II B.Sc Computer Science
18BCS42C – DATABASE MANAGEMENT SYSTEM
UNIT-V

DATABASE SECURITY, INTEGRITY CONTROL: Security and Integrity threats – Defense mechanisms – Transaction and concurrency control mechanisms - ACID properties, Serializability and concurrency control, Lock based concurrency control (2PL, Deadlocks), Time stamping methods, optimistic methods, Database recovery management.

Definitions

- SECURITY: Protecting the database from unauthorized access, alteration or deletion.
- INTEGRITY: It refers to accuracy or validation of the data

THREATS To Security And Integrity

- A threat is any situation, event or personnel that will adversely effect the database security and smooth and efficient functioning of the organization.
- Threat to a database may be intentional or accidental.
- Given below are some database security threats
    - Data tampering
    - Eavesdropping and data theft
    - Falsifying User's identities
    - Password related threats
    - Unauthorized access to data
    - Lack of accountability

DEFENCE MECHANISMS

- Generally four levels of defence are recognized for a database security:
    - Physical security
    - Human factors
    - Operating system
    - Database system Data Security

REQUIREMENTS

- The basic security standards which technologies can assure are :
- CONFIDENTIALITY
    - Access control - Access to data is controlled by means of privileges, roles and user accounts.
    - Authenticated users – Authentication is a way of implementing decisions of whom to trust. It can be employ passwords, finger prints etc.
    - Secure storage of sensitive data – It is required to prevent data from hackers who could damage the sensitive data.

    - Privacy of communication - The DBMS should be capable of controlling the spread of confidential personal information from unauthorized people such as credit cards etc.
- INTEGRITY –
    - Integrity contributes to maintaining a secure database by preventing the data from becoming invalid and giving misleading results.
    - It consists of following aspects :
        - System and object privileges control access to applications tables and system commands so that only authorized users can change the data.
        - Integrity constraints are applied to maintain the correctness and validity of the data in the database.
        - Database must be protected from viruses so firewalls and anti-viruses should be used.
        - Ensures that access to the network is controlled and data is not vulnerable to attacks during transmission across network.
- AVAILABILITY
    - Data should always be made available for the authorized user by the secure system without any delays.
    - Availability is often thought of as a continuity of service assuring that database is available.
    - Denial of service attacks are attempts to block authorized users ability to access and use the system when needed.
    - It has number of aspects.
        - Ease of use – Resources managed by users for working with databases should be effectively managed so that it is available all the time to valid users.
        - Flexibility – Administrators must have all the relevant tools for managing user population.
        - Scalability - System performance should not get affected by the increase in number of users or processes which require services from system.
        - Resistance – User profiles must be defined and the resource used by any user should be limited.

IMPORTANT SECURITY FEATURES

- Views
- Authorization and controls
- User defined procedures or privileges
- Encryption procedures

Authorization

- AUTHORIZATION is a PROCESS OF PERMITTING USERS to perform certain operations on certain data objects in a shared database.
- For example: Let us consider the authorization that a salesperson undertakes;

Authorization	Customer Records	Order Records
READ	Y	Y
INSERT	Y	Y
MODIFY	Y	N
DELETE	N	N

- Where N stands for NO and Y stands for YES to authorization for salesperson
- To explain the concept of view, let us consider the example of a bank clerk who needs to know the names of customers of each branch but is not authorized to see specific loan information.
- The view is defined as follows: CREATE VIEW CUST_LOAN AS SELECT BRANCHNAME, CUSTOMER_NAME FROM BORROWER, LOAN Where BORROWER.LOAN_NO = LOAN.LOAN_NO; since the clerk is authorized to see this view so clerk can execute a query to see the result.
- SELECT * from CUST_LOAN; When the query processor translates the result into a query on actual base table in the database we obtain a query on BORROWER and LOAN tables.
- This permission must be checked on clerk's query processor begins.

DATABASE INTEGRITY

Constraints:

- It can be defined in 3 ways
  - Business constraints
  - Entity constraints
  - Referential constraints

BUSINESS CONSTRAINTS

- A value in one column may be constrained by value of some another or by some calculation or formulae.

ENTITY CONSTRAINTS

- Individual columns of a table may be constrained eg. Not null.

REFERENTIAL CONSTRAINTS

- Sometimes referred to as key constraints. Eg. Table two depends upon table one.

BENEFITS OF USING CONSTRAINTS

- Guaranteed integrity and consistency
- Defined as a part of table definition

- Applies across all applications
- Cannot be circumvented
- Application development and productivity
- Requires no special programming
- Easy to specify and maintain
- Defined once only

CONCURRENCY CONTROL

- What is it ? The coordination of simultaneous requests for the same data, from multiple users.
- Why is it important ? Simultaneous execution of transactions over a shared database may create a several data integrity and consistency problems.

THREE MAIN INTEGRITY PROBLEMS ARE

- Lost updates
- Uncommitted data
- Inconsistent retrievals

DATABASE RECOVERY

- The process of restoring database to a correct state in the case of failure.
- E.g.
    - System crashes
    - Media failures
    - Application software errors
    - Natural physical disasters
    - Carelessness

BASIC RECOVERY CONCEPTS

- Backup mechanism – it makes periodic backup copies of the database.
- Logging concept – that keeps the track of current state of transaction and the changes made in the database.
- Check pointing mechanism – that enables update to be made permanent.
- The choice of the best possible strategy depends upon the
    - Extent of damage that had occurred to the database.
    - If there has been a physical damage like disk crash then the last backup copy of the data is restored.
    - However if database has become inconsistent but not physically damaged then changes caused inconsistency must be undone.
    - It may also be required to redo some transactions so as to ensure that the updates are reflected in the database.

DBMS Concurrency Control: Timestamp & Lock-Based Protocols

What is Concurrency Control?

- **Concurrency Control** in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.

- It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.
- DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system.
- Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Potential problems of Concurrency

Here, are some issues which you will likely to face while using the DBMS Concurrency Control method:

- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
- **Incorrect Summary issue** occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Example

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

Concurrency Control Protocols

- Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.
- Following are the Concurrency Control techniques in DBMS:
  - Lock-Based Protocols
  - Two Phase Locking Protocol
  - Timestamp-Based Protocols
  - Validation-Based Protocols

Lock-based Protocols

- **Lock Based Protocols** in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock.
- Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.
- A lock is a data variable which is associated with a data item.
- This lock signifies that operations that can be performed on the data item.
- Locks in DBMS help synchronize access to the database items by concurrent transactions.
- All lock requests are made to the concurrency-control manager.
- Transactions proceed only once the lock request is granted.
- **Binary Locks:** A Binary lock on a data item can either lock or unlock states.
- **Shared/exclusive:** This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

- A shared lock is also called a Read-only lock.
- With the shared lock, the data item can be shared between transactions.
- This is because you will never have permission to update data on the data item.
- For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

- With the Exclusive Lock, a data item can be read as well as written.
- This is exclusive and can't be held concurrently on the same data item.
- X-lock is requested using lock-x instruction.
- Transactions may unlock the data item after finishing the 'write' operation.

- For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

3. Simplistic Lock Protocol

- This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation.
- Transactions may unlock the data item after finishing the 'write' operation.

4. Pre-claiming Locking

- Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process.
- In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

Starvation

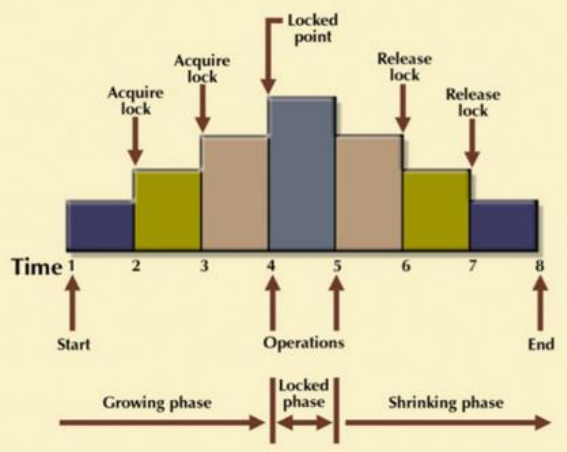
- Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.
- Following are the reasons for Starvation:
    - When waiting scheme for locked items is not properly managed
    - In the case of resource leak
    - The same transaction is selected as a victim repeatedly

Deadlock

- Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

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.

Strict Two-Phase Locking Method

- Strict-Two phase locking system is almost similar to 2PL.
- The only difference is that Strict-2PL never releases a lock after using it.
- It holds all the locks until the commit point and releases all the locks at one go when the process is over.

Centralized 2PL

- In Centralized 2 PL, a single site is responsible for lock management process.
- It has only one lock manager for the entire DBMS.

Primary copy 2PL

- Primary copy 2PL mechanism, many lock managers are distributed to different sites.
- After that, a particular lock manager is responsible for managing the lock for a set of data items.
- When the primary copy has been updated, the change is propagated to the slaves.

Distributed 2PL

- In this kind of two-phase locking mechanism, Lock managers are distributed to all sites.
- They are responsible for managing locks for data at that site.

- If no data is replicated, it is equivalent to primary copy 2PL.
- Communication costs of Distributed 2PL are quite higher than primary copy 2PL

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

Validation Based Protocol

- **Validation based Protocol** in DBMS also known as Optimistic Concurrency Control Technique is a method to avoid concurrency in transactions.
- In this protocol, the local copies of the transaction data are updated rather than the data itself, which results in less interference while execution of the transaction.
- The Validation based Protocol is performed in the following three phases:
  1. Read Phase
  2. Validation Phase
  3. Write Phase

Read Phase

- In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase

- In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase

- In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

Characteristics of Good Concurrency Protocol

An ideal concurrency control DBMS mechanism has the following objectives:

- Must be resilient to site and communication failures.
- It allows the parallel execution of transactions to achieve maximum concurrency.
- Its storage mechanisms and computational methods should be modest to minimize overhead.
- It must enforce some constraints on the structure of atomic actions of transactions.

Summary

- Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another.
- Lost Updates, dirty read, Non-Repeatable Read, and Incorrect Summary Issue are problems faced due to lack of concurrency control.
- Lock-Based, Two-Phase, Timestamp-Based, Validation-Based are types of Concurrency handling protocols
- The lock could be Shared (S) or Exclusive (X)
- Two-Phase locking protocol which is also known as a 2PL protocol needs transaction should acquire a lock after it releases one of its locks. It has 2 phases growing and shrinking.
- The timestamp-based algorithm uses a timestamp to serialize the execution of concurrent transactions. The protocol uses the **System Time or Logical Count as** a Timestamp.

ACID PROPERTIES

- A transaction is a unit of program that updates various data items
- To ensure the integrity of data during a transaction, the database system maintains the following properties.
- These properties are widely known as ACID properties:

- **Atomicity**:
    - This property ensures that either all the operations of a transaction reflect in database or none.
    - Let's take an example of banking system to understand this: Suppose Account **A** has a balance of 400$ & **B** has 700$. Account **A** is transferring 100$ to Account **B**. This is a transaction that has two operations a) Debiting 100$ from A's balance b) Creating 100$ to B's balance. Let's say first operation passed successfully while second failed, in this case A's balance would be 300$ while B would be having 700$ instead of 800$. This is unacceptable in a banking system. Either the transaction should fail without executing any of the operation or it should process both the operations.
    - The Atomicity property ensures that.
- **Consistency**:
    - To preserve the consistency of database, the execution of transaction should take place in isolation (that means no other transaction should run concurrently when there is a transaction already running).
    - For example account A is having a balance of 400$ and it is transferring 100$ to account B & C both. So we have two transactions here. Let's say these transactions run concurrently and both the transactions read 400$ balance, in that case the final balance of A would be 300$ instead of 200$. This is wrong. If the transaction were to run in isolation then the second transaction would have read the correct balance 300$ (before debiting 100$) once the first transaction went successful.
- **Isolation**:
    - For every pair of transactions, one transaction should start execution only when the other finished execution.
- **Durability**:
    - Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure.
    - The recovery-management component of database systems ensures the durability of transaction.

Serializability and Concurrency Control

DBMS Serializability

- When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state.
- Serializability is a concept that helps us to check which schedules are serializable.

- A serializable schedule is the one that always leaves the database in consistent state.

Serializable schedule

- A serializable schedule always leaves the database in consistent state.
- A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution.
- However a non-serial schedule needs to be checked for Serializability.
- A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions.
- A serial schedule doesn't allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Types of Serializability

- There are two types of Serializability.
    - Conflict Serializability
    - View Serializability

Conflict Serializability

- A Serial schedule doesn't support concurrent execution of transactions while a non-serial schedule supports concurrency.
- **Conflict Serializability** is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.
- A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

- Two operations are said to be in conflict, if they satisfy all the following three conditions:
    - Both the operations should belong to different transactions.
    - Both the operations are working on same data item.
    - At least one of the operation is a write operation.
- Lets see some examples to understand this:
- **Example 1:** Operation W(X) of transaction T1 and operation R(X) of transaction T2 are conflicting operations, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on same data item X, one of the operation in write operation.
- **Example 2:** Similarly Operations W(X) of T1 and W(X) of T2 are conflicting operations.
- **Example 3:** Operations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on same data item so these operations don't satisfy the second condition.

- **Example 4:** Similarly R(X) of T1 and R(X) of T2 are non-conflicting operations because none of them is write operation.
- **Example 5:** Similarly W(X) of T1 and R(X) of T1 are non-conflicting operations because both the operations belong to same transaction T1.

Conflict Equivalent Schedules

- Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Conflict Serializable check

- If a schedule is conflict Equivalent to its serial schedule then it is called Conflict Serializable schedule.
- Example of Conflict Serializability

  Lets consider this schedule:

T1	T2
R(A)	
R(B)	
	R(A)
	R(B)
	W(B)
W(A)	

- To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable**.
- Lets take another example:

T1	T2
R(A)	
	R(A)
	R(B)
	W(B)
R(B)	
W(A)	

  Lets **swap non-conflicting operations**:

  After swapping R(A) of T1 and R(A) of T2 we get:

T1	T2
	R(A)
R(A)	
	R(B)
	W(B)

R(B)
W(A)

After swapping R(A) of T1 and R(B) of T2 we get:

T1	T2
	R(A)
	R(B)
R(A)	
	W(B)
R(B)	
W(A)	

After swapping R(A) of T1 and W(B) of T2 we get:

T1	T2
	R(A)
	R(B)
	W(B)
R(A)	
R(B)	
W(A)	

- We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable**.

LOCK BASED CONCURRENCY CONTROL

Lock-based Protocols

- **Lock Based Protocols** in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock.
- Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.
- A lock is a data variable which is associated with a data item.
- This lock signifies that operations that can be performed on the data item.
- Locks in DBMS help synchronize access to the database items by concurrent transactions.
- All lock requests are made to the concurrency-control manager.
- Transactions proceed only once the lock request is granted.
- **Binary Locks:** A Binary lock on a data item can either lock or unlock states.
- **Shared/exclusive:** This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

- A shared lock is also called a Read-only lock.
- With the shared lock, the data item can be shared between transactions.
- This is because you will never have permission to update data on the data item.
- For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

- With the Exclusive Lock, a data item can be read as well as written.
- This is exclusive and can't be held concurrently on the same data item.
- X-lock is requested using lock-x instruction.
- Transactions may unlock the data item after finishing the 'write' operation.
- For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

3. Simplistic Lock Protocol

- This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation.
- Transactions may unlock the data item after finishing the 'write' operation.

4. Pre-claiming Locking

- Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process.
- In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

Starvation

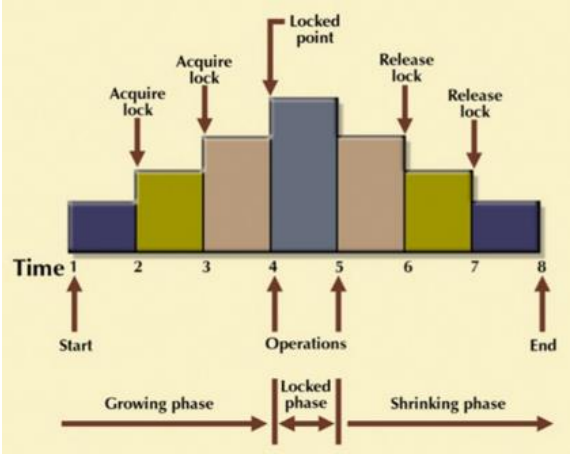
- Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.
- Following are the reasons for Starvation:
    - When waiting scheme for locked items is not properly managed
    - In the case of resource leak
    - The same transaction is selected as a victim repeatedly

Deadlock

- Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

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.

Strict Two-Phase Locking Method

- Strict-Two phase locking system is almost similar to 2PL.
- The only difference is that Strict-2PL never releases a lock after using it.
- It holds all the locks until the commit point and releases all the locks at one go when the process is over.

Centralized 2PL

- In Centralized 2 PL, a single site is responsible for lock management process.
- It has only one lock manager for the entire DBMS.

Primary copy 2PL

- Primary copy 2PL mechanism, many lock managers are distributed to different sites.
- After that, a particular lock manager is responsible for managing the lock for a set of data items.
- When the primary copy has been updated, the change is propagated to the slaves.

Distributed 2PL

- In this kind of two-phase locking mechanism, Lock managers are distributed to all sites.
- They are responsible for managing locks for data at that site.
- If no data is replicated, it is equivalent to primary copy 2PL.
- Communication costs of Distributed 2PL are quite higher than primary copy 2PL

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.

DATABASE RECOVERY MANAGEMENT

- The entire DBMS is a very complex structure with multiple transactions being performed and carried out every second.
- The toughness and strength of a system depend not only on the complex and secured architecture of a system but also in the way how data are managed and maintained in the worst cases.
- If the underlying architecture fails or crashes, then there must be some techniques and procedures by which the lost data during a transaction gets recovered.

DATA RECOVERY

- It is the method of restoring the database to its correct state in the event of a failure at the time of the transaction or after the end of a process.
- The storage of data usually includes four types of media with an increasing amount of reliability:
    - The main memory
    - The magnetic disk
    - The magnetic tape
    - The optical disk.
- Many different forms of failure can affect database processing and/or transaction, and each of them has to be dealt with differently.
- Some data failures can affect the main memory only, while others involve non-volatile or secondary storage also.
- Among the sources of failure are:
    - Due to hardware or software errors, the system crashes, which ultimately resulting in loss of main memory.
    - Failures of media, such as head crashes or unreadable media that results in the loss of portions of secondary storage.
    - There can be application software errors, such as logical errors that are accessing the database that can cause one or more transactions to abort or fail.
    - Natural physical disasters can also occur, such as fires, floods, earthquakes, or power failures.
    - Carelessness or unintentional destruction of data or directories by operators or users.

    - Damage or intentional corruption or hampering of data (using malicious software or files) hardware or software facilities.
- Whatever the grounds of the failure are, there are two principal things that you have to consider:
    - Failure of main memory, including that database buffers.
    - Failure of the disk copy of that database.

Recovery Facilities

- Every DBMS should offer the following facilities to help out with the recovery mechanisms:
    - **Backup mechanism** makes backup copies at a specific interval for the database.
    - **Logging facilities** keep tracing the current state of transactions and any changes made to the database.
    - **Checkpoint facility** allows updates to the database for getting the latest patches to be made permanent and keep secure from vulnerability.
    - **Recovery manager** allows the database system for restoring the database to a reliable and steady-state after any failure occurs.

Crash Recovery

- DBMS is a highly complex system with hundreds of transactions being executed every second.
- The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software.
- If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification

- To see where the problem has occurred, we generalize a failure into various categories, as follows −

Transaction failure

- A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further.
- This is called transaction failure where only a few transactions or processes are hurt.
- Reasons for a transaction failure could be –
    - **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.
    - **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

- There are problems external to the system that may cause the system to stop abruptly and cause the system to crash.
- For example, interruptions in power supply may cause the failure of underlying hardware or software failure.
- Examples may include operating system errors.

Disk Failure - Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Storage Structure

- We have already described the storage system.
- In brief, the storage structure can be divided into two categories –
    - **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
    - **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Recovery and Atomicity

- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.
- Transactions are made of various operations, which are atomic in nature.
- But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.
- When a DBMS recovers from a crash, it should maintain the following –
    - It should check the states of all the transactions, which were being executed.
    - A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
    - It should check whether the transaction can be completed now or it needs to be rolled back.
    - No transactions would be allowed to leave the DBMS in an inconsistent state.
- There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction –

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Log-based Recovery

- Log is a sequence of records, which maintains the records of actions performed by a transaction.
- It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.
- Log-based recovery works as follows –
  - The log file is kept on a stable storage media.
  - When a transaction enters the system and starts execution, it writes a log about it.

    $<T_n$, Start$>$
  - When the transaction modifies an item X, it write logs as follows

    $<T_n, X, V_1, V_2>$
  - It reads T_n has changed the value of X, from V_1 to V_2.
  - When the transaction finishes, it logs −

    $<T_n$, commit$>$
- The database can be modified using two approaches –
  - **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
  - **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent Transactions

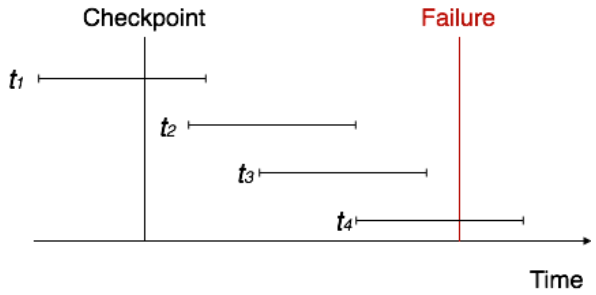
- When more than one transaction are being executed in parallel, the logs are interleaved.
- At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering.
- To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

- Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system.
- As time passes, the log file may grow too big to be handled at all.
- Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.
- Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

- When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <T_n, Start> but no commit or abort log found, it puts the transaction in undo-list.
- All the transactions in the undo-list are then undone and their logs are removed.
- All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

REFERENCES

1. Silberschatz A., Korth H. and Sudarshan S., "Database System Concepts", Tata McGraw Hill, 2011.
2. Elmasri R. and Navathe S.B., "Fundamentals of Database Systems", Pearson Education, 2016.
3. www.w3schools.com
4. www.guru99.com
5. www.tutorialspoint.com
6. www.beginnersbook.com
7. www.ecomputernotes.com

----------**********----------