Chapter 10

Transaction Management and Concurrency Control

CENGAGE

Learning Objectives (1 of 2)

- 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

Learning Objectives (2 of 2)

- 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? (1 of 2)

- 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? (2 of 2)

- **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

- Transactions are likely to contain many parts,
  - updating a customer's account,
  - adjusting product inventory, and
  - updating the seller's accounts receivable. All parts of a

- Transactions must be successfully completed to prevent data integrity problems.

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

Example

- On January 18, 2016, the credit sale of one unit of product 89-WRE-Q to customer 10016 for $277.55.

- The required transaction affects the
  - INVOICE,
  - LINE,
  - PRODUCT,
  - CUSTOMER, and
  - ACCT_TRANSACTION tables.

```
INSERT INTO INVOICE
        VALUES (1009, 10016,'18-Jan-2016', 256.99, 20.56, 277.55, 'cred', 0.00, 277.55);
INSERT INTO LINE
        VALUES (1009, 1, '89-WRE-Q', 1, 256.99, 256.99);

UPDATE  PRODUCT
SET     PROD_QOH = PROD_QOH – 1
WHERE   PROD_CODE = '89-WRE-Q';

UPDATE  CUSTOMER
SET     CUST_BALANCE = CUST_BALANCE + 277.55
WHERE   CUST_NUMBER = 10016;

INSERT INTO ACCT_TRANSACTION
        VALUES (10007, '18-Jan-16', 10016, 'charge', 277.55);
COMMIT;
```

FIGURE 10.2 TRACING THE TRANSACTION IN THE CH10_SALECO DATABASE

Database name: Ch10_SaleCo

Table name: INVOICE

INV_NUMBER	CUST_NUMBER	INV_DATE	INV_SUBTOTAL	INV_TAX	INV_TOTAL	INV_PAY_TYPE	INV_PAY_AMOUNT	INV_BALANCE
1001	10014	16-Jan-16	54.92	4.39	59.31	cc	59.31	0.00
1002	10011	16-Jan-16	9.98	0.80	10.78	cash	10.78	0.00
1003	10012	16-Jan-16	270.70	21.66	292.36	cc	292.36	0.00
1004	10011	17-Jan-16	34.87	2.79	37.66	cc	37.66	0.00
1005	10018	17-Jan-16	70.44	5.64	76.08	cc	76.08	0.00
1006	10014	17-Jan-16	397.83	31.83	429.66	cred	100.00	329.66
1007	10015	17-Jan-16	34.97	2.80	37.77	chk	37.77	0.00
1008	10011	17-Jan-16	1033.08	82.65	1115.73	cred	500.00	615.73
1009	10016	18-Jan-16	256.99	20.56	277.55	cred	0.00	277.55

Table name: LINE

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE	LINE_AMOUNT
1001	1	13-Q2/P2	3	14.99	44.97
1001	2	23109-HB	1	9.95	9.95
1002	1	54778-2T	2	4.99	9.98
1003	1	2238/QPD	4	38.95	155.80
1003	2	1546-QQ2	1	39.95	39.95
1003	3	13-Q2/P2	5	14.99	74.95
1004	1	54778-2T	3	4.99	14.97
1004	2	23109-HB	2	9.95	19.90
1005	1	PVC23DRT	12	5.87	70.44
1006	1	SM-18277	3	6.99	20.97
1006	2	2232/QTY	1	109.92	109.92
1006	3	23109-HB	1	9.95	9.95
1006	4	89-WRE-Q	1	256.99	256.99
1007	1	13-Q2/P2	2	14.99	29.98
1007	2	54778-2T	1	4.99	4.99
1008	1	PVC23DRT	5	5.87	29.35
1008	2	WR3/TT3	4	119.95	479.80
1008	3	23109-HB	1	9.95	9.95
1008	4	89-WRE-Q	2	256.99	513.98
1009	1	89-WRE-Q	1	256.99	256.99

Table name: PRODUCT

PROD_CODE	PROD_DESCRIPT	PROD_INDATE	PROD_QOH	PROD_MIN	PROD_PRICE	PROD_DISCOUNT	VEND_NUMBER
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-15	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-15	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-15	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-16	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-16	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-15	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-15	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-16	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-16	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-16	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-15	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Jan-16	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	06-Jan-16	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-16	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-16	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-16	18	5	119.95	0.10	25595

Table name: CUSTOMER

CUST_NUMB	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE	CUST_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	615.73
10012	Smith	Kathy	W	615	894-2285	0.00
10013	Olowski	Paul	F	615	894-2180	0.00
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	277.55
10017	Williams	George		615	290-2556	0.00
10018	Farriss	Anne	G	713	382-7185	0.00
10019	Smith	Olette	K	615	297-3809	0.00

Table name: ACCT_TRANSACTION

ACCT_TRANS_NUM	ACCT_TRANS_DATE	CUST_NUMBER	ACCT_TRANS_TYPE	ACCT_TRANS_AMOUNT
10003	17-Jan-16	10014	charge	329.66
10004	17-Jan-16	10011	charge	615.73
10006	29-Jan-16	10014	payment	329.66
10007	18-Jan-16	10016	charge	277.55

A scenario…

- The DBMS completes the first three SQL statements.
- During the execution of the fourth statement (the UPDATE of the
- CUSTOMER table's CUST_BALANCE value for customer 10016), the computer system loses electrical power.
  - If the computer does not have a backup power supply, the transaction cannot be completed.
- The INVOICE and LINE rows were added, and the PRODUCT table was updated to represent the sale of product 89-WRE-Q, but customer 10016 was not charged, nor was the required record written in the ACCT_TRANSACTION table.
- The database is now in an inconsistent state, and it is not usable for subsequent transactions.

Transaction Properties (ACID)

- Atomicity
  - All operations of a transaction must be completed
    - If not, the transaction is aborted
- Consistency
  - Permanence of database's consistent state
  - A transaction takes a database from one consistent state to another.
  - When a transaction is completed, the database must be in a consistent state.
  - If any of the transaction parts violates an integrity constraint, the entire transaction is aborted.
- Isolation
  - Data used during transaction cannot be used by second transaction until the first is completed

Transaction Properties (ACID)

- Durability
  - Ensures that once transactions are committed, they cannot be undone or lost
- Serializability
  - This property is important in multiuser and distributed databases in which multiple transactions are likely to be executed concurrently
  - For example, let's assume that the DBMS has three transactions (T1, T2 and T3) executing at the same time.
    - To properly carry out transactions, the DBMS must schedule the concurrent execution of the transaction's operations.
  - 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
    - Equivalent to a COMMIT
  - Program is abnormally terminated
    - Equivalent to a ROLLBACK

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

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.)

- The transaction log stores the following:
    - A record for the beginning of the transaction.
    - For each transaction component (SQL statement):
        - The type of operation being performed (INSERT, UPDATE, DELETE).
        - The names of the objects affected by the transaction (the name of the table).
        - The “before” and “after” values for the fields being updated.
        - Pointers to the previous and next transaction log entries for the same transaction.
- The ending (COMMIT) of the transaction.

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

Lost update

TABLE 10.2	
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

CENGAGE

Lost update

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

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 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

Uncommitted data

TABLE 10.5

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

Uncommitted data

CORRECT 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	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 – 30	
7	T2	Write PROD_QOH	5

Uncommitted data

CORRECT 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	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 – 30	
7	T2	Write PROD_QOH	5

AN UNCOMMITTED DATA PROBLEM			
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

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

- Why must we run database operations concurrently?
- Why not on a first-come-first serve basis.

TABLE 10.11

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

CENGAGE

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:** good for batch processing
  - **Table-level lock**
  - **Page-level lock:** suitable for multi-user DBMS
    - **Page** or **diskpage**: Directly addressable section of a disk
  - **Row-level lock**
  - **Field-level lock**

Figure 10.3 - Database-Level Locking Sequence

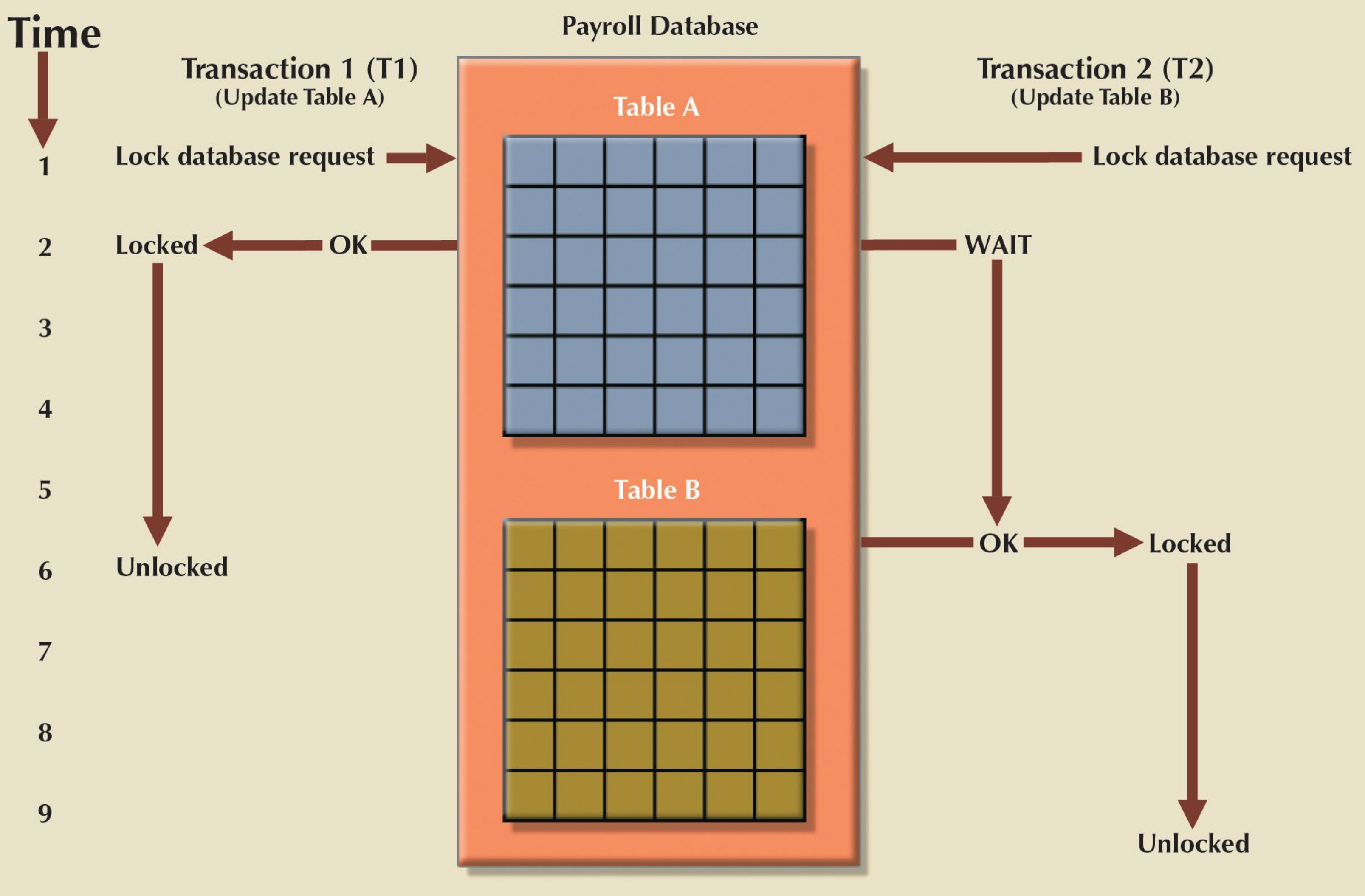


Figure 10.4 - An Example of a Table-Level Lock

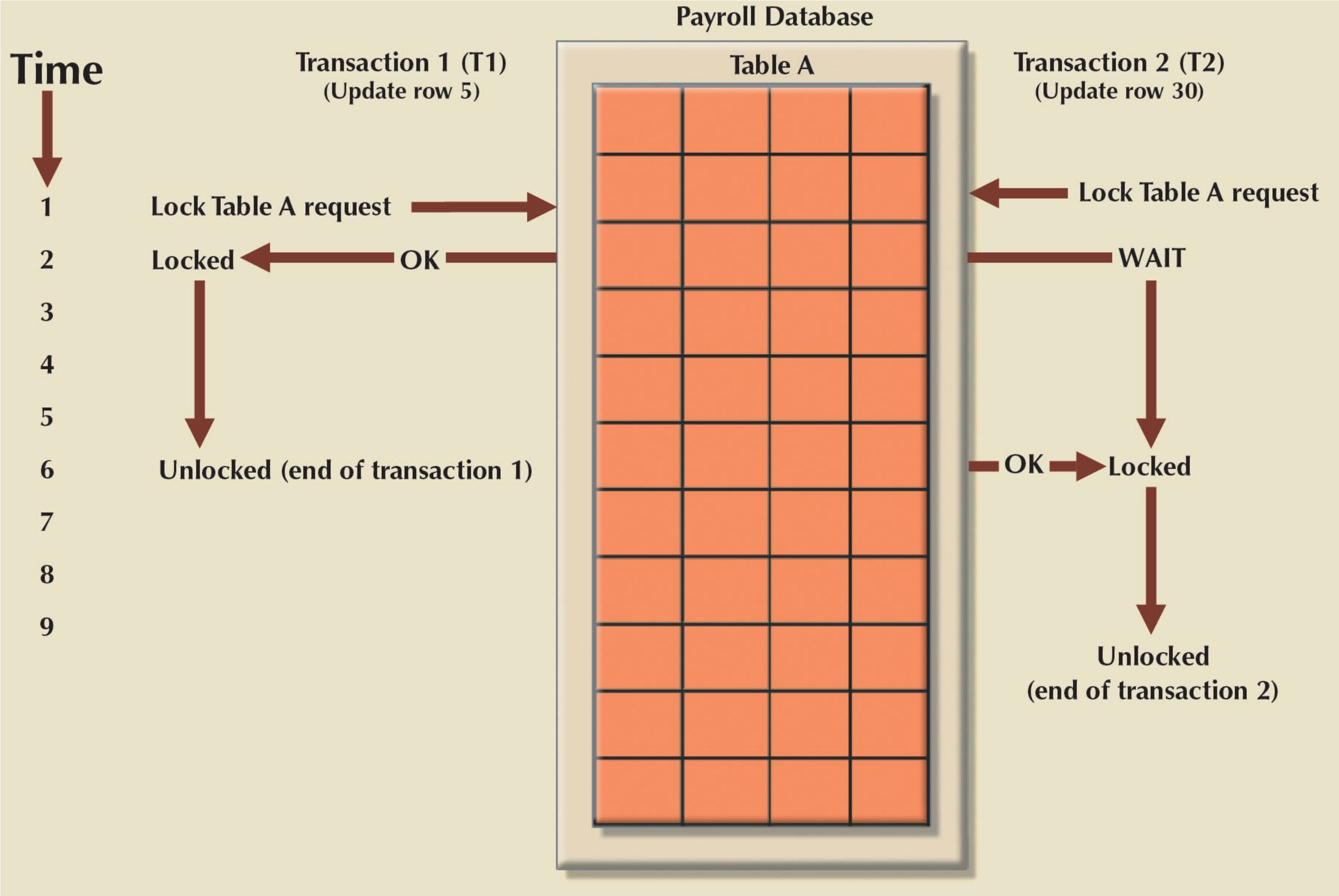


Figure 10.5 - An Example of a Page-Level Lock

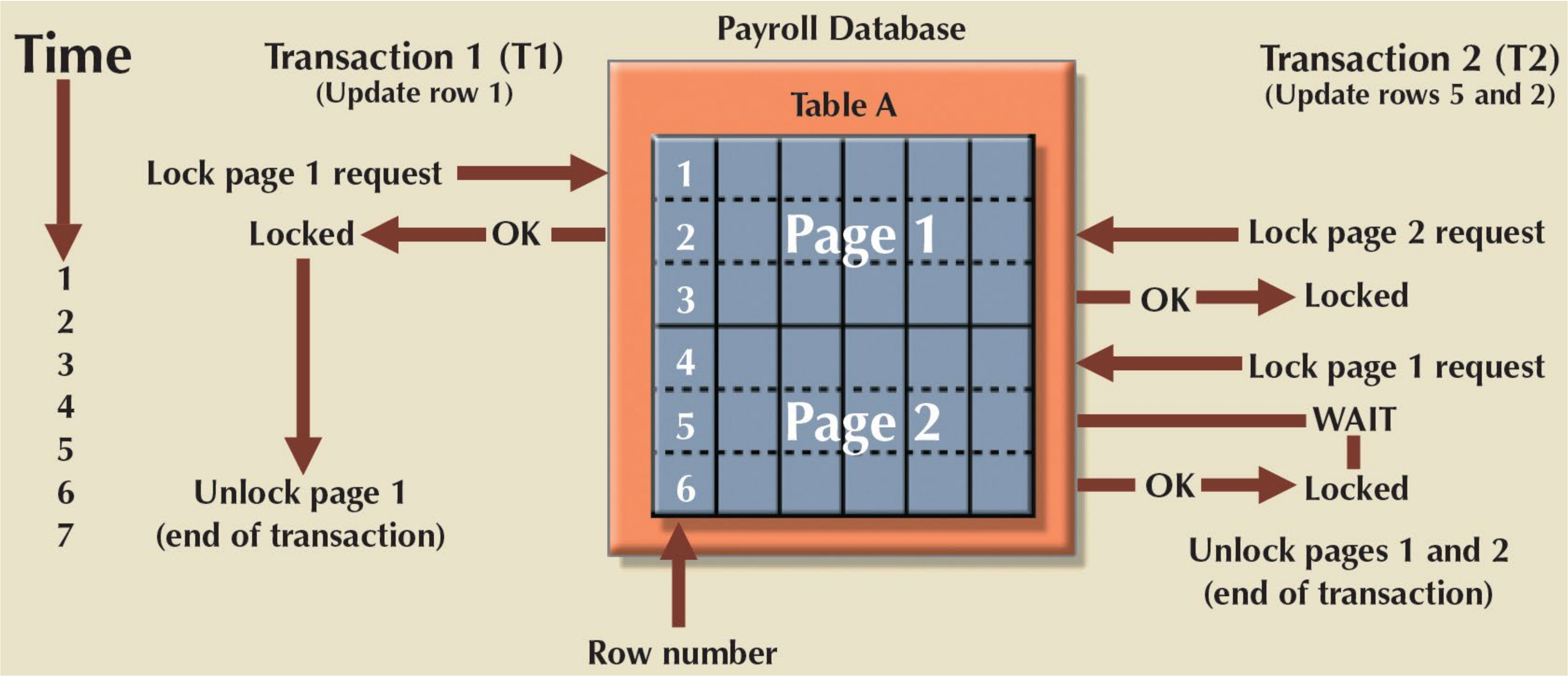
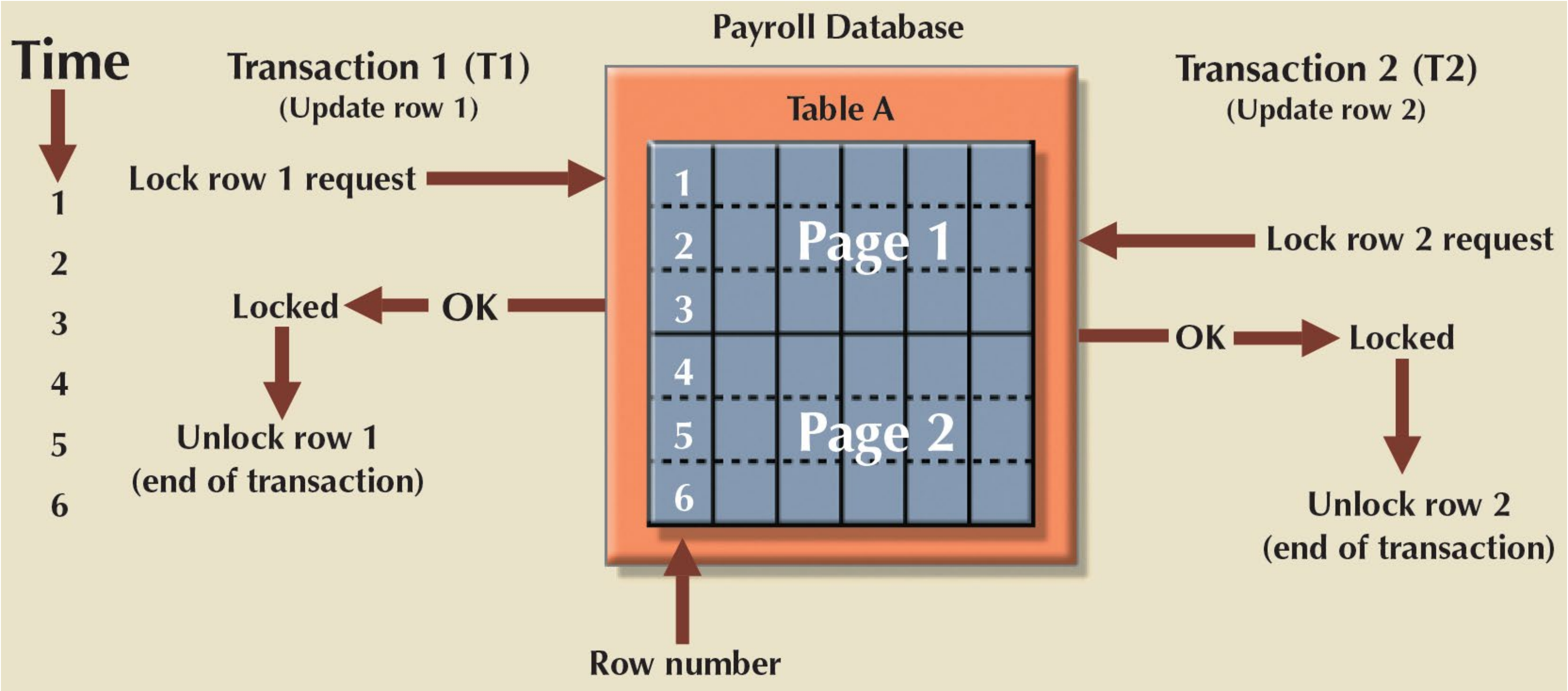


Figure 10.6 - An Example of a Row-Level Lock



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/Shared lock
  - Exclusive: Exists when access is reserved for the transaction that locked the object
  - Shared: Exists when concurrent transactions are granted read access on the basis of a common lock
  - Three states: unlocked, shared (read), and exclusive (write).

Problems in Using Locks

- Resulting transaction schedule might not be serializable
  - Resolved by two-phase locking
- Schedule might create **deadlocks**
  - A deadlock occurs when two transactions wait indefinitely for each other to unlock data
  - Deadlock detection and prevention techniques

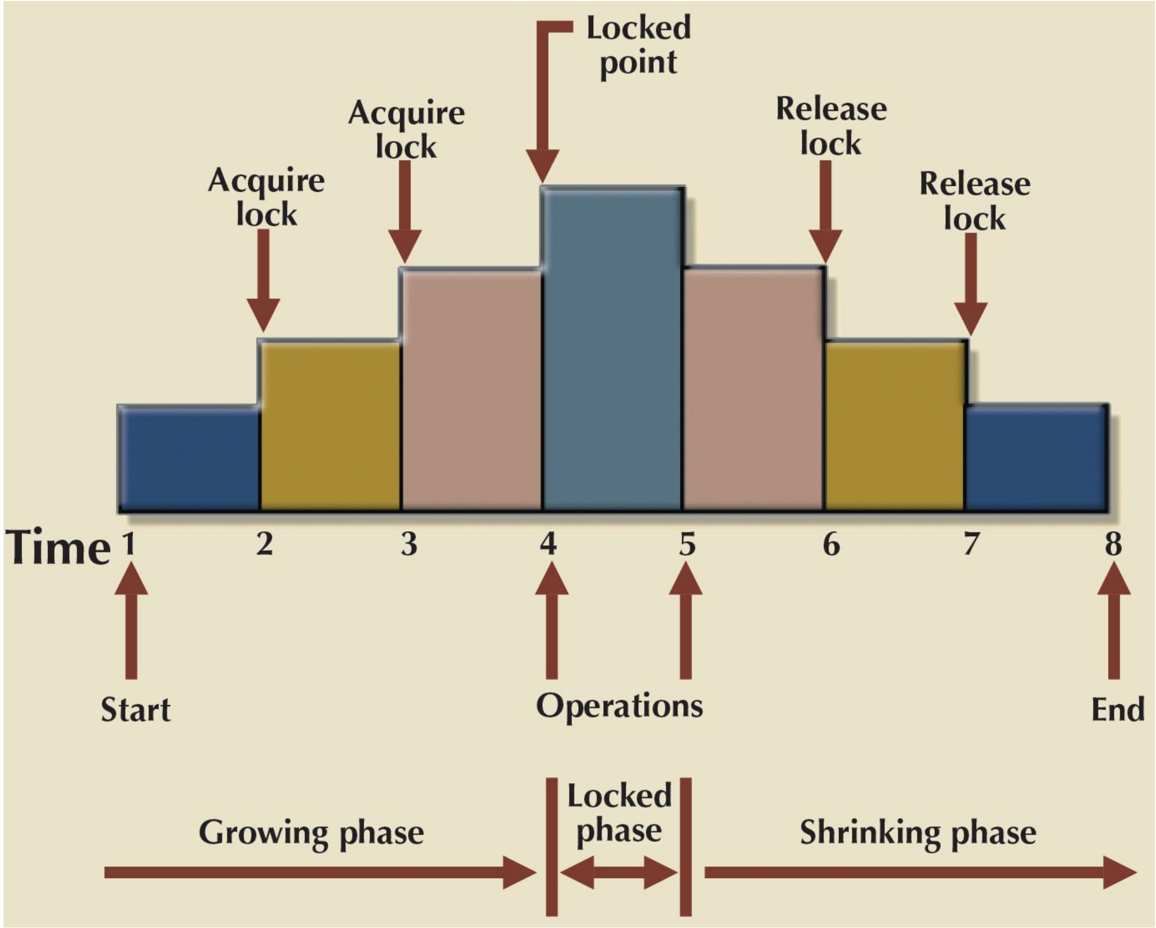
Two-Phase Locking (2PL) (1 of 2)

- 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) (2 of 2)

- 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



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

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

CENGAGE

Time Stamping (1 of 2)

- 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 (2 of 2)

- 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
- Two schemes used to decide which transaction is rolled back and which continues executing:
  - the wait/die scheme and
  - the wound/wait scheme

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)	• 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.

The Wait/Die Scheme

- If the transaction requesting the lock is the older of the two transactions, it will wait until the other transaction is completed and the locks are released.
- If the transaction requesting the lock is the younger of the two transactions, it will die (roll back) and is rescheduled using the same time stamp.
- In short, in the wait/die scheme, the older transaction waits for the younger one to complete and release its locks.

The wound/wait scheme

- If the transaction requesting the lock is the older of the two transactions, it will preempt (wound) the younger transaction by rolling it back.
    - T1 preempts T2 when T1 rolls back T2. The younger, preempted transaction is rescheduled using the same time stamp.
- If the transaction requesting the lock is the younger of the two transactions, it will wait until the other transaction is completed and the locks are released.
- In short, in the wound/wait scheme, the older transaction rolls back the younger transaction and reschedules it.

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

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
  - uses data in the transaction log to recover a database from an inconsistent state to a consistent state.

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 used to speed up
  - disk operations.
- **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 because the database was never updated.

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

CENGAGE