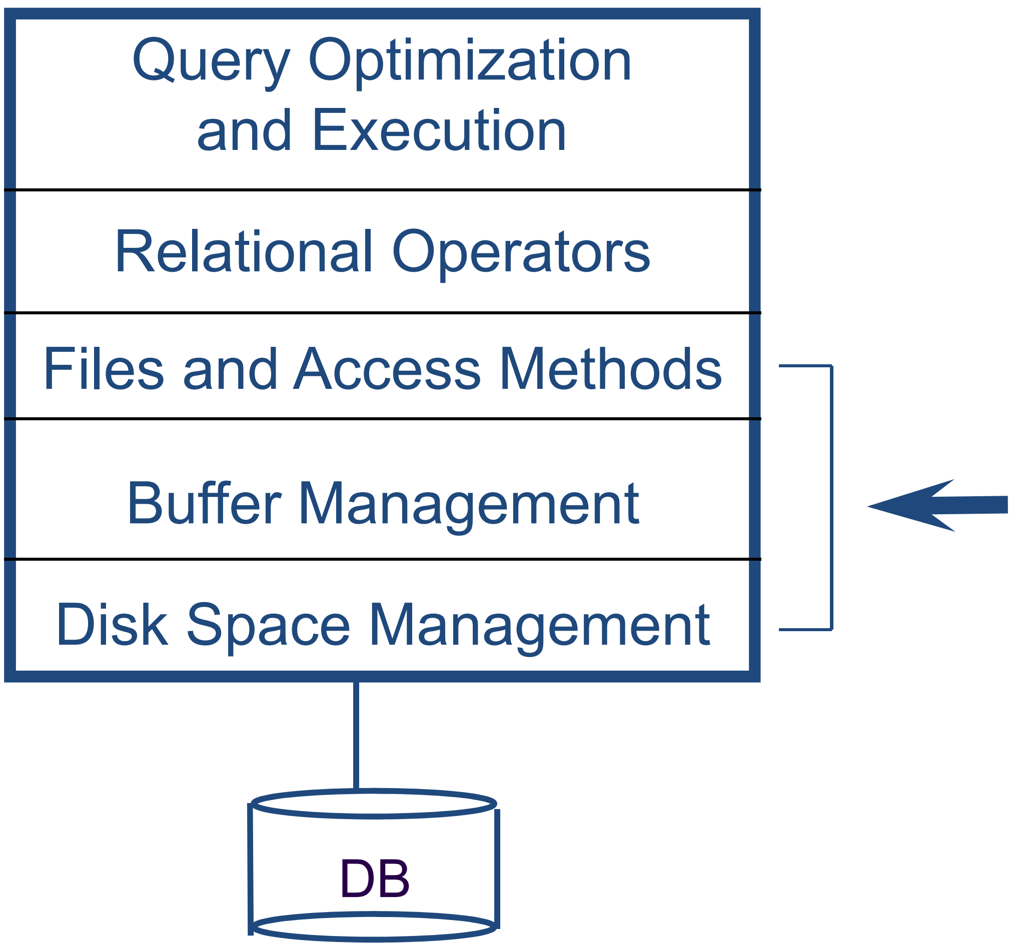
Database Management System

Sumayyea Salahuddin (Lecturer)
Dept. of Computer Systems Eng.
UET Peshawar

Overview

- 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

DBMS Structure



These layers must consider concurrency control and recovery (Transaction, Lock, Recovery Managers)

What is Transaction?

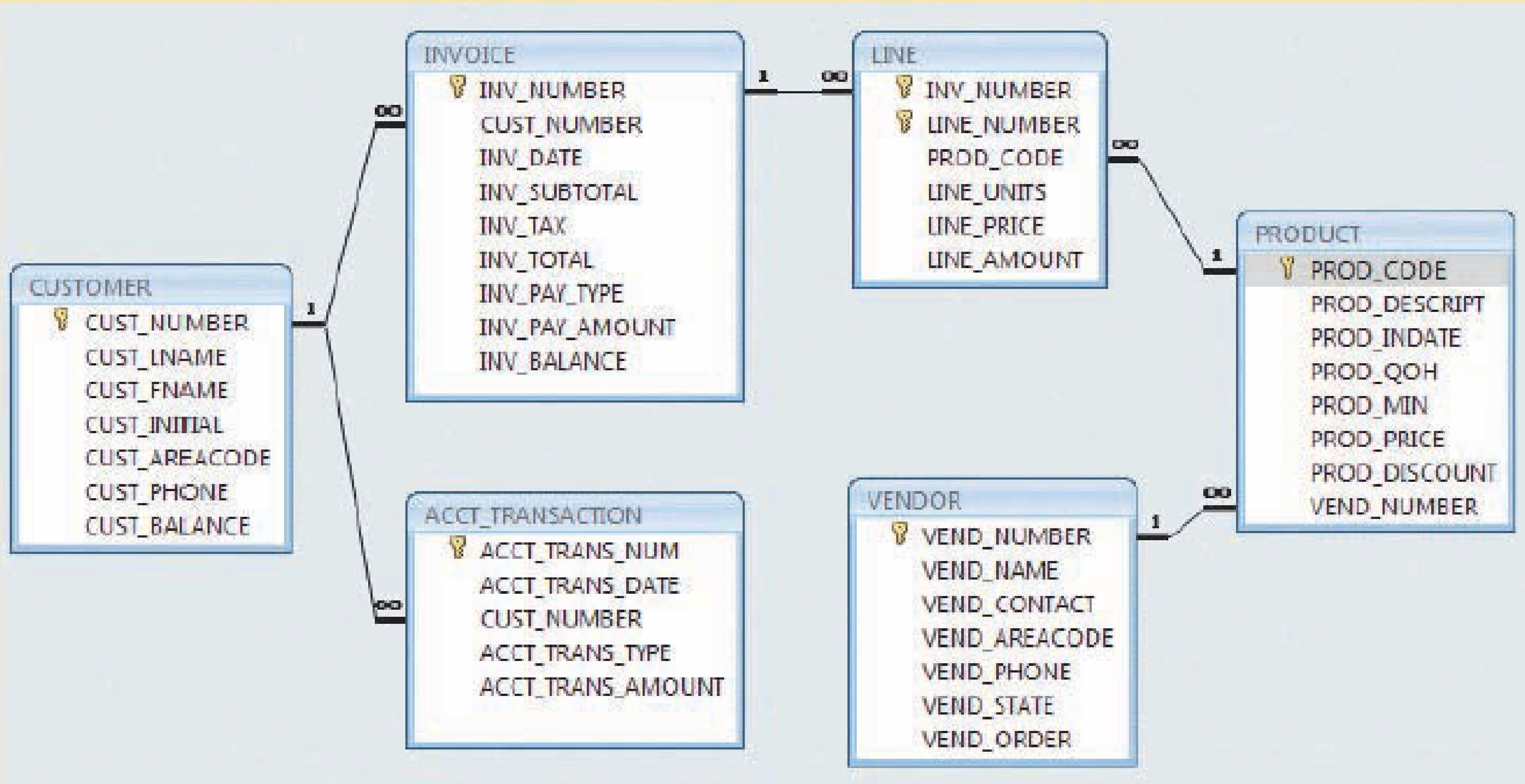
- Logical unit of work that must be either entirely completed or aborted
- Successful transaction changes database from one consistent state to another
  - One in which all data integrity constraints are satisfied
- Most real-world database transactions are formed by two or more database requests
  - Equivalent of a single SQL statement in an application program or transaction
    - if a transaction is composed of two UPDATE statements and one INSERT statement, the transaction uses three database requests. In turn, each database request generates several input/output (I/O) operations that read from or write to physical storage media.

What is Transaction? (Cont…)

- A transaction is any action that reads from and/or writes to a database & may consist of:
  - Simple SELECT statement to generate a list of table contents
  - Series of related UPDATE statements to change values of attributes in various tables
  - Series of INSERT statements to add rows to one or more tables
  - Combination of SELECT, UPDATE, and INSERT statements
- Example in next slide

Sample Relational Database

FIGURE 10.1 The Ch10_SaleCo database relational diagram



Example: Sales Transaction

- START TRANSACTION;
- INSERT INTO INVOICE VALUES (1009, 10016,'18-Jan-2008', 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-08', 10016, 'charge', 277.55);
- COMMIT/ROLLBACK;

Transaction Results

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-08	54.92	4.39	59.31	cc	59.31	0.00
1002	10011	16-Jan-08	9.98	0.80	10.78	cash	10.78	0.00
1003	10012	16-Jan-08	270.70	21.66	292.36	cc	292.36	0.00
1004	10011	17-Jan-08	34.87	2.79	37.66	cc	37.66	0.00
1005	10018	17-Jan-08	70.44	5.64	76.08	cc	76.08	0.00
1006	10014	17-Jan-08	397.83	31.83	429.66	cred	100.00	329.66
1007	10015	17-Jan-08	34.97	2.80	37.77	chk	37.77	0.00
1008	10011	17-Jan-08	1033.08	82.65	1115.73	cred	500.00	615.73
1009	10016	18-Jan-08	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-07	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-07	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-07	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-08	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-08	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-07	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-07	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-08	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-08	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-08	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-07	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Jan-08	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	06-Jan-08	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-08	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-08	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-08	18	5	119.95	0.10	25595

Table name: CUSTOMER

CUST_NUMBER	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-08	10014	charge	329.66
10004	17-Jan-08	10011	charge	615.73
10006	29-Jan-08	10014	payment	329.66
10007	18-Jan-08	10016	charge	277.55

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
  - Some DBMSs provide means by which user can define enforceable constraints
  - Other integrity rules are enforced automatically by the DBMS
    - Primary key integrity
    - Referential integrity
    - Entity integrity
- No semantic/logical checking
- Transactions that violate integrity constraints are aborted

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
- Durability
  - Once transactions are committed, they cannot be undone
- Serializability
  - Concurrent execution of several transactions yields consistent results

Multtiuser databases 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 (Equivalent to COMMIT)
  - Program is abnormally terminated (Equivalent to ROLLBACK)

Transaction & Concurrent Execution

- Transaction Manager controls the execution of transactions.
- User program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database.
- Concurrent execution of multiple transactions is essential for good performance.
  - Disk is the bottleneck (slow, frequently used)
  - Must keep CPU busy w/many queries
  - Better response time

The Transaction Log

- Keeps track of all transactions that update the database
- Useful for recovering database
- **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

Sample 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
(*Note*: The transaction number is automatically assigned by the DBMS.)
PTR = Pointer to a transaction log record ID

```
START TRANSACTION;
UPDATE PRODUCT SET PROD_QOH = 23 WHERE PROD_ID = ‘1558-QW1’;
UPDATE CUSTOMER SET CUST_BALANCE = 615.73 WHERE CUST_ID =
10011;
COMMIT;
```

Concurrency Control

- Coordination of simultaneous transaction execution in a multiprocessing database
- Objective is to ensure serializability of transactions in a multiuser environment
- Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems
  - Lost updates
  - Uncommitted data
  - Inconsistent retrievals

Lost Updates

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

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

Lost Updates (Cont…)

TABLE 10.3 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

TABLE 10.4 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

Uncommitted Data

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

TABLE 10.5 Transactions Creating 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 (Cont…)

TABLE 10.6 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

TABLE 10.7 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

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 Retrievals (Cont…)

TABLE 10.8 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 10.9 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 Retrievals (Cont…)

TABLE 10.10 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

Scheduling Transactions

- <u>Serial schedule:</u> Schedule that does not interleave the actions of different transactions.

- <u>Equivalent schedules:</u> For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.

- <u>Serializable schedule:</u> A schedule that is equivalent to some serial execution of the transactions. (Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

The Scheduler

- As long as two transactions access unrelated data, there is no conflict in the execution (order is irrelevant to the final outcome)
- 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 some serial execution

The Scheduler (Cont…)

- Bases its actions on concurrency control algorithms
- Ensures computer's central processing unit (CPU) is used efficiently
  - First-come first-served scheduling wastes processing time when CPU waits for READ or WRITE operation
- Facilitates data isolation to ensure that two transactions do not update same data element at same time

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

For the same data unit

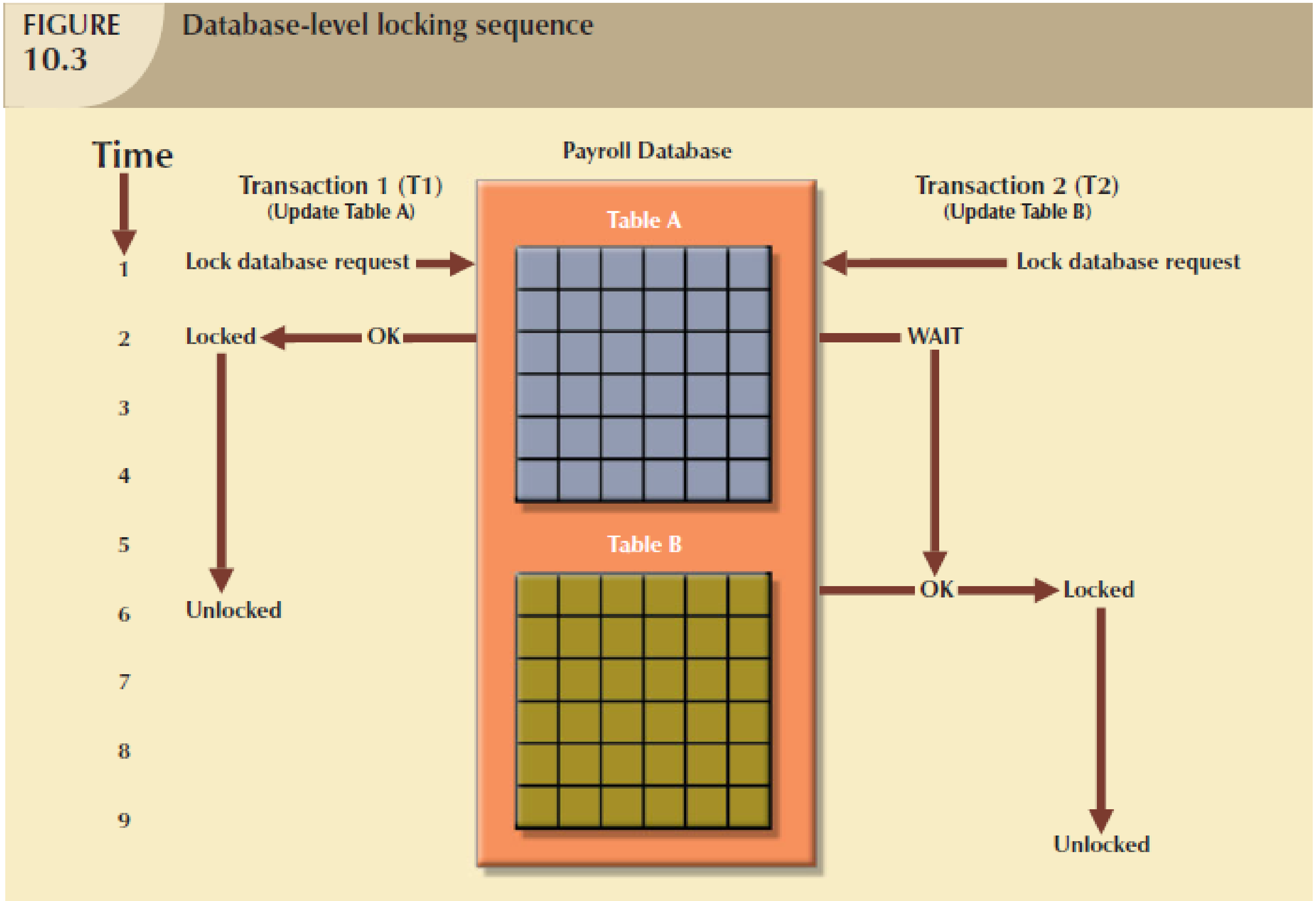
Concurrency Control with Locking Methods

- Lock
  - Guarantee 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 transactions

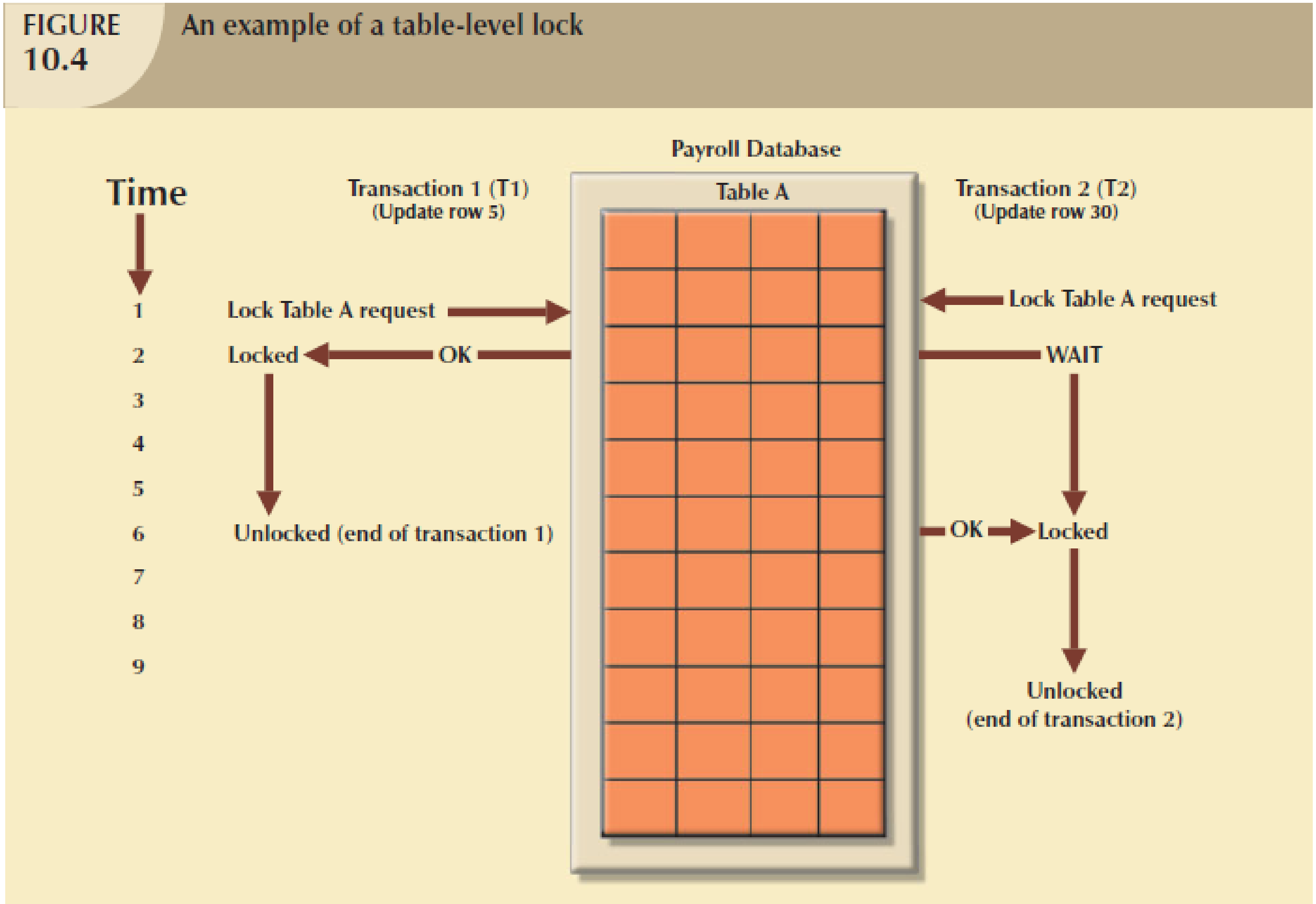
Lock Granularity

- 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:
    - Allows concurrent transactions to access different rows of same table, even if rows are located on same page
  - Field (attribute)
    - Allows concurrent transactions to access same row, as long as they require use of different fields (attributes) within that row

Database-level Locking Sequence

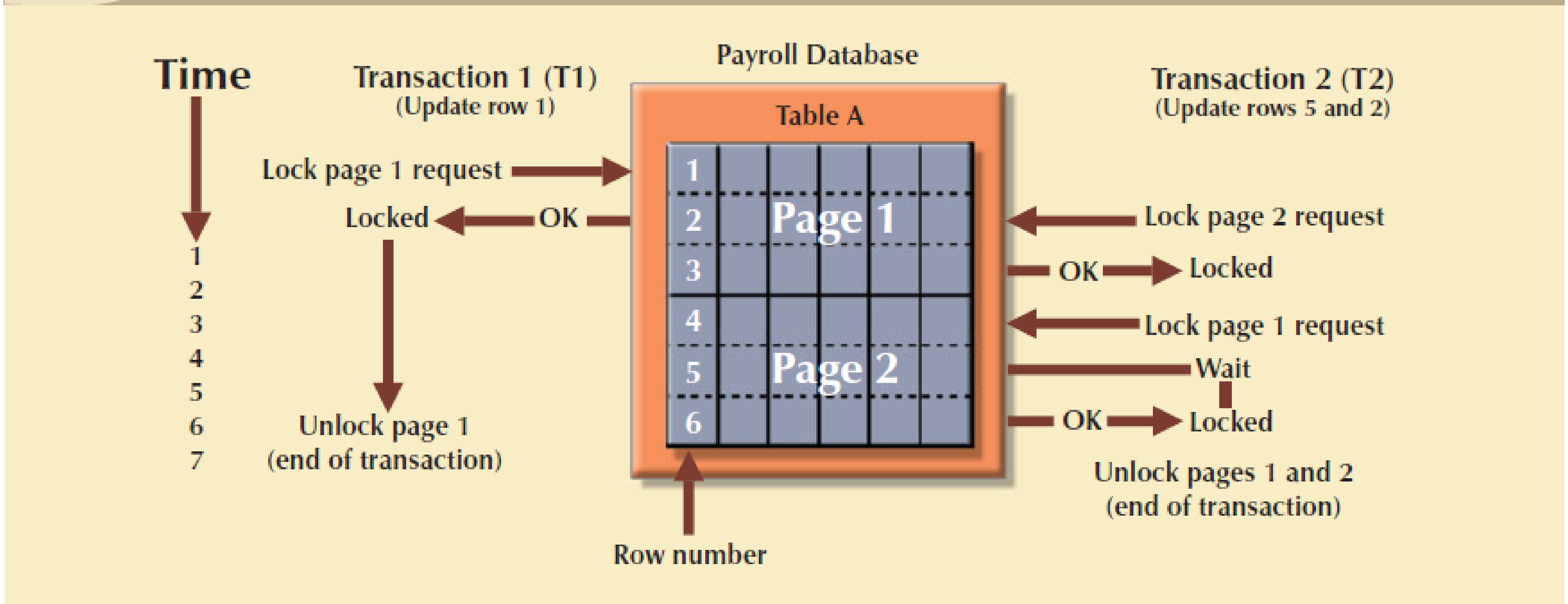


Example: Table-level Lock

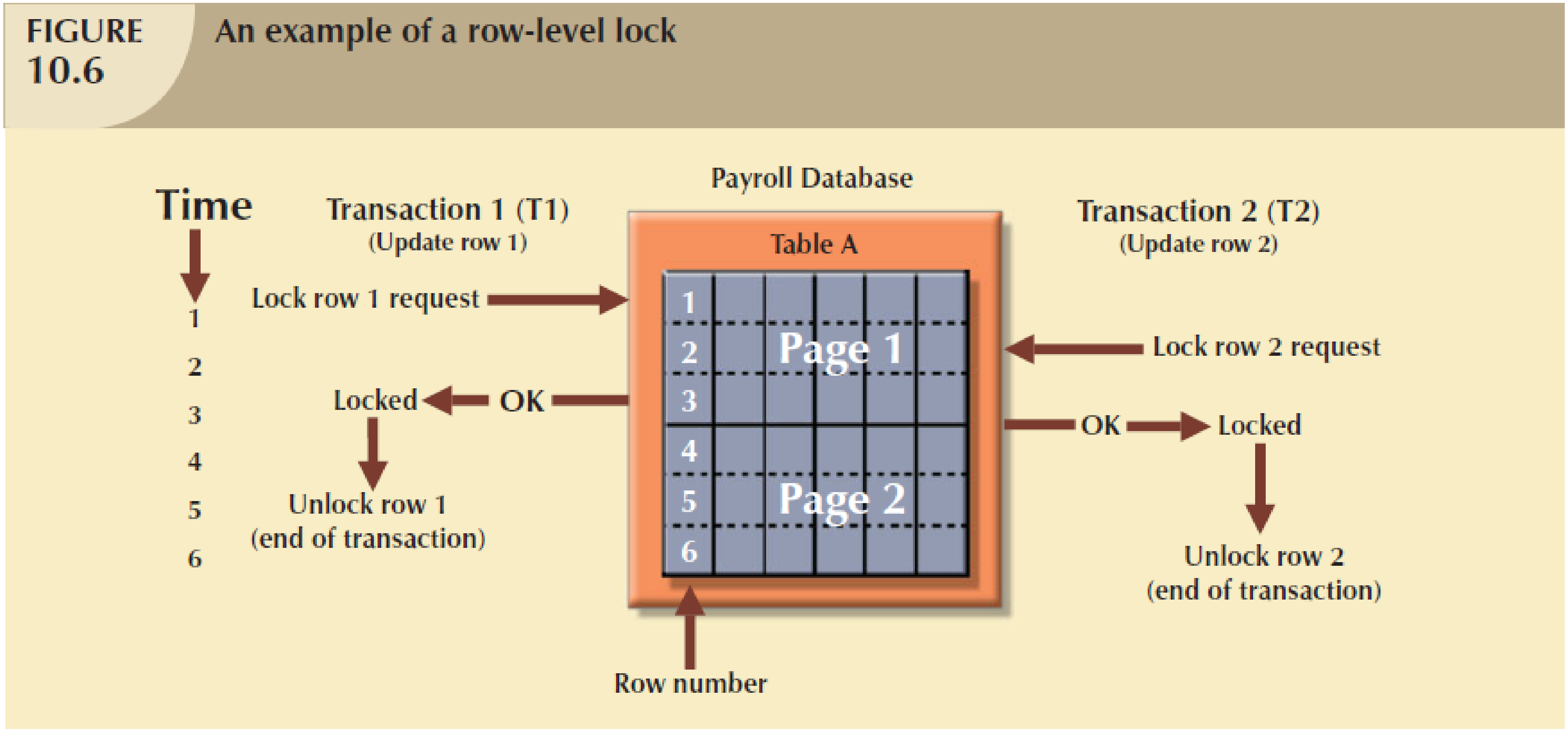


Example: Page-level Lock

FIGURE 10.5 An example of a page-level lock



Example: Row-level Lock



Lock Types

- Binary lock
  - Two states: locked (1) or unlocked (0)
  - Every transaction required a lock and unlock operation for each accessed data item, which are automatically managed by the DBMS
- Exclusive lock
  - Access is specifically reserved for transaction that locked object
  - Mutual exclusive rule
  - Must be used when potential for conflict exists
- Shared lock
  - Concurrent transactions are granted read access on basis of a common lock

Locking Conflict Table

Data Status \ Request	Not Locked	Share Locked	Exclusive Lock
Shared Lock	No Conflict	No Conflict	Conflict
Exclusive Lock	No Conflict	Conflict	Conflict

Example: Binary Lock

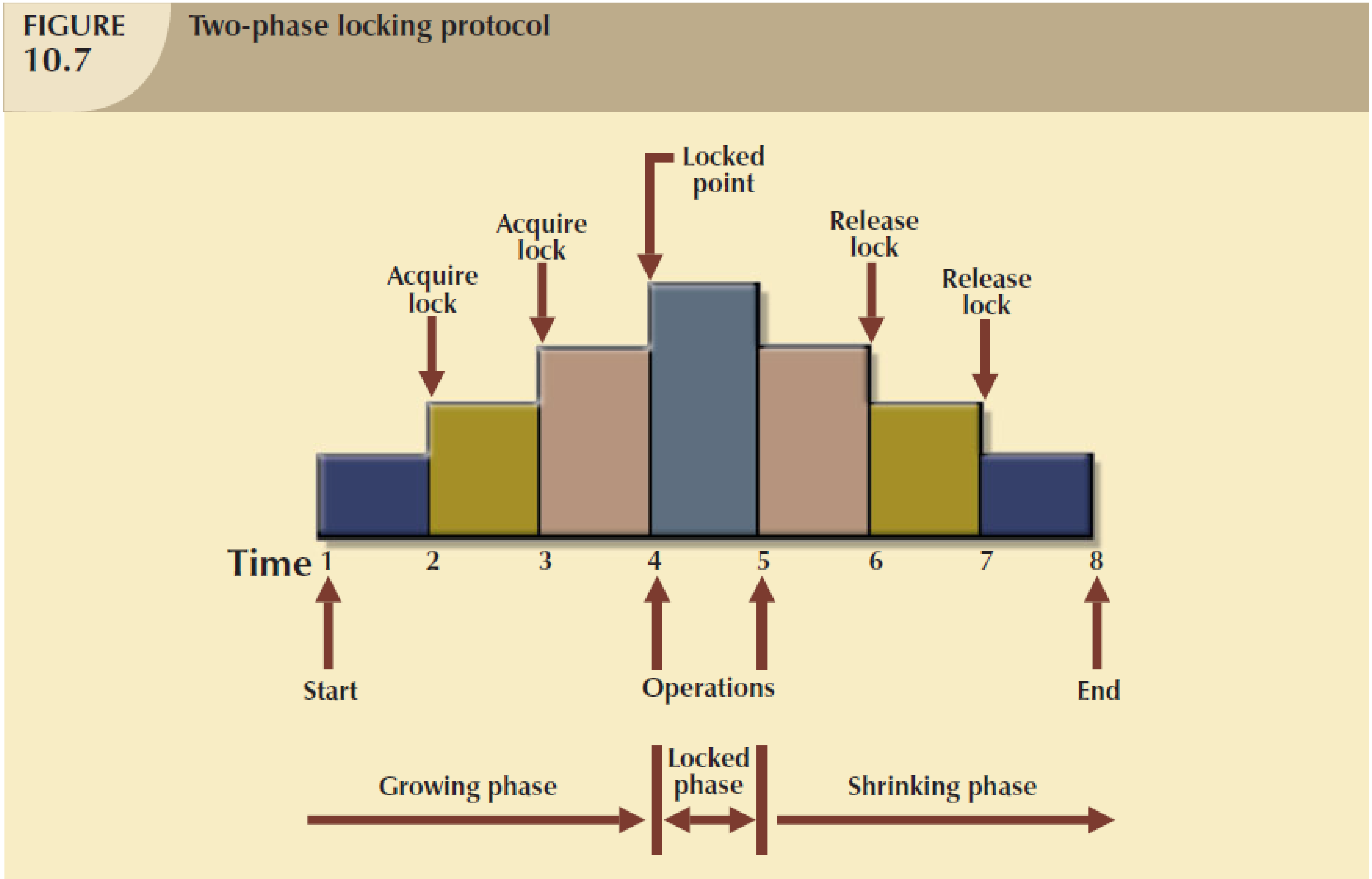
TABLE 10.12 An Example of a Binary Lock

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Lock PRODUCT	
2	T1	Read PROD_QOH	13
3	T1	PROD_QOH = 13 + 10	
4	T1	Write PROD_QOH	23
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 does not prevent deadlocks
  - Growing phase
    - Transaction acquires all required locks without unlocking any data
  - Shrinking phase
    - Transaction releases all locks and cannot obtain any new lock
- 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 transaction in in its locked point

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
- Three techniques to control deadlock:
  - Prevention: Abort & reschedule transaction if deadlock possible
  - Detection: Test database for deadlocks; abort victim, leave rest
  - Avoidance
- Choice of deadlock control method depends on database environment
  - Low probability of deadlock, detection recommended
  - High probability of deadlock, prevention recommended

How Deadlock Condition is Created?

TABLE 10.13 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
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....

Deadlock

Concurrency Control with Time Stamping Methods

- Assign global, unique time stamp to each transaction
- Produces explicit order in which transactions are submitted to DBMS
- Properties of time stamp
  - **Uniqueness**
    - Ensures that no equal time stamp values can exist
  - Monotonicity
    - Ensures that time stamp values always increase
- Disadvantages
  - Each value in database needs two fields: a) last time field was read, and b) last update
  - Increases memory needs and processing overhead

Wait/Die and Wound/Wait Schemes

- Wait/die
  - Older requesting transaction waits and
  - Younger requesting transaction is rolled back and rescheduled
- Wound/wait
  - Older requesting transaction preempts (rolls back) younger transaction and reschedules it
  - Younger requesting transaction waits

Wait/Die and Wound/Wait Schemes (Cont…)

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 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 assumption that 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: read, validation, and write
  - Good for read/query database systems requiring few update transactions; Poor for heavily used DBMS environment

Reasons for a Crash

- System Crash
- Transaction or System Error
- Local Error or Exception
- Concurrency Control
- Disk Failure
- Catastrophe

Crash Recovery

- Recovery Manager
  - When DBMS is restarted after crashes, the recovery manager must bring the database to a consistent state
  - Ensures transaction atomicity and durability
  - Undo actions of transactions that do not commit
  - Redo actions of committed transactions during system failures and media failures (corrupted disk).

- Recovery Manager maintains log information during normal execution of transactions for use during crash recovery

Database Recovery Management

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

Concept that Affect 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

- Make use of deferred-write and write-through techniques

- **Deferred-write technique**
  - Transaction operations do not immediately update physical database
  - Only transaction log is updated
  - Database is physically updated only after transaction reaches its commit point using transaction log information

Transaction Recovery (Cont…)

- Recovery process for deferred-write:
  - 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 or was left active
    - Do nothing because no updates were made

Transaction Recovery (Cont…)

- **Write-through technique**
  - Database is immediately updated by transaction operations during transaction's execution
- Recovery process for write-through
  - Identify last checkpoint
  - If transaction committed before checkpoint
    - Do nothing
  - If transaction committed after checkpoint
    - DBMS redoes the transaction using "after" values
  - If transaction had ROLLBACK operation or was left active
    - Uses the before value in the transaction log records to ROLLBACK (undo)

Transaction Recovery (Cont…)

TABLE 10.15 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-2008, …
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 exit 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…)

- 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 guarantee unique access to data item by transaction
- Two types of locks: binary locks and shared/exclusive locks

Summary (Cont…)

- 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

Summary (Cont…)

- Optimistic methods assume the majority of database transactions do not conflict
  - Transactions are executed concurrently, using private copies of the data
- Database recovery restores database from given state to previous consistent state