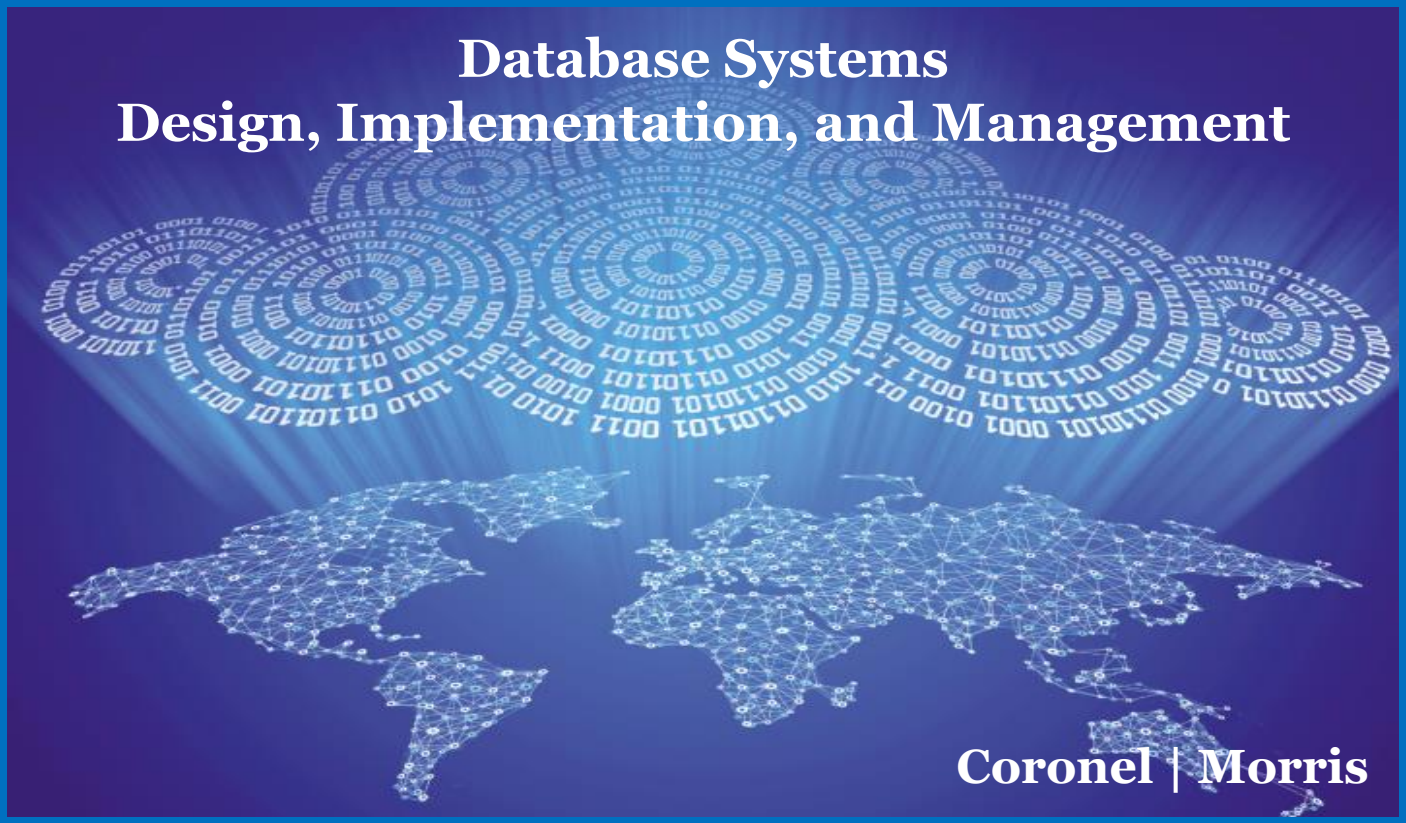
11e




# Chapter 10

# Transaction Management and Concurrency Control

# Learning Objectives

- In this chapter, students 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

- In this chapter, students 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

# 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

# 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
- Consists of a single SQL statement or a collection of related SQL statements

# 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

# 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

# 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

# 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

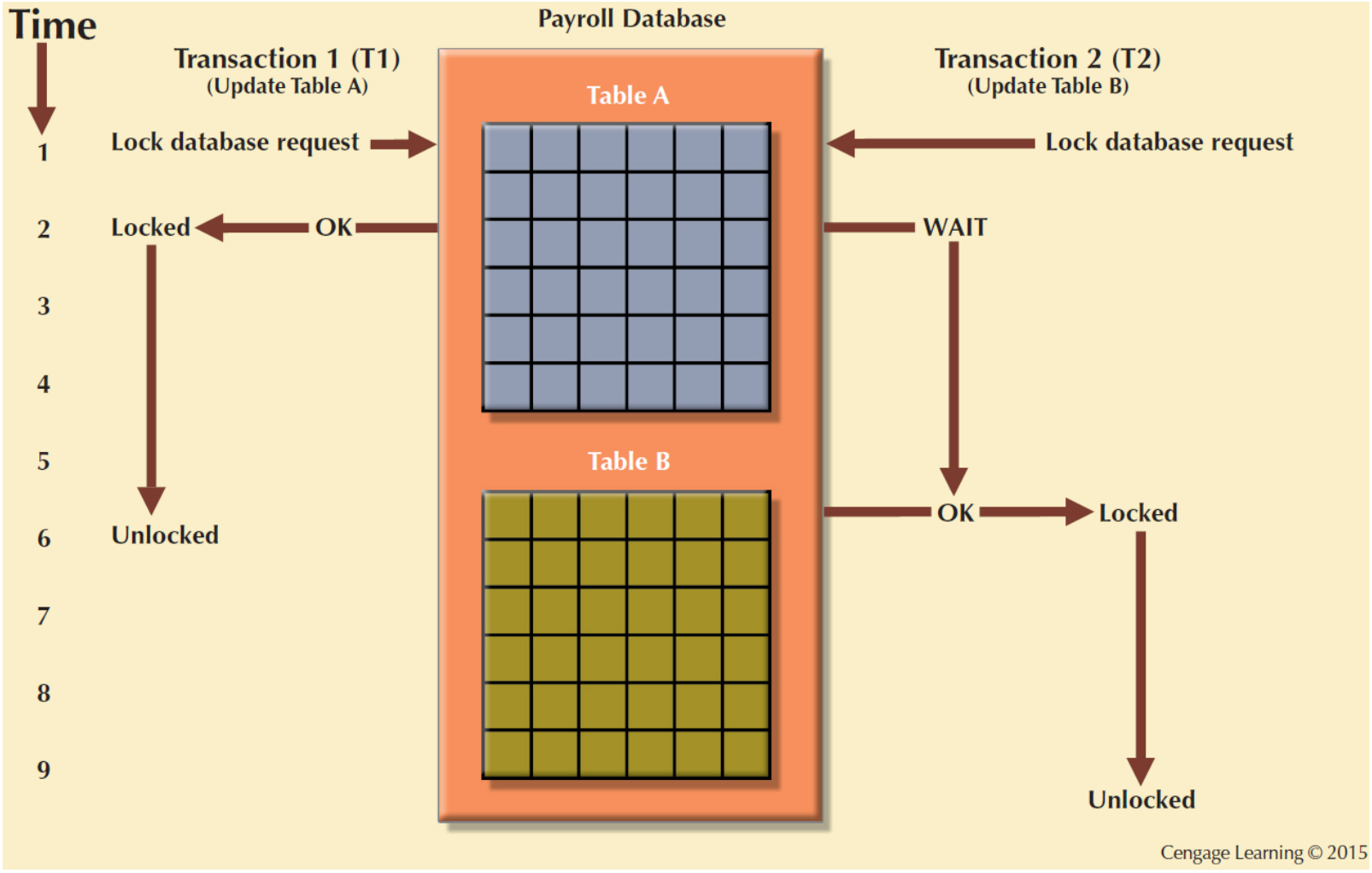
# Concurrency Control with Locking Methods

- 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

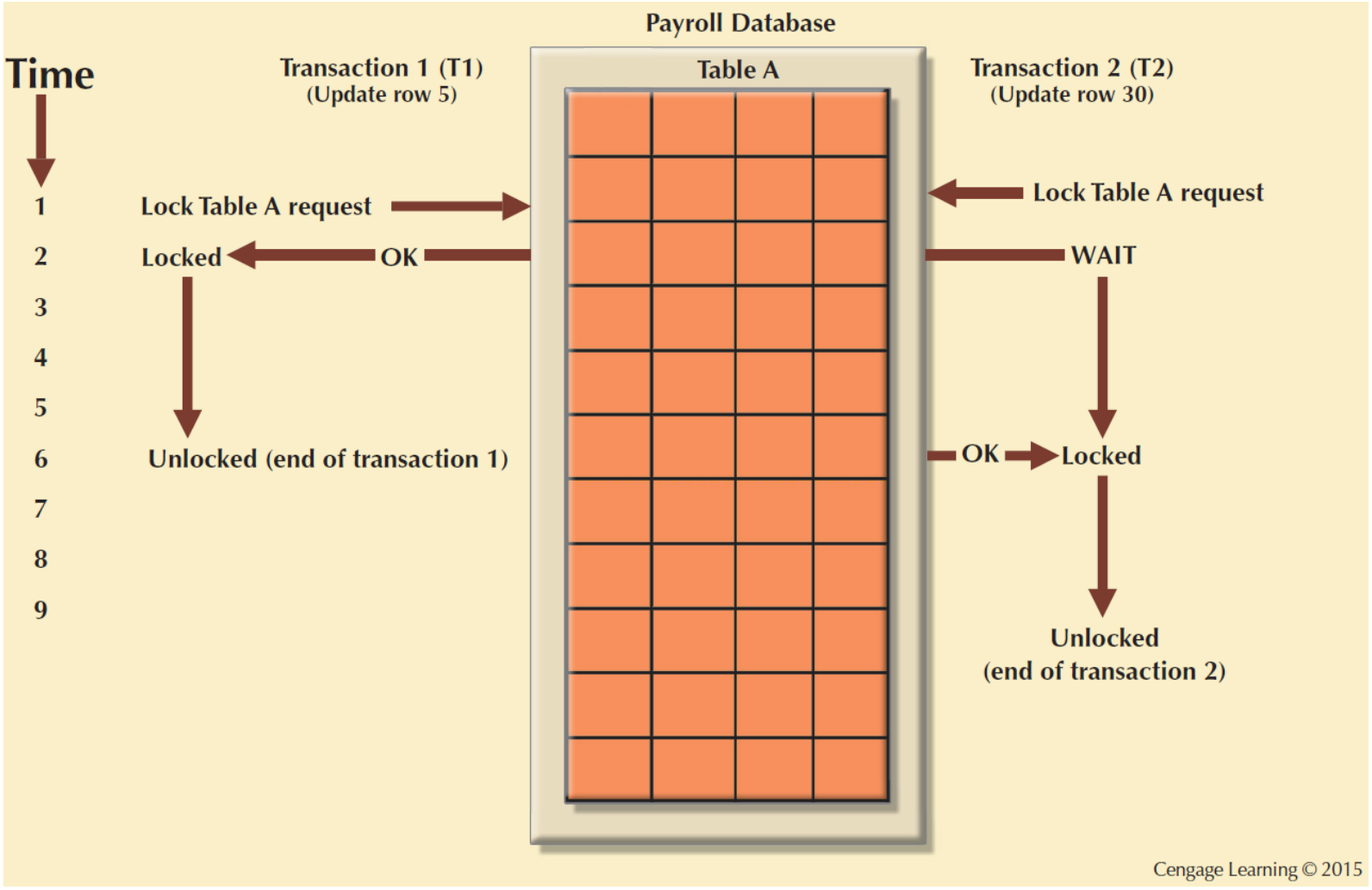
# Lock Granularity

- 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**

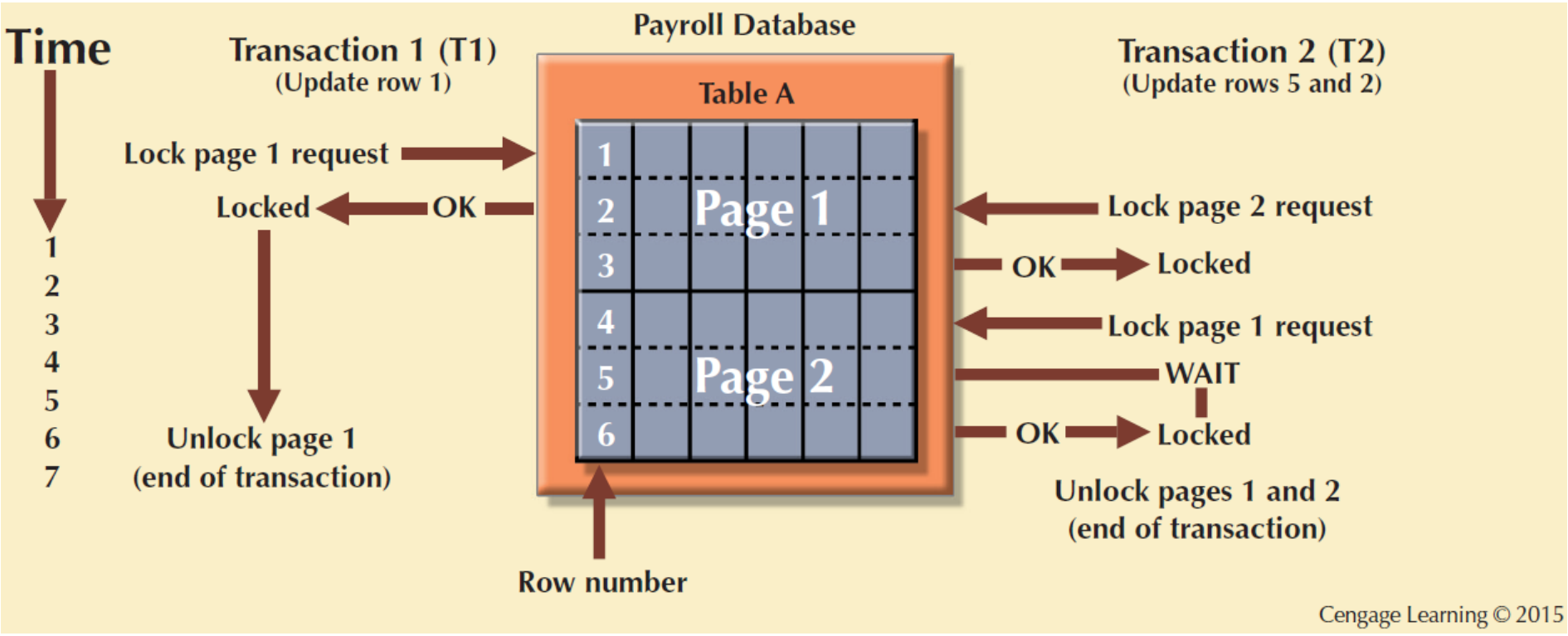
# Figure 10.3 - Database-Level Locking Sequence



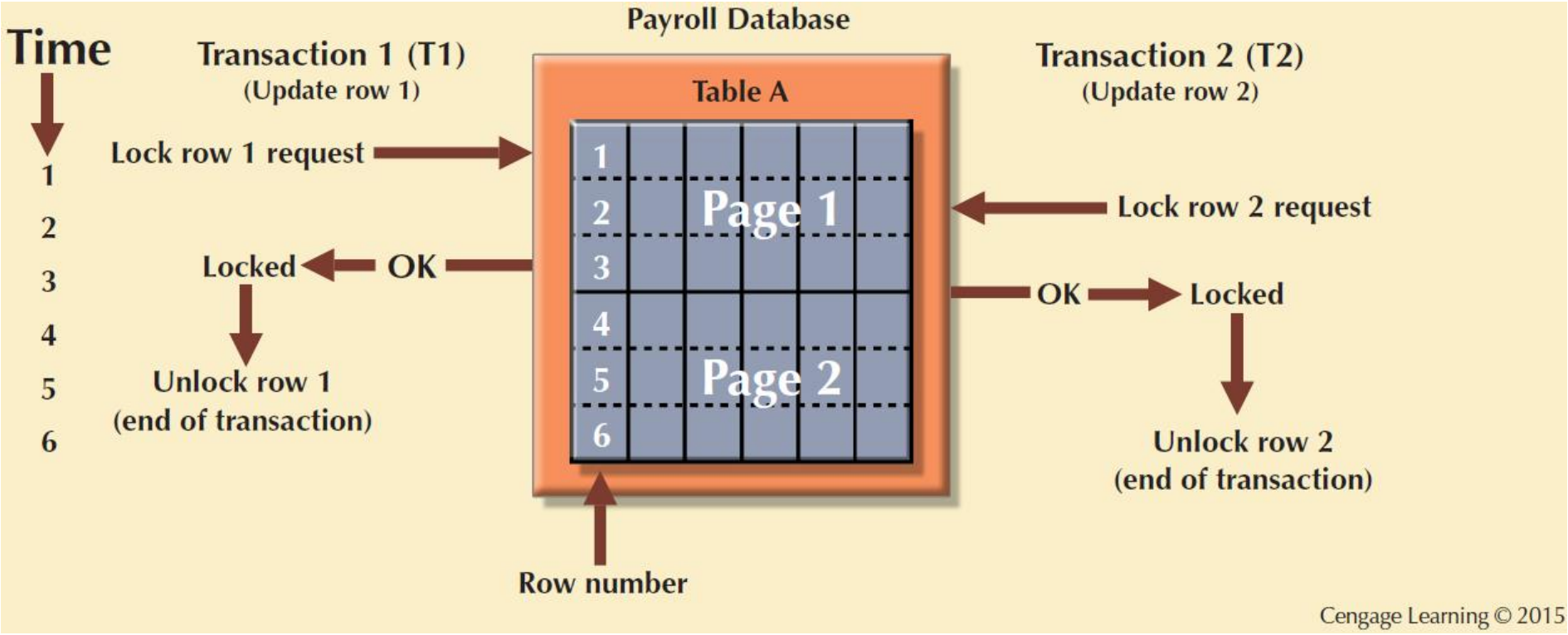
# Figure 10.4 - An Example of a Table-Level Lock



# 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 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**

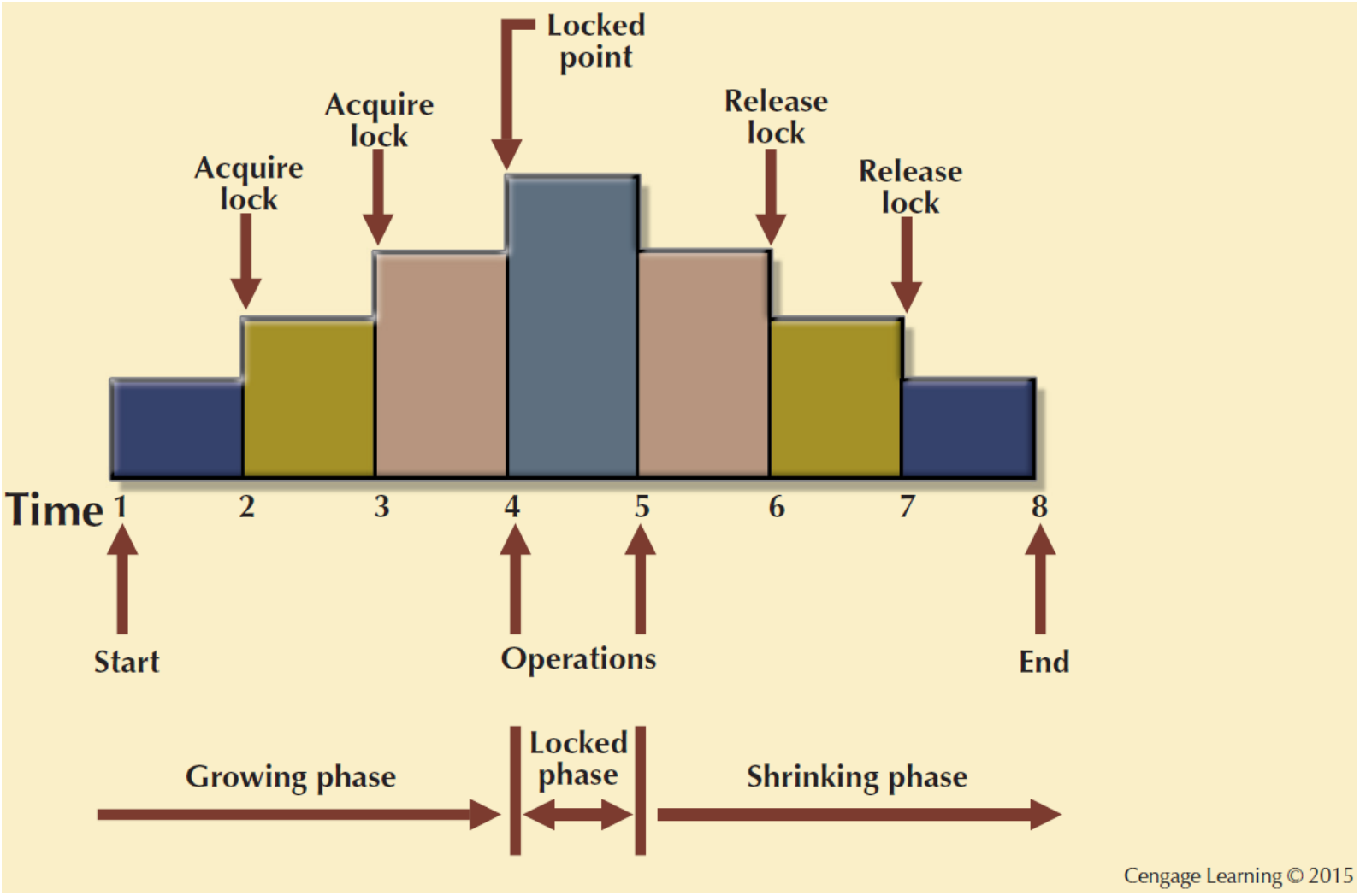
# Two-Phase Locking (2PL)

- 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



# 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 | |
|---|---|---|---|---|
| 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 |
| … | …… | …… | …… | …… |
| … | …… | …… | …… | …… |
| … | …… | …… | …… | …… |
| … | …… | …… | …… | …… |


Cengage Learning © 2015

# 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

| 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.<br>• T2 is rescheduled using the same timestamp. |
| T2 (19562545) | T1 (11548789) | • T2 dies (rolls back).<br>• T2 is rescheduled using the same times tamp. | • T2 waits until T1 is completed and T1 releases its locks. |

Cengage Learning © 2015

# 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

| | Isolation Level | Allowed | | | Comment |
|---|---|---|---|---|---|
| | | Dirty Read | Nonrepeatable Read | Phantom Read | |
| Less restrictive  More 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. |

Cengage Learning © 2015

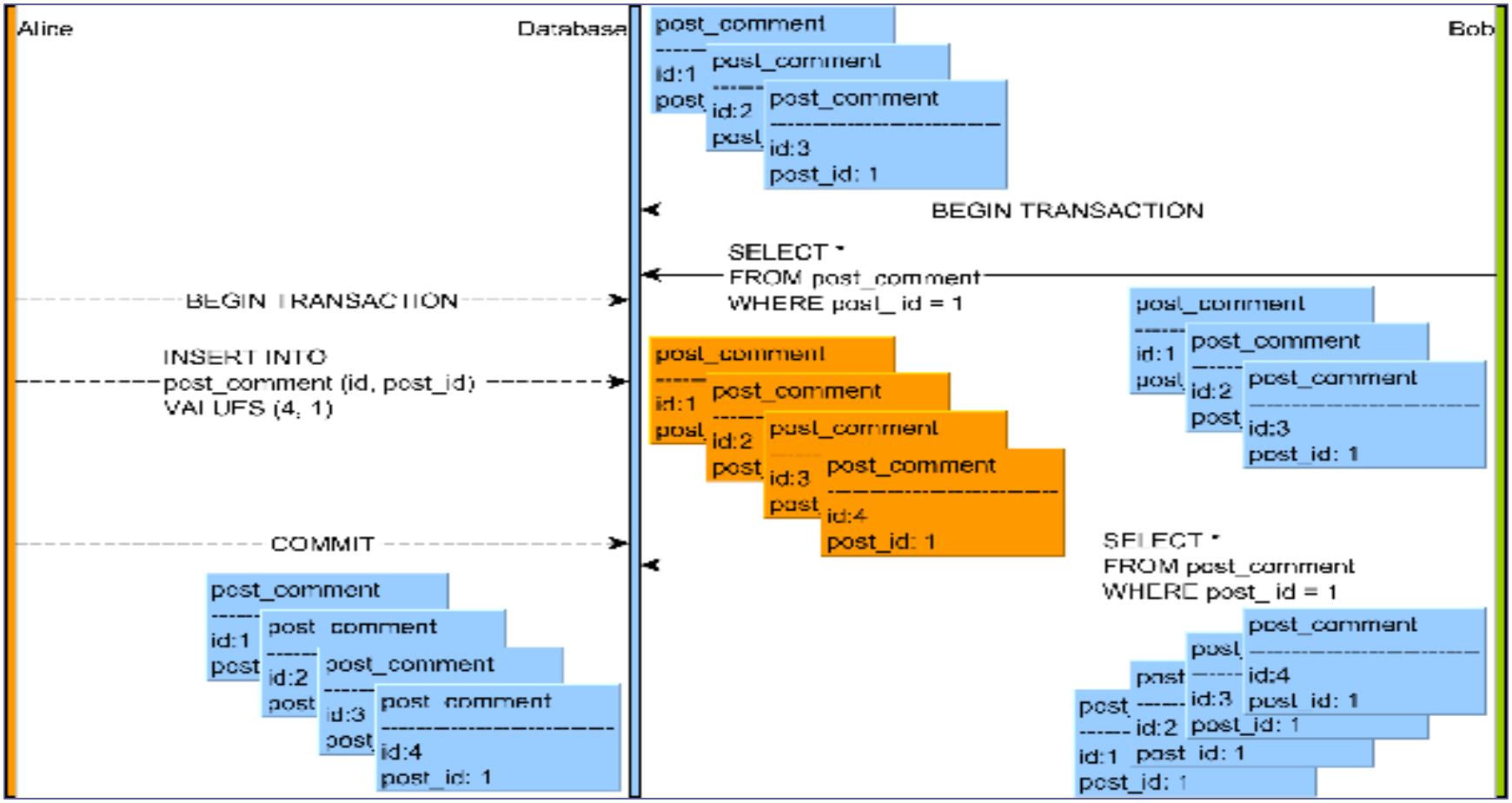
# Isolation Levels

- Read committed is an isolation level that guarantees that any data read was *committed* at the moment is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. IT makes no promise whatsoever that if the transaction re-issues the read, will find the *Same*data, data is free to change after it was read.

- Repeatable read is a higher isolation level, that in addition to the guarantees of the read committed level, it also guarantees that any data read *cannot change*, if the transaction reads the same data again, it will find the previously read data in place, unchanged, and available to read.

- The next isolation level, serializable, makes an even stronger guarantee: in addition to everything repeatable read guarantees, it also guarantees that *no **new** data* can be seen by a subsequent read.

# Phantom Read



# Phantom Read

In the diagram above, the flow of statements goes like this:

1. Alice and Bob start two database transactions.
2. Bob's reads all the post_comment records associated with the post row with the identifier value of 1.
3. Alice adds a new post_comment record which is associated with the post row having the identifier value of 1.
4. Alice commits her database transaction.
5. If Bob's re-reads the post_comment records having the post_id column value equal to `, he will observe a different version of this result set.

This phenomenon is typical for both Read Uncommitted, Read Committed and Repeatable Read isolation levels. The default isolation level being either Read Committed (Oracle, SQL Server or PostgreSQL) or Repeatable Read (MySQL) does not prevent this anomaly.

Nevertheless, preventing this anomaly is fairly simple. All you need to do is use a higher isolation level like Serializable. Or, if the underlying RDBMS supports predicate locks, you can simply lock the range of records using a share (read) lock or an exclusive (write) range lock.

# 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

**Deferred-write technique** or **deferred update**

- 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