Database Management System

Unit IV – Chapter 8

Introduction: Introduction to transaction processing, transaction and system concepts, desirable properties of transactions, transaction support in SQL. Concurrency control techniques: two-phase locking techniques, concurrency control based on timestamp ordering. Recovery techniques: recovery concepts, recovery in multi-database systems, database backup, and recovery from catastrophic failures.

Transaction processing

The concept of the transaction provides a mechanism for describing logical units of database processing. Transaction processing systems are with large databases and hundreds of concurrent users that are executing database transactions. These systems require high availability and fast response time for hundreds of concurrent users.

Examples of such systems include airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications. The concept of a transaction is used to represent a logical unit of database processing that must be completed in its entirety to ensure correctness. A transaction is typically implemented by a computer program that includes database commands such as retrievals, insertions, deletions, and updates.

One criterion for classifying a database system is the number of users who can use the system concurrently.

Single user and Multiuser Systems

- Single-User
- Multi-User
- Multiprogramming
- Interleaving

A DBMS is a **single-user if at most one user at** a time can use the system, and it is **multiuser if many users can use the system.**

Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser.

For example - an airline reservations system is used by hundreds of users and travel agents concurrently.

Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems.

Single-User: At most one user at a time can use the system

Multi-User: Many users can use the system concurrently.

Multiprogramming: This allows the computer to execute multiple programs at the same time.

Interleaving: **Keeps the CPU busy when a process requires an input or output operation**, the CPU switched to execute another process rather than remaining idle during I/O time.

Most of the theory concerning concurrency control in databases is developed in terms of interleaved concurrency.

Transactions, Read and Write Operations

- A Transaction is a logical unit of database processing, that includes one or more database access operations.
- These transactions are insertion, deletion, modification (update), or retrieval operations.
- All database access operations between **Begin Transaction** and **End Transaction** statements are considered as one logical transaction.
- If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**, otherwise, it is known as a **read-write transaction**.

Basic database access operations:

read_item(X) : Reads a database item X into program variable.

write_item(X): Writes the value of program variable X into the database item X.

read_item(X): reads a database item named X into a program variable also named X.

Execution of the command includes the following steps:

- Find the address of the disk block that contains item X.
- Copy that disk block into a buffer in the main memory.
- Copy item X from the buffer to the program variable named X.

write_item(X): writes the value of program variable X into the database item named X.

Execution of the command includes the following steps:

- Find the address of the disk block that contains item X.
- Copy that disk block into a buffer in the main memory.
- Copy item X from the program variable named X into its current location in the buffer.
- Store the updated block in the buffer back to disk (this step updates the database on disk).

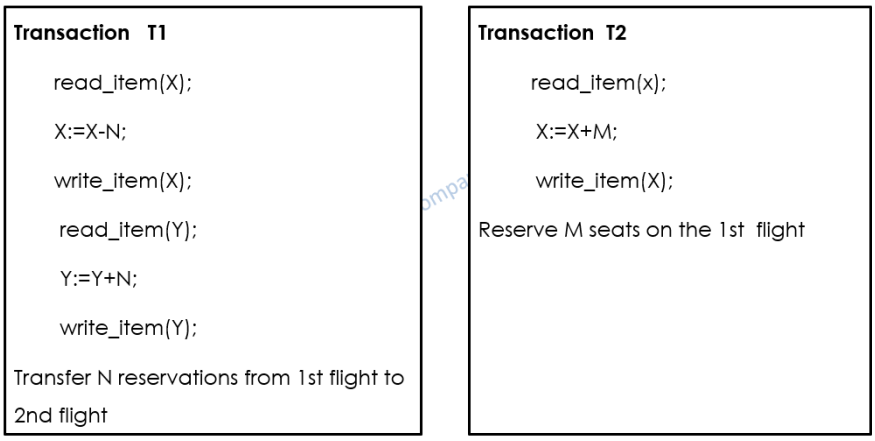
Why Concurrency Control Is Needed

Several problems can occur when concurrent transactions execute in an uncontrolled manner. **Example:** Airline's reservation database. Various types of problems occur when the two-transaction run concurrently.

Types of problems:

- The Lost Update Problem
- The temporary Update (or Dirty read) problem
- The Incorrect Summary Problem
- Unrepeatable Read problem

Transaction processing



Where X & Y refers to some reserved seats for a specific flight.

Transactions – Examples

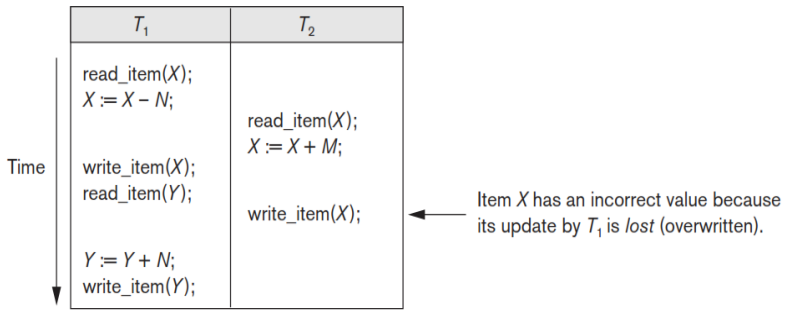
Example T1: Rs.2000 is deducted from my account and added into other-account

Example T2: Rs.1000 is added to my-account

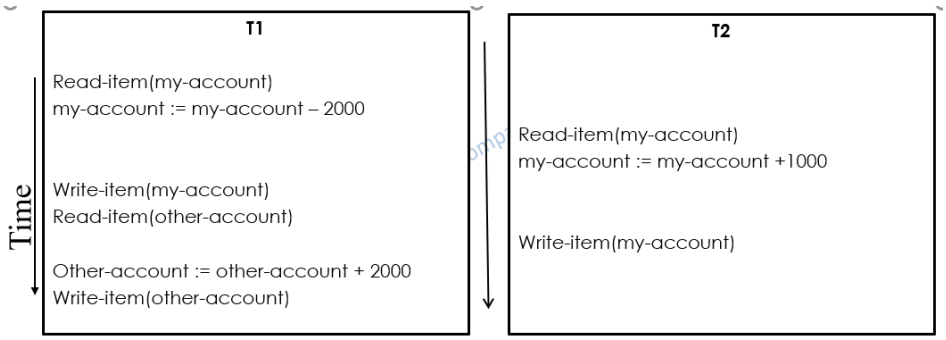
T1	T2
Read-item(my-account) my-account := my-account - 2000 Write-item(my-account) Read-item(other-account) Other-account := other-account + 2000 Write-item(other-account)	Read-item(my-account) my-account := my-account +1000 Write-item(my-account)

The Lost Update Problem:

This problem occurs when two transactions access the same database items concurrently.



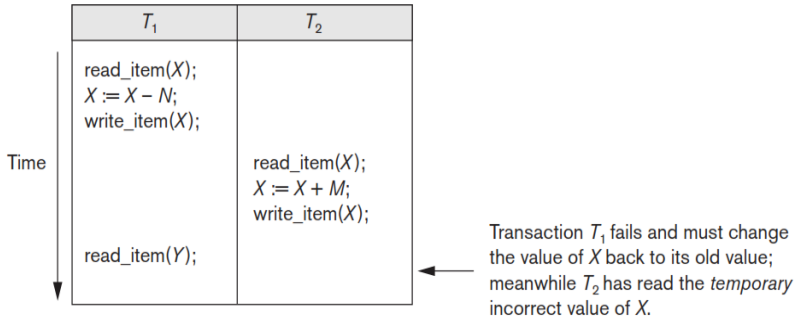
Example: Lost update problem



Example T1: Rs.2000 is deducted from my-account **Example T2:** Rs.1000 is added to my-account

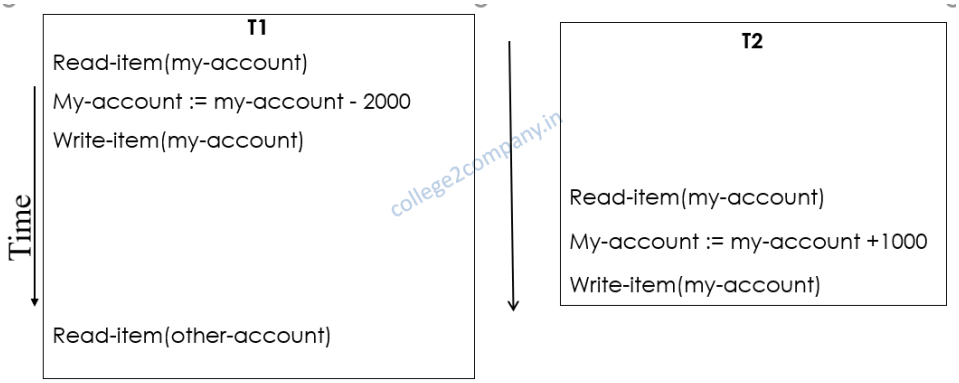
Temporary Update (or Dirty read) problem:

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The update item is accessed by another transaction before it is changed back to its original value.



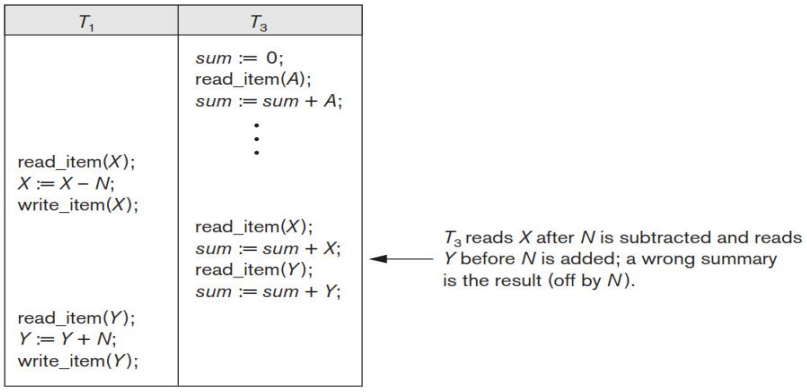
The Temporary Update (or Dirty Read) Problem. This problem occurs when one transaction updates a database item and then the transaction fails for some reason Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value. An example where T updates item X and then fails before completion, so the system must roll back X to its original value. Before it can do so, however, transaction T1 reads the temporary value of X, which will not be recorded permanently in the database because of the failure of T1. The value of item X that is read by T is called dirty data because it has been created by a transaction that has not been completed and committed yet; hence, this problem is also known as the dirty read problem.

Example: Temporary Update problem



Incorrect Summary Problem

If one transaction is calculating an aggregated summary function on several records while another transaction is updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.



The Incorrect Summary Problem. If one transaction is calculating an aggregate summary function on several database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated. For example, suppose that a transaction T is calculating the total number of reservations on all the flights; meanwhile, transaction T3 is executing.

X := X ・ N;

write_item (X);

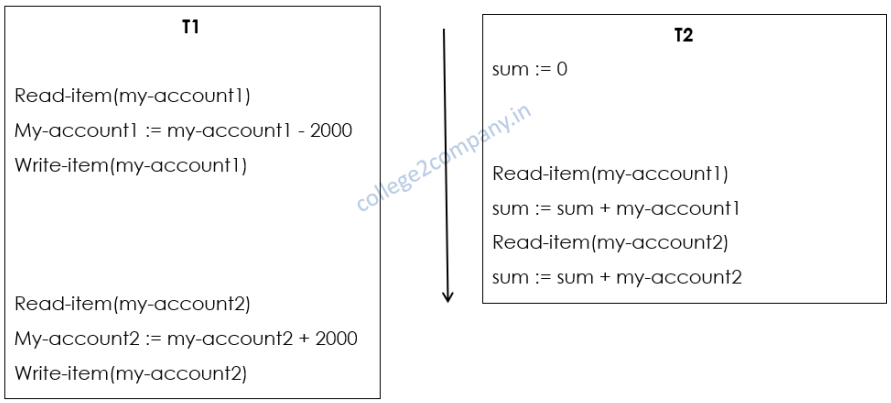
read_item (Y);

Y := Y + N;

write_item (Y);

If the interleaving of operations is shown, the result of T1 T3 will be off by an amount N because T is the value of X after N seats has been subtracted from it but reads the value of Y before those N seats have been added to it.

Example: Incorrect Summary Problem



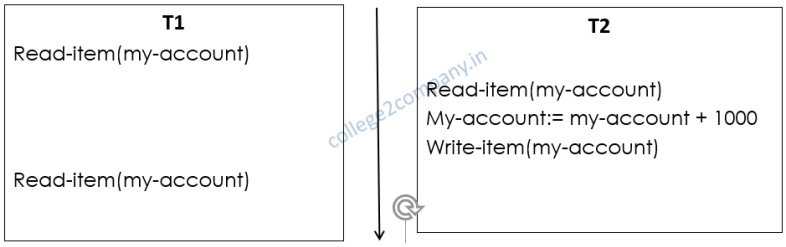
Unrepeatable Read Problem

A transaction reads items twice with two different values because it was changed by another transaction between the two reads. **Example:** Bank, Flight Reservation – A customer inquiries about seat availability on different flights. When he decides to book, the number of seats will be different because other transactions update the number of seats.

The Unrepeatable Read Problem. Another problem that may occur is called unrepeatable read, where a transaction T, reads the same item twice and the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item. This may occur, for example, during an airline reservation transaction, a customer inquiry about seat availability on several flights. When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up reading a different value for the item.

Example: Unrepeatable Read Problem

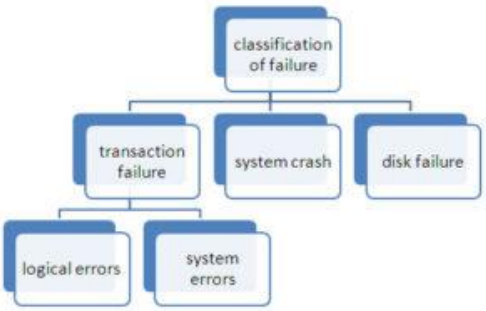
Example: Bank, Flight Reservation – A customer inquiries about seat availability in diff. flights. When he decides to book, the number of seats will be different because T2 updates the number of seats.



Why Recovery Is Needed

There are several possible reasons for a transaction to fail.

- A computer failure
- A transaction or system error
- Local errors or exception
- Concurrency control enforcement
- Disk failure
- Physical problems



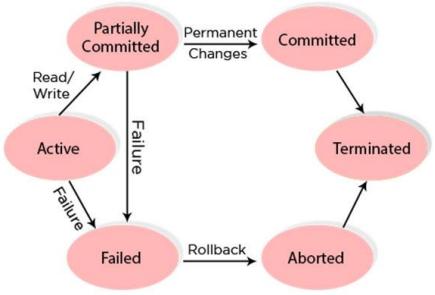
- **A computer failure:** A hardware, software, or network error occurs in the computer system during transaction execution.
- **A transaction or system error:** Some operations in the transaction may cause it to fail.
- **Local errors or exception** conditions detected by the transaction.
- ***Concurrency control enforcement***: The concurrency control method may decide to abort the transaction
- ***Disk failure***: all disk or some disk blocks may lose their data.

- ***Physical problems***: Disasters, theft, fire, etc.

The system must keep sufficient information to recover from the failure.

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts. The recovery manager keeps track of the following operations(Transaction states):

- BEGIN_TRANSACTION
- READ OR WRITE
- END_TRANSACTION
- COMMIT_TRANSACTION
- ROLLBACK



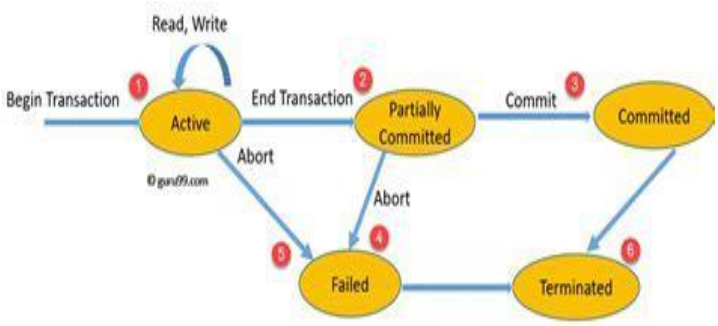
BEGIN_TRANSACTION: This marks the beginning of transaction execution.

READ OR WRITE: Specifies READ or WRITE transaction operations on the database items that are executed as part of a transaction.

END_TRANSACTION: Specifies the READ and WRITE transaction operations have ended and marks the end of transaction execution.

COMMIT_TRANSACTION: This signals a successful end of the transaction.

ROLLBACK (or ABORT): This signals that the transaction has ended unsuccessfully so that any changes or effects that the transaction may have applied to the database must be undone.



State transition diagram illustrating the states for transaction execution

A state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an active state immediately after it starts execution, where it can issue READ and WRITE operations. When the transaction ends it moves to the partially committed state. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently. Once this check is successful the transaction is said to have reached its commit point and enters the committed state. A transaction can go to the failed state if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database. The terminated state corresponds to the transaction leaving the system.

Transactions should possess several properties, often called the **ACID properties.**

ACID Properties should be enforced by the concurrency control and recovery methods of the DBMS.

ACID Properties of transactions

- Atomicity
- Consistency preservation
- Isolation
- Durability

Atomicity: a transaction is an atomic unit of processing; it is either performed entirely or not performed at all. **(It is the responsibility of recovery)**

Consistency: transfer the database from one consistent state to another consistent state. **(It is the responsibility of the applications and DBMS to maintain the constraints)**

Isolation: the execution of the transaction should be isolated from other transactions (Locking). **(It is the responsibility of concurrency)**

* **Isolation level:**

- Level 0 (no dirty read)
- Level 1 (no lost update)
- Level 2 (no dirty+ no lost)
- Level 3 (level 2+repeatable reads)

Durability: committed transactions must persist in the database, i.e., those changes must not be lost because of any failure. **(It is the responsibility of recovery)**

The basic definition of an SQL transaction is, it is a logical unit of work and is guaranteed to be atomic. A single SQL statement is always considered to be atomic either it completes execution without an error or it fails and leaves the database unchanged. With SQL, there is no explicit Begin Transaction statement. Transaction initiation is done implicitly when particular SQL statements are encountered. However, every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK.

Every transaction has certain characteristics attributed to it. These characteristics are specified by a SET TRANSACTION statement in SQL.

Characteristics are:

- Access mode
- the diagnostic area sizes
- the isolation levels

The **access mode** can be specified as READ ONLY or READ WRITE.

The **diagnostic area size** option, DIAGNOSTIC SIZE n, specifies an integer value n, which indicates the number of conditions that can be held simultaneously in the diagnostic area.

The **isolation level** option is specified using the statement ISOLATION LEVEL <isolation>, where the value for <isolation> can be READ UNCOMMITTED, READ COMMITTED.

Unit IV Chapter 9

Concurrency Control

Concurrency Control in the management procedure that is required for controlling the concurrent execution of the operations that take place on a database. In a multi-user system, multiple users can access and use the same database at one time. The simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations. In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.

Two-Phase Locking Techniques for concurrency control

The main technique used to control the concurrent execution of transactions is based on the concept of locking data items. A lock is a variable associated with a data item that describes the status of the item for possible operations that can be applied to it. Generally, there is one lock for each data item in the database.

Types of locks and system lock tables

- Binary Locks
- Shared/Exclusive (or Read/write) Locks
- Conversion of Locks

Binary Locks

A binary lock can have two states or values: locked and unlocked (or 1 and 0). 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. We refer to the current value (or state) of the lock associated with item X as lock(X).

The following code performs the lock operation:

```
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;
```

Two operations, lock_item, and unlock_item, are used with binary locking. A transaction requests access to an item X by first issuing a **lock_item(X)** operation. LOCK (X) = 1, the transaction is forced to wait. If LOCK(X) = 0, it is set to 1 (the transaction **locks the item), and the transaction is allowed to access item X.**

The following code performs the Unlock operation:

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

When the transaction is through using the item, it issues an **unlock_item(X) operation,** which sets LOCK(X) back to 0 (**unlocks the item) so that X may be accessed by** other transactions. Hence, a binary lock enforces **mutual exclusion on the data** item. A description of the lock_item(X) and unlock_item(X) operations.

The binary locking scheme described that every transaction must follow these rules:

A transaction T must issue the operation **lock_item(X)** before any read_item(X) or write_item(X) operations are performed in T.

A transaction T must issue the operation **unlock_item(X)** after all read_item(X) And write_item(X) operations are completed in T.

A transaction T, will not issue a **lock_item(X)** operation if it already holds the lock on item X.

A transaction T, will not issue an **unlock_item(X)** operation unless it already holds the lock on item X.

Shared/Exclusive (or Read/Write) Locks

The binary locking scheme is too restrictive for database items because at most one transaction can hold a lock on a given item. But we should allow several transactions to access the same item X if they all access X for reading purposes only. If a transaction is to write an item X, it must have exclusive access to X.

A different type of lock, called a multiple-mode lock, is used which is called a shared/exclusive or read/write locks.

There are three locking operations:

- ✓ read_lock(X)
- ✓ write_lock(X)
- ✓ unlock(X)

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

Locking operations for two-mode (read/write or shared/exclusive) locks

The following code performs the read lock operation: (read_lock is also called shared_locked)

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin 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;
```

The following code performs the write lock operation: (write_lock is also called exclusive_locked)

```
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;
```

Unlocking operations for two-mode (read/write or shared/exclusive) locks

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
            wakeup one of the waiting transactions, if any
            end
    else it 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. This rule maybe

relaxed for the downgrading of locks, as we discuss shortly.

5. A transaction T, will not issue a write_lock(X) operation if it already holds a read (shared) lock or writes (exclusive) lock on item X. This rule may also be relaxed for upgrading of locks, as we discuss shortly.

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 Locks

Lock Conversion: The transaction that already holds a lock on item X is allowed under certain conditions to convert the lock from one locked state to another.

Two Phases:

- Locking (Growing)
- Unlocking (Shrinking)

Locking (Growing) Phase: A transaction applies locks (read or write) on desired data items one at a time.

Unlocking (Shrinking) Phase: A transaction unlocks its locked data items one at a time.

Requirement: For a transaction, these two phases must be mutually exclusive, that is, during the locking phase unlocking phase must not start and during the unlocking phase-locking phase must not begin.

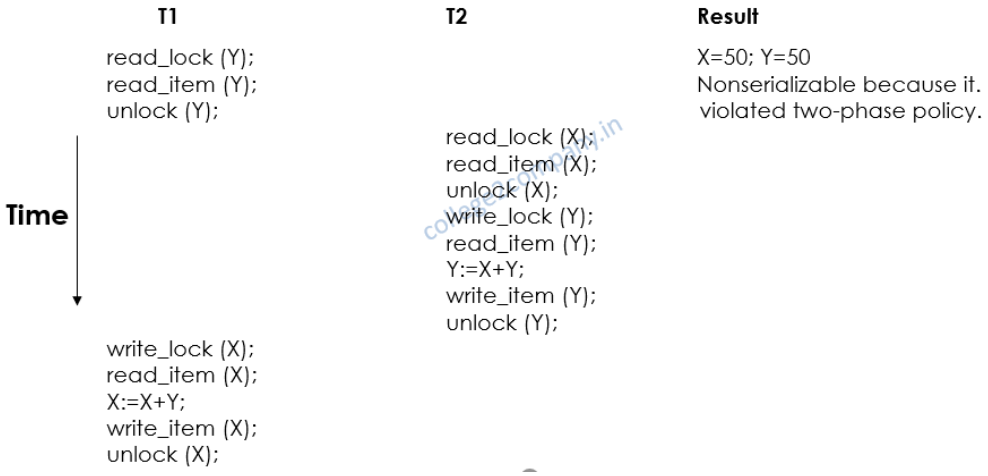
Example

A transaction T can issue a read_lock(X) and then, later on, 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 on, to downgrade the lock by issuing a read_lock(X) operation.

Two-Phase Locking Techniques: The algorithm

T1	T2	Result
read_lock (Y); read_item (Y); unlock (Y); write_lock (X); read_item (X); X:=X+Y; write_item (X); unlock (X);	read_lock (X); read_item (X); unlock (X); Write_lock (Y); read_item (Y); Y:=X+Y; write_item (Y); unlock (Y);	Initial values: X=20; Y=30 Result of serial execution T1 followed by T2 X=50, Y=80. Result of serial execution T2 followed by T1 X=70, Y=50

Two-Phase Locking Techniques: The algorithm



T1	T2	
read_lock (Y);	read_lock (X);	T1 and T2 follow two-phase
read_item (Y);	read_item (X);	policy but they are subject to
write_lock (X);	Write_lock (Y);	deadlock, which must be
unlock (Y);	unlock (X);	dealt with.
read_item (X);	read_item (Y);	
X:=X+Y;	Y:=X+Y;	
write_item (X);	write_item (Y);	
unlock (X);	unlock (Y);	

Two-Phase Locking Techniques: The algorithm

A two-phase policy generates two locking algorithms (a) Basic and (b) Conservative.

Conservative: Prevents deadlock by locking all desired data items before the transaction begins execution.

Basic: Transaction locks data items incrementally. This may cause deadlock which is dealt with.

Strict: A stricter version of the Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled back). This is the most commonly used two-phase locking algorithm.

Timestamp based concurrency control algorithm

A timestamp 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.

Basic Timestamp Ordering

Transaction T issues a write_item(X) operation:

- ✓ If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then a younger transaction has already read the data item so abort and roll back T and reject the operation.
- ✓ If the condition in part (a) does not exist, then execute write_item(X) of T and set write_TS(X) to TS(T).
- ✓ Transaction T issues a read_item(X) operation:
- ✓ If write_TS(X) > TS(T), then a younger transaction has already been written to the data item so abort and roll back T and reject the operation.
- ✓ If write_TS(X) ≤ TS(T), then execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Unit III Chapter – 10

Recovery techniques

Recovery concepts

Purpose of Database Recovery:

Is to bring the database into the last consistent state, which existed before the failure. To preserve transaction properties (Atomicity, Consistency, Isolation, and Durability). Example: If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have an incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

Types of Failure:

The database may become unavailable for use due to transaction failure. Transactions may fail because of incorrect input, deadlock, incorrect synchronization. System failure: The system may fail because of addressing error, application error, operating system fault, RAM failure, etc. Media failure: Disk head crash, power disruption, etc.

Data Update

Deferred Update: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

Immediate Update: As soon as a data item is modified in the cache, the disk copy is updated.

Recovery Techniques

- Recovery Techniques Based on Deferred Update (No Undo/Redo)
- Recovery Techniques Based on Immediate Update

Recovery Techniques Based on Deferred Update

Deferred update techniques are to defer or postpone any actual updates to the database until the transaction completes its execution successfully, and reaches its commit point. A set of transactions records their updates in the log and cache buffers. After the transaction reaches its commit point and the log is force-written to disk, the updates are

recorded in the database. If a transaction fails before reaching its commit point, there is no need to undo any operations, because the transaction has not affected the database on disk. We can state a typical deferred update protocol as follow: A transaction cannot change the database on disk until it reaches its commit point. A transaction does not reach its commit point until all its update operations are recorded in the log and the log is forced written to disk.

Deferred Update in a single-user system

No concurrent data is shared in a single user system. The data update goes as follows: A set of transactions records their updates in the log. After the transaction reaches its commit point and the log is force-written to disk, the updates are recorded in the database. After rebooting from a failure, the log is used to redo all the transactions affected by this failure.

Deferred Update in a single-user system:

(a)

T1	T2
read_item (A)	read_item (B)
read_item (D)	write_item (B)
write_item (D)	read_item (D)
write_item (D)	

(b)

[start_transaction, T1]
[write_item, T1, D, 20]
[commit T1]
[start_transaction, T2]
[write_item, T2, B, 10]
[write_item, T2, D, 25] ← system crash

The [write_item, ...] operations of T1 are redone.
T2 log entries are ignored by the recovery manager.

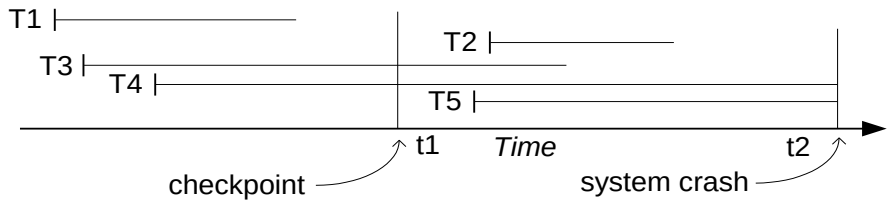
Deferred Update with concurrent users

This environment requires some concurrency control mechanism to guarantee the isolation property of transactions. In a system recovery transaction, which was recorded in the log after the last checkpoint was redone. The recovery manager may scan some of the transactions recorded before the checkpoint.

Example

When the checkpoint was taken at time t1, transaction T1 had committed, whereas transactions T3 & T4 had not. Before the system crash at time t2, T3 & T2 were committed but not T4 & T5. According to the RDU_M method, there is no need to redo the write_item operations of transaction T1. Write_item operation of T2 & T3 must be redone because both transactions reached their commit point after the last checkpoint. Recall that the log is force-written before committing a transaction. Transaction T4 & T5 are ignored. They are effectively canceled or rolled back because none of their write_item operations were recorded in the database.

Recovery in a concurrent user's environment



Recovery in multi-database system

A multi-database system is a special distributed database system where one node may be running a relational database system under Unix, another may be running an object-oriented system under a window, and so on. A transaction may run in a distributed fashion at multiple nodes. In this execution scenario, the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing. This commit scheme is referred to as a "two-phase commit" (2PC). If any one of these nodes fails or cannot commit part of the transaction, then the transaction is aborted. Each node recovers the transaction under its recovery protocol.

Database Backup and Recovery from Catastrophic Failures

A key assumption has been that the system log is maintained on the disk and is not lost as a result of the failure. The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as disk crashes. The main technique used to handle such crashes is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or another large capacity offline. storage devices. In case of a catastrophic system failure, the latest backup

copy can be reloaded from the tape to the disk, and the system can be restarted. For example, the data from critical applications such as banking, insurance, stock market, and other databases are periodically backed up in its entirety and moved to physically separate safe locations. To avoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape.

References

1. Elmasri and Navathe: Fundamentals of Database Systems, 7th Edition, Addison - Wesley, 2016.

2. Silberschatz, Korth and Sudharshan Database System Concepts, 7th Edition, Tata McGraw Hill,

2019.

3. C.J. Date, A. Kannan, S. Swamynatham: An Introduction to Database Systems, 8th Edition, Pearson

education, 2009

4. Database Management Systems: Raghu Ramakrishnan and Johannes Gehrke: 3rd Edition,

McGraw-Hill, 2003.

5. Reference - Web Content