Aktu

Database Management System

BCS-501

Unit-5

Syllabus

Concurrency Control Techniques: Concurrency Control, Locking Techniques for Concurrency Control, Time Stamping Protocols for Concurrency Control, Validation Based Protocol, Multiple Granularity, Multi Version Schemes, Recovery with Concurrent Transaction, Case Study of Oracle.

Content of Unit-5

- Concurrency Control
- Concurrency Problems
- Lock-Based Protocols in Concurrency Control
- Time Stamping Protocols for C.C. - 2PYQ
- Validation Based Protocol - 3PYQ
- Multiple Granularity
- Multi Version Schemes - 2PYQ
- Recovery with Concurrent Transaction
- Case Study of Oracle
- Graph Based Locking Protocol - 1PYQ

Concurrency Control

Concurrency control is a mechanism in database systems to manage simultaneous transactions while ensuring data consistency, integrity, and isolation. It prevents problems that can arise when multiple transactions access and manipulate the same data concurrently.

Importance of Concurrency Control

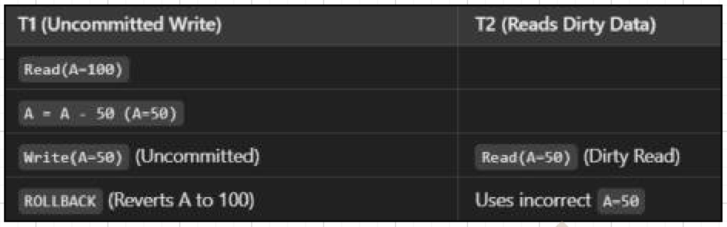
1. **Data Consistency:** Ensures the database remains accurate and reliable despite simultaneous transactions.
2. **Isolation:** Guarantees that one transaction's intermediate states are not visible to others.
3. **Fair Resource Sharing:** Allows multiple users to perform transactions efficiently.
4. **Prevention of Anomalies:** Resolves conflicts like lost updates, dirty reads, and other concurrency-related issues.

Concurrency Problems

Concurrency problems occur when multiple transactions execute simultaneously and interact with the same data, leading to unexpected or incorrect outcomes. The most common concurrency problems are:

1. Dirty Read

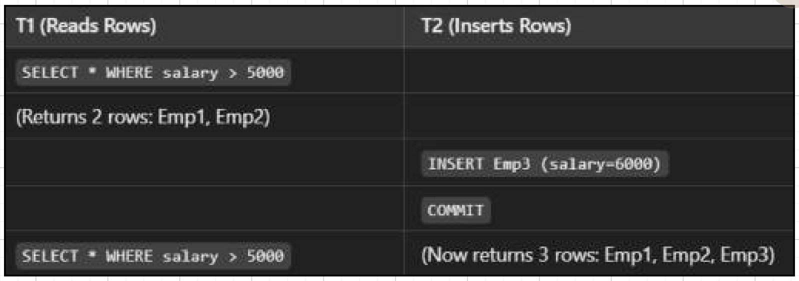
- **Definition:** A transaction reads uncommitted changes made by another transaction.
- **Impact:** Inconsistent and unreliable data.
- **Prevention:** Use the Read Committed isolation level or higher.



T1 (Uncommitted Write)	T2 (Reads Dirty Data)
`Read(A=100)`	
`A = A - 50 (A=50)`	
`Write(A=50)` (Uncommitted)	`Read(A=50)` (Dirty Read)
`ROLLBACK` (Reverts A to 100)	Uses incorrect `A=50`

3. Phantom Read

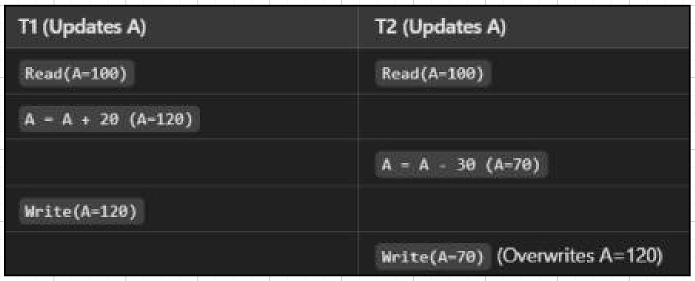
- **Definition:** A transaction re-executes a query and gets a different result set because another transaction has added or removed rows that match the query criteria.
- **Impact:** The result set changes unexpectedly.
- **Prevention:** Use the Serializable isolation level.



T1 (Reads Rows)	T2 (Inserts Rows)
`SELECT * WHERE salary > 5000`	
(Returns 2 rows: Emp1, Emp2)	
	`INSERT Emp3 (salary=6000)`
	`COMMIT`
`SELECT * WHERE salary > 5000`	(Now returns 3 rows: Emp1, Emp2, Emp3)

4. Lost Update

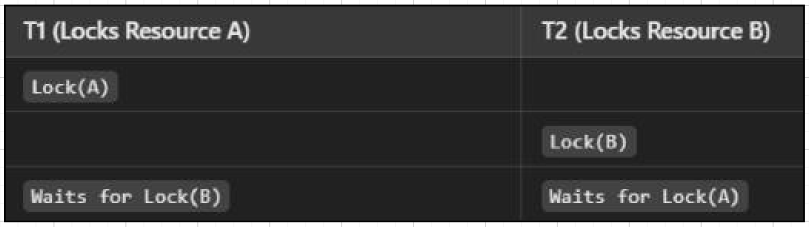
- **Definition:** Two transactions read the same data and update it concurrently, but one update overwrites the other, leading to a loss of data.
- **Impact:** One update is lost, leading to incorrect results.
- **Prevention:** Use locks or the Serializable isolation level.



T1 (Updates A)	T2 (Updates A)
`Read(A=100)`	`Read(A=100)`
`A = A + 20 (A=120)`	
	`A = A - 30 (A=70)`
`Write(A=120)`	
	`Write(A=70)` (Overwrites A=120)

5. Deadlock

- **Definition:** Two or more transactions wait indefinitely for resources locked by each other, creating a circular wait.
- **Impact:** Transactions are unable to proceed, leading to a system stall.
- **Prevention:** Deadlock detection and recovery mechanisms (e.g., timeout, resource ordering).



T1 (Locks Resource A)	T2 (Locks Resource B)
`Lock(A)`	
	`Lock(B)`
`Waits for Lock(B)`	`Waits for Lock(A)`

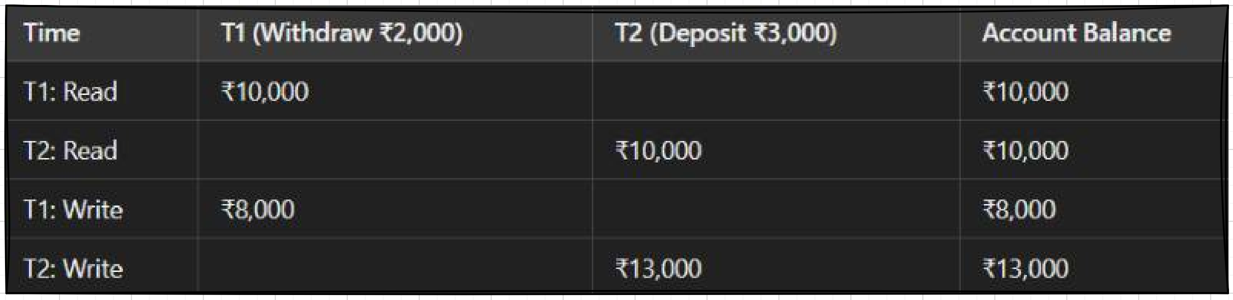
Techniques for Concurrency Control

1. **Lock-Based Protocols:** Control access using locks (e.g., shared, exclusive).
2. **Timestamp Ordering Protocols:** Assign timestamps to transactions and execute based on these timestamps.
3. **Validation-Based Protocols:** Transactions validate their operations before committing.

Example: Lost Update Problem

Let's consider a bank database:

1. Account Balance = ₹10,000.
2. Transaction 1 (T1): Withdraws ₹2,000.
3. Transaction 2 (T2): Deposits ₹3,000.



Time	T1 (Withdraw ₹2,000)	T2 (Deposit ₹3,000)	Account Balance
T1: Read	₹10,000		₹10,000
T2: Read		₹10,000	₹10,000
T1: Write	₹8,000		₹8,000
T2: Write		₹13,000	₹13,000

Here, T2 overwrites T1's result, leading to a lost update. The correct balance should be ₹11,000, but it's now incorrect.

Concurrency Control Solution
Using locks, T1 locks the account balance during withdrawal. T2 waits until T1 finishes, ensuring the balance is updated correctly.

Lock-Based Protocols in Concurrency Control

- To achieve isolation between transactions, locking operations are used in lock-based protocols.
- Isolation ensures that transactions are executed in such a way that their effects are not visible to others until they are complete.
- By locking operations, transactions are restricted from performing conflicting read/write actions on the same data, maintaining data integrity.

Types of Locks Used

1. Shared Lock (S):

- Prevents write operations but allows read operations on the data.
- It is also called a read-only lock.
- Example: Multiple transactions can read the balance of a bank account, but no one can modify it until the lock is released.
- Symbol: S

2. Exclusive Lock (X):

- Allows both read and write operations.
- This lock is for exclusive access, meaning no other transaction can read or modify the data while the lock is held.
- Example: When one transaction updates a bank account balance, no other transaction can access it until the update is complete.
- Symbol: X

Lock Compatibility

Current Lock	Requested Lock	Compatible?
Shared (S)	Shared (S)	Yes
Shared (S)	Exclusive (X)	No
Exclusive (X)	Shared (S)	No
Exclusive (X)	Exclusive (X)	No

Lock-Based Protocols:

1. Simplistic Lock Protocol:

- **Description:** This is a simple approach where a transaction acquires locks on all data items it needs to modify before performing any write operations. Once the transaction completes the write, it releases the locks.
- **Usage:** It is not widely used because it can lead to unnecessary locking and may cause delays in other transactions.

Example:
Scenario: T1 wants to update the account balance by depositing ₹200, and T2 wants to read the balance.

Steps:

1. T1 acquires a lock on the account balance.
2. T1 updates the balance: 500 + 200 = 700.
3. T1 releases the lock.
4. Now T2 can read the balance (700).

Problem: While T1 is holding the lock, T2 must wait, even though it only wants to read the data and doesn't interfere with the update. This makes it inefficient.

2. Pre-claiming Lock Protocol:

- **Description:** Before a transaction starts executing, it first claims all the locks it needs for the data items involved. It requests all the locks at once, and if all the locks are granted, it proceeds with the operations. If any lock is not granted, the transaction is rolled back.
- **Usage:** This protocol is less flexible because it locks all data at the beginning, which can lead to inefficiencies or deadlocks.

Example:
Scenario: T1 wants to update the balance, and T2 wants to read the balance.

Steps:

1. Before starting, T1 requests a write lock on the balance.
2. If the system grants the lock, T1 updates the balance: 500 + 200 = 700.
3. After completing the update, T1 releases the lock.
4. T2 can now acquire a read lock and read the updated balance (700).

What if locks are unavailable?
If T2 already holds a read lock, T1 rolls back (doesn't proceed with its operation).

Problem: Pre-claiming locks upfront may cause deadlocks (e.g., T1 and T2 waiting for each other's locks) or make transactions unnecessarily wait.

3. Two-Phase Locking Protocol (2PL):

- **Description:** This is the most commonly used protocol in databases. It consists of two phases:

1. **Growing Phase:** The transaction can acquire locks but cannot release any locks.
2. **Shrinking Phase:** Once a transaction releases any lock, it cannot acquire any more locks.

- **Advantages:** It ensures serializability (transactions appear to be executed in a serial order).

Example:

- **Phase 1 (Growing):** T1 requests and acquires locks on data items (read or write).
- **Phase 2 (Shrinking):** Once T1 starts releasing locks, no more locks can be acquired.

Scenario: T1 is transferring ₹200 from account A to account B, and T2 is reading the balance of account A.

Steps:
1. Growing Phase (Lock Acquisition):

- T1 acquires a write lock on account A (to debit ₹200).
- T1 acquires a write lock on account B (to credit ₹200).

2. Shrinking Phase (Lock Release):

- After transferring, T1 releases the lock on account B.
- Then it releases the lock on account A.

Advantages: Transactions follow a systematic order: lock first, work, then release.

Problem: T2 cannot read the balance of account A while T1 is holding the lock.

4. Strict Two-Phase Locking Protocol (Strict 2PL):

- This is a stricter version of the 2PL protocol. It makes sure that the transaction keeps the locks until it's completely finished and committed.

How it works:

- A transaction holds on to its locks and doesn't release them until it is ready to commit (finish and confirm the work).
- Once it commits, it releases all its locks at once.

Why it's useful:

- This avoids situations like cascading rollbacks (when one failure causes many other transactions to fail).
- By holding all locks until commit, it ensures that no other transaction can interfere before the work is finalized.

Scenario: T1 is transferring ₹200 from account A to account B, and T2 is reading the balance of account A.

Steps:

1. T1 acquires a write lock on account A and account B (Growing Phase).
2. T1 performs the transfer: Debit ₹200 from A: 500 - 200 = 300. & Credit ₹200 to B: 1000 + 200 = 1200.
3. T1 holds all locks until it commits.
4. After the transaction commits, T1 releases all locks (Shrinking Phase).

Advantage: Prevents cascading rollbacks. If T1 fails halfway, other transactions like T2 won't see incomplete or incorrect data.

Protocol	Locks Held	Lock Release	Example Use Case
Simplistic Lock Protocol	Locks only while writing	After writing	Updating a single data item without interference.
Pre-claiming Lock Protocol	All locks at the start	After completing the transaction	Safe execution of short transactions with known locks.
Two-Phase Locking (2PL)	Locks during growing phase	Gradually in shrinking phase	Transaction requiring multiple locks in sequence.
Strict 2PL	Locks during transaction	All at commit	Preventing cascading rollbacks in critical updates.

Timestamping Protocols

AKTU - 2023-24 & 2021-22

Timestamping protocols help in maintaining the serializability of transactions without using locks. Each transaction is assigned a unique timestamp that determines the order of execution.

1. Basic Timestamp Ordering Protocol

How it works:

1. Each transaction is assigned a timestamp when it starts.
2. All operations (read/write) are executed in the order of their timestamps.
3. The database maintains two timestamps for each data item:

- **Read Timestamp (RTS):** The largest timestamp of any transaction that successfully read the data.
- **Write Timestamp (WTS):** The largest timestamp of any transaction that successfully wrote the data.

4. Rules:

- If a transaction tries to read or write in a way that violates timestamp order, it is aborted and restarted with a new timestamp.

Example:

- T1 (TS = 1) starts first and writes to data item A.
- T2 (TS = 2) tries to read A after T1 has written to it.
- Since T2 > T1, the operation is valid because it respects the timestamp order.

But if T2 tries to write A before T1 completes, it would violate the timestamp order (because T2 is newer) and would be aborted.

2. Strict Timestamp Ordering Protocol

- **How it differs:**

Strict timestamp ordering is more restrictive. It ensures that:

1. Write operations by a transaction are delayed until all earlier transactions (with smaller timestamps) have finished.
2. Read operations are delayed if there is a pending write operation by an earlier transaction.

- **Why it's useful:**

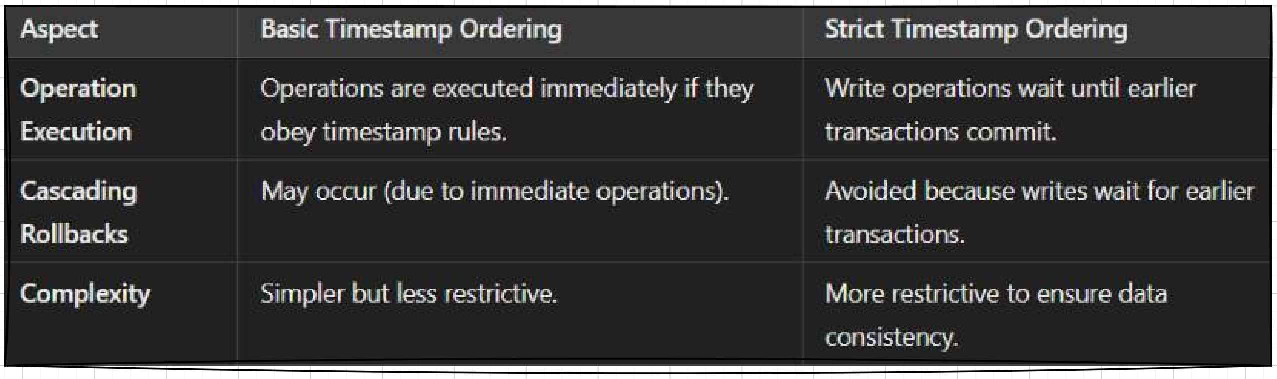
Avoids issues like cascading rollbacks, which occur in basic timestamp ordering.

Example:

- T1 (TS = 1) writes to A.
- T2 (TS = 2) wants to write to A but must wait for T1 to complete.

Even if T2 is ready, it cannot proceed until T1 commits, ensuring strict order.

Key Differences



Aspect	Basic Timestamp Ordering	Strict Timestamp Ordering
Operation Execution	Operations are executed immediately if they obey timestamp rules.	Write operations wait until earlier transactions commit.
Cascading Rollbacks	May occur (due to immediate operations).	Avoided because writes wait for earlier transactions.
Complexity	Simpler but less restrictive.	More restrictive to ensure data consistency.

Example in a Timestamp Queue

Scenario:
Two transactions T1 (TS = 1) and T2 (TS = 2) are working on a bank account balance (initial = ₹500).

Basic Timestamp Ordering:

1. T1 (TS = 1) reads the balance (₹500) and writes a new balance (₹400).
   - RTS(A) = 1, WTS(A) = 1.
2. T2 (TS = 2) tries to write ₹300.
   - Allowed because TS(T2) > WTS(A).
3. T2 writes ₹300, updating WTS(A) = 2.

Result: Final balance = ₹300.

Strict Timestamp Ordering:

1. T1 (TS = 1) reads the balance (₹500) and writes a new balance (₹400).
   - T1 must commit before any operation from T2 can proceed.
2. T2 (TS = 2) waits until T1 commits before writing ₹300.

Result: Final balance = ₹300, but changes are only applied after T1 commits.

Validation-Based Protocols

AKTU - 2021-22 & 2022-23 & 2023-24

Validation-Based Protocols are also known as Optimistic Concurrency Control techniques. These protocols assume that conflicts are rare and allow transactions to execute without locking resources. Conflicts are checked only at the time of validation before committing the transaction.

Working Phases of Validation-Based Protocols

1. Read Phase:

- The transaction reads data from the database and performs calculations or operations locally.
- During this phase, no actual changes are made to the database.
- Data is stored in a temporary workspace (buffer).

2. Validation Phase:

- Before committing, the system validates whether the transaction can be executed without conflicting with other transactions.
- Validation ensures serializability, meaning the transaction can appear to have been executed in isolation.

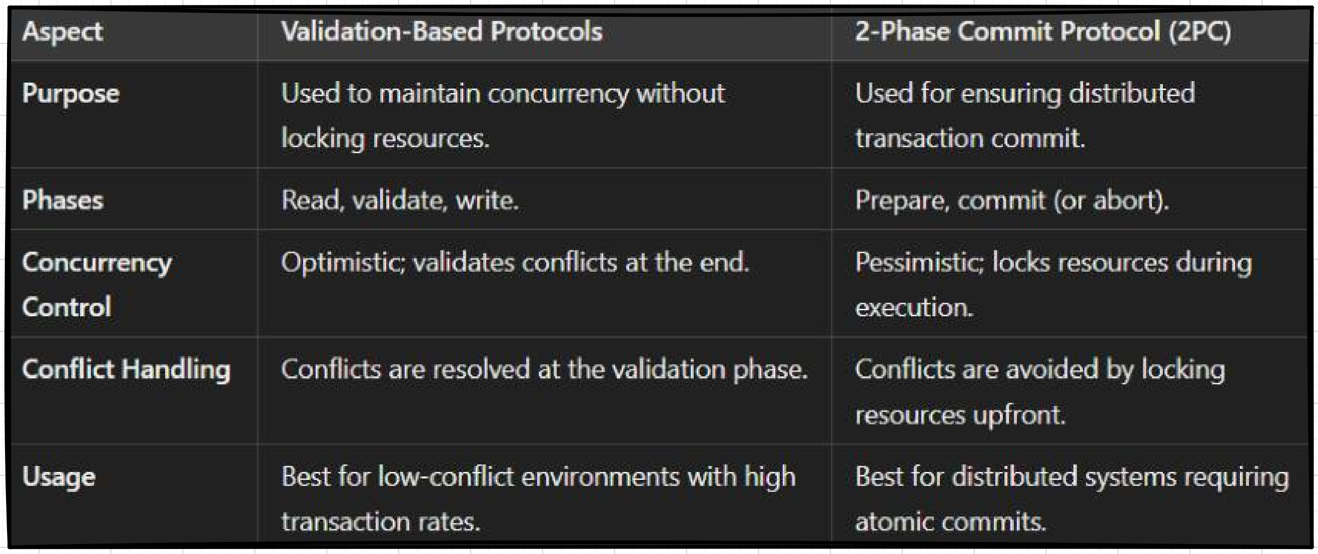
The validation checks:

- No other transaction has modified the data read by the current transaction.
- The current transaction does not overlap with the execution of conflicting transactions.

3. Write Phase:

- If the transaction passes the validation phase, it writes changes to the database permanently.
- If the validation fails, the transaction is aborted and restarted.

Differences from 2-Phase Commit (2PC)



Aspect	Validation-Based Protocols	2-Phase Commit Protocol (2PC)
Purpose	Used to maintain concurrency without locking resources.	Used for ensuring distributed transaction commit.
Phases	Read, validate, write.	Prepare, commit (or abort).
Concurrency Control	Optimistic; validates conflicts at the end.	Pessimistic; locks resources during execution.
Conflict Handling	Conflicts are resolved at the validation phase.	Conflicts are avoided by locking resources upfront.
Usage	Best for low-conflict environments with high transaction rates.	Best for distributed systems requiring atomic commits.

Example of Validation-Based Protocol

Scenario: Two transactions T1 and T2 updating account balances.
Initial database:

- Account A = ₹500, Account B = ₹1000.

Steps:

1. T1 (Read Phase):

- Reads A = ₹500, B = ₹1000.
- Plans to debit ₹200 from A and credit ₹200 to B.

2. T2 (Read Phase):

- Reads A = ₹500, B = ₹1000.
- Plans to credit ₹300 to A and debit ₹300 from B.

3. Validation Phase:

- T1 checks:
- No other transaction modified A or B after T1 read them.
- T2 checks:
- T1 modified A and B, which conflicts with T2's plans.

4. Write Phase:

- T1 passes validation and writes changes:
- A = ₹300, B = ₹1200.
- T2 fails validation, aborts, and restarts.

Multiple Granularity

What is Granularity?

Granularity refers to the size of the data that can be locked in a database.

- **Small Granularity:** Locking a single record (fine-grained).
- **Large Granularity:** Locking an entire table or database (coarse-grained).

Why Multiple Granularity Locking?

Problem:

- If a transaction (T1) needs to lock the entire database, locking every single record is inefficient.
- If another transaction (T2) needs just one record, locking the whole database affects concurrency unnecessarily.

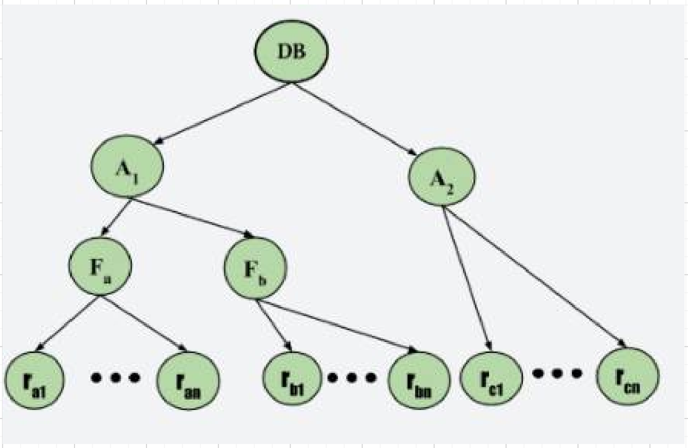
Solution:

- Multiple granularity allows locking different parts of the database (like records, files, or areas) depending on the needs of the transaction. This balance improves both efficiency and concurrency.

Hierarchy of Locks

The database is divided into levels, like a tree:

1. Database (highest level).
2. Area (e.g., a category in the database).
3. File (e.g., a table).
4. Record (lowest level, individual row in a table).



How It Works?

- A transcription can lock any node (e.g., database, table, or record).
- When a node is locked:

All descendants are implicitly locked in the same mode.
For example: If you lock a file (table) in exclusive mode, all its records are locked exclusively.

Intention Locks

To manage hierarchical locks, additional intention lock modes are introduced:

1. Intention-Shared (IS):

- Indicates a plan to place shared locks at lower levels.
- Example: Locking a table in IS mode to read specific records.

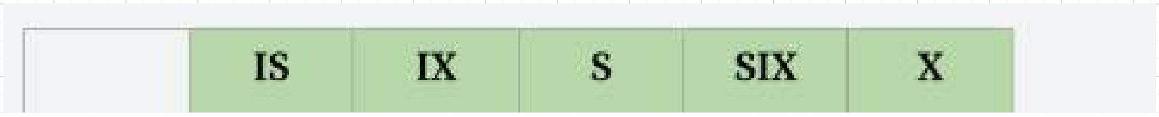
2. Intention-Exclusive (IX):

- Indicates a plan to place exclusive locks at lower levels.
- Example: Locking a table in IX mode to update specific records.

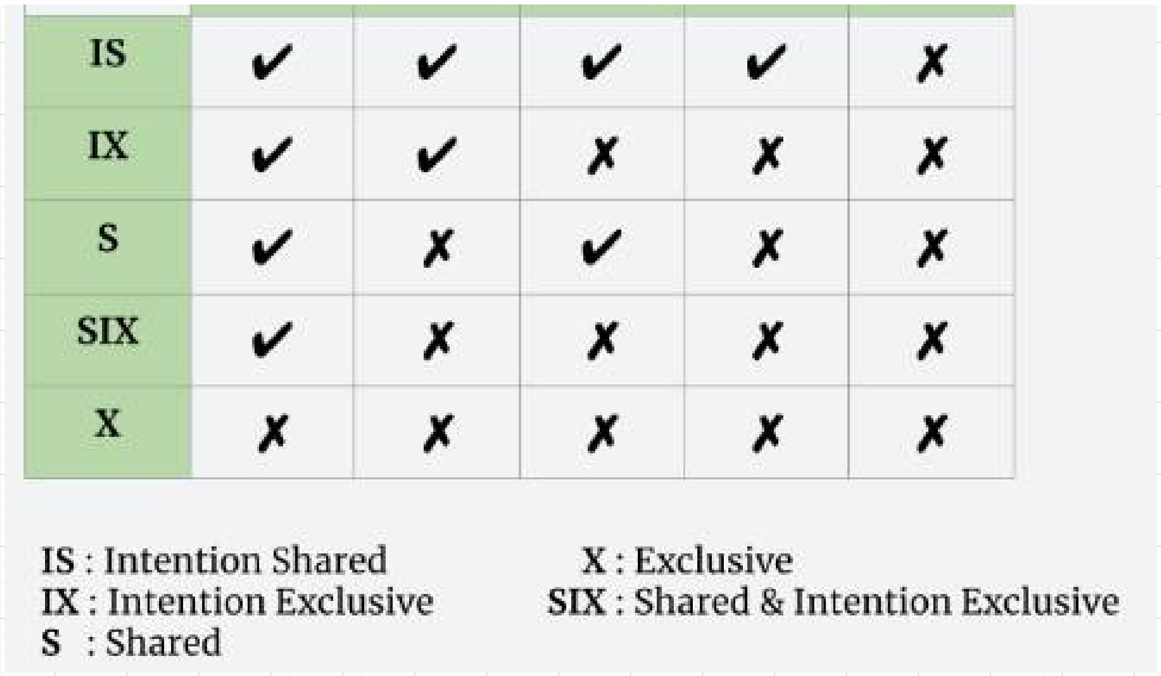
3. Shared & Intention-Exclusive (SIX):

- Locks a subtree in shared mode but allows exclusive locks at lower levels.
- Example: Locking a table for reading all records while updating specific ones.

Lock Compatibility Matrix



	IS	IX	S	SIX	X



IS	✓	✓	✓	✓	✗
IX	✓	✓	✗	✗	✗
S	✓	✗	✓	✗	✗
SIX	✓	✗	✗	✗	✗
X	✗	✗	✗	✗	✗

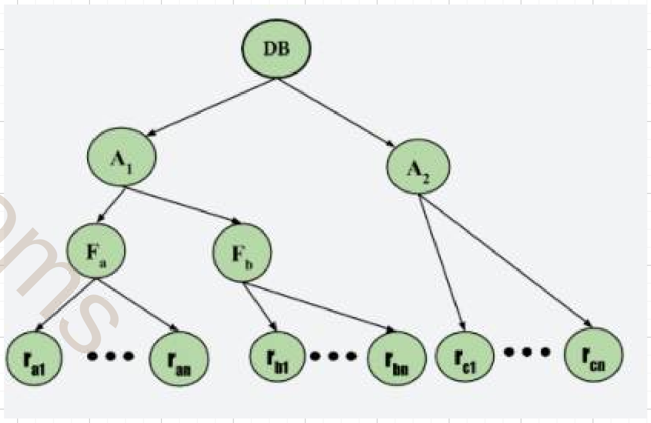
IS : Intention Shared
IX : Intention Exclusive
S : Shared
X : Exclusive
SIX : Shared & Intention Exclusive

Example Scenarios

1. Transaction T1 (Reads a Record)

- Action: T1 reads record Ra2 in file Fa.
- Locks Needed:

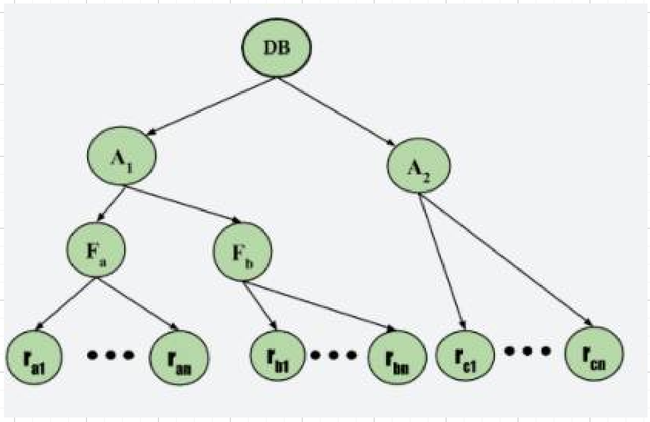
1. Database: IS mode.
2. Area A1: IS mode.
3. File Fa: IS mode.
4. Record Ra2: S mode.



2. Transaction T2 (Updates a Record)

- Action: T2 modifies record Ra9 in file Fa.
- Locks Needed:

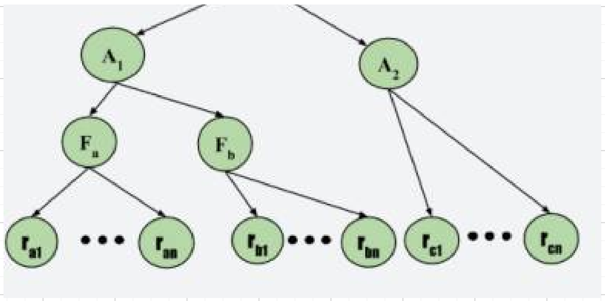
1. Database: IX mode.
2. Area A1: IX mode.
3. File Fa: IX mode.
4. Record Ra9: X mode.



3. Transaction T3 (Reads All Records in a File)

- Action: T3 reads all records in file Fa.
- Locks Needed:

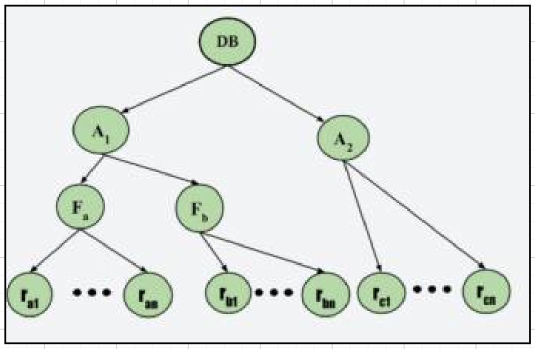
1. Database: IS mode.
2. Area A1: IS mode.
3. File Fa: S mode.



4. Transaction T4 (Reads Entire Database)

- **Action: T4 reads the entire database.**
- **Locks Needed:**

1. **Database: S mode.**



Concurrency in MGL

- **T1, T3, and T4 can run simultaneously because their operations don't conflict.**
- **T2 cannot run with T3 or T4 because it requires exclusive access.**

What is Multi-Version Scheme?

AKTU - 2023-24 & 2021-22

In multi-version schemes, the database maintains multiple versions of the same data item to allow better concurrency and avoid conflicts between transactions.

Why use it?

- To ensure **read consistency** (a transaction reads data as it existed at the start of the transaction, even if other transactions update it).
- To improve **concurrency** by letting read and write operations occur simultaneously.

How Does It Work?

1. Data Versions:

- Each data item has multiple versions, each with a timestamp or identifier.
- A new version is created when a transaction updates the data.
- Older versions remain available for transactions that need them.

2. Read and Write Operations:

- **Read Operation:** A transaction reads the version of the data valid at its start time.
- **Write Operation:** A transaction creates a new version with an updated value and timestamp.

Key Benefits

1. Read Consistency:

- Readers can access the version of the data item valid at the start of their transaction, even if other transactions update the data simultaneously.

2. No Blocking Reads:

- Read operations don't block write operations, improving concurrency.

3. Reduced Conflicts:

- Transactions can proceed without waiting for each other unless they modify the same data.

Examples of Multi-Version Schemes

1. Ticket Booking System
Scenario:

- A customer (T1) checks seat availability for a train.
- Another customer (T2) books a seat on the same train.

Process:

- When T1 starts, it reads the current version of the seat availability.
- T2 updates the seat availability, creating a new version.
- T1 still sees the old version, ensuring read consistency.

2. Banking System (Account Balance Updates)
Scenario:

- Transaction T1 reads an account balance.
- Transaction T2 updates the balance simultaneously.

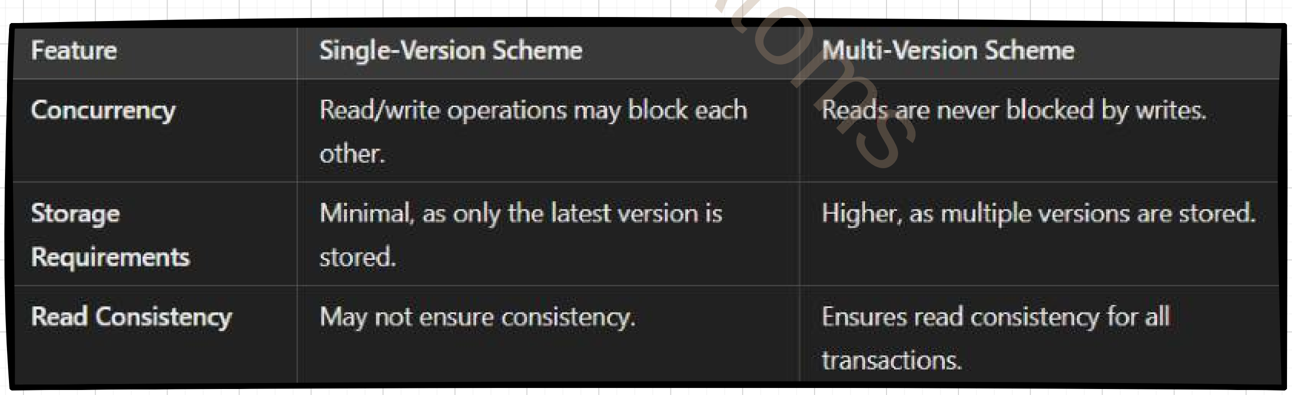
Process:

- T1 reads the old version of the balance.
- T2 creates a new version after updating the balance.
- T1 finishes without being blocked by T2.

Drawbacks

1. **Increased Storage:**
   - Multiple versions require more storage space.
2. **Complexity:**
   - Managing versions and ensuring they are valid for the right transactions can be challenging.
3. **Garbage Collection:**
   - Old versions that are no longer needed must be removed, adding overhead.

Difference Between Multi-Version and Single-Version Schemes



Feature	Single-Version Scheme	Multi-Version Scheme
Concurrency	Read/write operations may block each other.	Reads are never blocked by writes.
Storage Requirements	Minimal, as only the latest version is stored.	Higher, as multiple versions are stored.
Read Consistency	May not ensure consistency.	Ensures read consistency for all transactions.

What is Recovery with Concurrent Transactions?

When multiple transactions are running simultaneously, the system must ensure that the database remains consistent, even if there's a system failure (like a crash). Recovery techniques help restore the database to a consistent state.

1. Immediate Update Techniques

Definition: Changes made by a transaction are immediately written to the database as soon as they are performed, even before the transaction commits.

Benefit: Reduces recovery time since changes are already in the database.

Challenge: If the system crashes before the transaction commits, the database may become inconsistent.

Log-Based Recovery

A log is a sequential record of all operations performed by transactions. It is critical for recovery.

Steps in Log-Based Recovery

1.Write-Ahead Logging (WAL):

- Changes are logged before being applied to the database.
- Ensures that the log is available for recovery after a crash.

2.Two Types of Log Entries:

- **UNDO Entries:** Used to reverse incomplete transactions.
- **REDO Entries:** Used to reapply committed transactions.

3.During Recovery:

- UNDO Phase: Rollback all incomplete transactions using the log.
- REDO Phase: Reapply all changes of committed transactions.

Example:

Imagine two transactions:

- T1 (incomplete): Updated balance from ₹5000 to ₹4500.
- T2 (committed): Updated balance from ₹4500 to ₹4800.

Recovery steps:

- UNDO T1: Revert balance to ₹5000 using its UNDO log.
- REDO T2: Reapply balance update to ₹4800 using its REDO log.

Checkpoints

To optimize recovery, the system periodically creates checkpoints in the log.

What is a Checkpoint?

- A checkpoint is a snapshot of the current state of the database.
- It marks a point where all committed transactions have been written to the database.

Benefits of Checkpoints:

1.Speeds up Recovery:

- During recovery, the system starts from the last checkpoint instead of scanning the entire log.

2. Reduces Overhead:

- Limits the amount of undo/redo work during recovery.

Example of Checkpoints in Action:
At checkpoint C1, the database writes all changes from committed transactions up to that point.

If the system crashes after C1:

- **Undo:** Transactions started after C1 but didn't commit.
- **Redo:** Transactions committed after C1.

Single/Multi-User Environments

Single-User Environment:

- Easier to handle because only one transaction is active at a time.
- Recovery involves straightforward undo/redo of that transaction.

Multi-User Environment:

- More complex due to concurrent transactions.
- Requires logs, timestamps, and locking protocols to manage recovery and ensure consistency.

Case Study on Oracle: Concurrency Control Techniques

Oracle Database, developed by Oracle Corporation, is one of the most popular relational database management systems (RDBMS) globally. It is designed to handle large-scale data management needs for enterprise applications. Oracle implements robust Concurrency Control Techniques to ensure data integrity, consistency, and isolation when multiple transactions are executed simultaneously.

Key Concurrency Control Techniques in Oracle

1.Multiversion Concurrency Control (MVCC):

- Oracle uses MVCC to provide a non-blocking read mechanism. When a transaction modifies data, the database does not overwrite the original data. Instead, it creates a new version while retaining the old version for other transactions to read.
- This ensures read consistency, where each query sees a consistent snapshot of the database as of the query's start time.

Example:
If a user reads data while another transaction is modifying it, the user will see the old version of the data until the modifying transaction commits.

3.Undo Tablespace:
Oracle uses undo segments to store the original state of modified data. This enables:

- Rollback of uncommitted transactions.
- Generation of consistent snapshots for queries using MVCC.

Impact: Even if a transaction is rolled back, Oracle ensures that other users' views of the data remain unaffected.

4.Automatic Conflict Resolution:
Oracle employs sophisticated algorithms to handle write conflicts between transactions. When a conflict occurs:

- The database determines the transaction with the higher priority based on timing or access order.
- The lower-priority transaction may be delayed or restarted.

Advantages of Oracle's Concurrency Control:

- **High performance with minimal blocking.**
- **Improved scalability for multi-user environments.**
- **Data consistency and integrity are guaranteed.**

Graph-Based Locking Protocols

Aktu- 2023-24

Graph-based locking is a concurrency control mechanism designed to manage data item access and prevent deadlocks by defining a partial ordering of data items. This protocol ensures that transactions access data in a specific order, avoiding circular waits that cause deadlocks.

Key Concepts

1.Partial Ordering of Data Items:

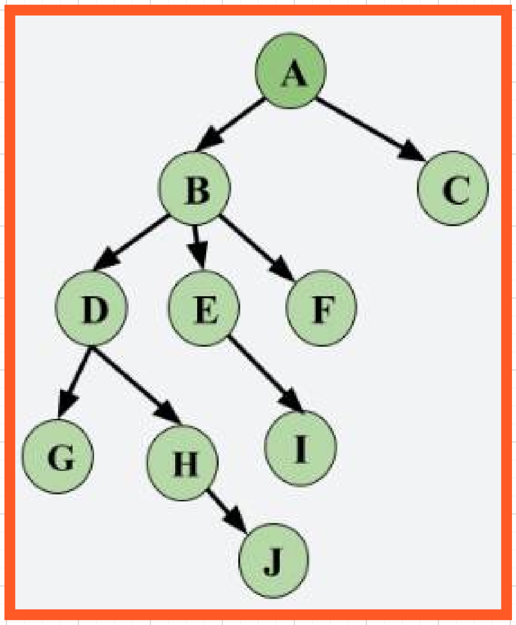
- All data items are organized in a directed acyclic graph (DAG).
- Each node represents a data item.
- Edges represent allowed access sequences (e.g., if there's an edge from A → B, a transaction must lock A before locking B).

2. Protocol Rules:

- A transaction can acquire a lock on a data item only if it has already locked its parent in the graph.
- Once a transaction releases a lock, it cannot acquire any more locks.

3. Deadlock Avoidance:

- By enforcing a strict order of locking based on the graph structure, circular waits are avoided, thereby eliminating deadlocks.

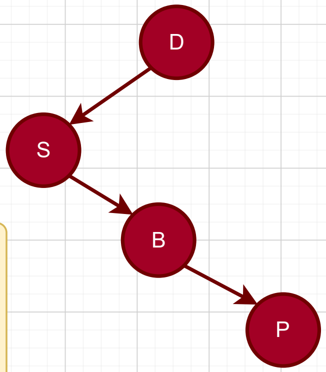


Steps in Graph-Based Locking

1. Identify the hierarchy of data items and arrange them in a DAG.
   (Example: Database → Table → Row → Column.)
2. A transaction locks data items starting from the highest node in the hierarchy.
3. The transaction can only lock children of a locked parent.
4. When finished, the transaction releases locks in a reverse order (from leaf to root).

Example

Hierarchy of Data Items in a Library System:
Database → Section → Book → Page



Transactions:
1. T1: Read a specific page (P1) of a book (B1):
   - Lock order: Database → Section → Book (B1) → Page (P1).
2. T2: Modify a book (B2):
   - Lock order: Database → Section → Book (B2).
3. T3: Read all books in a section:
   - Lock order: Database → Section.

Deadlock Prevention:
- If T1 locks Page P1, T2 cannot directly jump to modify Page P1 without following the lock hierarchy.
- Transactions must respect the lock order, preventing circular dependencies.

Advantages of Graph-Based Locking
- **Deadlock-Free:** Ensures no circular waits by enforcing a strict locking order.
- **Efficient Use of Resources:** Transactions lock only the required data.
- **Improved Concurrency:** Minimizes unnecessary locking.

Disadvantages
- **Complex Implementation:** Managing a DAG for large datasets can be challenging.
- **Reduced Flexibility:** Transactions are forced to follow a rigid locking order.