AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

ONE SHOT Revision

Unit-5

Concurrency Control Techniques

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -5
Concurrency control technique
Lecture-1

Today's Target

- Lock
- Two phase locking protocol
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

(BCS-501- Database Management System)

Unit-V : Concurrency Control Techniques

AKTU : 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.

LOCK

- A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- There is one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database items.

TYPE OF LOCKS

1.Binary lock

- A binary lock can have two states or values: locked and unlocked (or 1 and 0, for simplicity).
- A distinct lock is associated with each database item X.
- If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item.
- If the value of the lock on X is 0, the item can be accessed when requested, and the lock value is changed to 1.

Two operations, lock_item and unlock_item, are used with binary locking

```
//lock opeartion on binary lock
lock_item(X):
B: if LOCK(X) = 0 (* item is unlocked *)
then LOCK(X) ←1 (* lock the item *)
else
begin
wait (until LOCK(X) = 0
and the lock manager wakes up the transaction);
go to B
end
```

```
//unlock opeartion on binary lock
unlock_item(X):
LOCK(X) ← 0; (* unlock the item *)
if any transactions are waiting
then wakeup one of the waiting transactions;
```

If the simple binary locking scheme described here is used, every transaction must obey the following rules:

1. A transaction T must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T.
2. A transaction T must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T
3. A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X
4. A transaction T will not issue an unlock_item(X) operation unless it already holds the lock on item X.

2. Shared/Exclusive (or Read/Write) Locks

- The preceding binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item.
- We should allow several transactions to access the same item X if they all access X for reading purposes only.

This is because read operations on the same item by different transactions are not conflicting

- However, if a transaction is to write an item X, it must have exclusive access to X. For this purpose, a different type of lock called a multiple-mode lock is used. In this scheme—called shared/exclusive or read/write locks.
- there are three locking operations: read_lock(X), write_lock(X), and unlock(X).

A lock associated with an item X, LOCK(X), now has three possible states: read-locked, write-locked, or unlocked.

A read-locked item is also called share-locked because other transactions are allowed to read the item, whereas a write-locked item is called exclusive-locked because a single transaction exclusively holds the lock on the item.

```
read_lock(X):
B: if LOCK(X) = “unlocked”
thenbegin LOCK(X) ← “read-locked”;
no_of_reads(X) ← 1
end
else if LOCK(X) = “read-locked”
then no_of_reads(X) ← no_of_reads(X) + 1
else begin
wait (until LOCK(X) = “unlocked”
and the lock manager wakes up the transaction);
go to B
end;
```

```
write_lock(X):
B: if LOCK(X) = “unlocked”
then LOCK(X) ← “write-locked”
else begin
wait (until LOCK(X) = “unlocked”
and the lock manager wakes up the transaction);
go to B
end;
```

```
unlock (X):
if LOCK(X) = “write-locked”
thenbegin LOCK(X) ← “unlocked”;
wakeup one of the waiting transactions, if any
end
else if LOCK(X) = “read-locked”
then begin
    no_of_reads(X) ← no_of_reads(X) –1;
      if no_of_reads(X) = 0
      then begin LOCK(X) = “unlocked”;
      wakeup one of the waiting transactions, if any
      end
end
```

- When we use the shared/exclusive locking scheme, the system must enforce the following rules:
- 1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T.
- 2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.

3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.

4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.

5. . A transaction T will not issue a write_lock(X) operation if it already holds a read (shared) lock or write (exclusive) lock on item X.

6. A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X

Conversion of Lock

- Sometimes it is desirable to relax conditions 4 and 5 in the preceding list in order to allow lock conversion; that is, a transaction that already holds a lock on item X is allowed under certain conditions to convert the lock from one locked state to another.
- For example, it is possible for a transaction T to issue a read_lock(X) and then later to upgrade the lock by issuing a write_lock(X) operation

If T is the only transaction holding a read lock on X at the time it issues the write_lock(X) operation, the lock can be upgraded; otherwise, the transaction must wait.

It is also possible for a transaction T to issue a write_lock(X) and then later to downgrade the lock by issuing a read_lock(X) operation. When upgrading and downgrading of locks is used, the lock table must include transaction identifiers in the record structure for each lock (in the locking_transaction(s) field) to store the information on which transactions hold locks on the item

Two-Phase Locking protocol

A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction.

Such a transaction can be divided into two phases: an expanding or growing (first) phase, during which new locks on items can be acquired but none can be released; and a shrinking (second) phase, during which existing locks can be released but no new locks can be acquired

- If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase.

T1 do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T2

Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result of serial schedule T_2 followed by T_1: X=70, Y=50

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Result of schedule S X=50, Y=50 (nonserializable)

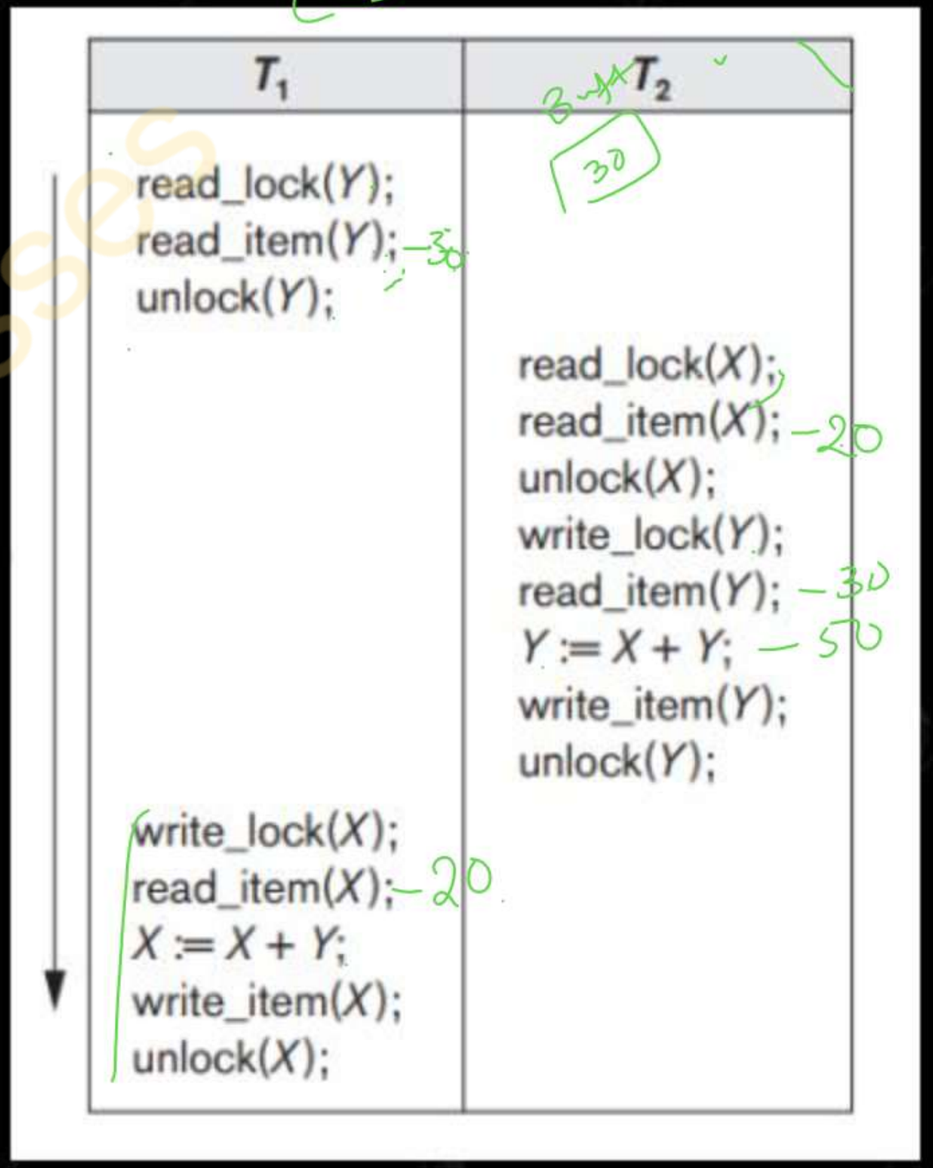
Transactions that do not obey two-phase locking.

(a) Two transactions T1 and T2.

(b) Results of possible serial schedules of T1 and T2.

(c) A nonserializable schedule S that uses locks.

Figure 22.3(c)



T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y)	unlock(X)
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

- The schedule shown in Figure 22.3(c) is not permitted for T1 and T2 (with their modified order of locking and unlocking operations) under the rules of locking.
- T1 will issue its write_lock(X) before it unlocks item Y; consequently, when T2 issues its read_lock(X), it is forced to wait until T1 releases the lock by issuing an unlock (X) in the schedule.

If every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules.

The locking protocol, by enforcing two-phase locking rules, also enforces serializability

- Two-phase locking may limit the amount of concurrency that can occur in a schedule because a transaction T may not be able to release an item X after it is through using it if T must lock an additional item Y later; or conversely, T must lock the additional item Y before it needs it so that it can release X.
- Hence, X must remain locked by T until all items that the transaction needs to read or write have been locked; only then can X be released by T

- Meanwhile, another transaction seeking to access X may be forced to wait, even though T is done with X; conversely, if Y is locked earlier than it is needed, another transaction seeking to access Y is forced to wait even though T is not using Y yet.
- This is the price for guaranteeing serializability of all schedules without having to check the schedules themselves.

- The two-phase locking protocol guarantees serializability (that is, every schedule that is permitted is serializable), it does not permit all possible serializable schedules (that is, some serializable schedules will be prohibited by the protocol)

AKTU PYQS

Q.1	What is lock? Explain different types of locks	AKTU 2016-17 AKTU 2022-23
Q.2	What is two phase locking protcol	AKTU2015-16 AKTU 2018-19 AKTU 2022-23
Q.3	Discuss the immediate update recovery technique in both single-user and multiuser environments. What are the advantages and disadvantages of immediate update.	AKTU2022-23

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -5
Concurrency control technique
Lecture-2

Today's Target

- Two phase locking protocol variation
- Timestamp ordering protocol
- MVCC protocol
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Two-Phase Locking (2PL) Variants

1.Basic 2PL:

- Transactions follow two phases: growing phase (locks are acquired but not released) and shrinking phase (locks are released but not acquired).
- Ensures conflict-serializability.

2. Conservative 2PL:

- All locks are acquired before the transaction begins execution.
- Avoids deadlocks but can cause delays due to pre-locking.

3.Strict 2PL:

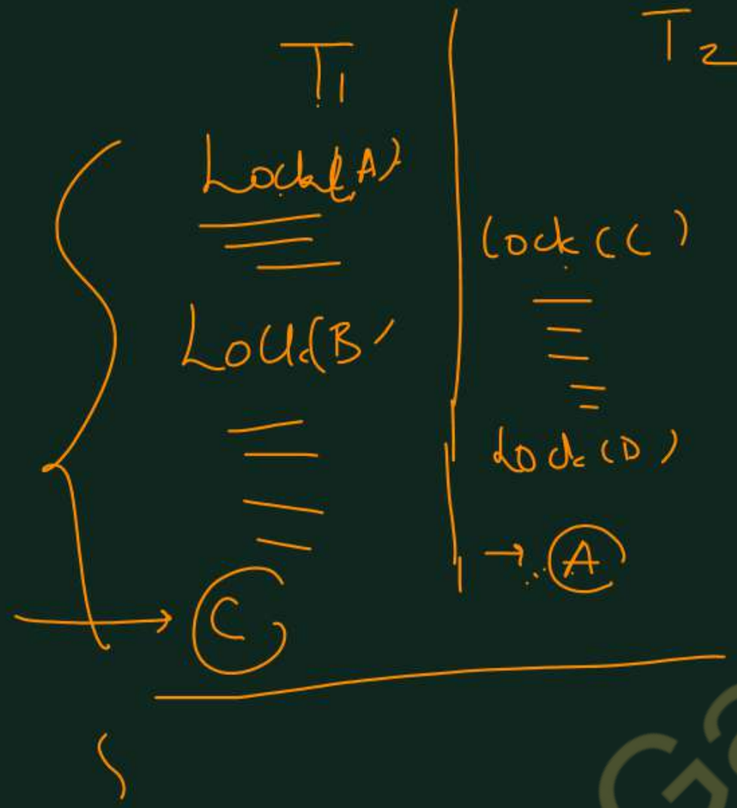
All exclusive (write) locks are released only after the transaction commits or aborts.

Ensures recoverability by avoiding cascading rollbacks.

4.Rigorous 2PL:

All locks (shared and exclusive) are held until the transaction commits or aborts.

Provides stronger recoverability and simplifies implementation.



Basic 2PL

Deadlock

Conservative 2PL

→ No Deadlock.

Lock(A) Lock(B) Lock(A) Lock

Concurrency control based on time stamp ordering

- A different approach that guarantees serializability involves using transaction timestamps to order transaction execution for an equivalent serial schedule.

Timestamps

Tmestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the transaction start time

- Timestamp of transaction T as TS(T). Concurrency control techniques based on timestamp ordering do not use locks; hence, deadlocks cannot occur.

The Timestamp Ordering Algorithm

- The Timestamp Ordering (TO) algorithm ensures serializability by ordering transactions according to their timestamps (TS), which are assigned when the transactions begin.
- A schedule is valid if it is equivalent to the serial execution of transactions in the order of their timestamps.

Key Features:

Ordering Guarantee: Ensures that the execution order follows the timestamp order.

Difference from 2PL:

- 2PL allows schedules equivalent to any serial order based on locking.
- TO enforces a specific serial order determined by the transaction timestamps.

Timestamps Associated with Each Item X:

read_TS(X):

- Records the largest timestamp of any transaction that successfully read
- Indicates the timestamp of the youngest transaction that read X

Formula:

read_TS(X)=TS(T), where T is the youngest transaction to read X

write_TS(X):

- Records the largest timestamp of any transaction that successfully wrote to X.
- Indicates the timestamp of the youngest transaction that wrote X.
- Formula: write_TS(X)=TS(T), where T is the youngest transaction to write X

How TO Ensures Serializability:

- For conflicting operations on an item X (read vs write, or write vs write), the algorithm ensures that the order of execution does not violate the timestamp order:
- If a transaction T tries to perform an operation that conflicts with an operation already performed by a transaction with a later timestamp, T is aborted or its operation is rejected.
- This guarantees that the final schedule respects the serial order of transaction timestamps.

Advantages:

- No locks are used, so there is no chance of deadlocks.
- Directly enforces a specific serial order based on timestamps.

Disadvantages:

- Frequent transaction restarts if timestamp violations occur, especially in high-contention scenarios.
- Lower throughput compared to protocols like 2PL in certain cases.

Basic Timestamp Ordering (TO)

Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated.

If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

- If T is aborted and rolled back, any transaction T1 that may have used a value written by T must also be rolled back.
- Similarly, any transaction T2 that may have used a value written by T1 must also be rolled back, and so on.
- This effect is known as cascading rollback and is one of the problems associated with basic TO, since the schedules produced are not guaranteed to be recoverable

with basic TO, since the schedules produced are not guaranteed to be recoverable.

- An additional protocol must be enforced to ensure that the schedules are recoverable, cascadeless, or strict.
- We first describe the basic TO algorithm here. The concurrency control algorithm must check whether conflicting operations violate the timestamp ordering in the following two cases:

Whenever a transaction T issues a write_item(X) operation, the following is checked:

a. If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering

b. If the condition in part (a) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T)

2. Whenever a transaction T issues a read_item(X) operation, the following is checked:

a. If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of item X before T had a chance to read X.

b. If write_TS(X) ≤ TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Whenever the basic TO algorithm detects two conflicting operations that occur in the incorrect order, it rejects the later of the two operations by aborting the transaction that issued it.

Strict Timestamp Ordering (TO).

- A variation of basic TO called strict TO ensures that the schedules are both strict (for easy recoverability) and (conflict) serializable.
- In this variation, a transaction T that issues a read_item(X) or write_item(X) such that
- TS(T) > write_TS(X) has its read or write operation delayed until the transaction T that wrote the value of X (hence TS(T) = write_TS(X)) has committed or aborted.

- To implement this algorithm, it is necessary to simulate the locking of an item X that has been written by transaction T until T is either committed or aborted.
- This algorithm does not cause deadlock, since T waits for T only if TS(T) > TS(T)

Thomas's Write Rule.

A modification of the basic TO algorithm, known as Thomas's write rule, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows:

1. If read_TS(X) > TS(T), then abort and roll back T and reject the operation.
2. If write_TS(X) > TS(T), then do not execute the write operation but continue processing. This is because some transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of X.
   - Thus, we must ignore the write_item(X) operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).
3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

Multi-version concurrency control technique

- Multiversion Concurrency Control (MVCC) maintains multiple versions of data items to allow concurrent transactions to access the database without conflicts, improving performance and flexibility in concurrency management

Key Features:

1.Multiple Versions:

- When a transaction writes to a data item, a new version is created, and the older version is retained.
- Read operations can access an appropriate version to maintain serializability.

Advantages:

1.Increased Concurrency: Some read operations that would be rejected in traditional methods (like 2PL) are allowed by reading older versions.

2.Historical Data: Useful for applications like temporal databases, where past states of data are needed.

3.Reduced Blocking: Transactions are less likely to block each other, as reads and writes access different versions.

Drawbacks:

1.Storage Overhead: Multiple versions require additional storage.

2.Version Management: Keeping track of versions and deciding which to keep or discard adds complexity.

Common MVCC Schemes:

1. MVCC Based on Timestamp Ordering.
2. MVCC Based on 2PL.
3. Validation-Based MVCC.

AKTU PYQS

Q.1	What is time stamp ordering protocol	AKITU 2022-22
Q.2	Explain the different variation of 2PL protocol	AKTU 2012-13 AKTU 2021-22
Q.3	Explain the Thomson's write rule, strict timestamp ordering protocol, Basic time stamp ordering protocol	AKTKU 2018-19 AKTU 2017-18 AKTUY 2022-23
Q.4	Explain mvcc protocol	AKITU 2022-23

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -5
Concurrency control technique
Lecture-3

Today's Target

- MVCC techniques
- Phantom phenomena
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Multi-version technique based on timestamp odering

- This description outlines a Multiversion Concurrency Control (MVCC) scheme that maintains multiple versions of a data item X to handle read and write operations.
- Each version has associated timestamps to track when it was created and read. The rules ensure serializability while allowing multiple transactions to execute concurrently.

Key Concepts:

Versions of X:

Multiple versions (X1,X2,…,Xk) are stored for each data item X

Each version Xihas:

- Value of Xi: The data stored in that version.
- write_TS(X_i): The timestamp of the transaction that created Xi
- read_TS(X_i): The highest timestamp of all transactions that read Xi

New Version Creation:

A new version is created when a transaction writes to X, with:

- write_TS(X_{k+1}) =TS(T)
- read_TS(X_{k+1})=TS(T).

Rules for Serializability:

Write Operation (write_item(X)):

1 When a transaction T tries to write to a data item X:

Find the Latest Version to Work With:

Look for the version of X (say Xi) that was written most recently before or at the time of T.

In other words, find the version Xi with the highest write_TS(Xi) ≤ TS(T))

2.Check for Conflicts:

See if any newer transaction (with a timestamp higher than T) has already read Xi.

If read_TS(Xi)>TS(T):

Abort T because T's write would mess up the order required for serializability.

If No Conflict:

Create a new version of (say Xi) specifically for T:

write_TS(Xj)=TS(T) (the timestamp of T).

read_TS(Xj)=TS(T) (initially, no one else has read it yet

Example:

Initial Versions of X:

X1: write_TS=5,read_TS=10.

X2: write_TS=15,read_TS=20.

Transaction T3 with (T3)=18 wants to write:

Latest suitable version: X2 (write_TS(X2)=15≤18).

Conflict check: read_TS(X2)=20>18.

Conflict! Abort T3

T4 with TS(T4)=25 wants to write:

Latest suitable version: X2 (write_TS(X2)=15≤25).

Conflict check: read_TS(X2)=20≤25. **No conflict! Create a new version X3**

write_TS(X3)=25, read_TS(X3)=25.

Read Operation (read_item(X)):

When a transaction T wants to read a data item X:

1.Find the Latest Suitable Version:

Look for the version of X (say Xi) that was last written by a transaction before T's timestamp.

This means finding the version with the highest write_TS(Xi) ≤ TS(T)).

2. Read the Value and Update read_TS:

Return the value of Xi to T.

Update the read_TS (Xi) to reflect that T has read this version:

read_TS(Xi) =max(read_TS(Xi),TS(T))

3.Guaranteed Success:

A read operation is always successful because there will always be a version of X suitable for T's timestamp.

Example

Initial Versions of X:

X1: write_TS=5,read_TS=10.

X2: write_TS=15,read_TS=20

2.Transaction T3 with TS(T3)=18 wants to read X:

Suitable version: X2 (since write_TS(X2)=15≤18)

Return the value of X2 to T3.

Update read_TS(X2) :

read_TS(X2)=max(20,18)=20

Transaction T4 with TS(T4)=25 wants to read X:

Suitable version: X2 (since write_TS(X2)=15≤25)

Return the value of X2 to T4.

Update read_TS(X2):

read_TS(X2)=max(20,25)=25.

This ensures that transactions can safely read the most relevant version while keeping track of when it was last accessed.

Concurrency Control techniques
→ 2PL → Timestamp
→ MVCC

2. Multiversion Two-Phase Locking (MV2PL) Using Certify Locks:

- This concurrency control mechanism combines multiversioning with a two-phase locking protocol, adding a third type of lock called the certify lock. It ensures serializability while allowing higher concurrency.

Key Components:

Locks Used:

Read Lock (RL): Acquired when a transaction reads a version of a data item.

Write Lock (WL): Acquired when a transaction creates a new version of a data item.

Certify Lock (CL): Ensures no other transaction can read or write to a version after it has been committed by the current transaction.

Version Management:

Multiple versions of a data item X are maintained.

Phases of MV2PL:

1.Growing Phase: Transactions acquire read, write, and certify locks.

2.Shrinking Phase: No new locks are acquired; existing locks are released after the transaction completes.

- In a standard locking scheme, if a transaction writes to a data item, no other transaction can read or write that item until the first transaction is finished.
- Multiversion 2PL improves this by allowing other transactions to read the old, committed version of the data while the current transaction writes to a new version.

Two Versions of Each Item:

- **Committed Version:** Written by a completed transaction.
- **Uncommitted Version:** Created by the current transaction holding the write lock.

While one transaction writes, other transactions can still read the committed version.

Steps in Multiversion 2PL:

1.Write Lock:

- When a transaction T starts writing to an item X, it creates a new version of X
- Other transactions continue to read the old, committed version.

2. Commit Phase (Certify Lock):

- Before T can commit, it must get a certify lock for all items it updated.
- A certify lock ensures no other transaction is reading the old version while T commits.

3.Replace the Old Version:

- After getting the certify locks, T makes its updated version the new committed version.
- The old version is discarded.

Release Locks:

Once done, T releases all locks.

Advantages:

1. Concurrent Reads: Other transactions can read the old version while a write is happening.

2. Avoids Cascading Aborts: Transactions only read committed data, so rollbacks don't cause a chain of aborts

Validation(optimistic) concurrency control technique

- This method assumes that conflicts are rare and doesn't check for them while transactions are running.
- Instead, all the checking happens after the transaction finishes but before it updates the database.
- It's called "optimistic" because it assumes transactions can proceed without interference most of the time.

How It Works:(phases)

1.Read Phase:

- The transaction reads committed data from the database.
- Updates are made to a local copy of the data (not the database).

2.Validation Phase:

- After the transaction finishes, a check is performed to ensure that its updates won't conflict with other transactions.
- If there's a conflict, the transaction is aborted and restarted.

3.Write Phase:

- If validation is successful, the updates from the local copy are written to the database.

Validation Rules:

- When validating a transaction Ti, the system ensures it doesn't interfere with another transaction Tj (committed or validating).

To avoid conflicts, one of these three conditions must hold:

- **Condition 1:**
- Tj's write phase ends before Ti starts its read phase.
- Means Ti read data after Tj completed. No overlap.
- **Condition 2:**

Ti's write phase starts after Tj's write phase ends, and Ti's read set and Tj's write set don't overlap.

- Tj and Ti didn't modify or read the same data during their active phases.

Condition 3:

- Tj's read phase ends before Ti's read phase ends, and Ti's read/write sets don't overlap with Tj's write set.
- Ensures no shared data between Ti and Tj.

If Validation Fails:

- The transaction Ti is aborted and restarted because conflicts occurred.

Advantages:

- Minimal overhead during transaction execution (no locking or checks).
- Works well when transaction conflicts are rare.

Disadvantages:

- If conflicts are frequent, many transactions may be aborted and restarted, wasting resources.

Phantom Phenomenon:

- Phantom phenomenon occurs in concurrent database systems when a transaction retrieves a set of rows satisfying a certain condition, and another transaction inserts or deletes rows that also meet that condition.
- This leads to inconsistent query results because the retrieved data changes during the transaction's execution.

Example:

- **Transaction T1:** Reads all rows in a table where salary > 5000.
- **Transaction T2:** Inserts a new row with salary = 6000 while T1 is still executing.
- When T1 re-executes the query, it sees an extra row ("phantom row") that was not present earlier.

Timestamp protocol that avoid phantom phenomena

The timestamp ordering protocol can avoid the phantom phenomenon by ensuring that transactions are serialized according to their timestamps.

Here's how it works:

Assign Timestamps: Each transaction is assigned a unique timestamp TS(T) when it begins. This determines its order relative to other transactions.

Handle operations:

1. Read Operation (read_item X):

A transaction T can read a data item X only if TS(T) (transaction's timestamp) is greater than or equal to the write timestamp (write_TS) of X.

If TS(T)<writeTS(X), it means T is trying to read an item modified by a newer transaction, so it is aborted.

2. Write Operation (write_item X):

A transaction T can write to X only if TS(T) is greater than or equal to the read timestamp (read_TS) and write timestamp (write_TS) of X.

If TS(T)<readTS(X)or TS(T)<writeTS(X) , T conflicts with a previously completed transaction, so it is aborted.

Range Queries to Avoid Phantoms:

To handle range queries (e.g., "SELECT WHERE salary > 5000"), the protocol applies the following:

- Lock on ranges of data rather than individual rows.
- The transaction T checks the timestamps of all rows that currently exist in the range and also prevents new rows from being inserted into the range by conflicting transactions.

Ensuring Serializability:

- The protocol ensures that transactions are serialized in the order of their timestamps, which avoids inconsistencies caused by the insertion or deletion of rows that match a query condition.

AKTU PYQS

Q.1	Discuss the timestamp ordering protocol for concurrency control. How does strict timestamp ordering differ from basic timestamp ordering?	AKTU 2023-24
Q.2	How do optimistic concurrency control techniques differ from other concurrency control techniques? Why they are also called validation or certification techniques? Discuss the typical phases of an optimistic concurrency control method.	AKTU 2023-24
Q.3	Explain Time Stamp Based Concurrency Control technique	AKTU 2021-22

AKTU PYQS

Q.4	Explain the Validation Based protocol for concurrency control	AKTU2021-22
Q.5	What is phantom phenomena Discuss the timestamp protocol that avoid phantom phenomena	AKTU 2021-22

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -5
Concurrency control technique
Lecture-4

Today's Target

- Recovery from concurrent transaction
- Granularity on data items and multiple granularity locks
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

How validation differ from 2PL

Validation (Optimistic Concurrency Control)

Approach: Transactions execute without locking resources, assuming conflicts are rare.

Validation Phase: Before committing, the system checks if the transaction conflicts with others. If it passes validation, it commits; otherwise, it aborts.

Performance: Suitable for low-contention environments.

2PL Protocol (Two-Phase Locking)

Approach: Transactions acquire locks on resources to ensure no conflicts occur during execution.

Phases:

- **Growing Phase:** Locks are acquired.
- **Shrinking Phase:** Locks are released, and no new locks can be acquired.

Performance: Provides strict consistency but may cause delays due to locking contention.

Key Difference: Validation allows optimistic execution with a check at the end, while 2PL ensures safety by locking throughout the transaction.

Recovery from concurrent transaction

Recovery from Concurrent Transactions refers to the process of ensuring the consistency of a database system after a failure, where multiple transactions were executing concurrently. The goal is to maintain the ACID properties (Atomicity, Consistency, Isolation, Durability) of the system despite crashes or failures that occur during the execution of concurrent transactions.

There are various strategies and techniques for handling recovery from concurrent transactions.

KEYPOINTS:

1. Transaction Log:

- A log records all the operations (read, write) and changes made by each transaction.
- The log is essential for the redo and undo operations during recovery.

2. Undo/Redo

Mechanisms:

- **Undo:** Reverts any incomplete changes made by a transaction before it failed or was aborted.
- **Redo:** Re-applies changes made by transactions that were committed but whose changes were not written to the database due to a failure.

Recovery Process from Concurrent Transactions

Before and After Images:

A database keeps track of before and after images of a transaction's effects:

Before image: The value of a data item before it is updated.

After image: The value of a data item after it has been updated.

These images help determine which actions need to be undone or redone.

Transaction States:

During recovery, transactions can be in various states, which help in determining the recovery actions:

- **Committed:** The transaction has completed successfully and its changes are permanent.
- **Active:** The transaction is still in execution and hasn't yet completed.
- **Aborted:** The transaction has been terminated due to a failure.

Recovery Protocols:

The recovery protocols are designed to handle scenarios where multiple transactions execute concurrently, such as:

1. **Write-Ahead Logging (WAL):** Ensures that changes to the database are first recorded in the log before they are applied to the actual data. This helps in preventing partial updates when a failure occurs before a transaction commits.

Saves the transaction log locally to prepare for possible recovery.

Participants respond to the coordinator:

Yes (Prepared): If they can commit.

No (Abort): If they cannot commit.

Phase 2: Commit/Abort Phase

Based on the responses:

- If all participants reply Yes, the coordinator sends a COMMIT request to all participants, and they commit the transaction.

If any participant replies No, the coordinator sends an ABORT request, and all participants abort the transaction.

Each participant: Logs the commit or abort.

Releases any locks held for the transaction.

2. Checkpoints:

- A checkpoint in DBMS is a mechanism to reduce the work needed during database recovery after a failure.
- It is a snapshot of the database's current state, taken at a specific point in time, which includes information about active transactions and the data modified by them. Checkpoints help in limiting the amount of redo and undo operations required during recovery

Two phase commit protocol

The Two-Phase Commit (2PC) protocol ensures atomicity in distributed transactions, ensuring that all participating nodes in a transaction either commit or abort together, even in the presence of failures.

Phases of the 2PC Protocol:

Phase 1: Prepare Phase

- The coordinator sends a PREPARE request to all participating nodes, asking them if they can commit.
- Each participant checks:
  - If it can commit the transaction (e.g., no resource conflicts, all changes can be applied).

Phase 2: Commit/Abort Phase

Based on the responses:

- If all participants reply Yes, the coordinator sends a COMMIT request to all participants, and they commit the transaction.
- If any participant replies No, the coordinator sends an ABORT request, and all participants abort the transaction.

Each participant:

- Logs the commit or abort.
- Releases any locks held for the transaction.

Recovery Procedure:

Identify Transactions to Undo or Redo:

- First, identify which transactions need to be undone or redone using the transaction log.
- Transactions that were active at the time of failure must be undone (rollback).
- Transactions that were committed before the failure should have their changes redone.

- **Undo Phase**: Active transactions are rolled back using the before images in the transaction log to restore data to its state before the transaction started.
- **Redo Phase**: For transactions that were committed but did not complete successfully before the crash, their changes are redone using the after images from the log to ensure their updates are applied.

Handling Cascading Aborts:

- Cascading aborts occur when one transaction failure leads to the need to abort other transactions that depend on it.
- To avoid this, protocols like Strict Two-Phase Locking (2PL) ensure that a transaction only reads committed data, thus preventing such scenarios.
- Recovery mechanisms, like Write-Ahead Logging, also help in ensuring that partial updates are not propagated, and only fully committed transactions are included in the redo phase.

Granularity on data items and multiple granularity locks

All concurrency control techniques assume that the database is formed of a number of named data items.

A database item could be chosen to be one of the following:

A database record , A field value of a database record , A disk block ,A whole file ,The whole database

- The size of data items is often called the data item granularity.
- The granularity level refers to the size of the data item that a lock applies to, such as a record, block, or file.
- Choosing the right size is important because it affects concurrency (how many transactions can run at the same time) and overhead (system effort to manage locks).

Tradeoffs in Granularity

1. Fine Granularity (Small Items like Records)

More Concurrency: Smaller items mean fewer conflicts between transactions.

For example: If Transaction A locks Record 1, and Transaction B wants Record 2 (in the same block), both can proceed

Higher Overhead:

More items mean:

More locks to manage.

More operations (lock/unlock).

Larger lock tables, which need more storage.

2.Coarse Granularity (Large Items like Blocks or Files):Less Concurrency:

Larger items mean more conflicts.

For example: If Transaction A locks a block to access Record 1, and Transaction B wants Record 2 (in the same block), Transaction B must wait.

Lower Overhead:

Fewer items mean:

- Fewer locks to handle.
- Smaller lock tables and fewer operations.

Choosing the Right Granularity

If transactions access a few specific records, fine granularity (record-level) is better for higher concurrency.

If transactions access many records in the same block or file, coarse granularity (block/file-level) is better to reduce overhead.

Example:

- Fine granularity works well for banking transactions that update specific accounts.
- Coarse granularity is better for batch processing, where large amounts of data in a file are read or updated.

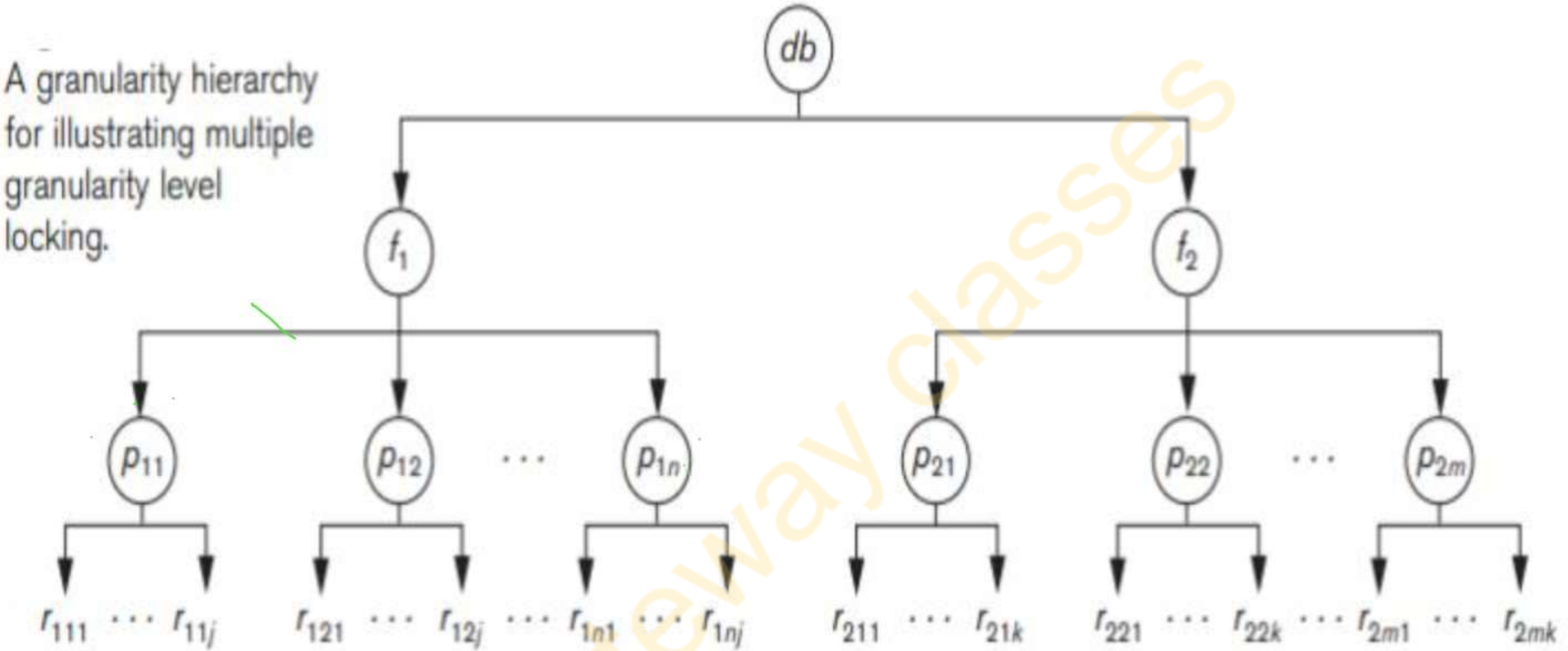
Multiple Granularity Level Locking

Problem with Basic Locking

- If a large transaction locks a file (exclusive lock), small transactions trying to access individual records in that file must wait.
- If a small transaction locks a record (shared lock), it's inefficient for the system to check all individual locks when a large transaction wants to lock the file.

multiple granularity level 2PL protocol, where a lock can be requested at any level. However, additional types of locks will be needed to support such a protocol efficient.

A granularity hierarchy for illustrating multiple granularity level locking.



To solve this, intention locks were introduced to indicate the type of locks needed for child nodes in a hierarchy (e.g., database → file → page → record).

Types of Intention Locks:

IS (Intention-Shared): Shows the intention to request shared locks on child nodes.

IX (Intention-Exclusive): Shows the intention to request exclusive locks on child nodes.

SIX (Shared-Intention-Exclusive): Locks the current node in shared mode but allows exclusive locks on child nodes.

The compatibility table of the three intention locks, and the shared and exclusive locks

	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

The multiple granularity locking (MGL) protocol consists of the following rules:

1. The lock compatibility must be adhered to.

2. The root of the tree must be locked first, in any mode.

3. A node N can be locked by a transaction T in S or IS mode only if the parent node N is already locked by transaction T in either IS or IX mode.

4. A node N can be locked by a transaction T in X, IX, or SIX mode only if the parent of node N is already locked by transaction T in either IX or SIX mode.

5. A transaction T can lock a node only if it has not unlocked any node (to enforce the 2PL protocol).

6. A transaction T can unlock a node, N, only if none of the children of node N are currently locked by T

AKTU PYQS		
Q.1	Explain the concurrent control mechanism	AKTU 2022-23
Q.2	Explain the multiple granularity lock	AKTU 2018-19 AKTU 2021-22