Concurrency Control Methods

Dr. Rabah Mokhtari

Department of Computer Science,
UNIVERSITY OF M'SILA

September 27, 2025

Content

Concurrency Control with Locking Methods

Locking Methods Overview

Most Common Technique

Locking methods are one of the most common techniques used in concurrency control because they facilitate the isolation of data items used in concurrently executing transactions.

How Locks Work

- A **lock** guarantees exclusive use of a data item to a current transaction
- Transaction T2 cannot access data item currently used by T1
- Lock acquired prior to data access, released when transaction completes
- Enables sequential access to shared data resources

Pessimistic Locking

- Assumes conflict between transactions is likely
- Proactive approach to concurrency control
- Prevents conflicts before they occur

Locking Necessity and Implementation

Why Locks Are Required

- Data consistency cannot be guaranteed during transaction execution
- Database may be in temporary inconsistent state during updates
- Locks prevent other transactions from reading inconsistent data
- Maintains isolation property during multi-step transactions

Automatic Lock Management

- Most multi-user DBMSs automatically initiate and enforce locking
- **Lock manager** responsible for assigning and policing locks
- Handles all lock information for concurrent transactions
- Transparent to application developers and users

Lock Manager Responsibilities

- Assigns appropriate locks to transactions
- Monitors lock conflicts and resolves deadlocks
- Releases locks upon transaction completion
- Ensures efficient lock utilization

Lock Granularity

Definition

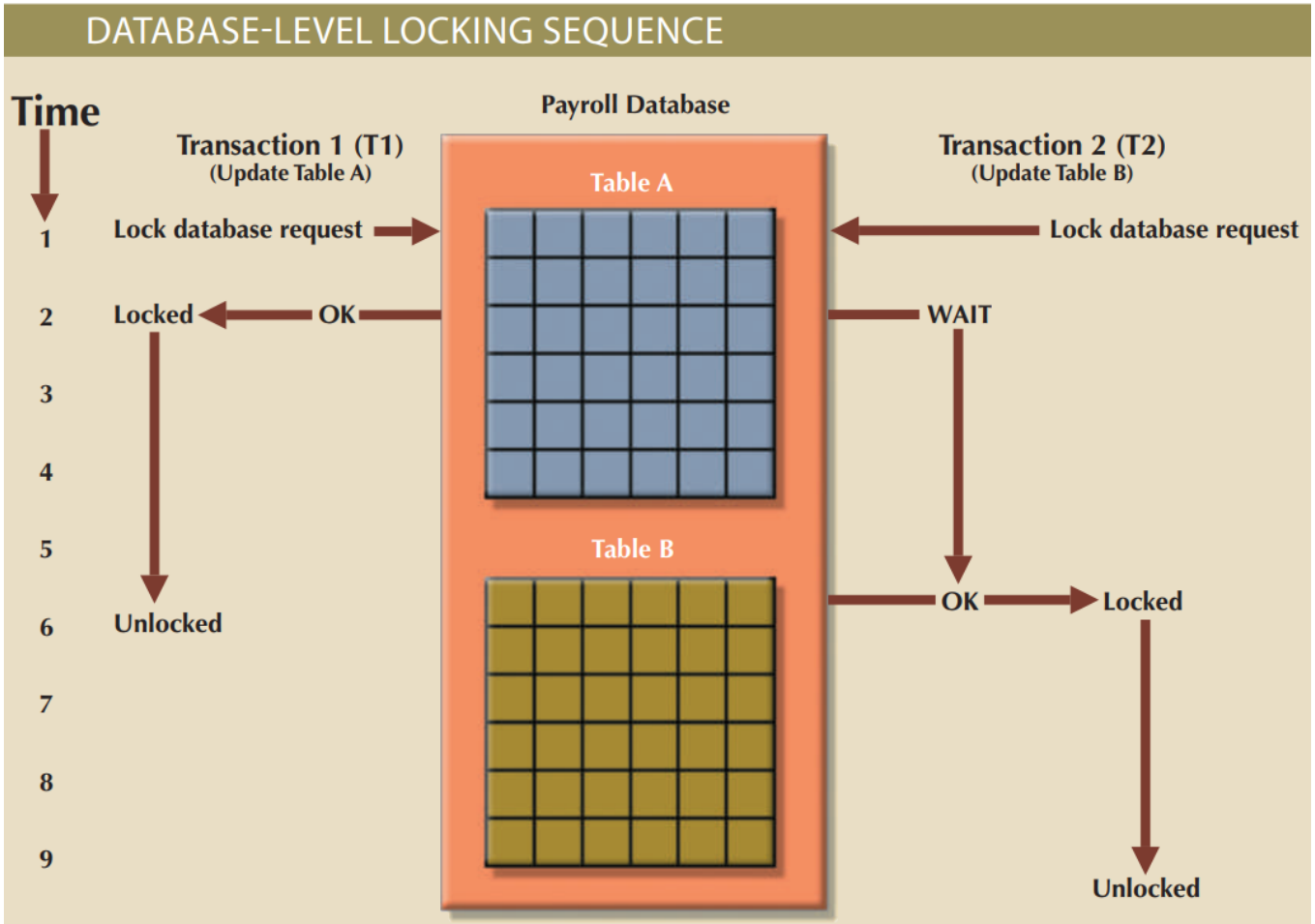
Lock granularity indicates the level of lock use. Locking can take place at the following levels:

- Database level
- Table level
- Page level
- Row level
- Field (attribute) level

Database-Level Lock

- Entire database is locked
- Prevents use of any tables by other transactions
- Suitable for batch processes
- Unsuitable for multiuser DBMSs

Database-Level Lock Illustration



Database-Level Lock Illustration (Cont.)

Performance Impact

- Very slow data access in multiuser environments
- Thousands of transactions must wait sequentially
- Even transactions using different tables cannot proceed concurrently

Key Limitation

Transactions T1 and T2 cannot access the same database concurrently even when they use different tables.

Appropriate Use Cases

- Batch processing operations
- Database maintenance tasks
- Backup operations
- Major schema changes

Table-Level Lock

Definition

In a table-level lock, the entire table is locked, preventing access to any row by transaction T2 while transaction T1 is using the table.

Locking Mechanism

- If transaction requires access to several tables, each table may be locked
- Two transactions can access same database if they access different tables
- Less restrictive than database-level locks

Performance Issues

- Causes traffic jams when many transactions wait for same table
- Problematic when transactions need different parts of same table
- Forces delays even when transactions wouldn't interfere with each other
- Not suitable for multiuser DBMSs

Table-Level Lock Illustration

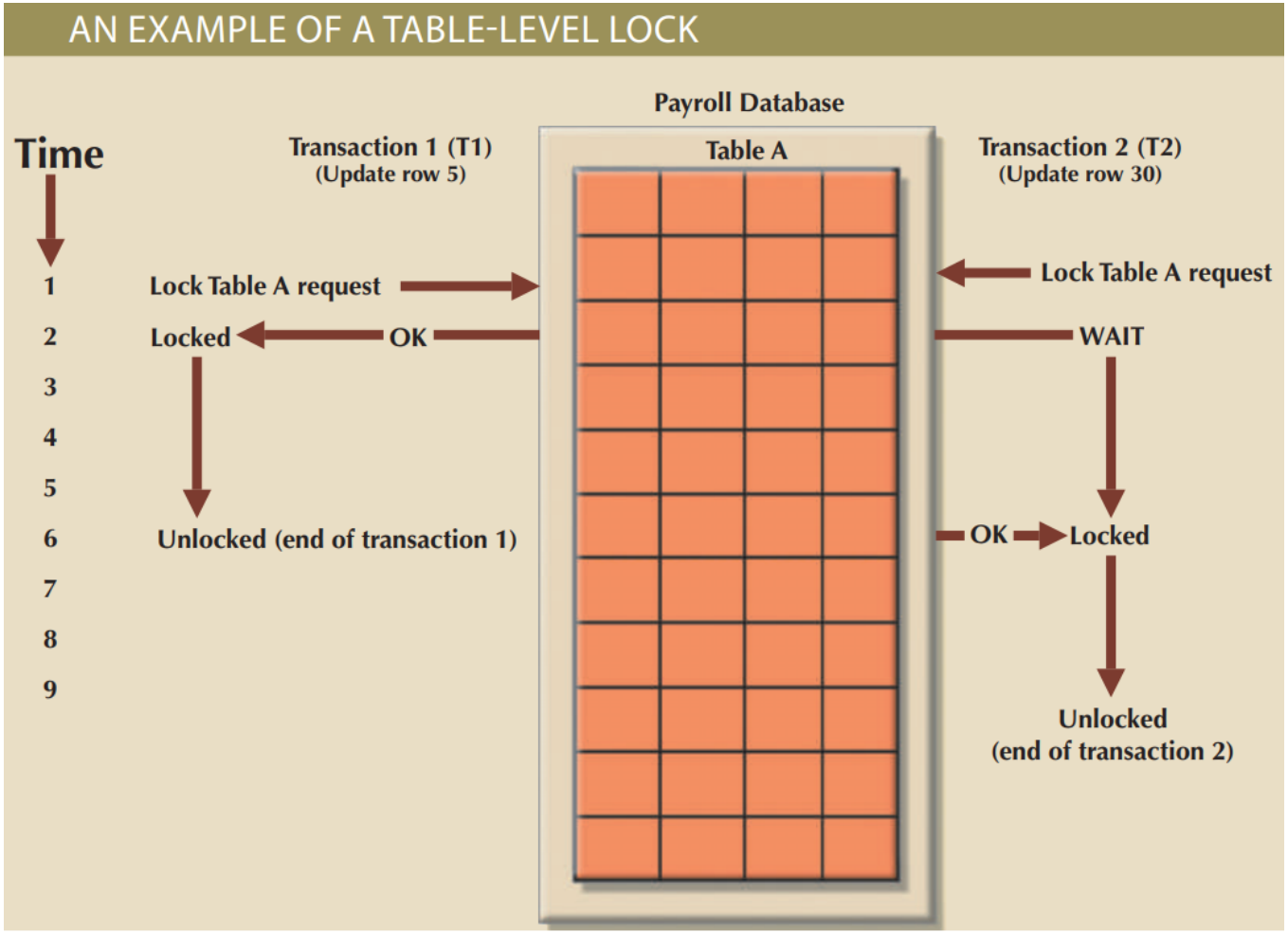


Figure: Table-level lock prevents concurrent row access within same table

Table-Level Lock Illustration (Cont.)

Key Limitation

Transactions T1 and T2 cannot access the same table even when they try to use different rows; T2 must wait until T1 unlocks the table.

Concurrency Impact

- Reduces parallelism for unrelated operations
- Creates unnecessary waiting queues
- Limits throughput in high-volume systems
- Better than database-level but still inefficient

Page-Level Lock

Definition

In a page-level lock, the DBMS locks an entire diskpage. A diskpage (or page) is a directly addressable section of a disk with fixed size (4K, 8K, or 16K).

Page Characteristics

- Fixed size: 4K, 8K, or 16K
- Entire page must be read/written even for small updates
- Table can span several pages
- Page can contain rows from one or more tables

Current Usage

- Page-level locks are most frequently used in multiuser DBMSs
- Balances granularity and performance
- Better concurrency than table-level locks

Page-Level Lock Illustration

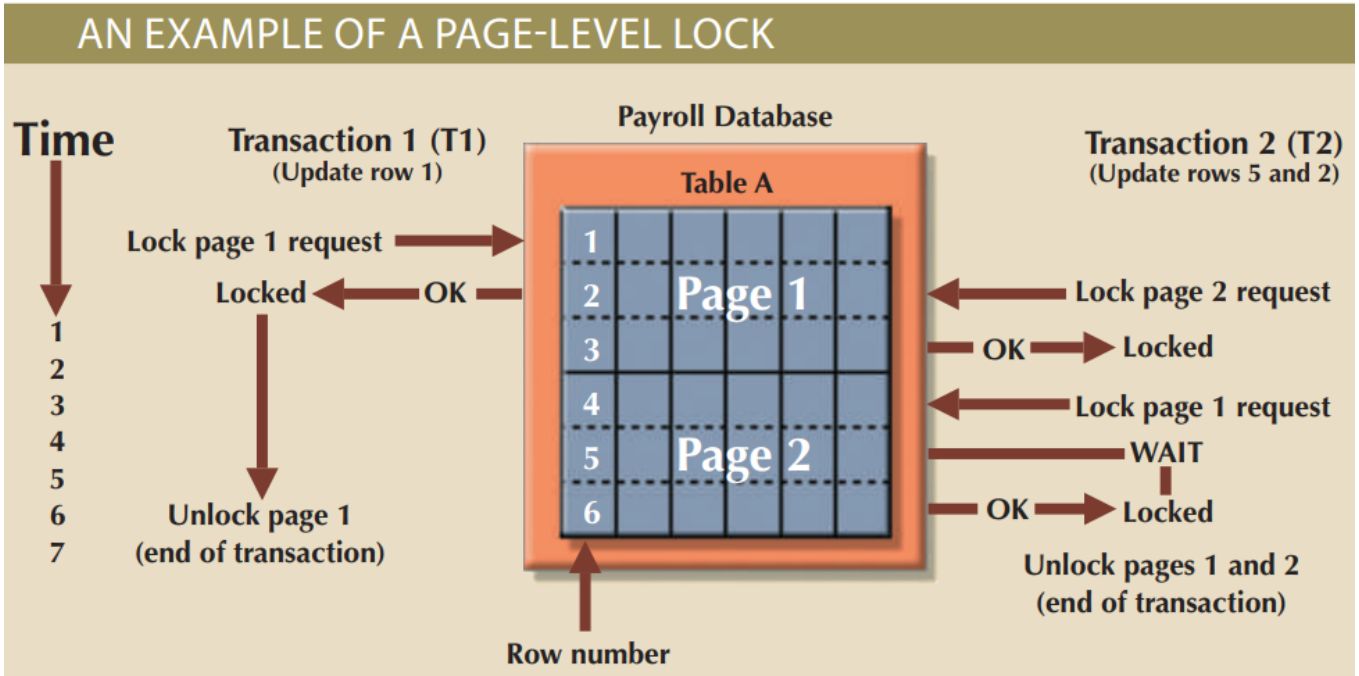


Figure: Page-level lock allows concurrent table access with page-level isolation

Concurrent Access

- T1 and T2 can access same table while locking different diskpages
- Enables better parallelism than table-level locks
- More granular control than database-level locks

Row-Level Lock

Definition

A row-level lock is much less restrictive than previous locking methods. The DBMS allows concurrent transactions to access different rows of the same table even when the rows are located on the same page.

Advantages

- Maximum data availability and concurrency
- Different transactions can access different rows simultaneously
- Works even when rows are on the same disk page
- Highest granularity level for locking

Management Overhead

- Requires high overhead due to many individual locks
- Lock exists for each row involved in conflicting transaction
- Modern DBMSs automatically escalate row locks to page locks
- Escalation occurs when multiple locks requested on same page

Row-Level Lock Illustration

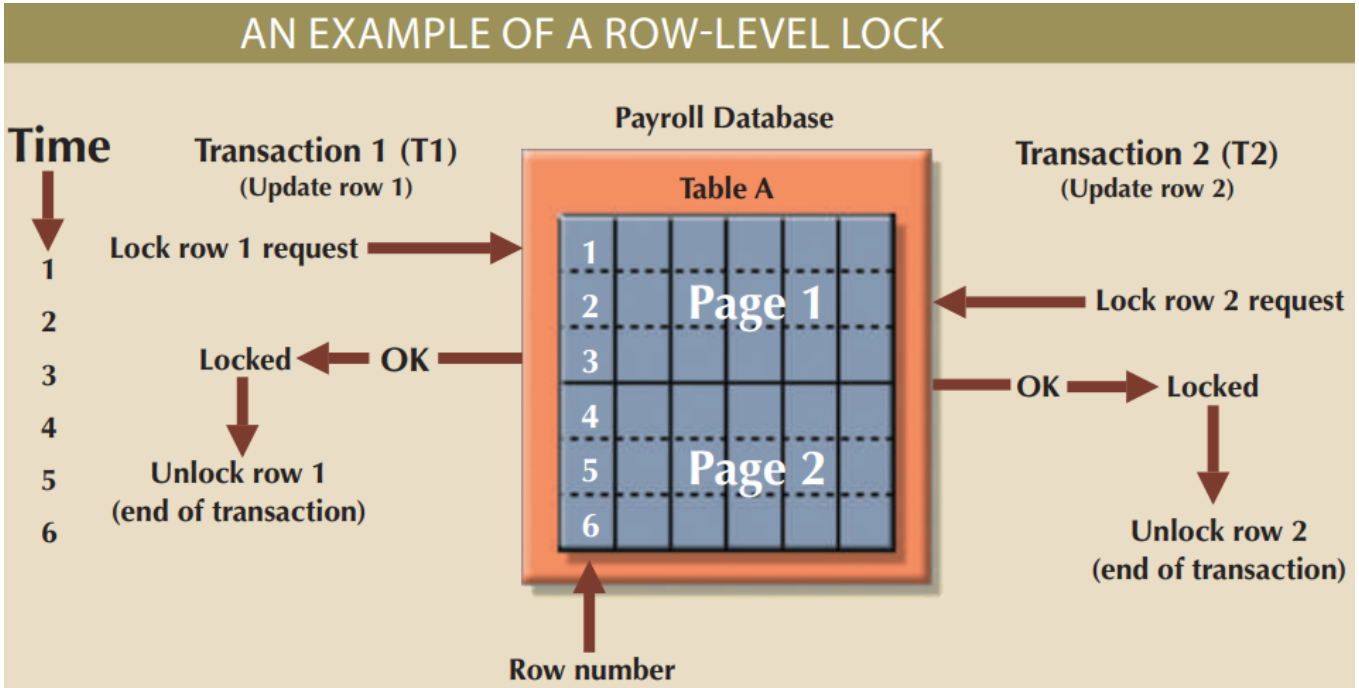


Figure: Row-level lock enables maximum concurrency within same table

Note

Note that both transactions can execute concurrently, even when the requested rows are on the same page. T2 must wait only if it requests the same row as T1

Row-Level Lock Illustration (Cont.)

Concurrency Benefits

- Maximum parallelism for transactions
- Minimal waiting time for unrelated operations
- Ideal for high-transaction volume environments
- Best for OLTP (Online Transaction Processing) systems

Automatic Lock Escalation

- DBMS escalates row locks to page locks when efficient
- Reduces lock management overhead
- Balances concurrency with system performance
- Transparent to applications and users

Field-Level Lock

Definition

The field-level lock allows concurrent transactions to access the same row as long as they require the use of different fields (attributes) within that row.

Maximum Flexibility

- Highest granularity of locking available
- Multiple transactions can access same row simultaneously
- Each transaction locks only the specific fields it needs
- Maximum possible concurrency for data access

Practical Limitations

- Rarely implemented in commercial DBMS
- Requires extremely high computer overhead
- Management complexity outweighs benefits
- Row-level locking is more practical and useful

Field-Level Lock Practicality

Why Field-Level Locks Are Rare

- Excessive overhead for minimal practical benefit
- Most applications don't need field-level concurrency
- Row-level locking satisfies 99% of use cases
- Implementation complexity doesn't justify benefits

Real-World Usage

- Primarily theoretical concept
- Rarely found in commercial database systems
- Row-level locks provide sufficient granularity
- Better to optimize row-level locking performance

Performance Consideration

The computational cost of managing field-level locks outweighs the concurrency benefits for most practical database applications.

Lock Types Overview

Two Main Lock Types

Regardless of the level of granularity, DBMS may use different lock types or modes:

- **Binary locks** - Simple locked/unlocked states
- **Shared/Exclusive locks** - More sophisticated read/write locking

Automatic Lock Management

- Lock operations automatically managed by DBMS
- User does not manually lock/unlock data items
- Default locking mechanism provided by DBMS
- Override possible with SQL commands like LOCK TABLE

Binary Locks

Simple Two-State System

- Only two states: locked (1) or unlocked (0)
- Locked object cannot be used by other transactions
- Unlocked object available to any transaction
- Every operation requires lock/unlock cycle

Advantages and Limitations

- **Advantage**: Eliminates lost update problem
- **Limitation**: Too restrictive for optimal concurrency
- Prevents two transactions from reading same object
- Even when no updates occur (unnecessary restriction)

Concurrency Insight

Concurrency conflicts occur only when two transactions execute concurrently and one of them updates the database.

Binary Lock Example

Solving Lost Update Problem

The binary locking technique eliminates the lost update problem because the lock is not released until the WRITE statement is completed.

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	

Binary Locking Example (Cont.)

How Binary Locks Work

- PROD_QOH value cannot be used until properly updated
- Lock remains active throughout entire update process
- Prevents interference between concurrent transactions
- Ensures data consistency during write operations

Restrictive Nature of Binary Locks

- Considered too restrictive for optimal concurrency
- DBMS prevents two transactions from reading same object
- Even when both transactions are read-only
- Unnecessary restriction since no concurrency problems occur

Shared/Exclusive Locks

Three Lock States

- **Unlocked**: No locks held
- **Shared (read)**: Multiple transactions can read concurrently
- **Exclusive (write)**: Single transaction has exclusive write access

Lock Granting Rules

- **Shared lock**: Granted for reads when no exclusive lock exists
- **Exclusive lock**: Granted for writes when no other locks exist
- **Mutual exclusive rule**: Only one exclusive lock per object

Blocking Behavior

- Shared lock blocks exclusive (write) locks
- Exclusive lock blocks all other locks
- Shared locks allow concurrent reading

Shared/Exclusive Lock Examples

Safe Concurrent Operations

- Multiple transactions can hold shared locks simultaneously
- Safe for read-only transactions to execute concurrently
- Example: T1 and T2 can both read data item X with shared locks

Exclusive Lock Requirements

- Exclusive lock requires no other locks on the data item
- If shared lock exists, exclusive lock must wait
- Transactions wait until previous locks are released

Increased Overhead

- Lock manager must track lock types
- Three operations: READ_LOCK, WRITE_LOCK, UNLOCK
- Support for lock upgrades/downgrades

Locking Problems and Solutions

Major Locking Problems

- **Non-serializable schedules**: Transaction order may not be serializable
- **Deadlocks**: Two transactions wait indefinitely for each other
- Similar to traffic gridlock in database operations

Solutions

- **Serializability**: Achieved through two-phase locking protocol
- **Deadlock management**: Detection and prevention techniques
- Both problems can be effectively managed

Coming Next

- Two-phase locking protocol for serializability
- Deadlock detection and prevention methods
- Advanced concurrency control techniques

Two-Phase Locking Overview

Definition

Two-phase locking (2PL) defines how transactions acquire and relinquish locks. It guarantees serializability but does not prevent deadlocks.

The Two Phases

1. **Growing phase**: Transaction acquires all required locks without unlocking any data
2. **Shrinking phase**: Transaction releases all locks and cannot obtain new locks

Locked Point

- Point when transaction acquires all required locks
- No more locks can be acquired after this point
- Data modification occurs only after locked point

Two-Phase Locking Rules

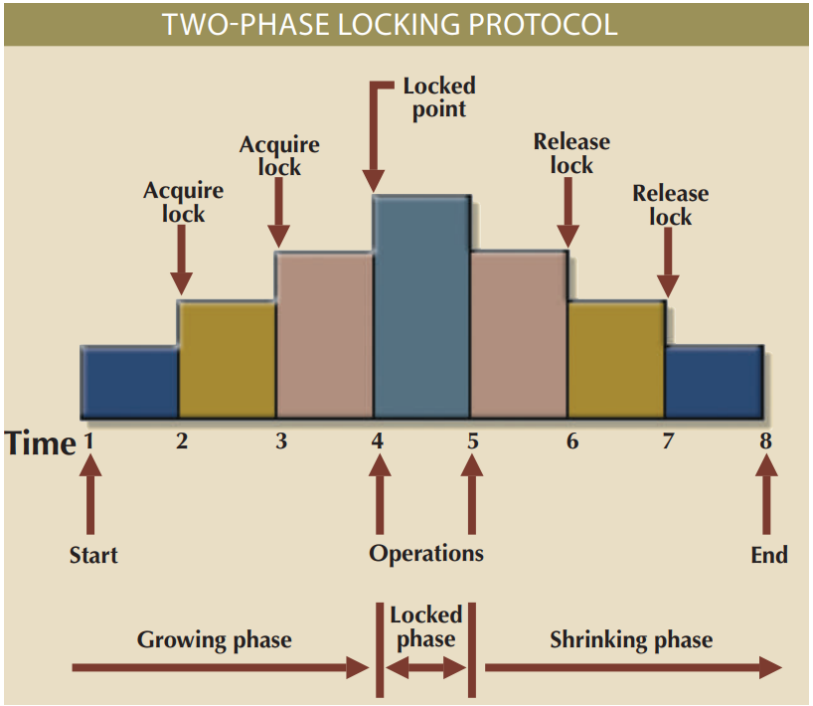
Protocol Rules

- Two transactions cannot have conflicting locks
- No unlock operation can precede a lock operation in same transaction
- No data is affected until all locks are obtained (locked point)
- Strict ordering of lock acquisition and release

Transaction Sequence

1. Acquire all necessary locks (growing phase)
2. Reach locked point - all locks acquired
3. Modify data as required
4. Release all locks (shrinking phase)
5. Transaction completes

Two-Phase Locking Illustration



Process Flow

- Lock acquisition phase (growing)
- Locked point reached
- Data modification occurs
- Lock release phase (shrinking)

Two-Phase Locking Implications

Process Flow

- Lock acquisition phase (growing)
- Locked point reached
- Data modification occurs
- Lock release phase (shrinking)

Advantages

- Guarantees serializability of transactions
- Ensures consistent transaction scheduling
- Prevents non-serializable execution orders

Disadvantages

- Increases transaction processing cost
- May cause additional deadlocks
- Reduced concurrency due to lock holding
- Potential for longer lock wait times

Deadlock Control Overview

Three Basic Techniques

1. **Deadlock prevention**: Abort transactions when deadlock possible
2. **Deadlock detection**: Periodically test for deadlocks
3. **Deadlock avoidance**: Require all locks upfront

Selection Criteria

- Choice depends on database environment
- Probability of deadlocks
- System response time requirements
- Transaction characteristics and workload

Deadlock Illustration

Deadlock Creation Analysis

- This example demonstrates how a deadlock condition is created
- Example uses two concurrent transactions for demonstration
- Real-world DBMS execute many more transactions simultaneously
- Increased transaction count raises probability of deadlocks
- Deadlocks possible only with exclusive lock requests
- No deadlock condition can exist among shared locks only

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

Deadlock Prevention

How It Works

- Transaction aborted when deadlock possibility detected
- All changes rolled back, locks released
- Transaction rescheduled for execution
- Avoids conditions that lead to deadlocking

When to Use

- Recommended when probability of deadlocks is high
- Proactive approach to deadlock management
- Prevents deadlocks before they occur
- May cause unnecessary transaction aborts

Trade-off

- Sacrifices some transactions to prevent system-wide issues
- Better than dealing with actual deadlock situations
- Conservative but safe approach

Deadlock Detection

How It Works

- DBMS periodically tests database for deadlocks
- If deadlock found, "victim" transaction is aborted
- Aborted transaction rolled back and restarted
- Other transaction continues execution

When to Use

- Recommended when probability of deadlocks is low
- Reactive approach - deals with deadlocks as they occur
- All current DBMSs support deadlock detection
- Most common approach in transactional databases

Advantages

- Less intrusive than prevention
- Only acts when deadlock actually occurs
- Good balance of performance and safety

Deadlock Avoidance

How It Works

- Transaction must obtain all locks before execution
- Locks obtained in succession (serial assignment)
- Avoids rolling back of conflicting transactions
- Preemptive lock acquisition strategy

When to Use

- When response time is not high priority
- In environments with predictable lock patterns
- Some DBMSs use blend with prevention techniques
- Used for data warehouses or XML data in some systems

Performance Impact

- Serial lock assignment increases response times
- Reduces concurrency and system throughput
- Conservative approach with performance trade-offs

Current DBMS Practices

Industry Standards

- All current DBMSs support deadlock detection
- Most common for transactional databases
- Some systems use prevention/avoidance blends
- Choice depends on data type and workload

Application by Data Type

- **Transactional databases**: Primarily detection
- **Data warehouses**: Prevention/avoidance blends
- **XML data**: Specialized avoidance techniques
- **Real-time systems**: Customized approaches

Key Consideration

The optimal deadlock control strategy depends on specific database environment characteristics and performance requirements.

Concurrency Control with Time Stamping Methods

Time Stamping Approach

Basic Concept

- Assigns global, unique time stamp to each transaction
- Time stamps must have: uniqueness and monotonicity
- Uniqueness: No equal time stamp values can exist
- Monotonicity: Time stamp values always increase

Transaction Execution

- All operations within same transaction have same time stamp
- DBMS executes conflicting operations in time stamp order
- Ensures serializability of transactions
- Conflicting transactions: one stopped, rolled back, rescheduled

Disadvantages

- Each database value requires two additional time stamp fields
- Increases memory needs and processing overhead
- Many transactions may need to be stopped and restamped
- Demands significant system resources

Wait/Die and Wound/Wait Schemes

Time Stamping Methods

Time stamping methods are used to manage concurrent transaction execution. In this section, you will learn about two schemes used to decide which transaction is rolled back and which continues executing: the wait/die scheme and the wound/wait scheme.

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.

Wait/Die Scheme

Scenario Setup

- T1 (older): time stamp 11548789
- T2 (newer): time stamp 19562545
- Lower time stamp = older transaction

Wait/Die Rules

- **Older transaction waits**: If requesting lock is older, waits for younger
- **Younger transaction dies**: If requesting lock is younger, rolls back
- Younger transaction rescheduled with same time stamp
- Older transaction waits for younger to complete

Behavior Summary

In wait/die scheme, the older transaction waits for the younger one to complete and release its locks.

Wound/Wait Scheme

Scenario Setup

- Same transaction setup: T1 (older), T2 (newer)
- Different conflict resolution approach

Wound/Wait Rules

- **Older transaction wounds**: If requesting lock is older, rolls back younger
- **Younger transaction waits**: If requesting lock is younger, waits for older
- Younger, preempted transaction rescheduled with same time stamp
- Older transaction rolls back younger transaction

Behavior Summary

In wound/wait scheme, the older transaction rolls back the younger transaction and reschedules it.

Scheme Comparison and Deadlock Prevention

Commonality

- Both schemes: one transaction waits for another
- Both ensure serializable transaction execution
- Both require transaction rollback and rescheduling

Multiple Lock Requests

- Transactions often request multiple locks
- Potential for indefinite waiting (deadlock)
- Each lock request has associated time-out value
- If time-out expires, transaction rolled back

Deadlock Prevention

- Time-out mechanism prevents indefinite waiting
- Ensures system progress despite conflicts
- Critical for maintaining database availability
- Balances fairness with system efficiency

Concurrency Control with Optimistic Methods

Optimistic Approach Overview

Basic Assumption

The optimistic approach is based on the assumption that the majority of database operations do not conflict. It requires neither locking nor time stamping techniques.

Transaction Execution

- Transaction executed without restrictions until committed
- Moves through two or three phases: read, validation, and write
- Suitable for environments with low update contention
- Minimal overhead during normal operation

Key Characteristic

A transaction is executed without restrictions until it is committed, unlike locking methods that restrict access proactively.

Optimistic Method Phases

1. Read Phase

- Transaction reads database and executes computations
- Makes updates to a private copy of database values
- All update operations recorded in temporary update file
- Temporary file not accessed by other transactions

2. Validation Phase

- Transaction validated for integrity and consistency
- Positive validation test: proceeds to write phase
- Negative validation test: transaction restarted, changes discarded
- Ensures changes won't affect database integrity

3. Write Phase

- Changes permanently applied to database
- Only occurs after successful validation
- Atomic application of all transaction changes

Optimistic Method Applications

Suitable Environments

- Read or query database systems with few updates
- Environments with low conflict probability
- Applications with predominantly read operations
- Systems where conflicts are rare exceptions

Heavy DBMS Environments

- Deadlock management is crucial DBMS function
- DBMS uses combination of discussed techniques
- May include variations and hybrid approaches
- Choice depends on specific workload characteristics

Next Topic: Transaction Isolation Levels

- Important for understanding transaction management
- Defined in ANSI SQL 1992 standard
- Critical for database consistency and performance

ANSI Levels of Transaction Isolation

ANSI Transaction Isolation Levels

Definition

Transaction isolation levels refer to the degree to which transaction data is "protected or isolated" from other concurrent transactions. The ANSI SQL standard (1992) defines transaction management based on these isolation levels.

Types of Read Operations

- **Dirty read**: Reading data that is not yet committed
- **Nonrepeatable read**: Same row read at different times yields different results
- **Phantom read**: Same query at different times yields additional rows

ANSI Transaction Isolation Levels (Cont.)

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.

Figure: ANSI SQL Transaction Isolation Levels and their characteristics

Read Committed Isolation Level

Characteristics

- Forces transactions to read only committed data
- Default mode for most databases (Oracle, SQL Server)
- Uses exclusive locks on data during writes
- Other transactions wait until original transaction commits

Behavior

- Prevents dirty reads
- Allows nonrepeatable reads and phantom reads
- Good balance of performance and consistency
- Most commonly used in production systems

Locking Mechanism

- Exclusive locks prevent reading of uncommitted data
- Ensures data consistency for committed data only
- Suitable for most OLTP applications

Repeatable Read Isolation Level

Characteristics

- Ensures queries return consistent results
- Uses shared locks to prevent updates after reads
- Other transactions cannot update rows after original query reads them
- May still allow phantom reads for new rows

Behavior

- Prevents dirty reads and nonrepeatable reads
- Allows phantom reads (new rows appear in subsequent reads)
- Higher consistency than Read Committed
- Increased locking overhead

Phantom Read Issue

- New rows read because they didn't exist during first query
- Shared locks don't prevent insertion of new rows
- Consistent for existing rows but not for new additions

Serializable Isolation Level

Most Restrictive Level

- Most restrictive level defined by ANSI SQL standard
- Highest level of data consistency
- Prevents dirty reads, nonrepeatable reads, and phantom reads
- Maximum locking requirements

Important Note

- Even with Serializable level, deadlocks are always possible
- Most databases use deadlock detection approach
- Deadlocks detected during transaction validation phase
- Transactions rescheduled when deadlocks occur

Trade-off

- Highest consistency but lowest concurrency
- Significant performance impact
- Used when absolute data consistency is critical

Isolation Level Purpose and Syntax

Purpose of Different Levels

- Increase transaction concurrency while maintaining consistency
- Levels range from least restrictive (Read Uncommitted) to most restrictive (Serializable)
- Higher isolation = more locks = better consistency but lower performance

ANSI SQL Syntax

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED
SQL STATEMENT;
COMMIT TRANSACTION;
```

Database-Specific Implementations

- SQL Server: `SET TRANSACTION ISOLATION LEVEL`
- Oracle: Consistent statement-level reads for Read Committed/Repeatable Read
- MySQL: `START TRANSACTION WITH CONSISTENT SNAPSHOT`

Transaction Management Complexity

Complex Subject

- Transaction management involves complex techniques
- Databases use various methods for concurrent execution
- Balance between consistency and performance
- Different isolation levels for different requirements

Database Recovery Importance

- Sometimes necessary to restore database to consistent state
- Recovery techniques complement transaction management
- Essential for maintaining data integrity f comprehensive database management strategy

Key Takeaway

Transaction isolation levels provide a spectrum of consistency vs. performance trade-offs, allowing database designers to choose the appropriate level for their specific application requirements.