Attribute level inconsistency

```
/* Client 1 - Deposit */
    UPDATE CheckingAccount
    SET Balance = Balance+100 WHERE
                  AccountNumber=123456;
--------
/* Client 2 - Deposit */
    UPDATE CheckingAccount
    SET Balance = Balance+150 WHERE
                  AccountNumber=123456;
```

Relation level inconsistency

```
/* Client 1 - Give High Cost Area increases to
              top performers */
UPDATE Employees
SET Hours = Hours * 1.25
WHERE Rating>90;

/* Client 2 - Tri-State hourly rate Increases for
              full-time workers */
UPDATE Paychecks
SET Rate = Rate * 1.05
WHERE (State=NY OR State=CT OR State=NJ) AND
Empid IN
    (SELECT Empid FROM Employees
     WHERE Hours>=40);
```

Multiple statement inconsistency

```
/* Client 1 - promote students based on
              hours earned */
INSERT INTO Seniors
       (SELECT *
        FROM Juniors WHERE Hours > 90);
DELETE FROM Juniors WHERE Hours > 90;
...

/* Client 2 - Calculate class sizes */
SELECT COUNT(*) FROM SENIORS;
SELECT COUNT(*) FROM JUNIORS;
```

Transactions

```
BEGIN TRANSACTION;
-- get input from something or
someone
Do some SQL commands using that
input;
-- Confirm the results with
something or someone
IF (OK?) THEN Commit; ELSE Undo;
```

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 Demo

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.)

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

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

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

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

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;

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

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

Transactions that read and write data in the same row

Transaction A

```
START TRANSACTION;

UPDATE invoices SET credit_total = credit_total + 100
WHERE invoice_id = 6;

-- the SELECT statement in Transaction B
--     won't show the updated data
-- the UPDATE statement in Transaction B
--     will wait for transaction A to finish

COMMIT;

-- the SELECT statement in Transaction B
--     will display the updated data
-- the UPDATE statement in Transaction B
--     will execute immediately
```

From Murach, 4th Ed.

Transactions that read and write data in the same row

Transaction B

```
START TRANSACTION;

SELECT invoice_id, credit_total
FROM invoices WHERE invoice_id = 6;

UPDATE invoices SET credit_total = credit_total
+ 200 WHERE invoice_id = 6;

COMMIT;
```

From Murach, 4th Ed.

Concurrency problems that locking can prevent

- Lost updates
- Dirty reads
- Nonrepeatable reads
- Phantom reads

Isolation levels can also help

Isolation level	Problems prevented
READ UNCOMMITTED	None
READ COMMITTED	Dirty reads, ~~lost updates, nonrepeatable reads, phantom reads~~
REPEATABLE READ	Dirty reads, lost updates, nonrepeatable reads, ~~phantom reads~~
SERIALIZABLE	Dirty reads, lost updates, nonrepeatable reads, phantom reads

Example transaction with Repeatable Read

```
– MySQL default isolation level:  REPEATABLE READ
START TRANSACTION;
– specify level with: “START TRANSACTION ISOLATION LEVEL xxx”

SELECT ... ;

-- do some complex calculation using the following result
SELECT COUNT (*) FROM ENROLLMENT WHERE ClassDept = "CompSci";
-- do some other stuff, then get that same result again to
-- finish the calculation, and this count had better be the
-- same as before!

SELECT COUNT (*) FROM ENROLLMENT WHERE ClassDept = "CompSci";

COMMIT; -- This ends the transaction
```

Setting isolation level

```
SET {GLOBAL|SESSION} TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED|READ COMMITTED|
     REPEATABLE READ|SERIALIZABLE}
```

Set the transaction isolation level to…

SERIALIZABLE for the next transaction

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

READ UNCOMMITTED for the current session

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

READ COMMITTED for all sessions

```
SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED
```

From Murach, 4th Ed.

Database-Level Lock

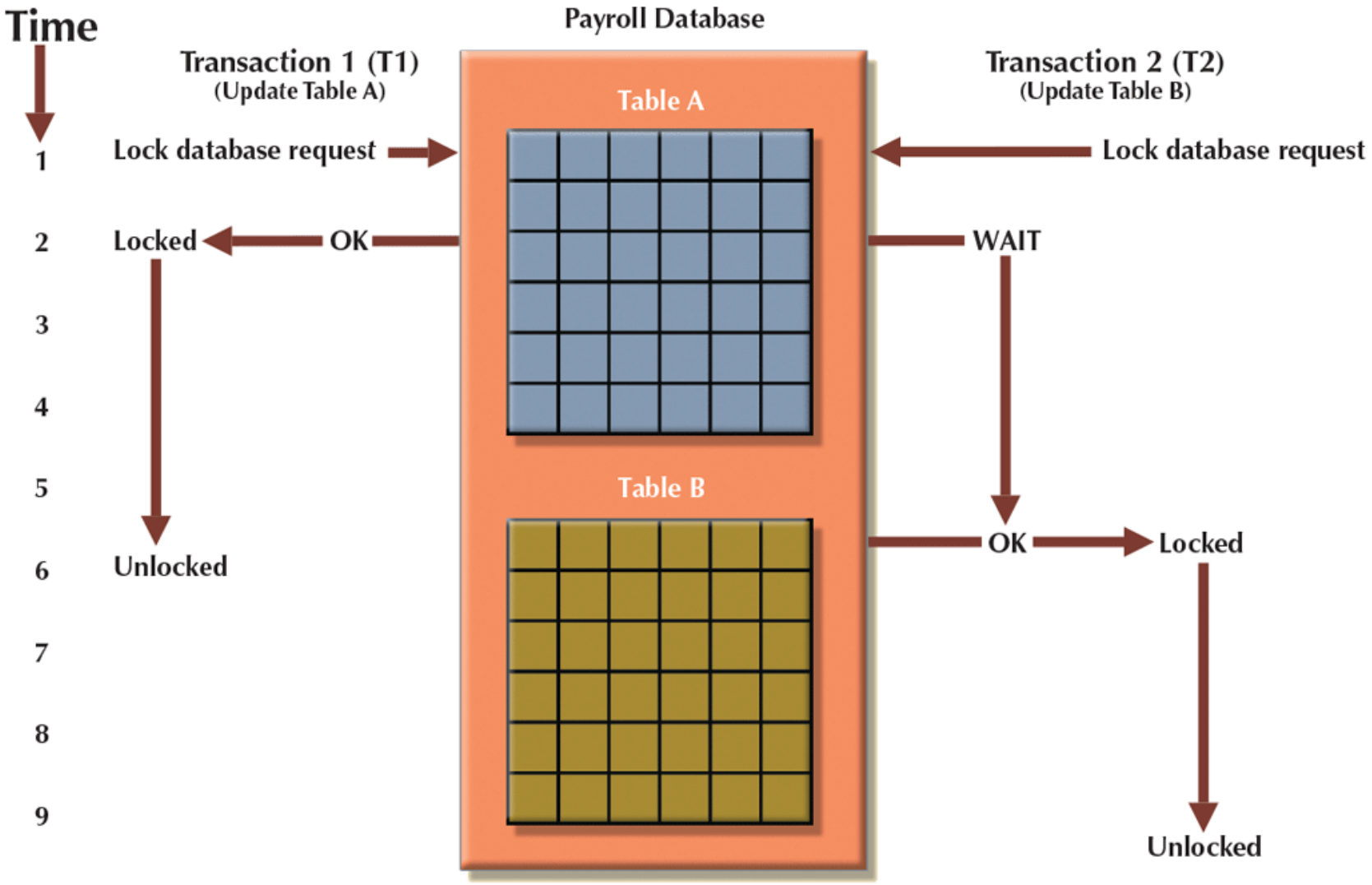
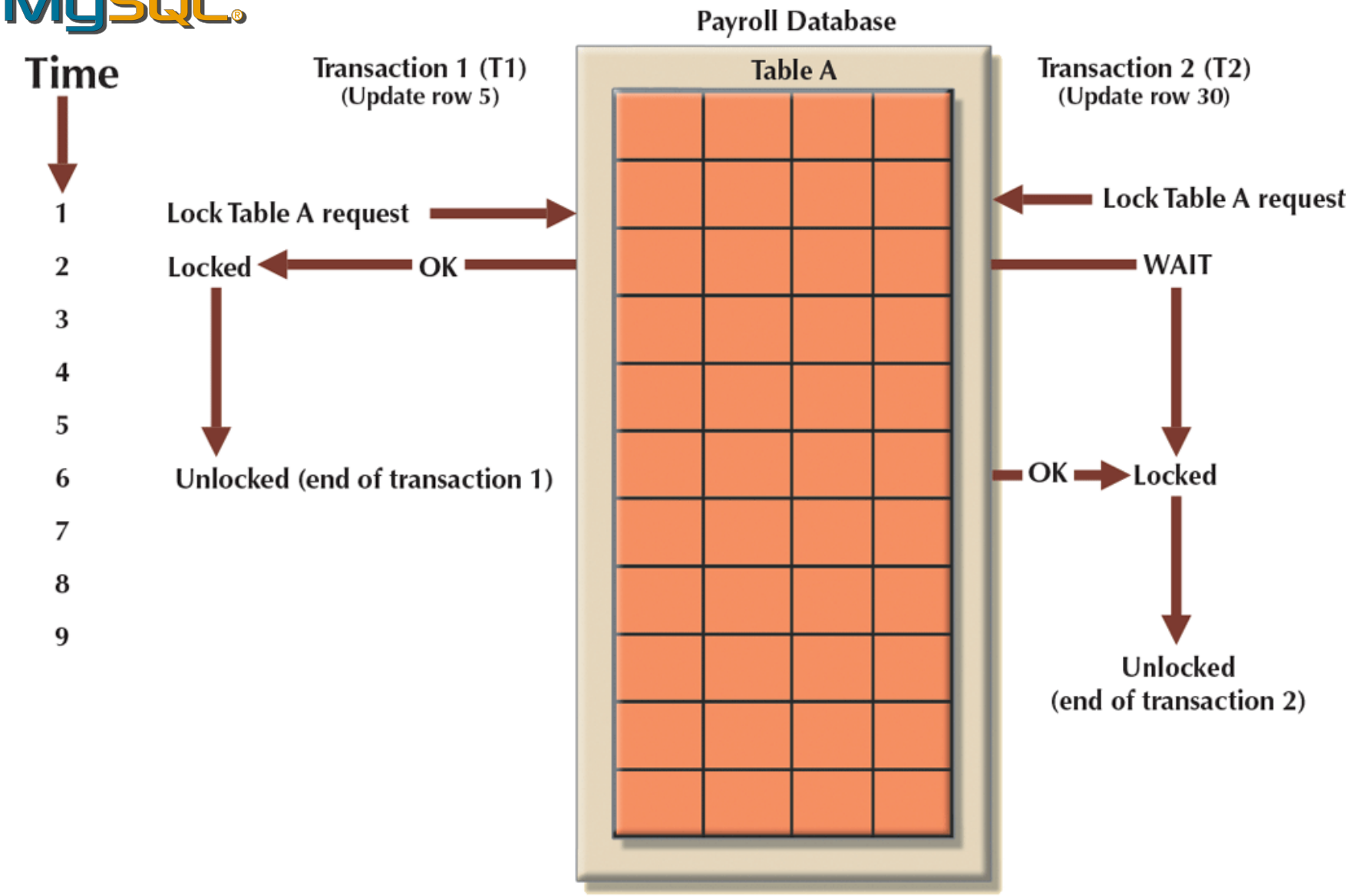
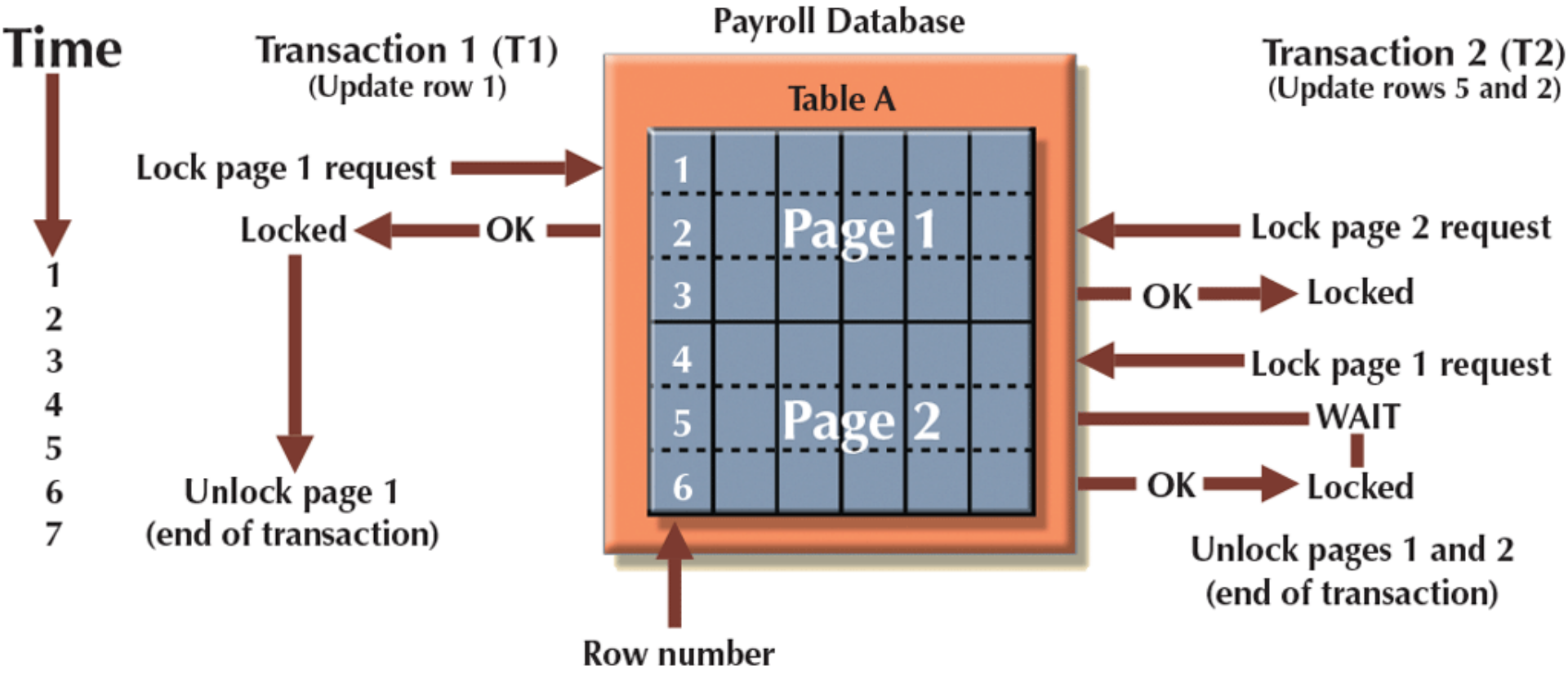


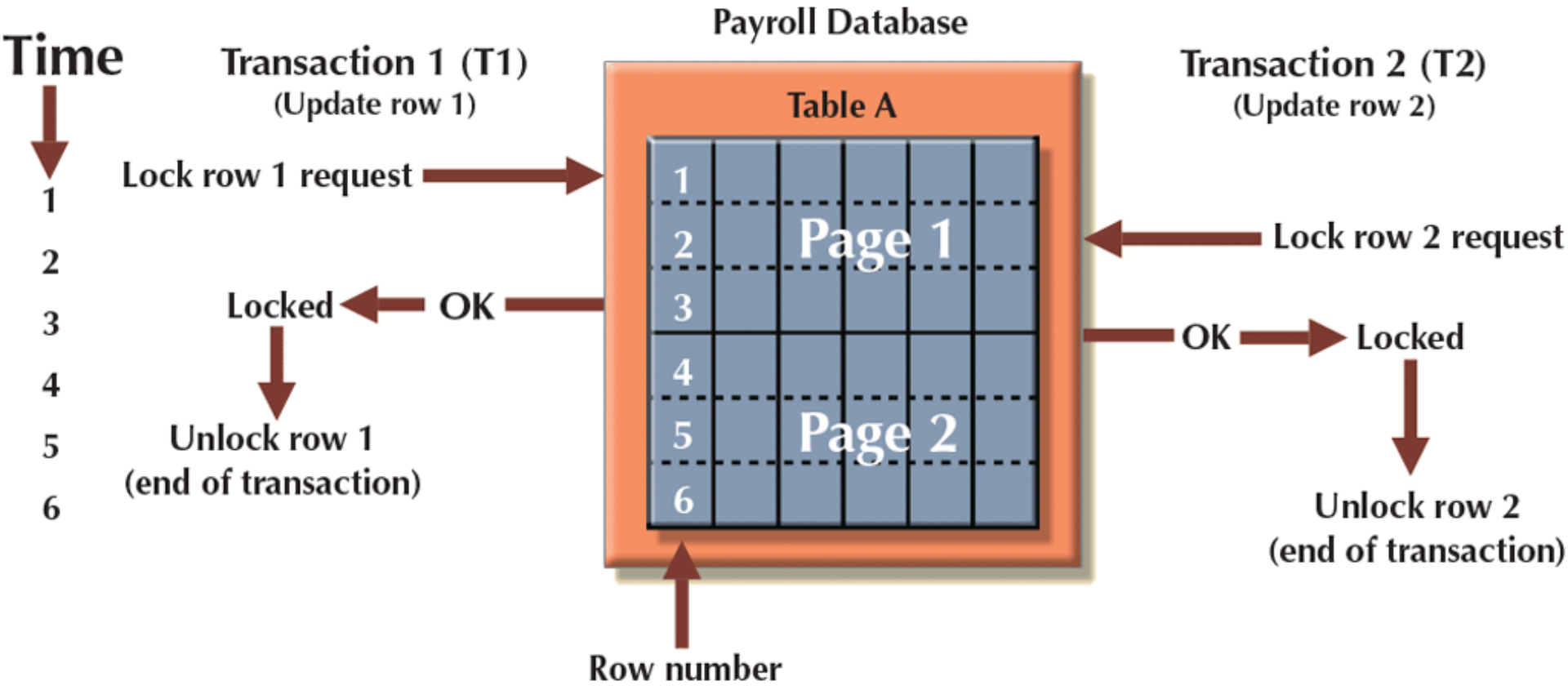
Table-level lock



A Page level lock



A row-level lock



Four transactions that show how to work with locking reads

Transaction A

```
-- lock row with rep_id of 2 in parent table
SELECT * FROM sales_reps WHERE rep_id = 2 FOR SHARE;

-- Transaction B waits for transaction A to finish
-- Transaction C returns an error immediately
-- Transaction D skips the locked row and returns
--     the other rows immediately

 -- insert row with rep_id of 2 into child table
INSERT INTO sales_totals
    (rep_id, sales_year, sales_total)
VALUES (2, 2023, 138193.69);

COMMIT;  -- Transaction B executes now
```

From Murach, 4th Ed.

Four transactions that show how to work with locking reads (continued)

Transaction B

```
START TRANSACTION;
SELECT * FROM sales_reps WHERE rep_id < 5 FOR UPDATE;
COMMIT;
```

Transaction C

```
START TRANSACTION;
SELECT * FROM sales_reps WHERE rep_id < 5
    FOR UPDATE NOWAIT;
COMMIT;
```

Transaction D

```
START TRANSACTION;
SELECT * FROM sales_reps WHERE rep_id < 5
    FOR UPDATE SKIP LOCKED;
COMMIT;
```

From Murach, 4th Ed.

Optional Locking Algorithm Details

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 to do **write**

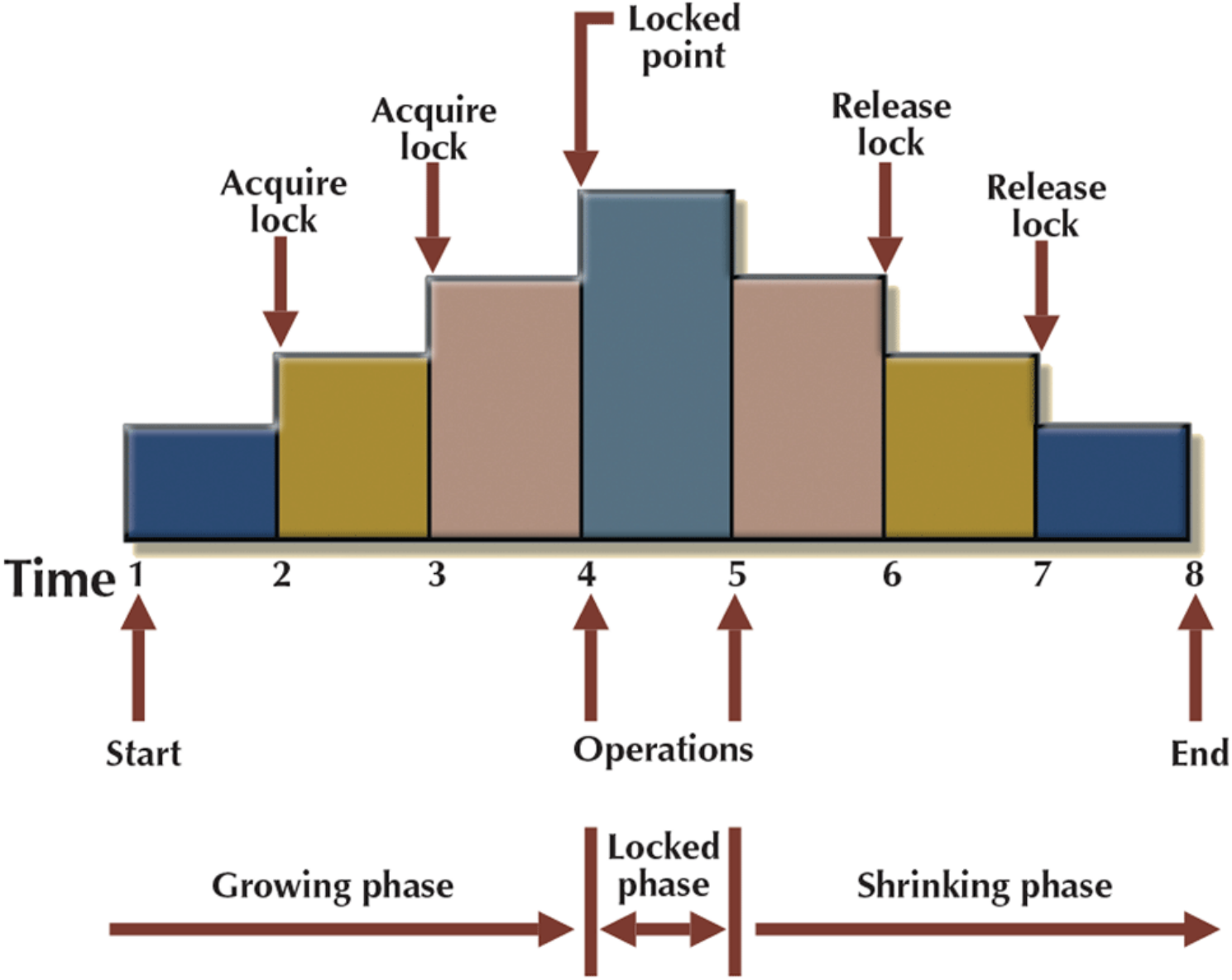
Shared lock

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

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 Protocol



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
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....

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.

Guidelines to avoid deadlocks

- Don’t allow transactions to remain open for very long.
- Don’t use a transaction isolation level higher than necessary.
- Make large changes when you can be assured of nearly exclusive access.
- Consider locking when coding your transactions.

Transaction log example

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-2018, ...
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 * * * *									