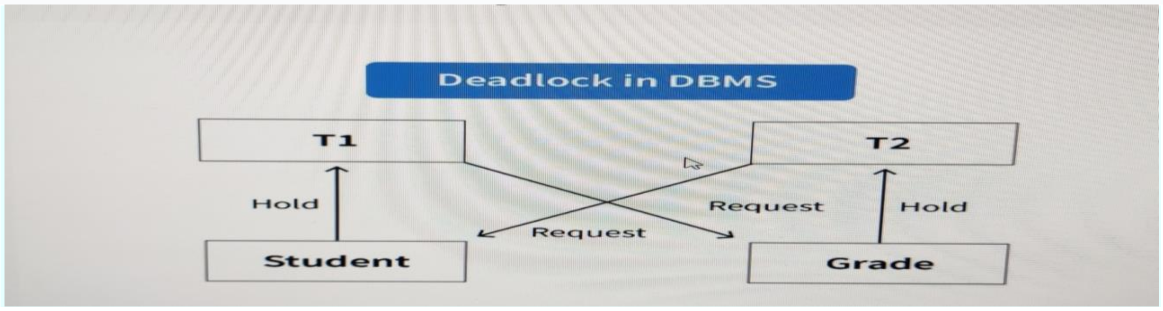
Unit 4

Deadlock

Deadlock in a database management system (DBMS) is an undesired **situation in which two or more transactions have to wait indefinitely for each other in order to get terminated, but none of the transactions is willing to give up the allocated CPU and memory resources that the other one needs**. Deadlock brings the whole system to a halt as no task ever gets finished and is in waiting state forever.

Let's understand deadlock in DBMS with an example of students database, where transaction 1 holds a lock on certain records of the Student table and needs to update those records in the Grade table. Simultaneously, there is transaction 2, which holds locks on some other records in the Grade table and needs to update those records in the Student table, which are already held by transaction 1.

Now, the main problem of deadlock arises when transaction 1 is waiting for Transaction 2 to release its lock and similarly, transaction Transaction 2 is waiting for Transaction 1 to release its lock. All activities come to a halt. *Both the transaction involved in such a situation will remain at a standstill until the DBMS detects the deadlock and aborts one of the two transactions that are causing deadlock.*



The three basic techniques to control deadlocks are:

• Deadlock prevention

A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.

• Deadlock detection

The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the "victim") is aborted (rolled back and restarted) and the other transaction continues.

• **Deadlock avoidance**

The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rolling back conflicting transactions by requiring that locks be obtained in succession. However, the serial lock assignment required in deadlock avoidance increases action response times.

Concurrency Control in DBMS

Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems. Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data. For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

Advantages of Concurrency:

In general, concurrency means, that more than one transaction can work on a system.

The advantages of a concurrent system are:

- **Waiting Time:** It means if a process is in a ready state but still the process does not get the system to get execute is called waiting time. So, concurrency leads to <u>less waiting time</u>.
- **Response Time:** The time wasted in getting the response from the cpu for the first time, is called response time. So, concurrency leads to <u>less Response Time</u>.
- **Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to <u>more Resource Utilization</u>.
- **Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to <u>more Efficiency</u>.

Major goals of concurrency control mechanisms

- **Serializability: -** Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read and write operations.
- Recoverability.
- Distributed serializability and commitment ordering.
- Distributed recoverability.
- Recovery.
- Replication.

Concurrency control techniques
The concurrency control techniques are as follows –
Locking
Lock guaranties exclusive use of data items to a current transaction. It first accesses the data items by acquiring a lock, after completion of the transaction it releases the lock. Types of Locks

Introduction to Locking

Locking is one of the most commonly used concurrency control schemes in DBMS. This works by associating a variable ***lock*** on the data items. This variable describes the status of the data item with respect to the possible operations that can be applied on it. The value of this variable is used to control the concurrent access and the manipulation of the associated data item.

The concurrency control technique in which the value of the lock variable is manipulated is called **locking**. The technique of locking is one way to ensure Serializability in DBMS.

In DBMS, locking is the responsibility of a subsystem called **lock manager**.

The types of locks are as follows –

Binary Locks

A binary lock has two states or values associated with each data item. These values are:

1. **Locked – 1**
2. **Unlocked – 0**

If a data item is **locked**, then it cannot be accessed by other transactions i.e., other transactions are forced to wait until the lock is released by the previous transaction.

But, if a data item is in the **unlocked** state, then, it can be accessed by any transaction and on access the lock value is set to locked state.

These locks are applied and removed using **Lock** () and **Unlock** () operation respectively.

In binary locks, at a particular point in time, only one transaction can hold a lock on the data item. No other transaction will be able to access the same data concurrently. Hence, Binary locks are very **simple** to apply but are **not used practically**.

1. **Shared Lock:-**Transaction can read only the data item values] It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction. It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.
2. **Exclusive Lock: -** Used for both read and write data item values] In the exclusive lock, the data item can be both reads as well as written by the transaction. This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

Time Stamping

Time stamp is a unique identifier created by DBMS that indicates relative starting time of a transaction. Whatever transaction we are doing it stores the starting time of the transaction and denotes a specific time.

This can be generated using a system clock or logical counter. This can be started whenever a transaction is started. Here, the logical counter is incremented after a new timestamp has been assigned.

Optimistic

It is based on the assumption that conflict is rare and it is more efficient to allow transactions to proceed without imposing delays to ensure serializability.

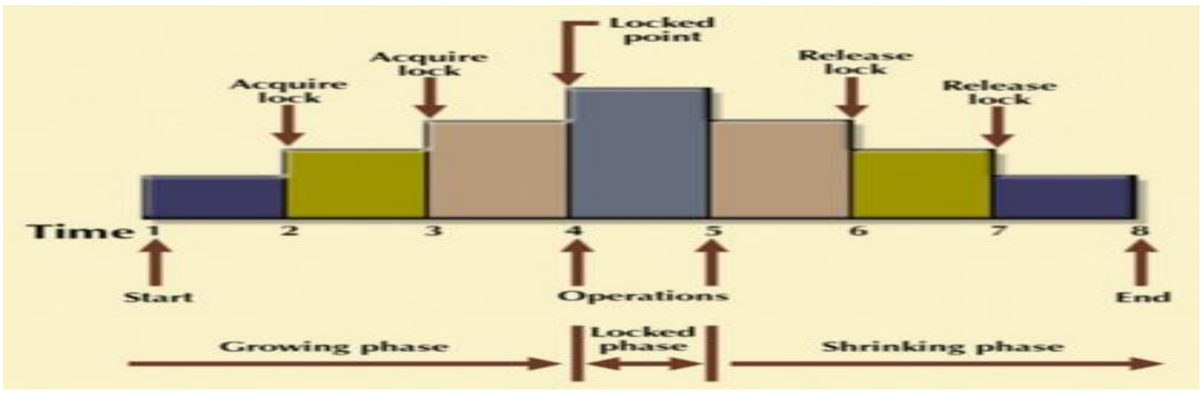
Concurrency Control Protocols

Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.

This locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.

In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.

The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.

Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

Example:

Suppose there are there transactions T1, T2, and T3.

T1 has entered the system at time 0010

T2 has entered the system at 0020

T3 has entered the system at 0030

Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:

Starvation is possible if the same transaction is restarted and continually aborted

(Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.)

Method used for DBMS Deadlock Recovery

The most common method to recover from a deadlock is to rollback one or more transactions until a no deadlock situation is reached. The **actions** that would be taken for deadlock recovery will be as follows:

- **Choice of the Victim Transaction**
- **Roll back**
- **Starvation**

Choice of the Victim Transaction

The transaction which is chosen for a rollback when a deadlock situation is reached is known as a **victim transaction** and the method is known as **victim selection**.

The victim may be selected based on any of the following **criterions**.

- Transaction may be selected randomly.
- Transaction will be selected depending on when the transaction started to run. That is the youngest transaction may be selected as the victim because it would have done the least amount of work.
- The transaction which needs a greater number of data items to be locked.
- The transaction which has done least number of data updates to the database because all the data written or changed in the database must be undone in the case of all a rollback.
- The transaction which would affect the roll back of a least number of other transactions. This means the victim transaction must lead to **least** amount of **cascade rollback**.

So, we can see that the victim transaction must be such a transaction which would incur the **minimum cost**.

Roll back

Once the victim transaction is selected the next important point to decide is how far this transaction should be rolled back. There are two ways in which the transaction can be rolled back. These are as follows:

- One of the simplest solutions is to **roll back** the transaction **entirely** and restart it later. This is referred to as a **full rollback**.
- On the other hand, some transactions can be rolled back to a **particular point** which is enough to break the deadlock. This type of rollback is known as a **partial rollback**. In such transactions additional information regarding the state of the current transaction execution, bookmark information, locks required etc. must be maintained by the system.

Starvation

Starvation is a condition which arises when one same transaction is always picked as the victim transaction. When the same transaction is rolled back again and again it will never complete its execution.

In other words, we can say that when a transaction cannot proceed for an indefinite period of time and other transactions in the system continue to execute normally, such a situation is called **starvation** or **Live Lock**.

For example, let us assume there are two transactions T1 and T2 which are trying to acquire a lock on a data item X. The scheduler grants the lock to T1. As a result, T2 has to wait for the lock. Now beforeT1 unlocks the data item X, a new transaction T3 also wants to acquire the lock on the data item X. Now, if the scheduler grants the lock to T3, T2 again has to wait for the lock. As a result, T2 has to wait indefinitely even if there is no deadlock. Hence, this situation is called starvation.

Solution to Starvation

In order to avoid the situation of starvation to implement DBMS Deadlock Recovery we can choose one of the following ways:

- A transaction must be picked as the victim transaction only a finite number of times. This can be done by maintaining an index to how many times a transaction has aborted and rolled back.
- The scheduler must follow the policy of first come first serve basis.

Transaction processing

Transaction processing means dividing information processing up into individual, indivisible operations, called transactions, that complete or fail as a whole; a transaction can't remain in an intermediate, incomplete, state (so other processes can't access the transaction's data until either the transaction has completed or it has been "rolled back" after failure). Transaction processing is designed to maintain database integrity (the consistency of related data items) in a known, consistent state.

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)
2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)

2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations\

1. R(X);
2. X = X - 500;
3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.