- Serializability
  - Concurrent execution of several transactions yields consistent results
- Multiuser databases are subject to multiple concurrent transactions

TRANSACTION MANAGEMENT WITH SQL

- ANSI has defined standards that govern SQL database transactions
- Transaction support is provided by two SQL statements: COMMIT and 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

- Transaction log stores:
  - A record for the beginning of transaction
  - For each transaction component:
    - Type of operation being performed (update, delete, insert)
    - Names of objects affected by transaction
    - “Before” and “after” values for updated fields
    - Pointers to previous and next transaction log entries for the same transaction
  - Ending (COMMIT) of the transaction

TRANSACTION LOG - EXAMPLE

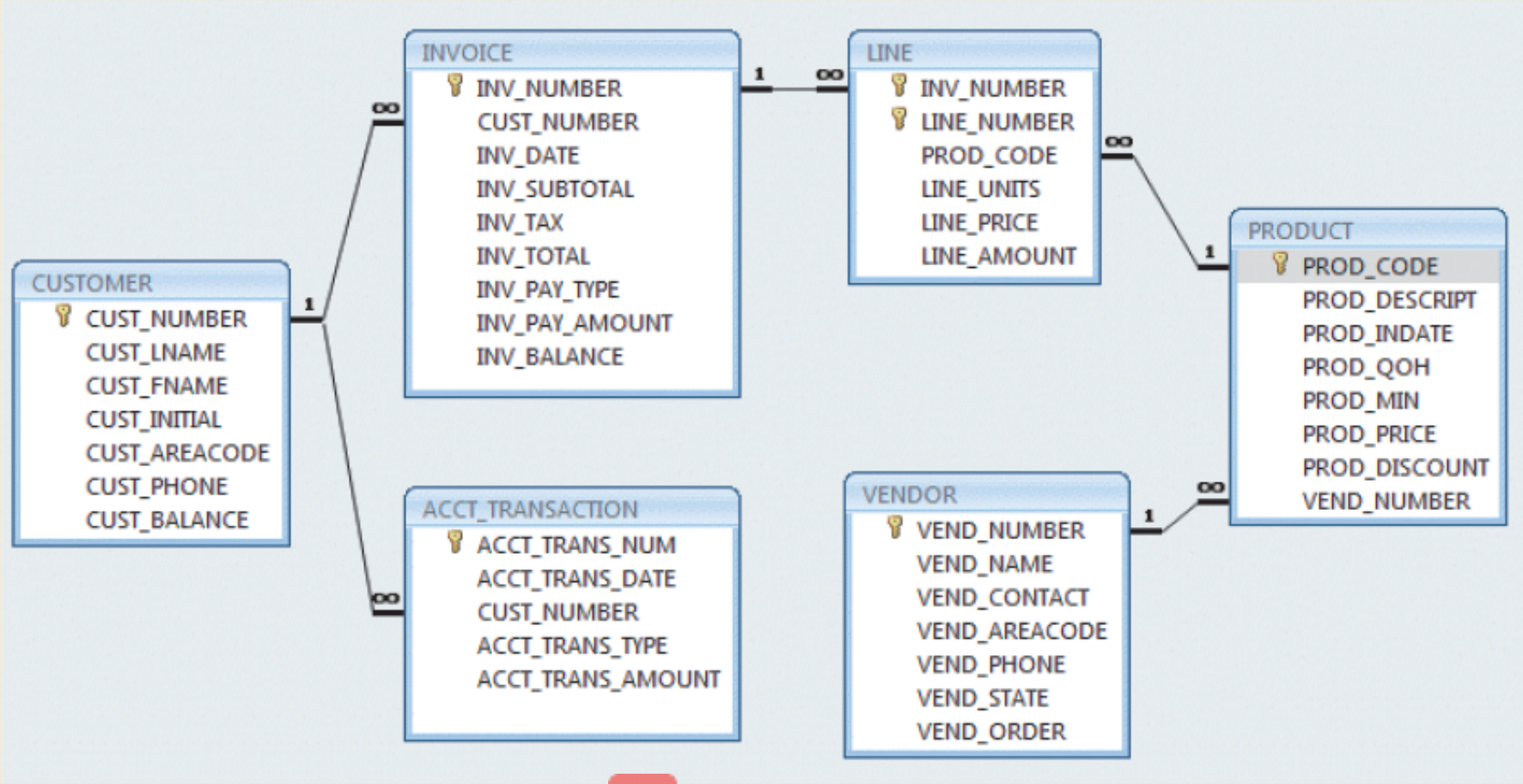
TABLE 13.1 A Transaction Log

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.)

SAMPLE DATABASE FOR DISCUSSION

FIGURE 13.1 The Ch13_SaleCo database relational diagram



SOURCE: Course Technology/Cengage Learning

CONCURRENCY CONTROL

- Coordination of simultaneous transaction execution in a multiprocessing database
- Objective is to ensure serializability of transactions in a multiuser environment
- Three main problems:
  - Lost updates
  - Uncommitted data
  - Inconsistent retrievals

LOST UPDATES

- Lost update problem:
  - Two concurrent transactions update same data element
  - One of the updates is lost
    - Overwritten by the other transaction

LOST UPDATE - EXAMPLE

TABLE 13.2 Two Concurrent Transactions to Update QOH

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

TABLE 13.3 Serial Execution of Two Transactions

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	135
5	T2	PROD_QOH = 135 – 30	
6	T2	Write PROD_QOH	105

LOST UPDATE - EXAMPLE (CONT')

TABLE 13.4 Lost Updates

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 – 30	
5	T1	Write PROD_QOH (Lost update)	135
6	T2	Write PROD_QOH	5

Another way of illustrating lost update for concurrent transaction T1 and T2:

Time	T1	T2	PROD_QOH
t1	begin_transaction		35
t2	read(PROD_QOH)	begin_transaction	35
t3	PROD_QOH = PROD_QOH +100	read(PROD_QOH)	35
t4	write(PROD_QOH)	PROD_QOH = PROD_QOH - 30	135
t5	COMMIT	write(PROD_QOH)	5
t6		COMMIT	5

UNCOMMITTED DATA

- Uncommitted data phenomenon:
  - Two transactions are executed concurrently
  - First transaction rolled back after second already accessed uncommitted data

UNCOMMITTED DATA - EXAMPLE

TABLE 13.5 Transactions Creating an Uncommitted Data Problem

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

Time	T1	T2	PROD_QOH
t1	begin_transaction		35
t2	read(PROD_QOH)		35
t3	PROD_QOH = PROD_QOH +100		35
t4	write(PROD_QOH)		135
t5	ROLLBACK	begin_transaction	35
t6		read(PROD_QOH)	35
t7		PROD_QOH = PROD_QOH - 30	35
t8		write(PROD_QOH)	5
T9		COMMIT	5

UNCOMMITTED DATA - EXAMPLE (CONT')

- Uncommitted data

Time	T1	T2	PROD_QOH
t1	begin_transaction		35
t2	read(PROD_QOH)		35
t3	PROD_QOH = PROD_QOH +100	begin_transaction	35
t4	write(PROD_QOH)	read(PROD_QOH)	135
t5	ROLLBACK	PROD_QOH = PROD_QOH - 30	135
t6		write(PROD_QOH)	105
t7		COMMIT	105

INCONSISTENT RETRIEVALS

- Inconsistent retrievals:
  - First transaction accesses data
  - Second transaction alters the data
  - First transaction accesses the data again
- Transaction might read some data before they are changed and other data after changed
- Yields inconsistent results

INCONSISTENT RETRIEVAL - EXAMPLE

TABLE 13.8 Retrieval During Update

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;

TABLE 13.9 Transaction Results: Data Entry Correction

	BEFORE	AFTER
PROD_CODE	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**

INCONSISTENT RETRIEVAL - EXAMPLE (CONT')

TABLE 13.10 Inconsistent Retrievals

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

THE SCHEDULER

- Special DBMS program
  - Purpose is to establish order of operations within which concurrent transactions are executed
- Interleaves execution of database operations:
  - Ensures serializability
  - Ensures isolation
- Serializable schedule
  - Interleaved execution of transactions yields same results as serial execution
- Two concurrent transactions are in conflict when
  - they access the same data, and
  - at least one of them executes WRITE operation

CONCURRENCY CONTROL WITH LOCKING METHODS

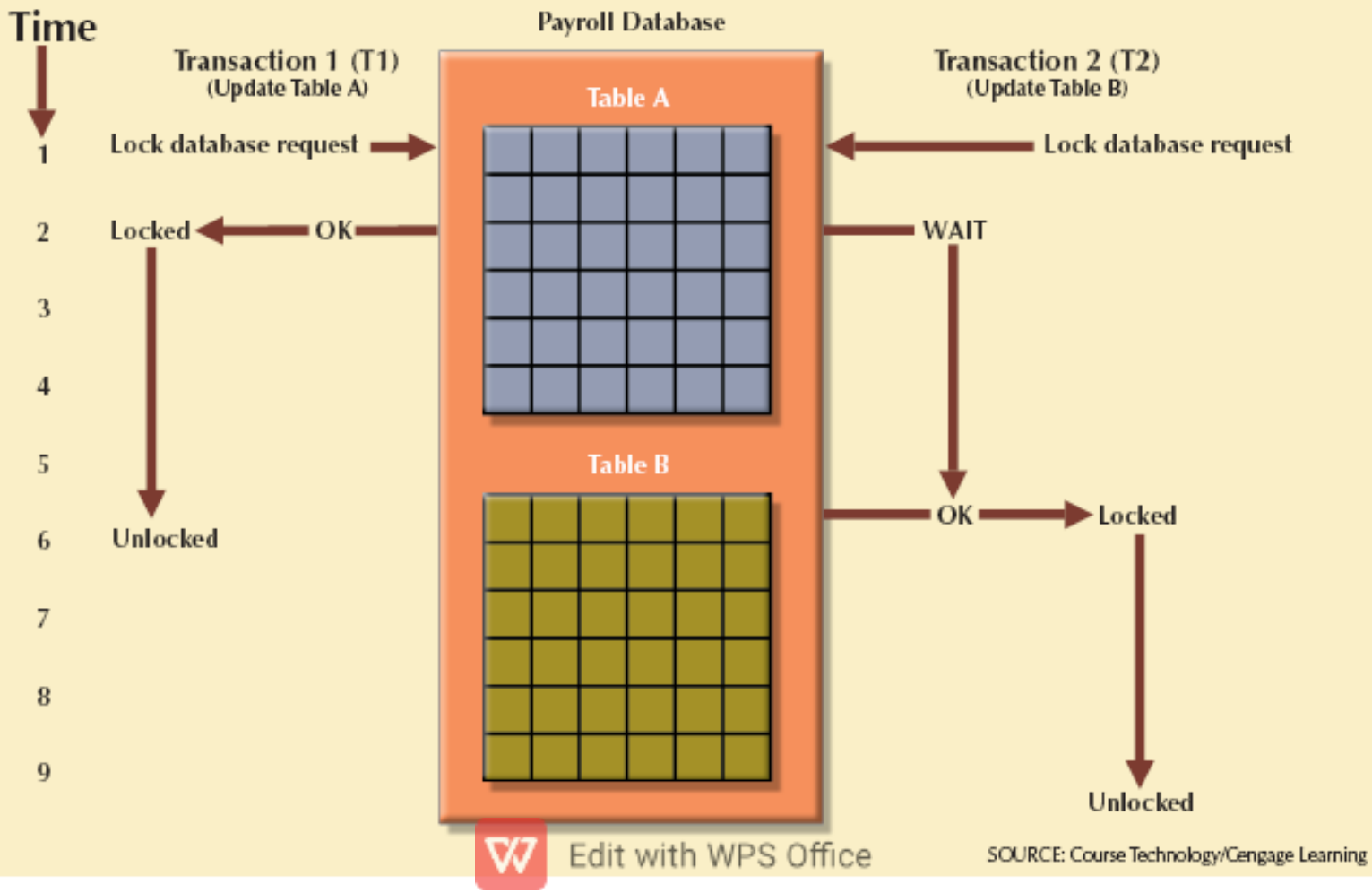
- Lock
  - Guarantees exclusive use of a data item to a current transaction
  - Required to prevent another transaction from reading inconsistent data
  - Pessimistic locking
    - Use of locks based on the assumption that conflict between transactions is likely
  - Lock manager
    - Responsible for assigning and policing the locks used by transactions

LOCK GRANULARITY

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

DATABASE-LEVEL LOCK

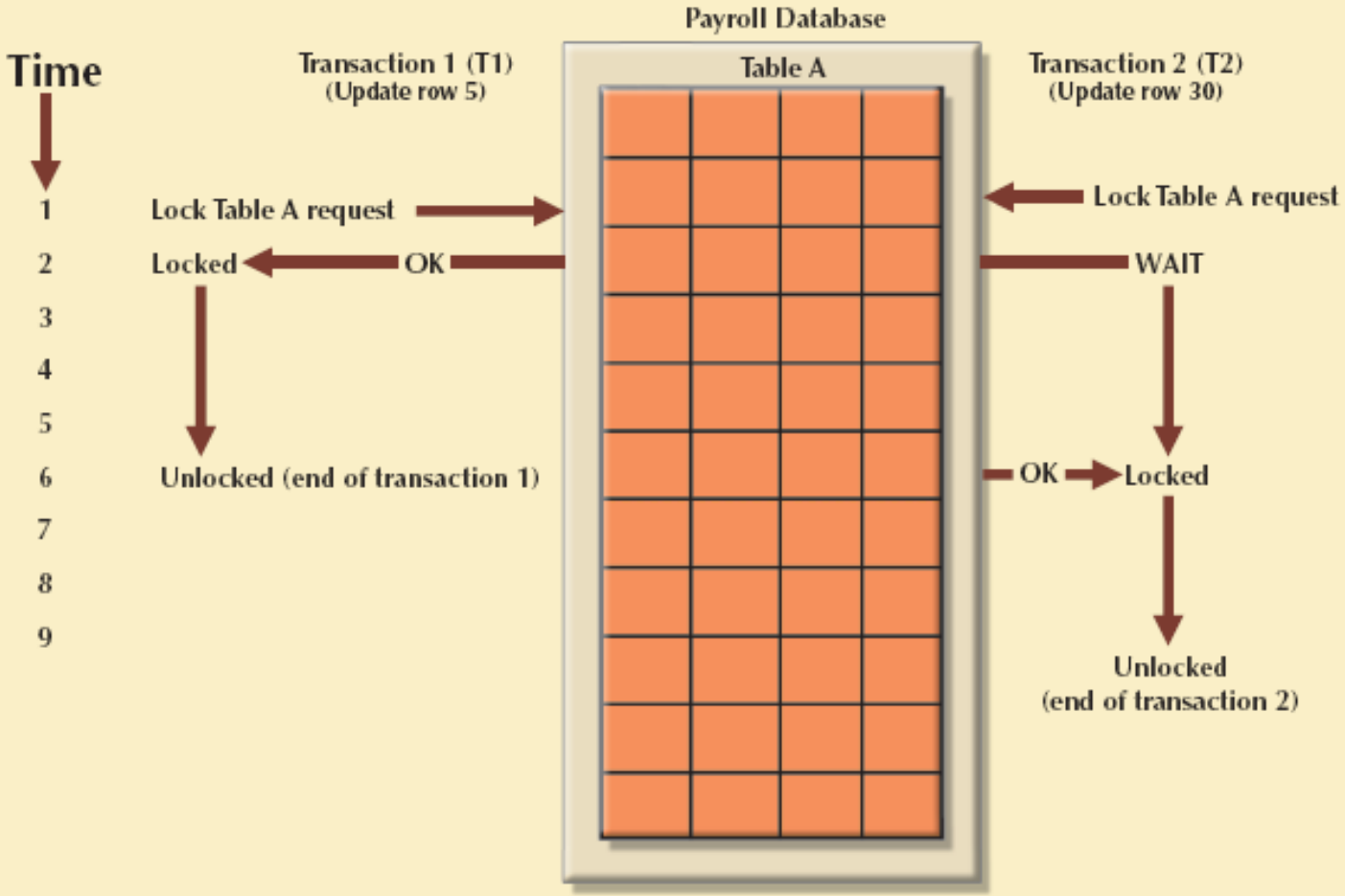
FIGURE 13.3 Database-level locking sequence



SOURCE: Course Technology/Cengage Learning

TABLE-LEVEL LOCK

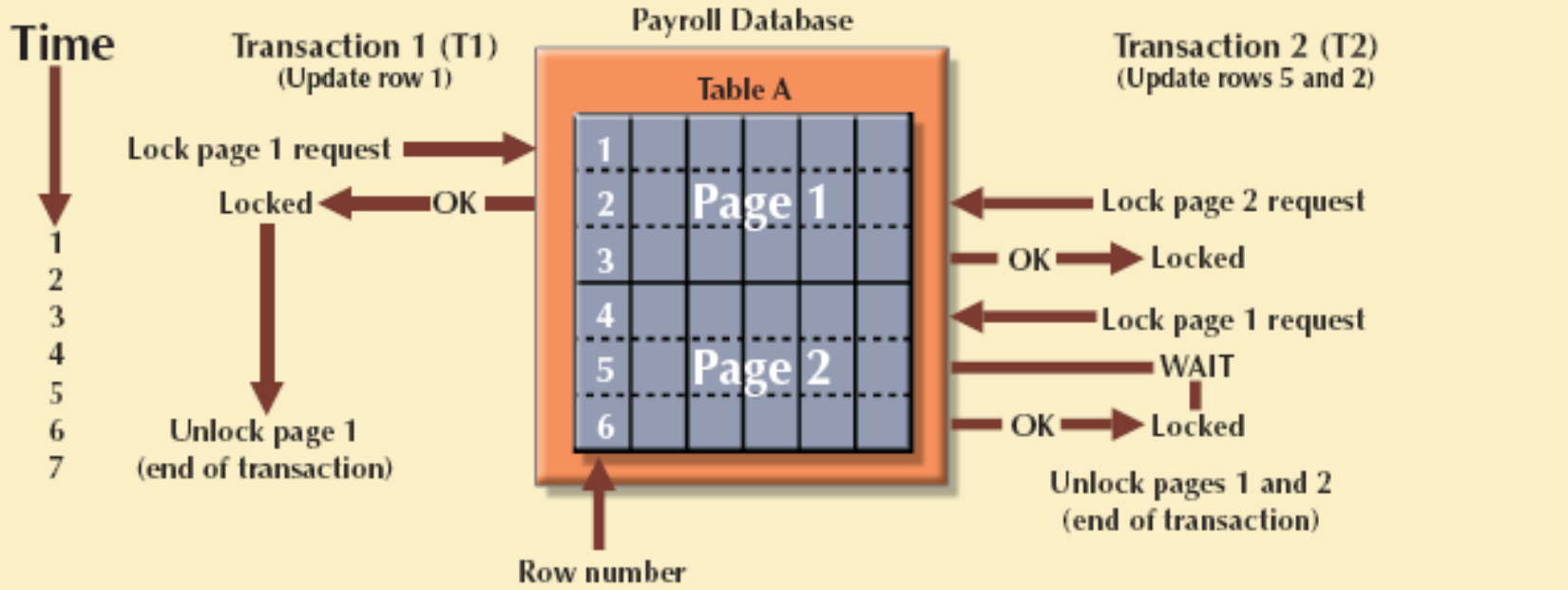
FIGURE 13.4 An example of a table-level lock



SOURCE: Course Technology/Cengage Learning

PAGE-LEVEL LOCK

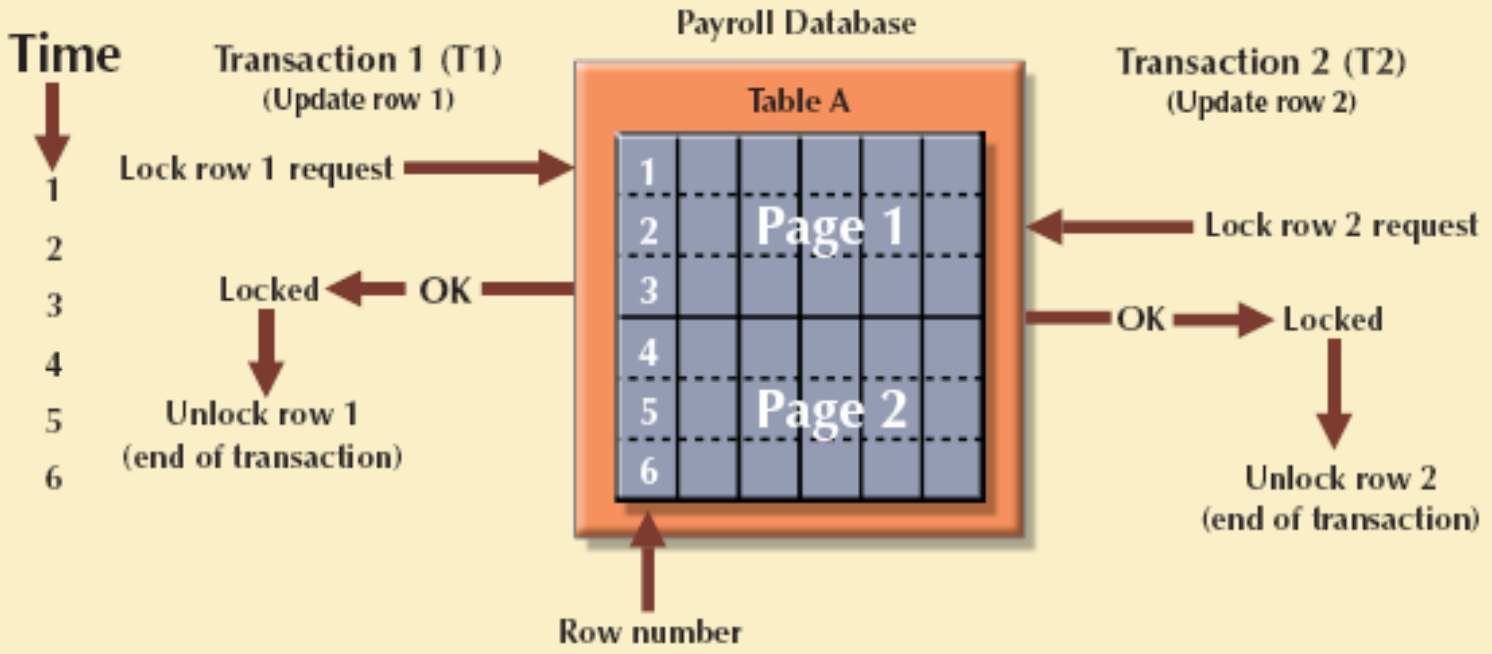
FIGURE 13.5 An example of a page-level lock



SOURCE: Course Technology/Cengage Learning

ROW-LEVEL LOCK

FIGURE 13.6 An example of a row-level lock



SOURCE: Course Technology/Cengage Learning

BINARY LOCKS

- Binary lock
  - Two states: locked (1) or unlocked (0)

TABLE 13.12 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	

SHARED/EXCLUSIVE LOCKS

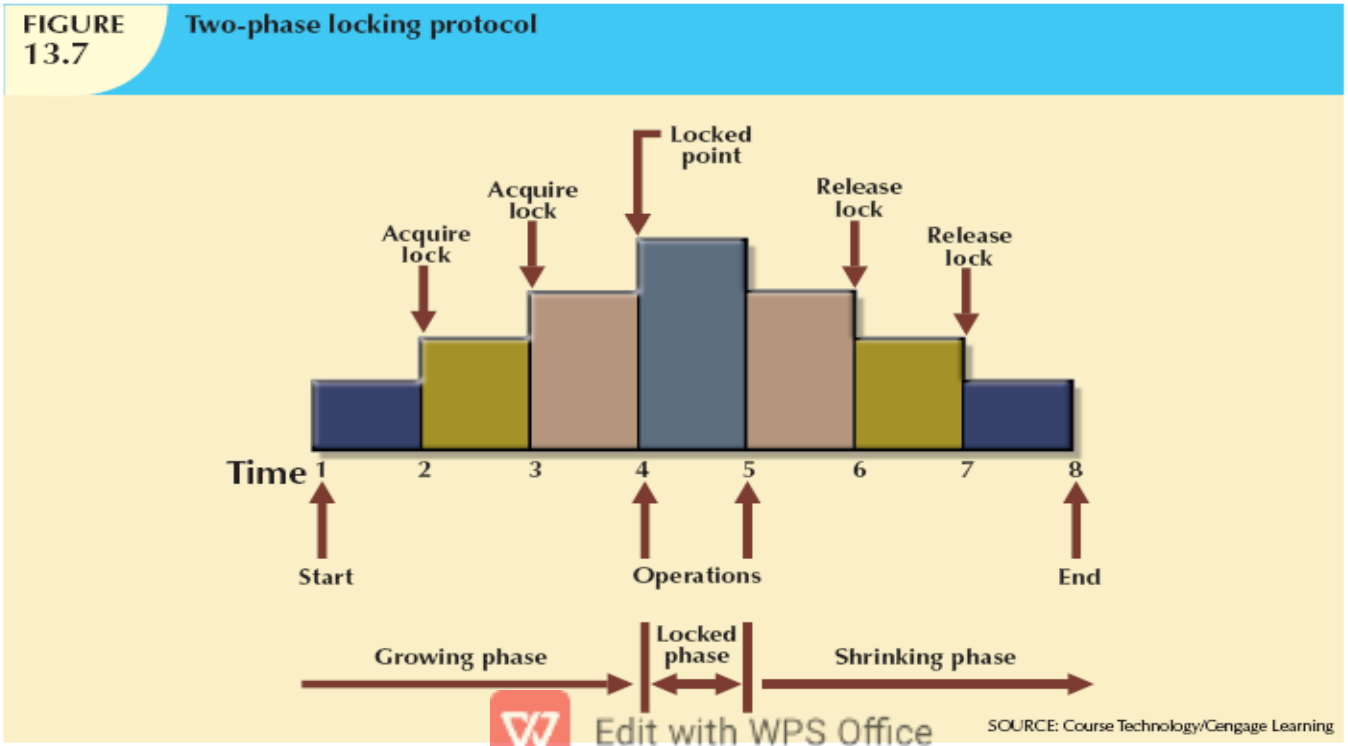
- Exclusive lock
  - Access is specifically reserved for transaction that locked object
  - Must be used when potential for conflict exists (when a transactions wants to write data)
- Shared lock
  - Issued when a transaction wants to read data and no exclusive lock is held on that data item
  - Concurrent transactions are granted read access on basis of a common lock
  - No conflict as long as all the concurrent transactions are read-only

TWO-PHASE LOCKING

- Defines how transactions acquire and relinquish locks
- Guarantees serializability, but does not prevent deadlocks
- Two 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 (CONT')

- 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



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
- Example:
  - T1 = access data items X then Y
  - T2 = access data items Y then X
  - T1 acquires the lock on X first and waits for T2 to release the lock on Y to commit the transaction
  - T2 acquires the lock on Y first and waits for T1 to release the lock on Y to commit the transaction

DEADLOCKS - EXAMPLE

TABLE 13.13 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	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
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....

DEADLOCKS (CONT')

- Three techniques to control deadlock:
  - Prevention
  - Detection
  - Avoidance
- Choice of deadlock control method depends on database environment
  - Low probability of deadlock; detection recommended
  - High probability; prevention recommended

CONCURRENCY CONTROL WITH TIME STAMPING METHODS

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

WAIT/DIE AND WOUND/WAIT SCHEMES

- Wait/die
  - Older transaction waits and younger is rolled back and rescheduled
- Wound/wait
  - Older transaction rolls back younger transaction and reschedules it

TABLE 13.14 Wait/Die and Wound/Wait Concurrency Control Schemes

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

DATABASE RECOVERY MANAGEMENT

- Restores database to previous consistent state
- Based on atomic transaction property
  - All portions of transaction are treated as single logical unit of work
  - All operations are applied and completed to produce consistent database
- If transaction operation cannot be completed:
  - Transaction aborted
  - Changes to database are rolled back

TRANSACTION RECOVERY

- Write-ahead-log protocol
  - Ensures transaction logs are written before data is updated
- Redundant transaction logs
  - Ensure physical disk failure will not impair ability to recover
- Buffers
  - Temporary storage areas in primary memory
- Checkpoints
  - Operations in which DBMS writes all its updated buffers to disk

TRANSACTION RECOVERY (CONT')

- Deferred-write technique
  - Only transaction log is updated
- Recovery process: identify last checkpoint
  - If transaction committed before checkpoint:
    - Do nothing
  - If transaction committed after checkpoint:
    - Use transaction log to redo the transaction
  - If transaction had ROLLBACK operation:
    - Do nothing

TRANSACTION RECOVERY (CONT)

- Write-through technique
  - Database is immediately updated by transaction operations during transaction's execution
- Recovery process: identify last checkpoint
  - If transaction committed before checkpoint:
    - Do nothing
  - If transaction committed after last checkpoint:
    - DBMS redoes the transaction using "after" values
  - If transaction had ROLLBACK or was left active:
    - DBMS undoes the operation using the "before" values in the transaction log

TRANSACTION RECOVERY - EXAMPLE

TABLE 13.15 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-2012, ...
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 database
  - Logical unit of work
    - No portion of transaction can exist by itself
  - Five main properties: atomicity, consistency, isolation, durability, and serializability
- COMMIT saves changes to disk
- ROLLBACK restores previous database state
- SQL transactions are formed by several SQL statements or database requests

SUMMARY (CONT'D.)

- Transaction log keeps track of all transactions that modify database
- Concurrency control coordinates simultaneous execution of transactions
- Scheduler establishes order in which concurrent transaction operations are executed
- Lock guarantees unique access to a data item by transaction
- Two types of locks: binary locks and shared/exclusive locks

SUMMARY (CONT'D.)

- Serializability of schedules is guaranteed through the use of two-phase locking
- Deadlock: when two or more transactions wait indefinitely for each other to release lock
- Three deadlock control techniques: prevention, detection, and avoidance
- Time stamping methods assign unique time stamp to each transaction
  - Schedules execution of conflicting transactions in time stamp order
- Database recovery restores database from given state to previous consistent state